



# Programación con JavaScript II

# Proyecto final

# Desarrollo del proyecto

**Nombre del proyecto:** Aplicación presupuesto

## Objetivo

Poner en práctica algunos de los conceptos más utilizados en el desarrollo con JavaScript aprendidos en el curso Programación con JavaScript II.

## Descripción

La aplicación busca recetas y las muestra.

## Instrucciones detalladas

Con la intención de que amplies tu portafolio de experiencias en programación, y de que practiques los conceptos adquiridos, desarrolla una aplicación web frontend que busque en una API remota las recetas que coincidan con el patrón introducido por el usuario. Puedes ver una muestra de la página funcionando en: <https://forkify-v2.netlify.app/>

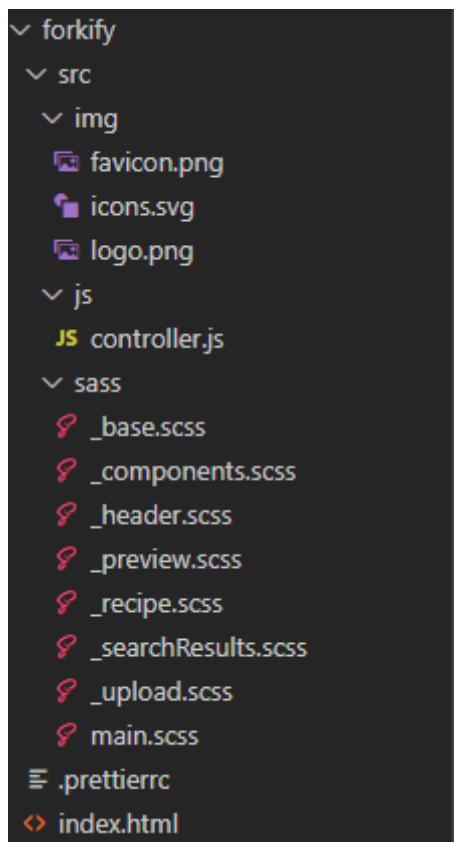
En este proyecto utilizarás:

- Clases
- Modularización
- AJAX
- Consumo de API
- Asincronía
- Promesas
- Manejo de eventos y manipulación del DOM

En el curso encontrarás cuatro avances, uno por semana, para que construyas tu proyecto final, el cual deberás entregar en la semana 5. Aunque estos avances no son evaluables, se sugiere que los realices para mayor organización.

# Avance 1

1. Toma la base del proyecto que se incluye en la sección Archivos adjuntos y ábrelo en Visual Studio Code.
2. Una vez que tengas tu código, analiza que tenga todos estos componentes:



3. Para este proyecto se utilizará la herramienta **parcel**, la cual procesa todo el código de la aplicación web y recorre todos los archivos enlazándolos y generando una nueva colección de archivos más apropiados para el navegador.
4. Abre una nueva terminal e inicializa un nuevo proyecto con la instrucción de npm. Utiliza la siguiente configuración:
  - a) En nombre del proyecto, coloca forkify si es que no aparece por defecto.
  - b) El siguiente parámetro a configurar es el *autor*, coloca tu nombre.
  - c) Todas las otras opciones déjalas como están por defecto.

## Avance 1

5. Podrás visualizar que se crea un archivo **package.json**.
  - a) Dentro de este archivo, modifica la línea 5, en la que aparece la propiedad **main** y el valor **index.js**. Modifícalo para que la propiedad sea **source** y en el valor coloca **indx.html**.
  - b) Dentro de los **scripts**, elimina la propiedad **text** y coloca como propiedad **start** y como valor coloca **parcel index.html**.
  - c) Crea una nueva propiedad **build** con el valor **parcel build index.html**.
6. Instala con npm **parcel V2** con todas sus dependencias.
7. Inicializa el parcel con el comando **npm start**. En tu consola se debe desplegar un mensaje similar al siguiente:

```
root@cdctecmi:/#npm start
> forkify@1.0.0 start
> parcel index.html

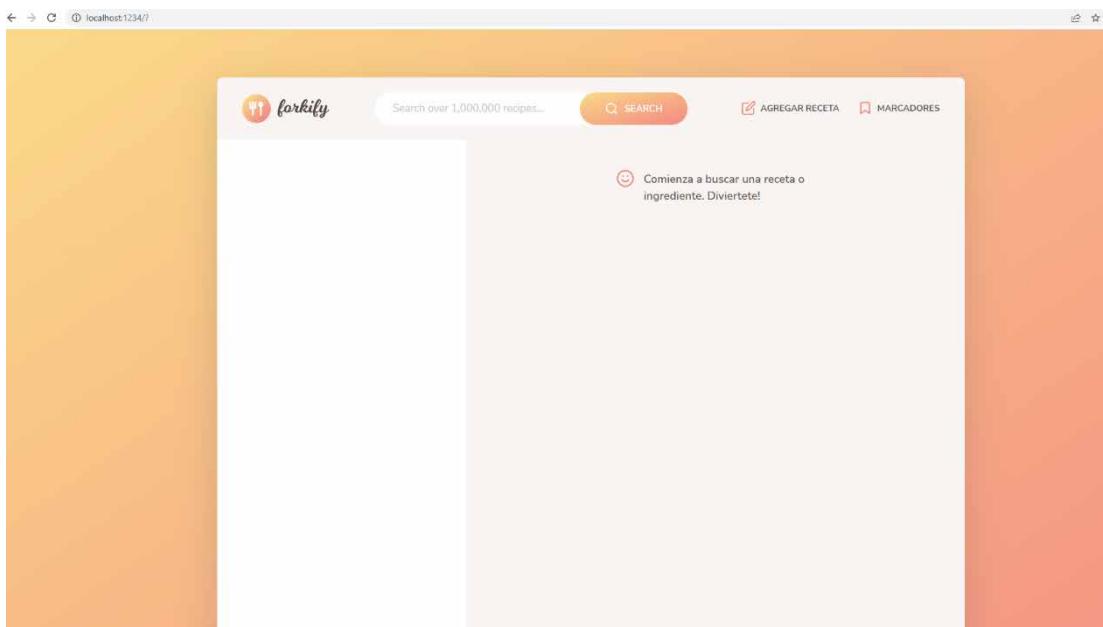
Server running at http://localhost:1234
Built in 13ms
```

8. Cuando ejecutas esta instrucción, observa que también deben aparecer las siguientes líneas en tu archivo json:

```
"devDependencies": {
  "@parcel/transformer-sass": "^2.8.0",
  "parcel": "^2.8.0"
}
```

9. En caso de que no aparezcan, analiza que no se te haya presentado ningún error en la terminal.
10. Valida también que hayan aparecido en tu explorador de archivos los directorios **node\_modules** y **dist**.
11. Valida que en la carpeta **dist** se encuentren los archivos listos para ser enviados al navegador.
12. En el archivo **index.html**, en la instrucción que manda a llamar al controller.js, coloca en la referencia que es una llamada al controller de tipo módulo.
13. Valida que tu proyecto esté funcionando, siguiendo el link generado <http://localhost:1234>.
14. Por ahora tu aplicación debe tener el siguiente aspecto:

## Avance 1



15. Para validar que puedas obtener información de la API, realiza la siguiente prueba:
  - a. Crea una función asíncrona llamada `showRecipe`. En el `try`, crea una constante `resp`, iguálala a un `await` para que espere el resultado de la función de búsqueda (`fetch`), la cual devolverá una promesa, y pásale como parámetro la siguiente URL válida para la API: <https://forkify-api.herokuapp.com/api/v2/recipes/5ed6604591c37cdc054bc886>.
  - b. Una vez que tengas el resultado, es necesario convertirlo a JSON. Para ello, declara una constante `data`, que será igual a `resp`, utilizando el método `json()` (no olvides utilizar el `await`).
  - c. Envía a la consola las constantes `resp` y `data`.
  - d. En caso de error, vas a enviar en una alerta el error generado.
  - e. Invoca a la función `showRecipe`.
  - f. ¿Cuál es el contenido de `resp`?
  - g. ¿Cuál es el contenido de `data`?
  - h. ¿Qué sucede si le pasas esta URL a la función?  
<https://forkify-api.herokuapp.com/api/v2/recipes/5ed6604591c37cdc054bc886zzz>
  - i. ¿Es la misma respuesta?
  - j. Para poder visualizar los datos que se necesitan desplegar en la pantalla, crea la variable `recipe` del tipo objeto e iguálala a `data.data`.

## Avance 1

k. Ahora desestructura recipe de la siguiente manera:

```
recipe = {
  id: recipe.id,
  title: recipe.title,
  publisher: recipe.publisher,
  sourceUrl: recipe.source_url,
  image: recipe.image_url,
  servings: recipe.servings,
  cookTime: recipe.cooking_time,
  ingredients: recipe.ingredients,
};
```

l. Imprime en la consola el contenido.

16. Como puedes notar, no se visualiza el contenido de la consulta en la página. Para hacerlo, ve al archivo **index.html** y copia desde la línea 161 a 253 al archivo **controller.js** dentro de la función en la variable **markup**. Utiliza **template string**.
17. Sustituye los valores fijos (datos duros) por los valores que contienen el objeto recipe, por ejemplo, el source de las imágenes en la línea:

```

sustitúyelo por:


La línea:
<span>Pasta with tomato cream sauce</span>
Sustitúyela por:
<span>${recipe.title}</span>

La línea:
<span class="recipe__info-data recipe__info-data--minutes">45</span>
Sustitúyela por:
<span class="recipe__info-data recipe__info-data--minutes">${recipe.cookTime}</span>

La línea:
<span class="recipe__info-data recipe__info-data--people">4</span>
Sustitúyela por:
<span class="recipe__info-data recipe__info-data--people">${recipe.servings}</span>

La línea:
<span class="recipe__publisher">The Pioneer Woman</span>. Please check out
Sustitúyela por:
<span class="recipe__publisher">${recipe.publisher}</span>. Please check out

La línea:
href="http://thepioneerwoman.com/cooking/pasta-with-tomato-cream-sauce/"
Sustitúyela por:
href="${recipe.sourceUrl}"
```

## Avance 1

18. Como puedes observar, aún se visualiza el mensaje inicial. Para quitarlo, utiliza el método innerHTML sobre recipeContainer y asignale una cadena vacía.
19. Para visualizar el nuevo contenido en la página, utiliza el método insertAdjacentHTML en el recipeContainer y pásale como parámetros afterbegin y la variable markup.
20. Para visualizar los ingredientes:
  - a. Recorre el arreglo **recipe.ingredients** y aplícale la función map para que se puedan visualizar cada uno de los elementos. Pásale como argumentos al **map** la función flecha img que envuelve al elemento de la lista con clase **recipe\_ingredient**.
  - b. Como el resultado se va a mostrar en el documento html, es necesario aplicar la función **join**.
  - c. Tu código debería quedar similar al siguiente:

```

${recipe.ingredients
    .map(ing => {
  return `
    <li class="recipe__ingredient">
      <svg class="recipe__icon">
        <use href="src/img/icons.svg#icon-check"></use>
      </svg>
      <div class="recipe__quantity">${ing.quantity}</div>
      <div class="recipe__description">
        <span class="recipe__unit">${ing.unit}</span>
        ${ing.description}
      </div>
    </li>
  `;
})
.join(")}
</div>

```

- d) Elimina todos los elementos con clase **recipe\_ingredient** del documento.
21. Como puedes notar, no se visualizan correctamente los íconos. En la parte superior del archivo **controller.js**, importa los íconos desde el archivo **../img/icons.svg**.
22. Sustituye en todos los lugares que aparezca **src/img/icons.svg** por **\${icons}**.
23. En la página terminada se ve un spinner que gira en lo que se carga la imagen de una receta. Para crearlo, toma el div con la clase **spinner** del archivo **index.html** y llévalo al archivo **controller.js**:
  - a. Crea una función que se llame **renderSpinner** que recibirá un elemento padre (**parentEl**). Dentro del cuerpo de la función, crea la variable markup y asignale el último código copiado (aplica el punto 22).
  - b. Utiliza el método **insertAdjacentHTML** al parámetro **parentEl**. Pasa como parámetro '**afterbegin**' y la variable markup.
  - c. Antes de esta última línea, borra el contenido del elemento padre con innerHTML, asignándole una cadena vacía.
  - d. Llama a la función **renderSpinner** desde el cuerpo de la función **showRecipe**. Se recomienda que lo pongas inmediatamente después del try y pásale como parámetro **recipeContainer**.

# Avance 1

## Consideraciones

**Instructor:**

En esta semana es importante validar que el aprendedor sepa utilizar NPM y la terminal de Visual Studio Code. Además, es esencial que sepa analizar el contenido de los archivos y modificar el archivo JSON.

Se sugiere que se fomente el uso de GIT y que el aprendedor pueda manipularlo.

**Aprendedor:**

Asegúrate de tener NPM en tu computadora para que puedas realizar las actividades del proyecto.

Una vez que tengas todos tus cambios realizados, recuerda subirlos a tu repositorio GIT.

## Avance 2

En este avance se van a crear los eventos que van a cargar dinámicamente los datos en el html y a manejar errores, además de modularizar la aplicación con una arquitectura.

Lo primero que se requiere es pasar el id de cada receta al URL y que, cada que cambie el **id**, cambie el producto del **div** principal **recipe**.

1. Agrega al final del archivo **controller.js** un evento con **addEventListener** a la ventana y pásale como parámetros el evento **hashchange** y **showRecipe**.
2. Con esto, ahora se llamará a la función **showRecipe** cada vez que el hash cambie, por tanto, se puede eliminar la llamada a la función **showRecipe**.
3. Para poder probar la funcionalidad, en el archivo **index.html**, en el área de resultados, crea un par de enlaces con hash de diferentes recetas, parecidos a estos ejemplos:

```
<a href="#5ed6604591c37cdc054bc886">Recipe 1</a>
<a href="#5ed6604591c37cdc054bccde">Recipe 2</a>
```

4. Si observas en la URL, cuando haces clic en cada uno de los enlaces, el hash cambia, sin embargo, no pasa nada en la página. Para que esto se vea reflejado en la página, haz lo siguiente:
  - a. En la función **showRecipe**, dentro del try, declara una variable **id** y asígnale el método **window.location.hash**.
  - b. Imprime en la consola el resultado.
  - c. Como no se requiere leer desde el primer carácter, agrégale el método slice(1) al final de la función del inciso a.
  - d. Modifica en la función de búsqueda la URL estática para que ahora reciba la URL dinámico, pasándole la variable \${id}.
5. Hasta aquí todo parece funcionar, pero si se copia la URL completa y la pones en una ventana del navegador nueva, no va a tener el funcionamiento adecuado. Para crear la funcionalidad esperada, haz lo siguiente:
  - a. Además del evento agregado al final del archivo **controller.js**, crea uno adicional, pero pásale como parámetro el evento **load** y **showRecipe**.
  - b. Como las funciones son iguales y se podrían tener más, optimiza el código creando un arreglo con los elementos 'hashchange', 'load' y aplícale el método **foreach**, el cual recibirá como parámetro una función flecha que, a su vez, recibe el evento (ev) y en el cuerpo de la función solo agrégale el evento **addEventListener** que recibirá como parámetros el evento (**ev**) y la función del controlador **showRecipe**.

## Avance 2

6. Ahora, si no le pasas ningún id, te muestra un error y se queda esperando eternamente. Para solucionarlo, después de la declaración de la variable id, valida si la variable id no existe, cuando esto suceda, solamente dale un return.

Es momento de organizar el código desarrollado. Para ello, se va a utilizar la arquitectura MVC, dicho en otras palabras, vamos a modularizar el código.

1. Modelo: aquí se escribirá el modelo completo y tendrá objetos para la receta, la búsqueda y los marcadores y una función para cargar las recetas. Para realizar esta funcionalidad, haz lo siguiente:

- En la carpeta js, crea el archivo model.js y dentro del cuerpo del objeto agrega lo siguiente:
  - Un objeto state que, a su vez, tendrá dentro de él un objeto recipe vacío.
  - Exporta este estado para que lo pueda visualizar el controlador.
  - Crea la función asíncrona loadRecipe que será la encargada de obtener la receta de la API.
  - Exporta la función loadRecipe.
- Toma el archivo controller.js y coloca dentro del cuerpo de la función loadRecipe lo siguiente:
  - Dentro del try:
    - La declaración de **res**.
    - La declaración de data.
    - La validación del estado de res.
    - El objeto recibe:
      - La desestructuración de recipe.
      - La impresión en consola de recipe.
  - Dentro del catch que recibe como parámetro el error (err):
    - Envía a una alerta el error.
  - Dentro del modelo, realiza las siguientes modificaciones para ajustar el código:
    - Pasa como parámetro a la función asíncrona la variable id.
    - Declara como const el objeto recipe.
    - Antepón el objeto state al recipe desestructurado.
    - En el console.log, también registra a recipe como objeto de state.
- En el archivo **controller.js**, realiza las siguientes modificaciones:
  - Importa todas las funciones como **model** desde el archivo model.js.
  - Manda a llamar la función **model.loadRecipe** con id como parámetro justo en donde se cortó el código que llevas al archivo **model.js** (como es una función asíncrona, recuerda colocar el await).
  - Declara una constante **recipe** desestructurada y asígnale **model.state**.

## Avance 2

- c. Prueba hasta aquí tu código, este debe continuar trabajando como hasta antes de la modularización.
2. Para separar la vista del controlador, haz lo siguiente:
- Dentro de la carpeta js, ahora crea la carpeta **views** y dentro de ella crea el archivo RecipeView.js para la vista de las recetas.
  - Dentro del archivo **RecipeView**, realiza lo siguiente:
    - Crea la clase **RecipeView** con las siguientes características:
      - Declara un elemento privado **parentElement** y trae del archivo **controller.js** el valor del **recipeContainer**.
      - Declara el elemento privado **data**.
      - Declara el método **render** y pásale como parámetro **data**.
      - Dentro de **render**, declara **this.#data** e iguálalos al parámetro que se acaba de recibir.
      - Crea la constante **markup** e instancia el método **generateMarkup** con **this.#generateMarkup**.
      - Declara un método privado **generateMarkup** y coloca dentro de su cuerpo todo lo que tenga la declaración de la variable **markup** en el archivo controller.js, pero regrésalo en un return.
      - Cambia todas las variables **recipe.x** por **this.#data.x**.
      - Quita el **innerHTML** y el **insertAdjacentHTML** de **#generateMarkup**.
      - Para el limpiar el **parentElement**:
        - Crea el método **#clean** y limpia al elemento utilizando **this.#parentElement.innerHTML = "";**
        - Instáncialo desde el **render** con **this.#clear**;
    - Para el **insertAdjacentHTML** puedes utilizar la misma instrucción en el **render**, solo referenciando al **parentElement** de la siguiente manera: **this.#parentElement.insertAdjacentHTML('afterbegin', markup);**
    - Trae del archivo **controller.js** la función **renderSpinner** y colócala dentro de la clase **RecipeView**:
      - Conviértela en un método público.
      - Elimina el parámetro que se le pasa.
      - Cambia el **parentEl** por **this.#parentElement**.
      - Corta la línea que importa los íconos del archivo **controller.js** y pégala en la parte superior de **RecipeView.js**.
      - Modifica la instancia del **renderSpinner** indicando que proviene de **renderView** y elimina el parámetro.

## Avance 2

- ii. Exporta por defecto no la clase directamente, sino una instancia de la clase utilizando **new RecipeView** para mantener la privacidad de sus elementos.
  - c. Dentro del archivo **controller.js**, realiza las siguientes modificaciones:
    - i. Importa **recipeView** desde **./view/recipeview.js**.
    - ii. Cambia el nombre de la función **showRecipe** por **controlRecipes**.
    - iii. Elimina la instrucción **const { recipe } = model.state;**, ya no se considera necesaria.
    - iv. Instancia **recipeView.render** y pásale como parámetro **model.state.recipe** en el lugar donde se cortó el código que se colocó en la vista.
  - d. Es momento de corregir un aspecto visual para que en los ingredientes, en lugar de presentar 0.5 coups, diga  $\frac{1}{2}$  coups. Para ello:
    - i. Instala con npm la librería **fractional**.
    - ii. En el archivo **recipeView**, importa **Fraction** desde fractional utilizando la desestructuración.
    - iii. Instancia **Fraction** en el campo cantidad de la siguiente manera:
      - 1) Valida con un operador ternario que exista un valor para la cantidad, si existe, entonces:
 

```
new Fraction(ing.quantity).toString();
```
      - 2) Si no existe, envía una cadena vacía.
  - e. Prueba tu aplicación y debería seguir funcionando correctamente.
3. Crea un archivo de configuración que contendrá todas las variables que se utilizarán durante todo el proyecto y que son responsables de definir algunos datos importantes:
- a. Crea el archivo **config.js**.
  - b. Declara y exporta la constante **API\_URL** y asígnale la URL de la API que se está consumiendo (hasta antes del **id**).
  - c. En el archivo **model.js**, importa la constante **API\_URL** (colócalo entre llaves, ya que se van a importar otras variables) desde el archivo **config.js**.
  - d. Modifica la variable **res** y utiliza la variable **API\_URL**.
4. Crea un archivo en donde se almacenen funciones que se reutilizarán una y otra vez en el proyecto:
- a. En la carpeta **js**, crea un archivo **helpers.js** y agrega lo siguiente:
    - i. Toma la declaración de **res**, la declaración de **data** y la validación del parámetro **ok** de **res** del archivo **model.js** de **loadRecipe** y llévalas al nuevo archivo **helpers.js**.
    - ii. Crea y exporta la función asíncrona **getJSON** que recibe un URL como parámetro
    - iv. La definición de **data** queda igual.
    - v. La validación queda igual.
    - vi. Haz que la función retorne el valor de **data**.

## Avance 2

- vii. En el catch, como lo hemos hecho en otras funciones, recibe el parámetro error y utiliza la instrucción **throw error**.
- viii. Dentro del archivo **model.js**, sustituye la declaración de la función asíncrona **data** (no olvides el await), pásale como parámetro la URL armada con el **API\_URL** y el **id**.
- ix. Optimiza el manejo de errores en la función **loadRecipe** y, en lugar de mandar un alert, envía un mensaje de error personalizado a la consola, podría ser del tipo:

```
console.log(` ${err} ⚡⚡⚡`);
```

- x. Prueba el nuevo funcionamiento del error.
- b. Mueve la función **timeout**, que básicamente lo que hace es devolver una promesa que rechazará después de cierta cantidad de tiempo:
  - i. Modifica la función **getJSON**. Dentro de su cuerpo, declara una constante **fetchPro** y asignale la función de búsqueda que recibe el URL como parámetro.
  - ii. Modifica la función **res** para que quede de esta manera:

```
const res = await Promise.race([fetchPro,timeout(TIMEOUT_SEC)]);
```

  - iii. Como podrás ver, hace uso de un **TIMEOUT\_SEC**, este debes declararlo en el archivo **config.js** y asignarle el valor de 5.

### Consideraciones

**Instructor:** Posiblemente a algunos de los aprendedores les cueste trabajo utilizar las funciones flecha, refuerza este tema.

**Aprendedor:** Asegúrate de saber utilizar las funciones flecha, ya que son las más fáciles de utilizar en los proyectos.

## Avance 3

En este avance se van a crear procedimientos para escuchar eventos y manejar eventos en la arquitectura MVC usando el patrón editor-suscriptor y manejo de errores.

1. Por el momento se han desarrollado algunas funcionalidades para poder “escuchar” el cambio de **hash** y los eventos de carga en el controlador, sin embargo, eso no tiene mucho sentido pues debería estar dentro de la vista. Para ello, realiza lo siguiente:
  - a. Corta del archivo **controller.js** los **listener** de **hash** y de carga que se encuentran al final del archivo.
  - b. En el archivo **recipeView.js** crea el método **addHandlerRender** dentro de la clase **RecipeView** y debajo del **renderSpinner**. Envíale como parámetro un **handler** y en el cuerpo, pega el código copiado del archivo controller.js
  - c. Modifica los parámetros que recibe el listener, pásale el evento (ev) y el handler recibido.
  - d. En el archivo controller, crea la función **init**:
    - i. Dentro del cuerpo instancia el método addHandlerRender recién creado y pásale como parámetro **controlRecipes**.
    - ii. Invoca a la función **init**.
2. Ahora se implementará el manejo de errores como se hace en el mundo real, es decir, mostrar en la pantalla de la aplicación el error que se presentó para que el usuario sepa lo que está ocurriendo en la aplicación. Para hacerlo realiza lo siguiente:
  - a. En la clase **RecipeView** crea un nuevo método responsable de mostrar el mensaje de error:
    - i. Nómbralo **renderError** y pásale como parámetro **message = this.\_errorMessage**.
    - ii. Crea la variable **\_errorMessage** y asígnale el siguiente texto: 'We could not find that recipe. Please try another one!'
    - iii. En el cuerpo de la función declara la constante **markup**.
    - iv. Colócale el código del **div** con clase **error** que se encuentra en el archivo **index.html**.
    - v. Modifica el archivo iconos del **href** y colócale la variable **\${icons}**.
    - vi. Modifica el mensaje del párrafo y ponle la variable **\${message}**.
    - vii. Llama al método privado **#clear** y el **insertAdjacentHTML** como en los otros **métodos**.
  - b. Manda a llamar el **errorRender** desde el catch de la función **controlRecipes** y borra el **console.log** que se utilizaba.
  - c. Para poder propagar el error generado en **loadRecipe**, utiliza la función **throw err** en el **catch**.

## Avance 3

3. Parecido al anterior, genera un manejador de mensajes exitosos. Utiliza la base del método **renderError**:

- Copia el método **renderError** y renómbralos como **renderMessage**.
- Pásale como parámetro **message=this.\_message**.
- Cambia el ícono de **icon-alert-triangle** a **icon-smile**.

Crea la funcionalidad para la búsqueda y presentación de resultados. Realiza las siguientes actividades:

1. En el archivo **model.js** crea la siguiente función:

- Declara y exporta una función asíncrona llamada **loadSearchResults** que recibirá como parámetro una búsqueda(query).
- En el cuerpo de la función crea un bloque try y catch.
- Dentro del cuerpo del try, declara una constante **data** e indícale que espere a la promesa getJSON que recibe como parámetro la cadena  
``${API_URL}/?search=${query}``.
- Del objeto **data.data.recipes** (que es la matriz con todos los objetos), crea una matriz con los nuevos objetos utilizando el siguiente código:

```
data.data.recipes.map(rec => {
  return {
    id: rec.id,
    title: rec.title,
    publisher: rec.publisher,
    image: rec.image_url,
  };
});
```

- En el catch recibe el error como parámetro e imprime en la consola la cadena **console.log(` \${err} ⚡⚡⚡⚡`)**.
  - Lanza el error nuevamente para que pueda ser utilizado por el controlador.
  - Para probar, llama a la función recién creada **loadSearchResults** con pizza como parámetro.
- Para almacenar esta información, realiza los siguientes cambios:
    - En el objeto **state**, crea una nueva propiedad llamada **search** y como valor tendrá un objeto que a su vez tendrá dos propiedades:
      - La propiedad **query** que tendrá inicialmente la cadena vacía como valor.
      - La propiedad **results** que tendrá como valor un arreglo vacío.

# Avance 3

- b. En la función **loadSearchResults** realiza estos cambios:

- Asigna la variable **query** a state.search.results.
- Asigna a **state.search.results** la matriz con los nuevos objetos.
- Envía a la consola los resultados, deberían ser similares a esto:

```
(59) [{}]
 0: {id: 'Sed6604591c77cd054bc090', title: 'Cauliflower Pizza Crust (with BBQ Chicken Pizza)', publisher: 'Closet Cooking', image: 'http://Forkify-api.herokuapp.com/images/BBQChickenPizzewithCauliflowerCrust5004699695624ce.jpg'}
 1: {id: 'Sed6604591c77cd054bc091', title: 'BBQ Chicken Pizza', publisher: 'Closet Cooking', image: 'http://Forkify-api.herokuapp.com/images/pizzaBBQChicken.jpg'}
 2: {id: 'Sed6604591c77cd054bc094', title: 'Homemade Pizza', publisher: 'Simply Recipes', image: 'http://Forkify-api.herokuapp.com/images/howtopizza2007225979.jpg'}
 3: {id: 'Sed6604591c77cd054bc097', title: 'How to Grill Pizza', publisher: 'Simply Recipes', image: 'http://Forkify-api.herokuapp.com/images/howtopizza20062006e0e1b.jpg'}
 4: {id: 'Sed6604591c77cd054bc098', title: 'Closet Cooking', publisher: 'All Recipes', image: 'http://Forkify-api.herokuapp.com/images/391236b085.jpg'}
 5: {id: 'Sed6604591c77cd054bc095d', title: 'Veggie Pizza', publisher: 'All Recipes', image: 'http://Forkify-api.herokuapp.com/images/7988559586.jpg'}
 6: {id: 'Sed6604591c77cd054bc0971', title: 'Greek Pizza', publisher: 'A Spicy Perspective', image: 'http://Forkify-api.herokuapp.com/images/451180x1804f4a.jpg'}
 7: {id: 'Sed6604591c77cd054bc0973', title: 'Valentine Pizza', publisher: 'My Baking Addiction', image: 'http://Forkify-api.herokuapp.com/images/PizzaDip210f14f405.jpg'}
 8: {id: 'Sed6604591c77cd054bc0910', title: 'Pizza Dips', publisher: 'BBC Good Food', image: 'http://Forkify-api.herokuapp.com/images/1649637_MEDIUMfc.jpg'}
 9: {id: 'Sed6604591c77cd054bc0990', title: 'Pizza Casserole', publisher: 'All Recipes', image: 'http://Forkify-api.herokuapp.com/images/5108898c05.jpg'}
 10: {id: 'Sed6604591c77cd054bc0966', title: 'Pizza Pinwheels', publisher: 'All Recipes', image: 'http://Forkify-api.herokuapp.com/images/56728fe.jpg'}
 11: {id: 'Sed6604591c77cd054bc0971', title: 'Pesto Pizza', publisher: 'All Recipes', image: 'http://Forkify-api.herokuapp.com/images/104254d419.jpg'}
 12: {id: 'Sed6604591c77cd054bc0958', title: 'Hummus Pizza', publisher: 'All Recipes', image: 'http://Forkify-api.herokuapp.com/images/636003d2a3.jpg'}
 13: {id: 'Sed6604591c77cd054bc0964', title: 'Puff pizza tart', publisher: 'BBC Good Food', image: 'http://Forkify-api.herokuapp.com/images/679637_MEDIUM765c.jpg'}
 14: {id: 'Sed6604601c77cd054bd0bc0', title: 'Pizza Monkey Bread', publisher: 'What's Gaby Cooking', image: 'http://Forkify-api.herokuapp.com/images/PizzaMonkeyBread67f8.jpg'}
 15: {id: 'Sed6604591c77cd054bc092', title: 'Veggi-Prosciutto Pizza', publisher: 'Epicurious', image: 'http://Forkify-api.herokuapp.com/images/5115060ff4cb.jpg'}
```

3. Elimina la llamada a la función para realizarla ahora desde el controlador, procede con los siguientes pasos:

- En el archivo **controller.js** crea la función asíncrona **controlSearchResults** con una estructura **try – catch**:
  - Dentro del try realiza lo siguiente:
    - Invoca a la función **model.loadsearchResults** con el parámetro **query**, recuerda que esta función debe esperar(await).
    - Imprime en la consola el resultado (state.search.results).
  - Dentro del catch imprime en la consola el error (**err**).
  - Prueba la funcionalidad invocando a la función **controlSerachResults**.

4. Para poder realizar la búsqueda colocando la palabra buscada en el campo de la interfaz y que al hacer clic o dar enter nos traiga solamente los resultados que coincidan, haz lo siguiente:

- En la carpeta **views**, crea el archivo searchViews.js.
- Dentro de este, crea la clase **SearchView**.
  - Crea el elemento padre privado (**#parentEl**) al cual le deberás asignar el elemento con la clase search utilizando un **querySelector**.
  - Crea un método **getQuery** que retorne el valor del elemento con la clase **search\_field** utilizando el método **querySelector** del elemento padre.
  - Crea un **listener** que va a escuchar el evento **clic**.
  - No olvides exportar por defecto una invocación a la clase recién creada.
- En el archivo **controller** realiza lo siguiente:
  - Importa la clase **SearchView**.
  - En la función **controlSearchResults** instancia la función **searchView.getQuery** y asígnala a la constante **query**.
  - Valida que, si no existe ninguna consulta, regrese inmediatamente.

# Avance 3

- d. Para hacer que se escuche el evento del botón, utiliza el patrón editor-suscriptor de la siguiente manera:

- i.. En la clase **SearchView** crea el método **addHandlerSearch** que recibirá a **handler** como **parámetro**.

- ii.. Para escuchar el evento, realiza lo siguiente:

1. Utiliza el método **addEventListener** de **this.#parentEl** y pásale como parámetros:

- a. El evento 'submit'.
  - b. Una función anónima con el evento(e) como parámetro y en el cuerpo

- debe:

  - Enviar la acción predeterminada con el método **preventDefault** del evento(s).

- iii. Llamar a la función del controlador (**handler()**)

- iii. En el **controller**, agrega a la función init el método

- iii. En el **controller**, agrega a la función init el método **searchView.addHandlerSearch** con el parámetro **controlSearchResults**).
  - iv. Prueba tu código, pero antes, elimina un poco de basura de la consola con las siguientes actividades:

- En el archivo **controller** elimina el **console.log** del **id**.
  - En la clase **RecipeView**, elimina el **console.log** de **Fraction**, si es que lo colocaste.
  - En el archivo del modelo, elimina el **console.log** del **state search results**.

- v. Tu salida en la consola utilizando la palabra pizza, debería ser similar a la siguiente imagen:

Esta pantalla se obtuvo directamente del software que se está explicando en la computadora, para fines educativos.

- vi. Inténtalo ahora con otra palabra de las permitidas según la documentación de la api: <https://forkify-api.herokuapp.com/v2>

- e. Como pudiste ver, cuando se hace la búsqueda, no se borra la palabra que introdujiste en el input: para lograr que se borre, realiza lo siguiente:

- i. En la clase SearchView, agrega el método privado clearInput y colócale la siguiente línea de código:

```
this.#parentEl.querySelector('search-field').value = "";
```

- ii. Llama a este método privado desde el método getQuery con this.

## Avance 3

iii. por lo anterior, será necesario crear la constante **query** y asignar la línea de código que había a dicha constante. El resultado debe quedar de la siguiente manera:

```
const query = this._parentEl.querySelector('.search__field').value;
```

iv. Y finalmente retornar la constante.

5. Para desplegar el resultado de la búsqueda en el área destinada, realiza lo siguiente:
  - a. Como se requieren prácticamente los mismos métodos de la clase **RecipeViews**, será mejor crear una clase padre **View** con las siguientes características:
    - i. En la carpeta **views** crea el archivo **View.js**.
    - ii. Declara y exporta por defecto la clase **View**.
  - b. En la clase **RecipeView** realiza lo siguiente:
    - i. Importa la clase **View**.
    - ii. Debido a la compatibilidad con ECMAScript anteriores a 6, cambiaremos todos los objetos privados a su forma antigua, es decir, cambiaremos el # por el \_ tanto en la clase **RecipeView** como en la clase **SearchView**.
    - iii. Indica que la clase **RecipeView** es hija de la clase **View**.
  - c. Copia **\_data**, **render()**, **\_clear()**, **renderSpinner()**, **renderError()**, **renderMessage** de la clase **recipeView** a la clase **View**.
  - d. Importa los iconos a la clase **View**.
  - e. Valida que todo esté funcionando correctamente.
  - f. En la carpeta **views** crea el archivo **ResultView.js** y dentro de este, realiza lo siguiente:
    - i. Importa la clase **View**.
    - ii. Crea a la clase **ResultsView** como hija de la clase **View**.
    - iii. Declara el elemento padre como privado (**\_parentElement**) e indica que este será el elemento con la clase results utilizando el querySelector.
    - iv. Exporta la instancia de la **claseResultview** por defecto.
  - g. En el archivo **controller** realiza los siguientes cambios:
    - i. Importa la clase **ResultsView** como **resultsView**.
    - ii. En la función **controlSearchResults** en el try, llama al método **resultsView.renderSpinner0**.
  - h. Como puedes ver, ya se presenta el **spinner** en el área donde se desplegarán los resultados de la búsqueda, ahora para presentar los resultados como tal, realiza lo siguiente:
    - i. Crea el método privado **generateMarkup** y en el cuerpo regresa el arreglo **this.data** convertido en cadena utilizando la siguiente línea de código:

```
return this._data.map(this._generateMarkupPreview).join('');
```

## Avance 3

- ii. Crea el método **\_generateMarkupPreview** en la clase **ResultsView** que recibirá el parámetro **result**, indica que regrese entre template strings todo el código HTML que se encuentra en el archivo **index.html** bajo la etiqueta li con clase preview, que a su vez se encuentra dentro de la lista desorganizada con clase results.
  - iii. Modifica los iconos colocando la variable \${icons}.
  - iv. Utiliza el **result.id** colocándolo en el **href**, el **result.image** en el src de la imagen, el **result.title** en la etiqueta h4 con **clase preview\_\_title** y **result.publisher** en el párrafo con la clase **preview\_\_publisher**.
  - v. Dentro de la función **ControlSearchResults** invoca el método **resultView.render** con el parámetro **model.state.search.results**.
  - vi. En el **método \_generateMarkupPreview** elimina la clase **preview\_\_link--active** del link en el div con clase **preview\_\_user-generated**.
- i. Una vez que puedes desplegar los resultados de la búsqueda, es necesario corregir el issue que se presenta cuando ingresas un valor no encontrado, actualmente no hace nada y debería informar que el elemento buscado no existe. Para lograrlo, realiza lo siguiente:
- i. Crea la propiedad privada error **Message** en la clase **Resultview** con la leyenda “No recipes found for your query”.
  - ii. Crea la propiedad privada mensaje y asígnale una cadena vacía.
  - iii. En la clase **View**, dentro del método render, antes de realizar cualquier otra actividad, valida que si no existen los datos, o si data es un array y su tamaño es 0, se debe regresar un mensaje de error. Utiliza el siguiente código para hacerlo:

```
if(!data ||(Array.isArray(data) && data.length === 0)) return this.renderError();
```

- iv. Con esto deberá estar trabajando correctamente, tanto si encuentra el patrón de búsqueda o si no lo encuentra.

### Consideraciones

**Aprendedor:** Asegúrate de comprender la diferencia entre clase y función y sigue los pequeños tips de nomenclatura indicados en la literatura. El test en la consola te puede ayudar para ver los resultados.

**Instructor:** Conforme a la planeación de tu clase, de ser posible refuerza el concepto de clases y resalta la diferencia entre las funciones y las clases.

## Avance 4

Ahora que ya se pueden visualizar los resultados de la búsqueda y cada resultado puede desplegar los detalles en el área principal de la aplicación, es necesario resolver algunos detalles como la paginación.

1. Para que los resultados de las búsquedas se presenten en páginas de hasta 10 resultados, haz lo siguiente:
  - a. Agrega y exporta al archivo de **config** la constante **RES\_PER\_PAGE = 10**.
  - b. En el archivo del modelo, realiza los siguientes cambios:
    - i. Importa la constante **RES\_PER\_PAGE**.
    - ii. En el state agrega las siguientes propiedades:
      1. Page con el valor 1 por defecto.
      2. A **resultsPerPage** asígnale el valor de la constante **RES\_PER\_PAGE**.
    - iii. En la parte final del archivo crea una función expresada con las siguientes características:
      1. Asígnale el nombre **getSearchResultsPage**.
      2. La función recibe el parámetro page (iguálalo a **state.search.page** para prevenir algún error si es que no se pasa nada en page).
      3. Exporta la función recién creada
      4. En el cuerpo de la función haz lo siguiente:
        - a. Asigna el valor de page pasado como parámetro a **starte.search.page**.
        - b. Crea las constantes start y end, para hacer que su contenido sea dinámico, realiza lo siguiente:
          - i. start asígnale el valor de (page -1) y multiplícalo por **state.search.resultPerPage**
          - ii. end solo multiplícalo por **state.search.resultPerPage**
        - c. Retorna una parte del arreglo **state.search.results** utilizando el método slice que recibirá como parámetros las constantes start y end.
    - c. En el archivo del controlador, en la función **SearchResults**, modifica la forma en la que se hace el render de los resultados para que se cambien los siguientes: en lugar de **resultsView.render(model.state.search.results)**, que ahora tome los resultados de la función **resultsView.render(model.getSearchResultsPage)**.

## Avance 4

2. Ahora ya puedes visualizar que se despliegan 10 resultados por página, sin embargo, aún no tenemos los botones para avanzar y retroceder para poder navegar en las siguientes páginas, para lograrlo, realiza lo siguiente:

- En el directorio **views** crea un nuevo archivo paginationView.js y realiza lo siguiente:
  - Importa la clase View.
  - Importa los iconos.
  - Crea la clase PaginationView como hija de la clase Vista y dentro del cuerpo, haz lo siguiente:
    - Declara al elemento padre como privado e indícale a través del método querySelector de document que dicho elemento padre es el que tenga como clase pagination.
    - Declara la constante página actual (curPage) que será igual a this.\_data.page.
    - Crea el método privado \_generateMarkup y programa lo siguiente:
      - Valida cuántas páginas existen creando la constante numPages de la siguiente manera:

```
const numPages = Math.ceil(this._data.results.length / this._data.resultsPerPage);
```

    - Crea los siguientes escenarios:
      - Estando en la página 1 y existen más páginas. En este escenario valida si la página es igual a 1 y si el número de páginas es mayor a 1. Si esa condición se cumple, regresa con template string el código html del botón con clase pagination\_\_btn--next del div pagination (obtén el código del archivo index.html). Modifica el numero de la página para que en lugar de mostrar 3, muestre currentPage + 1, además agrega a la etiqueta del botón el atributo data-goto="\${currentPage + 1}". Finalmente modifica los iconos.
      - Estando en la última página, valida si curPage es igual al número de páginas, es > 1, si esa condición se cumple, regresa con template string el código html del botón con clase pagination\_\_btn--prev del div pagination (obtén el código del archivo index.html). Modifica el número de la página para que en lugar de mostrar 1, muestre la diferencia de currentPage -1, además agrega a la etiqueta del botón el atributo data-goto="\${currentPage - 1}". Finalmente, modifica los iconos.
      - Estando en cualquier página diferente a la página 1 y diferente a la última página. Valida si la página es menor al número de páginas, si eso se cumple, regresa con template strings los códigos html de los dos escenarios anteriores
      - Estando en la página 1 y no existen más páginas. Es el caso por defecto y solo se regresaría una cadena vacía

## Avance 4

- c. Prueba que todos los escenarios realicen correctamente su función.
4. Crea el método addHandlerClick por encima del método \_generaMarkup. Envíale como parámetro un handler y en el cuerpo realiza lo siguiente:
  - a. Utiliza el método **addEventListener** de **this.\_parentEl** y pásale como parámetros:
    - i. El evento 'click'.
    - ii. Una función anónima con el evento(e) y en el cuerpo debes utilizar la delegación de eventos y crear la constante btn que es igual a el elemento que se hizo clic (e.target), sobre este utilizar el método closest y pasarle como parámetro la clase del botón ('.btn--inline').
    - iii. Crea la constante goToPage y asígnale la expresión btn.dataset.goto;
    - iv. Registra en la consola esta constante.
    - v. Exporta un nuevo objeto de **PaginationView** por defecto.
  - b. En el archivo controller realiza los siguientes cambios:
    - i. Importa la clase **PaginationView**.
    - ii. Dentro de la función **controlSearchResults**, después del render de los resultados, invoca al método paginationView.render y pasa como parámetro el objeto de búsqueda (model.state.search).

# Entrega de proyecto final

## Entregables:

- Documento txt con URL de Github con el código y URL de Netlify con publicación de tu trabajo.

Para esta entrega y revisión del proyecto, asegúrate de contar con todos los elementos solicitados. Apóyate en este checklist de puntos importantes a considerar:

Criterios de evaluación	Realizado
1. Instala las herramientas y dependencias necesarias para desarrollar la aplicación.	
2. Crea la funcionalidad para poder consumir la API.	
3. Crea los eventos y demás funcionalidades para cargar dinámicamente los datos en el html.	
4. Maneja los errores generados por la aplicación.	
5. Crea las clases necesarias en la aplicación.	
6. Modulariza la aplicación conforme a la arquitectura MVC.	
7. Genera las diferentes vistas.	
8. Sube al repositorio tu aplicación.	
9. Publica tu aplicación en Netlify.	

# Criterios de evaluación

## Rúbrica de evaluación de proyecto

Criterios de evaluación	Nivel de desempeño			%
	Altamente competente 100% - 86%	Competente 85% - 70%	Aún sin desarrollar la competencia 69% - 0%	
1. Ambienta su entorno de trabajo para poder desarrollar la aplicación.	<b>10 - 8</b>  Ambienta correctamente su entorno de trabajo para poder desarrollar la aplicación.	<b>7 - 5</b>  Ambienta parcialmente su entorno de trabajo para poder desarrollar la aplicación.	<b>4 - 0</b>  No logra ambientar correctamente su entorno de trabajo para poder desarrollar la aplicación.	10
2. Consume la API.	<b>15 - 14</b>  Consumе la API correctamente y puede enviar búsquedas.	<b>13 - 9</b>  Consumе la API pero no puede realizar correctamente las búsquedas.	<b>8 - 0</b>  No lograr consumir la API o realizar búsquedas correctamente.	15
3. Carga dinámica de los datos recuperados de la API en el HTML.	<b>15 - 14</b>  Logra desplegar los datos de la consulta a la API en el HTML.	<b>13 - 9</b>  Logra desplegar algunos datos de la consulta a la API en el HTML.	<b>8 - 0</b>  No logra desplegar correctamente los datos de la consulta a la API en el HTML.	15

# Criterios de evaluación

## Rúbrica de evaluación de proyecto

Criterios de evaluación	Nivel de desempeño			%
	Altamente competente 100% - 86%	Competente 85% - 70%	Aún sin desarrollar la competencia 69% - 0%	
4. Crea las clases necesarias en la aplicación.	<b>15 - 14</b>  Crea las clases suficientes para la aplicación.	<b>13 - 9</b>  Crea algunas clases para su uso en la aplicación.	<b>8 - 0</b>  Crea muy pocas clases para la aplicación o no las crea.	15
5. Modulariza la aplicación conforme a la arquitectura MVC.	<b>15 - 14</b>  Logra crear los módulos conforme a la arquitectura MVC.	<b>13 - 9</b>  Logra crear solamente algunos de los módulos conforme a la arquitectura MVC.	<b>8 - 0</b>  No crea módulos correctamente conforme a la arquitectura MVC.	15
6. Genera las diferentes vistas solicitadas en el proyecto.	<b>15 - 14</b>  Genera todas las vistas solicitadas en el proyecto.	<b>13 - 9</b>  Genera correctamente algunas de las vistas solicitadas en el proyecto.	<b>8 - 0</b>  No genera correctamente las vistas solicitadas en el proyecto.	15

# Criterios de evaluación

## Rúbrica de evaluación de proyecto

Criterios de evaluación	Nivel de desempeño			%
	Altamente competente 100% - 86%	Competente 85% - 70%	Aún sin desarrollar la competencia 69% - 0%	
7. Sube al repositorio la aplicación.	<b>7.5 - 5</b>  Realiza correctamente el versionamiento de su proyecto.	<b>4.9 - 3</b>  Realiza parcialmente el versionamiento de su proyecto.	<b>1 - 0</b>  No versiona su proyecto correctamente en GitHub.	7.5
8. Publica la aplicación realizada en Netlify.	<b>7.5 - 5</b>  Realiza exitosamente la publicación de su proyecto.	<b>4.9 - 3</b>  Publica parcialmente su proyecto.	<b>1 - 0</b>  No logra publicar su proyecto.	7.5
Total				100%

La obra presentada es propiedad de ENSEÑANZA E INVESTIGACIÓN SUPERIOR A.C. (UNIVERSIDAD TECMILENIO), protegida por la Ley Federal de Derecho de Autor; la alteración o deformación de una obra, así como su reproducción, exhibición o ejecución pública sin el consentimiento de su autor y titular de los derechos correspondientes es constitutivo de un delito tipificado en la Ley Federal de Derechos de Autor, así como en las Leyes Internacionales de Derecho de Autor.

El uso de imágenes, fragmentos de videos, fragmentos de eventos culturales, programas y demás material que sea objeto de protección de los derechos de autor, es exclusivamente para fines educativos e informativos, y cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por UNIVERSIDAD TECMILENIO.

Queda prohibido copiar, reproducir, distribuir, publicar, transmitir, difundir, o en cualquier modo explotar cualquier parte de esta obra sin la autorización previa por escrito de UNIVERSIDAD TECMILENIO. Sin embargo, usted podrá bajar material a su computadora personal para uso exclusivamente personal o educacional y no comercial limitado a una copia por página. No se podrá remover o alterar de la copia ninguna leyenda de Derechos de Autor o la que manifieste la autoría del material.