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**:
    a. Copia el método **renderError** y renómbralo como **renderMessage**.
    b. Pásale como parámetro **message=this._message**.
    c. Cambia el icono 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:
    a. Declara y exporta una función asíncrona llamada **loadSearchResults** que recibirá como parámetro una búsqueda(query).
    b. En el cuerpo de la función crea un bloque try y catch.
    c. 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}`**.
    d. 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,
      };
    });
```

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

Avance 3

b. En la función **loadSearchResults** realiza estos cambios:
   i. Asigna la variable **query** a state.search.results.
   ii. Asigna a **state.search.results** la matriz con los nuevos objetos.
   iii. Envía a la consola los resultados, deberían ser similares a esto:


```
▾(59) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…},
{…}, {…}, {…}, {…}, {…}] ℹ
▸0: {id: '5ed6604591c37cdc054bcd09', title: 'Cauliflower Pizza Crust (with BBQ Chicken Pizza)', publisher: 'Closet Cooking', image: 'http://forkify-api.herokuapp.com/images/BBQChickenPizzawithCauliflowerCrust5004699695624ce.jpg'}
▸1: {id: '5ed6604591c37cdc054bcc13', title: 'Cauliflower Pizza Crust (with BBQ Chicken Pizza)', publisher: 'Closet Cooking', image: 'http://forkify-api.herokuapp.com/images/BBQChickenPizzawithCauliflowerCrust5004699695624ce.jpg'}
▸2: {id: '5ed6604591c37cdc054bcb34', title: 'Homemade Pizza', publisher: 'Simply Recipes', image: 'http://forkify-api.herokuapp.com/images/pizza292x2007a259a79.jpg'}
▸3: {id: '5ed6604591c37cdc054bcb37', title: 'How to Grill Pizza', publisher: 'Simply Recipes', image: 'http://forkify-api.herokuapp.com/images/howtogrillpizzad300x20086a60e1b.jpg'}
▸4: {id: '5ed6604591c37cdc054bcac4', title: 'Pizza Dip', publisher: 'Closet Cooking', image: 'http://forkify-api.herokuapp.com/images/Pizza2BDip2B12B500c4c0a26c.jpg'}
▸5: {id: '5ed6604591c37cdc054bca5d', title: 'Veggie Pizza', publisher: 'All Recipes', image: 'http://forkify-api.herokuapp.com/images/391236ba85.jpg'}
▸6: {id: '5ed6604591c37cdc054bca57', title: 'Valentine Pizza', publisher: 'All Recipes', image: 'http://forkify-api.herokuapp.com/images/7988559586.jpg'}
▸7: {id: '5ed6604591c37cdc054bca3b', title: 'Greek Pizza', publisher: 'A Spicy Perspective', image: 'http://forkify-api.herokuapp.com/images/IMG_4351180x1804f4a.jpg'}
▸8: {id: '5ed6604591c37cdc054bca10', title: 'Pizza Dip', publisher: 'My Baking Addiction', image: 'http://forkify-api.herokuapp.com/images/PizzaDip21of14f05.jpg'}
▸9: {id: '5ed6604591c37cdc054bc990', title: 'Pitta pizzas', publisher: 'BBC Good Food', image: 'http://forkify-api.herokuapp.com/images/1649634_MEDIUMd3fc.jpg'}
▸10: {id: '5ed6604591c37cdc054bc96e', title: 'Pizza Casserole', publisher: 'All Recipes', image: 'http://forkify-api.herokuapp.com/images/5100898cc5.jpg'}
▸11: {id: '5ed6604591c37cdc054bc971', title: 'Pizza Pinwheels', publisher: 'All Recipes', image: 'http://forkify-api.herokuapp.com/images/567c8fe.jpg'}
▸12: {id: '5ed6604591c37cdc054bc958', title: 'Pesto Pizza', publisher: 'All Recipes', image: 'http://forkify-api.herokuapp.com/images/104254d419.jpg'}
▸13: {id: '5ed6604591c37cdc054bc8fd', title: 'Hummus Pizza', publisher: 'All Recipes', image: 'http://forkify-api.herokuapp.com/images/636003da23.jpg'}
▸14: {id: '5ed6604691c37cdc054bd0c0', title: 'Puff pizza tart', publisher: 'BBC Good Food', image: 'http://forkify-api.herokuapp.com/images/679637_MEDIUM765c.jpg'}
▸15: {id: '5ed6604691c37cdc054bd0bc', title: 'Pizza Monkey Bread', publisher: "What's Gaby Cooking", image: 'http://forkify-api.herokuapp.com/images/PizzaMonkeyBread67f8.jpg'}
▸16: {id: '5ed6604591c37cdc054bcfb2', title: 'Veggi-Prosciutto Pizza', publisher: 'Epicurious', image: 'http://forkify-api.herokuapp.com/images/51150600f4cb.jpg'}
```


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

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

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

Avance 3

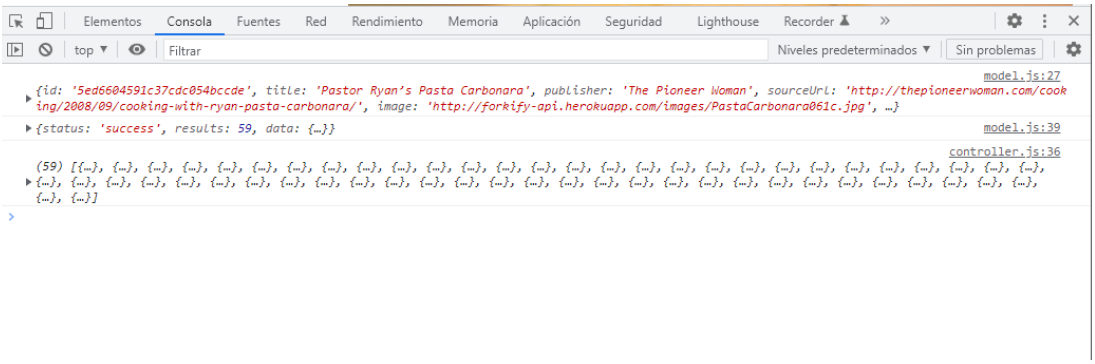
d. Para hacer que se escuche el evento del botón, utiliza el patrón editor-suscriptor de la siguiente manera:

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:
   i. Enviar la acción predeterminada con el método **preventDefault** del evento(e).
   ii. Llamar a la función del controlador (**handler**()).
   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 **controler** 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 queda 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__tittle** 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 puedas 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.