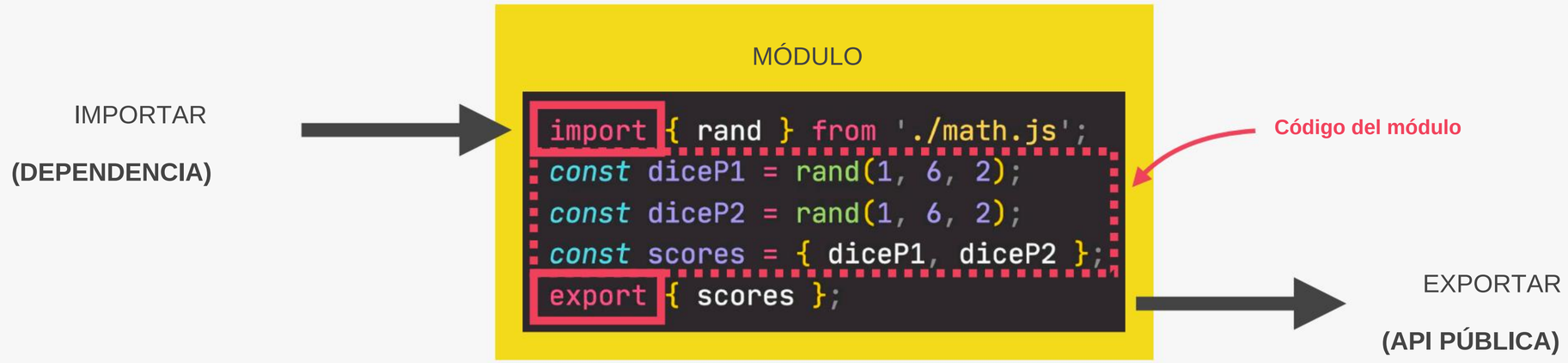
# UNA VISIÓN GENERAL DE LOS MÓDULOS

## MÓDULO

- Pieza de código reutilizable que **encapsula** los detalles de implementación;
- Por lo general, un **archivo independiente,** pero no tiene por qué serlo.

**¿POR QUÉ MÓDULOS?**

- **Componer software:** Los módulos son pequeños bloques de construcción que juntamos para construir aplicaciones complejas;
- **Aislar componentes: los** módulos se pueden desarrollar de forma aislada sin pensar en todo el código base;
- **Código abstracto:** implemente código de bajo nivel en módulos e importe estas abstracciones en otros módulos;
- **Código organizado:** los módulos conducen naturalmente a una base de código más organizada;
- **Reutilizar código:** los módulos nos permiten reutilizar fácilmente el mismo código, incluso en varios proyectos.



# MÓDULOS NATIVOS JAVASCRIPT (ES6)

## MÓDULOS ES6

Módulos almacenados en archivos,

**exactamente un módulo por archivo.**

```
import { rand } from './math.js';
const diceP1 = rand(1, 6, 2);
const diceP2 = rand(1, 6, 2);
const scores = { diceP1, diceP2 };
export { scores };
```

importar y exportar
sintaxis

| | MÓDULO ES6 | GUION |
|---|---|---|
| 👉 **Variables de nivel superior** | Alcance del módulo | Global |
| 👉 **Modo por defecto** | Modo estricto | Modo "descuidado" |
| 👉 **Nivel superior esto** | **indefinido** | **ventana** |
| 👉 **Importaciones y exportaciones** | ✅SÍ | 🚫 NO |
| 👉 **enlace HTML** | **<script tipo="módulo">** | **<script>** |
| 👉 **Descarga de archivos** | Asincrónico | Sincrónico |

☝️ que suceder al más alto nivel

¡Se alzan las importaciones!

# CÓMO SE IMPORTAN LOS MÓDULOS ES6


👉 índice.js

conexión en vivo,
NO copias


👉 matemáticas.js

conexión en vivo,
NO copias


👉 dom.js

Analizando index.js

Descarga asíncrona **de math.js**

Descarga asíncrona **dom.js**

Vinculación de importaciones a exportaciones **de math.js**

Vinculación de importaciones a exportaciones **dom.js**

Ejecución **math.js**

Ejecución **dom.js**

Ejecución **index.js**

**MÓDULOS DE IMPORTACIÓN**

ANTES DE LA **EJECUCIÓN**

- 👉 Los módulos se **importan sincrónicamente**
- 👉 Posible gracias a las importaciones de nivel superior ("estáticas"), que dan a **conocer las importaciones antes de la ejecución**
- 👉 Esto hace posible **la agrupación** y **la eliminación de código muerto.**

JONAS.IO
SCHMEDTMANN

# THE COMPLETE JAVASCRIPT COURSE

FROM ZERO TO EXPERT!

@JONASSCHMEDTMAN

## SECTION

MODERN JAVASCRIPT DEVELOPMENT: MODULES AND TOOLING

## LECTURE

REVIEW: WRITING CLEAN AND MODERN JAVASCRIPT

# REVISIÓN: CÓDIGO MODERNO Y LIMPIO

## CÓDIGO LEGIBLE

- 👉 Escribir código para que **otros** puedan entenderlo
- 👉 Escribe código para que **puedas** entenderlo en 1 año
- 👉 Evite soluciones demasiado “inteligentes” y demasiado complicadas
- 👉 Use nombres de variables descriptivos: **lo que contienen**
- 👉 Use nombres de funciones descriptivos: **lo que hacen**

## GENERAL

- 👉 Use el principio DRY (refactorice su código)
- 👉 No contamine el espacio de nombres global, encapsule en su lugar
- 👉 No use **var**
- 👉 Use comprobaciones de tipos fuertes **(===** y **!==)**

## FUNCIONES

- 👉 En general, las funciones deben hacer **una sola cosa**
- 👉 No use más de 3 parámetros de función
- 👉 Utilice parámetros predeterminados siempre que sea posible
- 👉 Por lo general, devuelve el mismo tipo de datos recibido
- 👉 Use funciones de flecha cuando hagan que el código sea más legible

## ABIERTO

- 👉 Usar clases ES6
- 👉 Encapsule datos y **no los mute** desde fuera de la clase
- 👉 Implementar el encadenamiento de métodos
- 👉 No use funciones de flecha como métodos (en objetos regulares )

# REVISIÓN: CÓDIGO MODERNO Y LIMPIO

## EVITAR CÓDIGO ANIDADO

- 👉 Utilizar **devolución anticipada** (cláusulas de guardia)
- 👉 Use operadores ternarios (condicionales) o lógicos en lugar de **if**
- 👉 Use múltiples **si** en lugar de **si/si no-si**
- 👉 Evite los bucles **for ,** use métodos de matriz en su lugar
- 👉 Evite las API asincrónicas basadas en devolución de llamada

## CÓDIGO ASINCRONICO

- 👉 Consuma promesas con async/await para una mejor legibilidad
- 👉 Siempre que sea posible, ejecute promesas en **paralelo (Promise.all)**
- 👉 Manejar errores y rechazar promesas

JONAS.IO
SCHMEDTMANN

# THE COMPLETE JAVASCRIPT COURSE

FROM ZERO TO EXPERT!

@JONASSCHMEDTMAN

## SECTION

MODERN JAVASCRIPT DEVELOPMENT: MODULES AND TOOLING

## LECTURE

DECLARATIVE AND FUNCTIONAL JAVASCRIPT PRINCIPLES

# IMPERATIVO VS. CÓDIGO DECLARATIVO

**Dos formas fundamentalmente diferentes de escribir código (paradigmas)**

## IMPERATIVO

- 👉 Programador explica **“CÓMO hacer las cosas”**
- 👉 Le explicamos al ordenador *cada uno de los pasos* que tiene que seguir para conseguir un resultado
- 👉 **Ejemplo:** Receta paso a paso de un pastel

```
const arr = [2, 4, 6, 8];
const doubled = [];
for (let i = 0; i < arr.length; i++)
  doubled[i] = arr[i] * 2;
```

## DECLARATIVO

- 👉 El programador dice **"QUÉ hacer"**
- 👉 Simplemente *describimos* la forma en que la computadora debe lograr el resultado.
- 👉 El **CÓMO** (instrucciones paso a paso) se abstrae
- 👉 **Ejemplo:** Descripción de un pastel

```
const arr = [2, 4, 6, 8];
const doubled = arr.map(n ⟹ n * 2);
```

# PRINCIPIOS DE PROGRAMACIÓN FUNCIONAL

## PROGRAMACION FUNCIONAL

- Paradigma de programación **declarativa**
- Basado en la idea de escribir software combinando muchas **funciones puras,** evitando **efectos secundarios** y **mutando** datos

- **Efecto secundario:** Modificación (mutación) de cualquier dato **fuera** de la función (mutar variables externas, iniciar sesión en la consola, escribir en DOM, etc.)
- **Función pura:** Función sin efectos secundarios. No es dependen de variables externas. **Dadas las mismas entradas, siempre devuelve las mismas salidas.**
- **Inmutabilidad:** ¡ El estado (datos) **nunca** se modifica! En su lugar, se **copia** el estado y la copia se muta y se devuelve.

- **Ejemplos:** 


## TÉCNICAS DE PROGRAMACIÓN FUNCIONAL

- Trate de evitar las mutaciones de datos
- Use métodos incorporados que no produzcan efectos secundarios
- Realice transformaciones de datos con métodos como como .map(), .filter() y .reduce()
- Intente evitar los efectos secundarios en las funciones: ¡esto, por supuesto, no siempre es posible!

## SINTAXIS DECLARATIVA

- Usar matriz y desestructuración de objetos
- Utilice el operador de propagación **(...)**
- Usar el operador ternario (condicional)
- Usar literales de plantilla

# APLICACIÓN FORKIFY:

# CONSTRUYENDO UN MODERNO

# SOLICITUD

JONAS.IO
SCHMEDTMANN

# THE COMPLETE JAVASCRIPT COURSE

FROM ZERO TO EXPERT!

@JONASSCHMEDTMAN

## SECTION

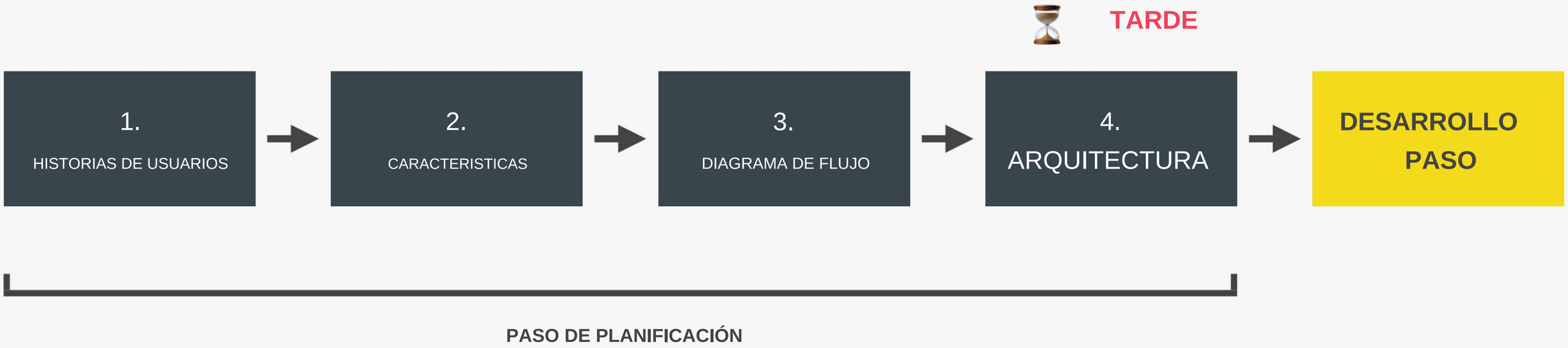
FORKIFY APP: BUILDING A MODERN APPLICATION

## LECTURE

PROJECT OVERVIEW AND PLANNING

JS

# PLANIFICACIÓN DE PROYECTOS



# 1. HISTORIAS DE USUARIOS

👉 **Historia de usuario:** Descripción de la funcionalidad de la aplicación desde la perspectiva del usuario.

👉 **Formato común:** Como *[tipo de usuario],* quiero *[una acción]* para que *[un beneficio]*

1 Como usuario, quiero **buscar recetas** para poder encontrar nuevas ideas para comidas.

2 Como usuario, quiero poder **actualizar el número de porciones,** para poder cocinar una comida para un número diferente de personas.

3 Como usuario, quiero **marcar recetas** para poder revisarlas más tarde.

4 Como usuario quiero poder **crear mis propias recetas,** para tenerlas todas organizadas en una misma app

5 Como usuario, quiero poder **ver mis marcadores y mis propias recetas cuando salgo de la aplicación y vuelvo más tarde,** para poder cerrar la aplicación de forma segura después de cocinar.

# 2. CARACTERÍSTICAS

forkify

**HISTORIAS DE USUARIOS** → **CARACTERISTICAS**

1. Buscar recetas
   - 👉 Funcionalidad de búsqueda: campo de entrada para enviar una solicitud a la API con las palabras clave buscadas
   - 👉 Mostrar resultados con paginación
   - 👉 Muestra la receta con el tiempo de cocción, las porciones y los ingredientes.
2. Actualizar el número de porciones
   - 👉 Cambie la funcionalidad de las porciones: actualice todos los ingredientes de acuerdo con la cantidad actual de porciones
3. Marcar recetas
   - 👉 Funcionalidad de marcadores: lista de visualización de todas las recetas marcadas
4. Crear mis propias recetas
   - 👉 El usuario puede cargar sus propias recetas.
   - 👉 Las recetas de los usuarios se marcarán automáticamente
   - 👉 El usuario solo puede ver sus propias recetas, no las recetas de otros usuarios
5. Ver mis marcadores y recetas propias cuando salgo de la aplicación y vuelvo más tarde
   - 👉 Almacenar datos de marcadores en el navegador usando el almacenamiento local
   - 👉 En la carga de la página, lea los marcadores guardados del almacenamiento local y muestre

# 3. DIAGRAMA DE FLUJO (PARTE 1)

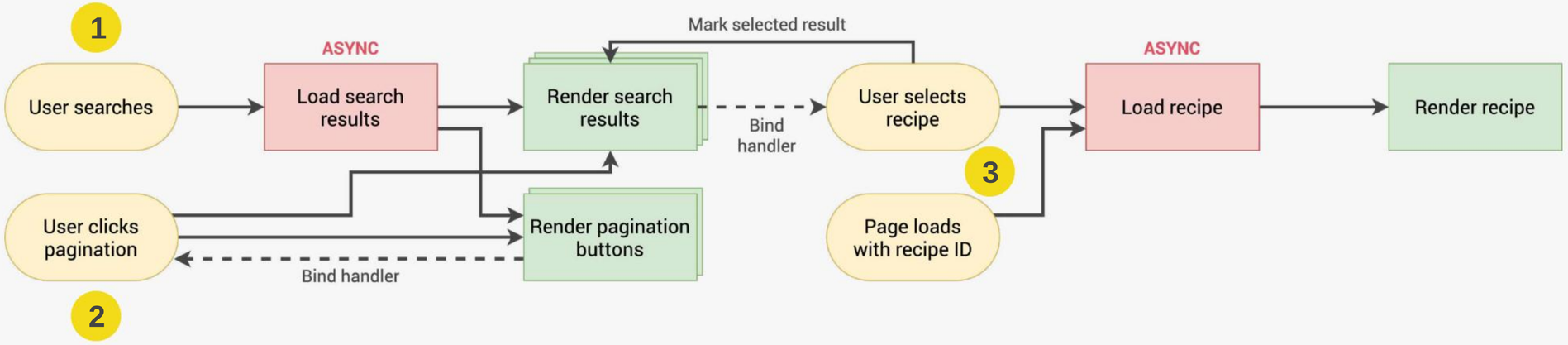
**CARACTERISTICAS**

1. Funcionalidad de búsqueda:
   solicitud de búsqueda API

2. Resultados con paginación

3. Mostrar receta

Otras características más adelante




# THE COMPLETE JAVASCRIPT COURSE

FROM ZERO TO EXPERT!


@JONASSCHMEDTMAN

## SECTION

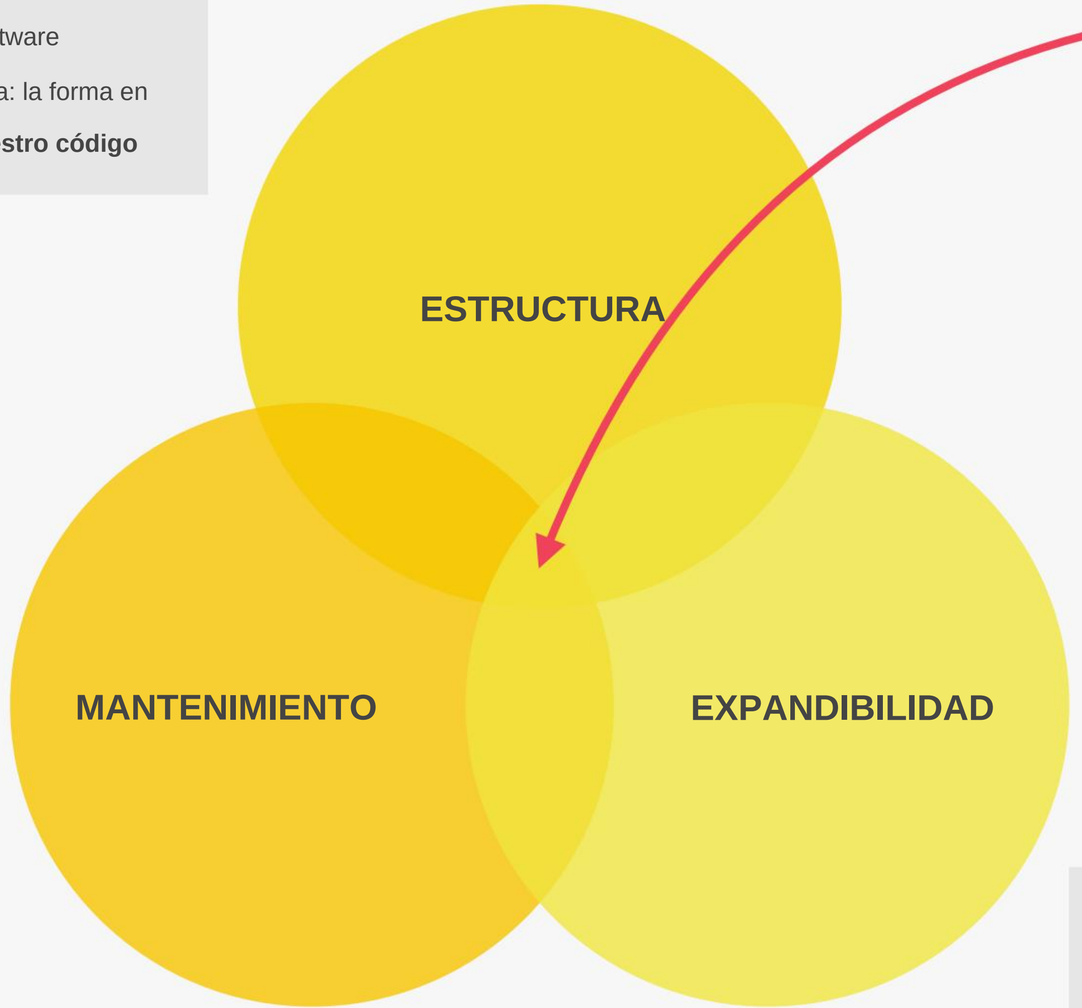
FORKIFY APP: BUILDING A MODERN APPLICATION

## LECTURE

THE MVC ARCHITECTURE

# ¿POR QUÉ PREOCUPARSE POR LA ARQUITECTURA?

👉 Como una casa, el software necesita una estructura: la forma en que **organizamos nuestro código**




## La arquitectura perfecta

👉 Podemos crear el nuestro arquitectura (proyecto Mapty)

👉 Podemos usar un bien establecido patrón de arquitectura como MVC, MVP, Flux, etc. **(este proyecto)**

👉 Podemos usar un marco como React, Angular, Vue, Svelte, etc.

👉 ¡Un proyecto nunca se termina! Necesitamos poder **cambiarlo fácilmente en el futuro.**

👉 También tenemos que ser capaces de agregue fácilmente **nuevas características**

# COMPONENTES DE CUALQUIER ARQUITECTURA

## LÓGICA DE NEGOCIOS

- Código que **resuelve el negocio real problema;**
- Directamente relacionado con lo que hace el negocio y lo que necesita;
- **Ejemplo:** envío de mensajes, almacenamiento de transacciones, cálculo de impuestos,…

## ESTADO

- Esencialmente **almacena todos los datos** sobre la aplicación
- debería ser el "única fuente de verdad"
- La interfaz de usuario debe mantenerse sincronizada con el estado
- Existen bibliotecas estatales

## BIBLIOTECA HTTP

- Responsable de hacer y recibir solicitudes AJAX
- Opcional pero casi siempre necesario en aplicaciones del mundo real

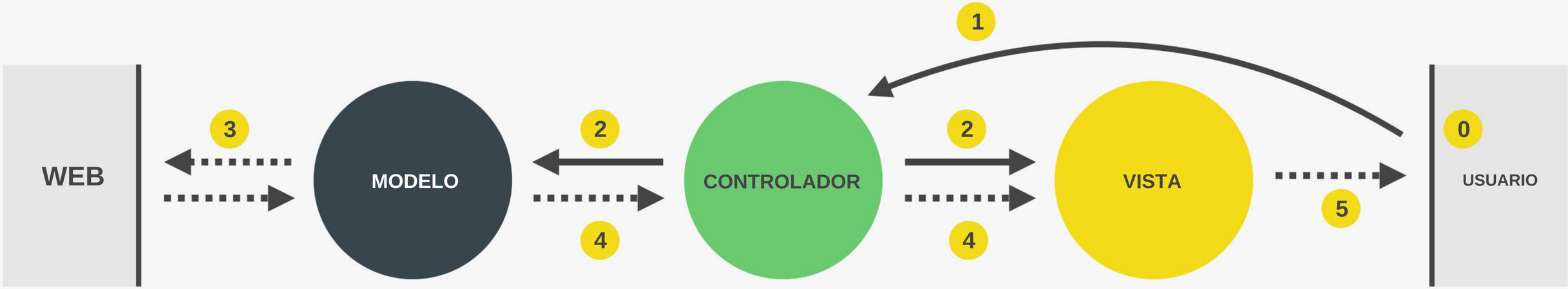
## LÓGICA DE LA APLICACIÓN (ENRUTADOR)

- Código que solo se preocupa por la **implementación de la propia aplicación;**
- Maneja la navegación y los eventos de la interfaz de usuario

## LÓGICA DE PRESENTACIÓN (CAPA DE IU)

- Código que es preocupado por la **parte visible** de la aplicación
- Esencialmente muestra el estado de la aplicación

# LA ARQUITECTURA MODELO-VISTA-CONTROLADOR (MVC)




**LÓGICA DE NEGOCIOS**

**ESTADO**

**BIBLIOTECA HTTP**

**LÓGICA DE LA APLICACIÓN**

👉 Puente entre el modelo y las vistas (que no se conocen entre sí)

👉 Maneja eventos de interfaz de usuario y

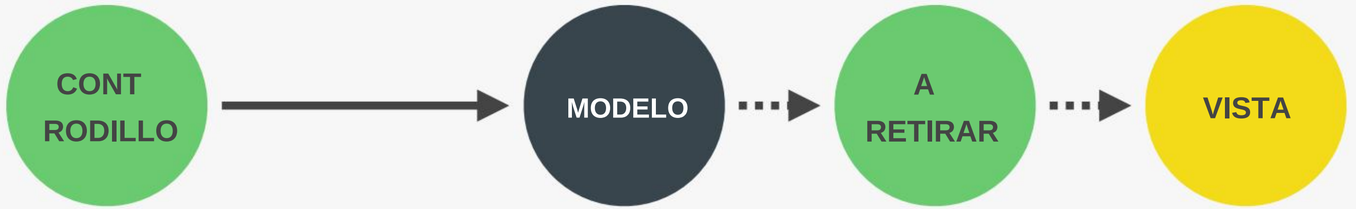
**Despacha tareas para modelar y ver.**

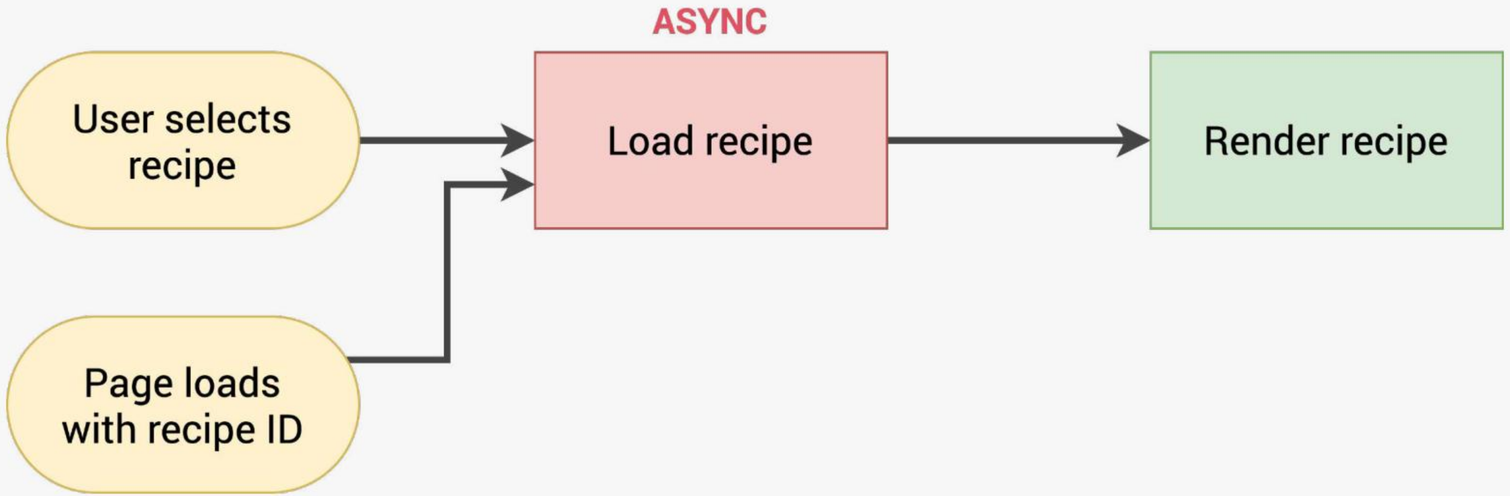
**LÓGICA DE PRESENTACIÓN**

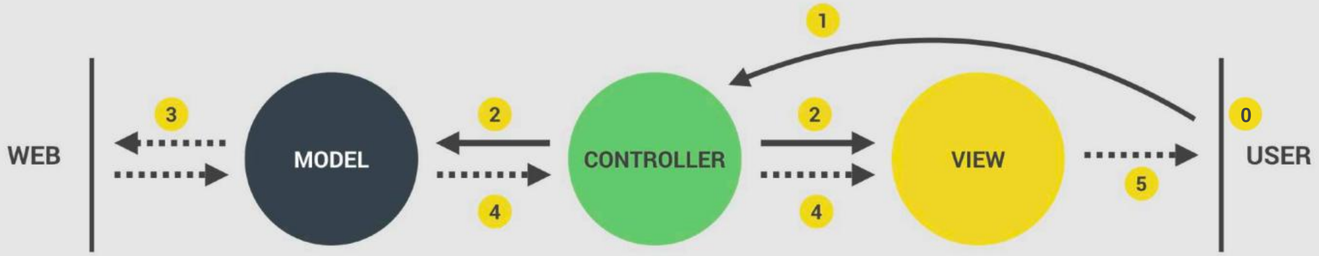
Conectado por llamada de función e importación

Flujo de datos

# MODELO, VISTA Y CONTROLADOR EN FORKIFY (SOLO VISUALIZACIÓN DE RECETAS)

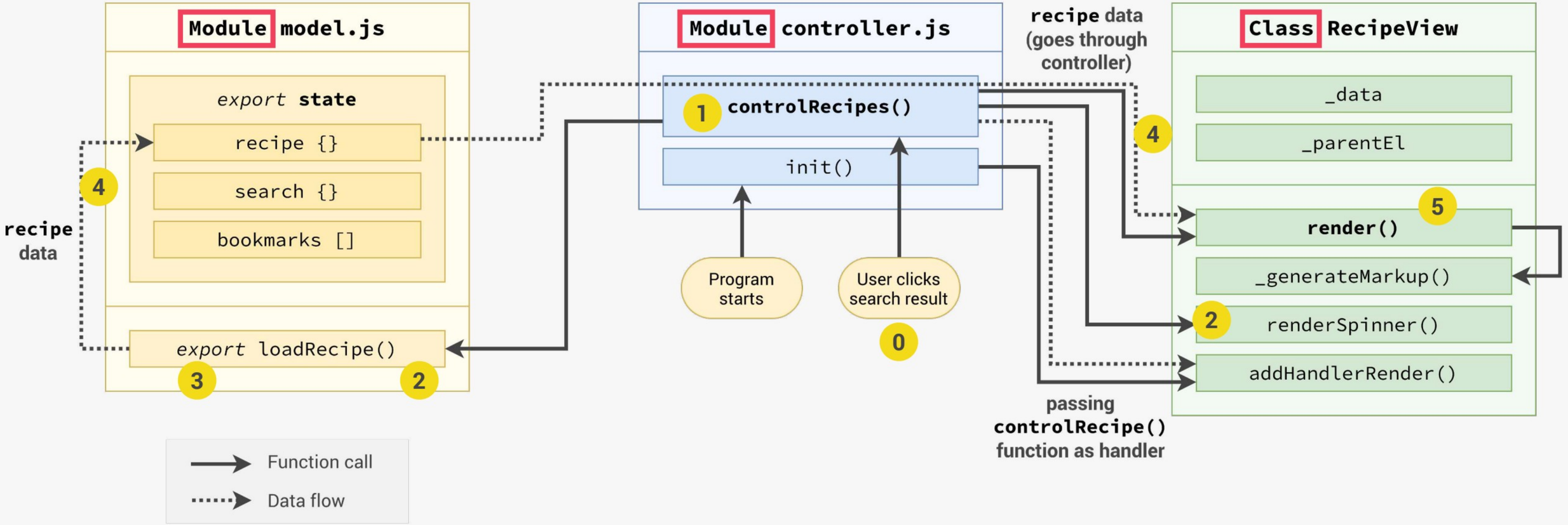


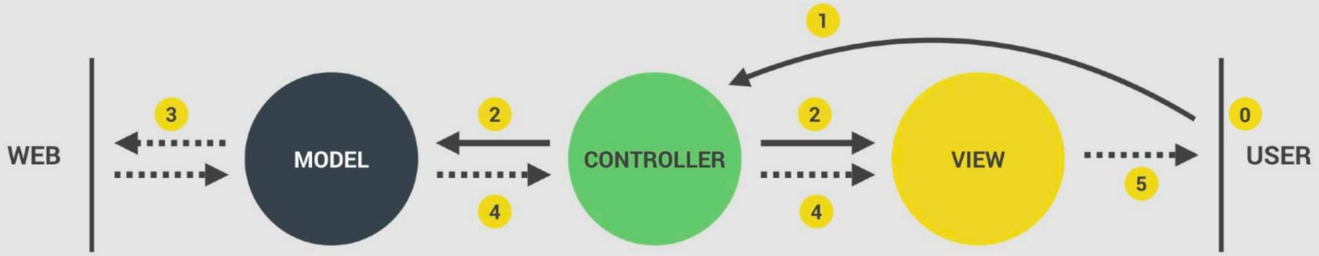



# MVC IMPLEMENTATION (RECIPE DISPLAY ONLY)

forkify




# THE COMPLETE JAVASCRIPT COURSE

FROM ZERO TO EXPERT!


@JONASSCHMEDTMAN

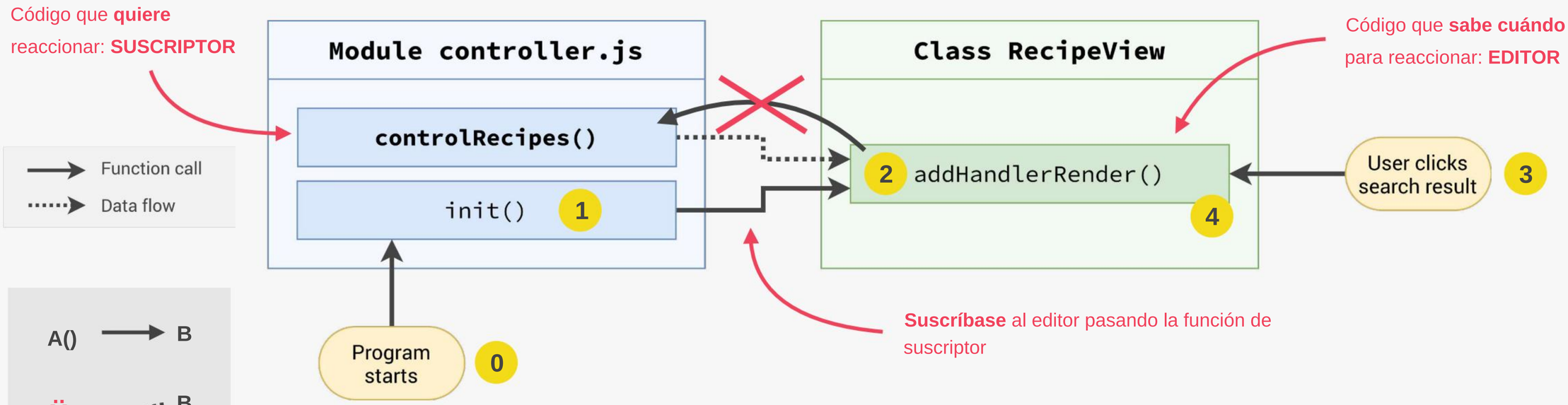
## SECTION

FORKIFY APP: BUILDING A MODERN APPLICATION

## LECTURE

EVENT HANDLERS IN MVC: PUBLISHER-SUBSCRIBER PATTERN

# MANEJO DE EVENTOS EN MVC: PATRÓN EDITOR-SUSCRIPTOR




- 👉 **controlRecipes** se pasará a **addHandlerRender** cuando se inicie el programa
- 👉 **addHandlerRender** escucha eventos (addEventListener) y usa **controlRecipes** como devolución de llamada

- 👉 Los eventos deben **manejarse** en el **controlador** (de lo contrario, tendríamos lógica de aplicación en la vista)
- 👉 Los eventos deben **escucharse** en la **vista** (de lo contrario, necesitaríamos elementos DOM en el controlador)


# THE COMPLETE JAVASCRIPT COURSE

FROM ZERO TO EXPERT!


@JONASSCHMEDTMAN

## SECTION

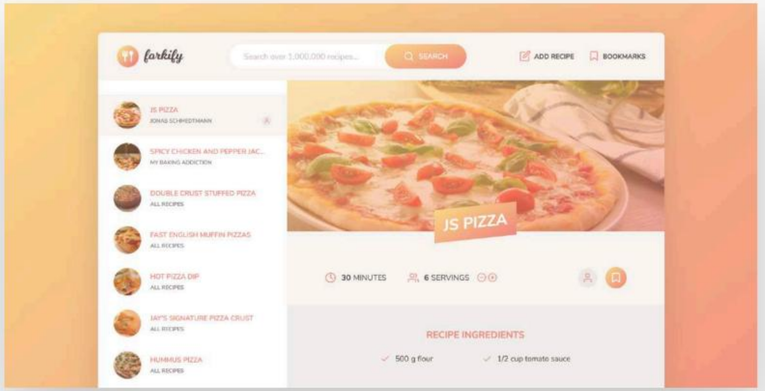
FORKIFY APP: BUILDING A MODERN APPLICATION

## LECTURE

WRAPPING UP: FINAL CONSIDERATIONS

# IDEAS DE MEJORA Y CARACTERÍSTICAS: DESAFÍOS🤓

- 👉 Muestra **el número de páginas** entre los botones de paginación;
- 👉 Posibilidad de **ordenar** los resultados de búsqueda por duración o número de ingredientes;
- 👉 Realizar la **validación de ingredientes** a la vista, antes de enviar el formulario;
- 👉 **Mejore la entrada de ingredientes de recetas:** sepárelos en varios campos y permita más de 6 ingredientes;




- 👉 **Función de lista de compras:** botón en la receta para agregar ingredientes a una lista;
- 👉 **Función de planificación de comidas semanales:** asigne recetas a los próximos 7 días y muéstrelas en un calendario semanal;
- 👉 **Obtenga datos nutricionales** de cada ingrediente de la API spoonacular (https:// spoonacular.com/food-api) y calcule las calorías totales de la receta.

END