Actividad 3.

Tarea: Crear el componente Carousel e instáncialo en la vista Ejercicio3. Implementa la lógica necesaria para que al cargarse la vista se muestre mediante el componente Carousel los datos de los personajes de la serie 'Rick and Morty' –nombre e imagen-, obtenidos mediante una petición GET a la API REST pública 'The Rick & Morty API'. Para ello deberás implementar las funciones pertinentes en el servicio creado para la actividad anterior y las interfaces necesarias para gestionar la respuesta a la petición GET. Empleando @Input, @Output y EventEmitter implementa la lógica necesaria para que se muestren en el componente Carousel los datos de los personajes de la serie, el componente debe permitir que cuando el usuario haga click sobre los botones anterior y siguiente del componente, se pueda navegar por los 20 personajes en que divide cada página la paginación de la API –se debe de poder pasar del primer personaje al último y viceversa-. Además, en la vista Ejercicio3 habrá otros dos botones anterior y siguiente, que permitirán navegar por las paginas de la paginación –también se deberá permitir pasar de la primera página a la última y viceversa-.

Interfaces de API

rickyMorty-episodes.interface.ts

examenFebrero > src > app > model > rickyMorty-episodes.interface.ts > Info

```ts
// Interfaz que representa la respuesta COMPLETA del endpoint
// https://rickandmortyapi.com/api/character?page=X
export interface RickyMortyResponse {

    // 'info' contiene metadatos de la paginación
    // (en este ejercicio solo necesitamos 'pages')
    info: Info;

    // 'results' es el array de personajes que usaremos en la vista
    results: Result[];
}

// Información de paginación de la API
export interface Info {

    // Número total de páginas disponibles en la API
    pages: number;
}

// Modelo de un personaje INDIVIDUAL
// Solo incluimos las propiedades que realmente usamos
export interface Result {

    // Nombre del personaje (se muestra en el Carousel)
    name: string;

    // URL de la imagen del personaje (se usa como background-image)
    image: string;
}
```

rickyMorty-characters.interface.ts

examenFebrero > src > app > model > rickyMorty-characters.interface.ts

```ts
// Respuesta del endpoint:
// https://rickandmortyapi.com/api/character?page=X
export interface RickyMortyResponse {

    // Información de paginación
    info: Info;

    // Array de personajes
    results: Result[];
}

// Metadatos de la paginación
export interface Info {

    // Número total de páginas
    pages: number;
}

// Modelo de un personaje individual
export interface Result {

    // Nombre del personaje
    name: string;

    // URL de la imagen del personaje
    image: string;
}
```

Servicio

TS morty-api.ts

examenFebrero > src > app > services > TS morty-api.ts > ...

```ts
import { HttpClient } from '@angular/common/http';
import { inject, Injectable } from '@angular/core';
import { Observable } from 'rxjs';
// Interfaces tipadas para cada endpoint
import { RickyMortyResponse } from '../model/rickyMorty-characters.interface';
import { RickyMortyEpisode } from '../model/rickyMorty-episodes.interface';

@Injectable({
  providedIn: 'root',
})
export class MortyApi {
  // Inyección de HttpClient usando inject()
  public http = inject(HttpClient);

  /* Obtiene PERSONAJES desde la API de Rick & Morty
  Endpoint: /api/character?page=X */
  public getPersonaje(page: number): Observable<RickyMortyResponse> {
    return this.http.get<RickyMortyResponse>(
      `https://rickandmortyapi.com/api/character/?page=${page}`
    )
  }

  /* Obtiene EPISODIOS desde la API de Rick & Morty
  Endpoint: /api/episode?page=X */
  public getEpisode(page: number): Observable<RickyMortyEpisode> {
    return this.http.get<RickyMortyEpisode>(
      `https://rickandmortyapi.com/api/episode?page=${page}`
    )
  }
}
```

Padre

ejercicio3.ts

examenFebrero > src > app > views > ejercicio3 > ejercicio3.ts > Ejercicio3

```ts
import { Component, inject, OnInit } from '@angular/core';
import { Carousel } from '../../components/carousel/carousel';
import { MortyApi } from '../../services/morty-api';
import { Result, RickyMortyResponse } from '../../model/rickyMorty-characters.interface';
import { EpisodeResult } from '../../model/rickyMorty-episodes.interface';


@Component({
  selector: 'app-ejercicio3',
  imports: [Carousel],
  templateUrl: './ejercicio3.html',
  styleUrl: './ejercicio3.css',
})
export class Ejercicio3 implements OnInit {
  // Inyección del servicio con inject() (Angular moderno)
  // El servicio se encarga de las peticiones HTTP y devuelve Observables tipados
  private rickService = inject(MortyApi);

  /* "mode" controla qué dataset se está mostrando:
        - 'characters' => se renderiza el carrusel de personajes
        - 'episodes'   => se renderiza el carrusel de episodios
  Esta variable se usa en el HTML con @if para decidir qué bloque pintar */
  public mode: 'characters' | 'episodes' = 'characters';

  // ─────────────────────────────────────────────────────────
  // PERSONAJES (estado + paginación + navegación dentro de página)
  // ─────────────────────────────────────────────────────────
  public characters: Result[] = [];   // Lista de personajes de la página actual (normalmente 20 por página).
  public totalPages?: number;         // Total de páginas disponibles de PERSONAJES (viene de response.info.pages).
  public currentPage: number = 1;     // Página actual de PERSONAJES (paginación de API). Importante: la API empieza en página 1 (no en 0).
  public currentIndex: number = 0;    // Índice actual del carrusel dentro de la página actual. Va de 0 a characters.length - 1.

  // ─────────────────────────────────────────────────────────
  // CICLO DE VIDA
  // ─────────────────────────────────────────────────────────
  ngOnInit(): void {
    this.loadPersonaje();
    this.loadEpisode();
  }

  // ─────────────────────────────────────────────────────────
  // ACCIONES DE UI: cambiar de modo
  // ─────────────────────────────────────────────────────────

  /* Botón "PERSONAJES":
  Los botones de modo cambian CONTEXTO, no navegan el carrusel */
  public showCharacters(): void {
    this.mode = 'characters';   // Cambia el modo
    this.currentIndex = 0;      // Resetea indice del modo
    this.currentPage = 1;       // Resetea pagina del modo
    this.loadPersonaje();       // Lanza la carga HTTP
  }
  /* Botón "EPISODIOS":
  Misma idea que showCharacters pero para episodios */
  public showEpisodes(): void {
    this.mode = 'episodes';
    this.currentIndexEp = 0;
    this.currentPageEp = 1;
    this.loadEpisode();
  }
```

```
61
62  // ──────────────────────────────────────────────────────────
63  // CARGA HTTP: PERSONAJES
64  // ──────────────────────────────────────────────────────────
65
66  /* Llama al servicio para obtener PERSONAJES de la página actual.
67  El subscribe se hace en el componente.
68  Guardamos:
69    - characters (lista)
70    - totalPages (paginación global) */
71  public loadPersonaje(): void {
72    this.rickService.getPersonaje(this.currentPage).subscribe((response) => {
73      this.characters = response.results;
74      this.totalPages = response.info.pages;
75      console.log('Personajes: ' , this.characters)
76    })
77  }
78
79  // ──────────────────────────────────────────────────────────
80  // NAVEGACIÓN CIRCULAR: PERSONAJES (dentro de la página)
81  // ──────────────────────────────────────────────────────────
82
83  /* Va al personaje anterior de forma circular
84  Esta lógica es "carrusel" (índice dentro de página), NO cambia currentPage */
85  public prevCharacter(): void {
86    if (this.currentIndex === 0) {                    // Si estás en 0, saltas al último
87      this.currentIndex = this.characters.length - 1;
88        } else {                                      // Si no, restas 1
89          this.currentIndex--;
90        }
91  }
92
93  /* Va al siguiente personaje de forma circular */
94  public nextCharacter(): void {
95    if (this.currentIndex < this.characters.length - 1) { // Si no estás en el último, sumas 1
96      this.currentIndex++;
97        } else {                                        // Si estás en el último, vuelves a 0
98          this.currentIndex = 0;
99        }
100 }
```

```
101
102   // ─────────────────────────────────────────────────────────
103   // EPISODIOS (estado + paginación + navegación)
104   // ─────────────────────────────────────────────────────────
105
106   /* */
107   public episodes: EpisodeResult[] = [];  // Lista de episodios de la página actual
108   public totalPagesEp?: number;           // Total de páginas disponibles de EPISODIOS.
109   public currentPageEp: number = 1;       // Página actual de EPISODIOS.
110   public currentIndexEp: number = 0;      // Índice actual del carrusel de EPISODIOS dentro de la página actual.
111   /* Imagen por defecto para episodios:
112       - La API de episodios NO devuelve imágenes, así que el padre le pasa una URL fija al componente Carousel para que pueda renderizar igual que con personajes.*/
113   public defaultPhoto: string = 'https://media.posterstore.com/site_images/68631d8a25436f8361d7687a_1426348943_WB0095-8.jpg';
114
115   // ─────────────────────────────────────────────────────────
116   // CARGA HTTP: EPISODIOS
117   // ─────────────────────────────────────────────────────────
118
119   /* Llama al servicio para obtener EPISODIOS de la página actual.
120   El subscribe se hace en el componente.
121   Guardamos:
122     - episodios (lista)
123     - totalPagesEp (paginación global) */
124   public loadEpisode(): void {
125     this.rickService.getEpisode(this.currentPageEp).subscribe((response) => {
126     this.episodes = response.results;
127     this.totalPagesEp = response.info.pages;
128     console.log('Episodios: ' , this.episodes)
129   })
130   }
131
132   // ─────────────────────────────────────────────────────────
133   // NAVEGACIÓN CIRCULAR: EPISODIOS
134   // ─────────────────────────────────────────────────────────
135
136   /* Anterior episodio de forma circular (misma lógica que personajes) */
137   public prevEpisode(): void {
138     if (this.currentIndexEp === 0) {
139       this.currentIndexEp = this.episodes.length - 1;
140       } else {
141         this.currentIndexEp--;
142       }
143   }
144   /* Siguiente episodio de forma circular */
145   public nextEpisode(): void {
146     if (this.currentIndexEp < this.episodes.length - 1) {
147       this.currentIndexEp++;
148       } else {
149         this.currentIndexEp = 0;
150       }
151   }
152 }
```

ejercicio3.html

examenFebrero > src > app > views > ejercicio3 > ejercicio3.html > div#page > section > div.content > div.grid > app-carousel

```
1   <!-- EJERCICIO 3 -->
2   <div id="page">
3       <section>
4           <!-- Botones de CAMBIO DE MODO:
5                   - NO navegan el carrusel
6                   - Cambian el contexto (mode) y lanzan la carga de datos correspondiente -->
7           <div id='render-more'>
8               <button (click)="showCharacters()">PERSONAJES</button>
9               <button (click)="showEpisodes()">EPISODIOS</button>
10          </div>
11          <div class="content">
12              <div class="grid">
13                  <!-- Renderizamos el carousel SOLO cuando:
14                          - mode coincide
15                          - y el array tiene datos
16                      Así evitamos errores del tipo: characters[currentIndex] undefined -->
17                  @if (mode === 'characters' && characters.length > 0) {
18                      <!-- Reutilización del componente hijo <app-carousel>:
19                              - @Input: datos que el hijo necesita para pintar (name, photo)
20                              - @Output: eventos que el HJO emite (prevItem/nextItem) y que el PADRE maneja con su lógica (prevCharacter/nextCharacter) -->
21                      <app-carousel
22                          [name]="characters[currentIndex].name"
23                          [photo]="characters[currentIndex].image"
24                          (prevItem)="prevCharacter()"
25                          (nextItem)="nextCharacter()"
26                          >
27                      </app-carousel>
28                  } @else if (mode === 'episodes' && episodes.length > 0){
29                      <app-carousel
30                          [name]="episodes[currentIndexEp].name"
31                          [photo]="defaultPhoto"
32                          (prevItem)="prevEpisode()"
33                          (nextItem)="nextEpisode()"
34                          >
35                      </app-carousel>
36                  }
37              </div>
38              <div id='render-more'>
39                  <button (click)="prevCharacter()">ANTERIORES</button>
40                  <button (click)="nextCharacter()">SIGUIENTES</button>
41              </div>
42          </div>
43      </section>
44  </div>
```

Hijo

carousel.ts

examenFebrero > src > app > components > carousel > carousel.ts > ...

```ts
import { NgStyle } from '@angular/common';
import { Component, EventEmitter, Input, Output } from '@angular/core';

@Component({
  selector: 'app-carousel',
  imports: [NgStyle],
  templateUrl: './carousel.html',
  styleUrl: './carousel.css',
})
export class Carousel {

  // ------------------------------------------------------------
  // INPUTS (Padre → Hijo)
  // ------------------------------------------------------------
  @Input() name: string = '';   // Texto que se muestra en el carrusel.
  @Input() photo: string = '';  // Imagen que se usa como background del carrusel.

  // ------------------------------------------------------------
  // OUTPUTS (Hijo → Padre)
  // ------------------------------------------------------------
  @Output() nextItem = new EventEmitter<void>();  // Evento que se emite cuando el usuario pulsa "SIGUIENTE".
  @Output() prevItem = new EventEmitter<void>();  // Evento que se emite cuando el usuario pulsa "ANTERIOR".

  // ------------------------------------------------------------
  // MÉTODOS DE UI (solo emiten eventos)
  // ------------------------------------------------------------

  // Llamado desde el botón "SIGUIENTE" del HTML
  public onNext() {
    this.nextItem.emit()
  };
  // Llamado desde el botón "ANTERIOR" del HTML.
  public onPrev() {
    this.prevItem.emit()
  };
}
```

carousel.html

examenFebrero > src > app > components > carousel > carousel.html > ...

```html
<!-- CAROUSEL -->
<div class="card">
    <div class='photo'
        [ngStyle]="{'background-image': 'url(' + photo + ')'}"> <!-- La imagen se pinta como background-image -->
    </div>
    <div class="footer">
        <div class="titles">
            <h2 class='card-title'>{{ name }}</h2> <!-- 'name' viene del padre por @Input -->
        </div>
    </div>
    <!-- Botones de navegación del carrusel -->
    <div id='render-more'>
        <!-- El hijo NO navega: solo emite eventos al padre -->
        <button (click)="onPrev()">ANTERIOR</button>
        <button (click)="onNext()">SIGUIENTE</button>
    </div>
</div>
```