# React intermedio

## Comunicación con el servidor

EJERCICIO: Creación de los servicios en nuestra App

# ÍNDICE

- Creación de nuestra aplicación: Usando create-react-app
- Estructurando nuestra aplicación en React
- Creación de los servicios base
- Creación de los modelos
- Desarrollo de servicios por entidad

# Creación de nuestra app con create-react-app

- La librería **create-react-app** nos proporciona una sencilla interfaz CLI para crear apps sencillas con React de manera rápida
- La instalación con el gestor de paquetes **NPM** es tan sencilla como:

  npm install -g create-react-app

- Crear una nueva app sólo requerirá unos simples comandos:

```
npx create-react-app mi-app
cd mi-app
npm start
```

# Creación de nuestra app con create-react-app

- La estructura inicial de la app queda de esta manera

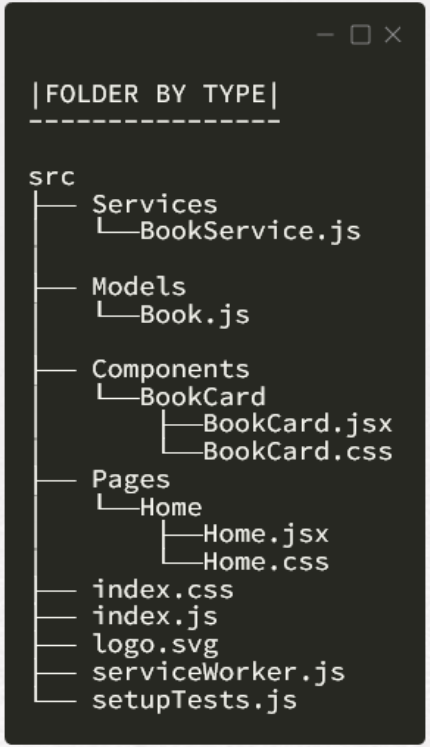
# Estructurando nuestra aplicación en React

- Las estructuras más usadas son **folder-by-type** y **folder-by-feature**

```
|FOLDER BY TYPE|
----------------

src
├── Services
│   └──BookService.js
│
├── Models
│   └──Book.js
│
├── Components
│   └──BookCard
│       ├──BookCard.jsx
│       └──BookCard.css
├── Pages
│   └──Home
│       ├──Home.jsx
│       └──Home.css
├── index.css
├── index.js
├── logo.svg
├── serviceWorker.js
└── setupTests.js
```

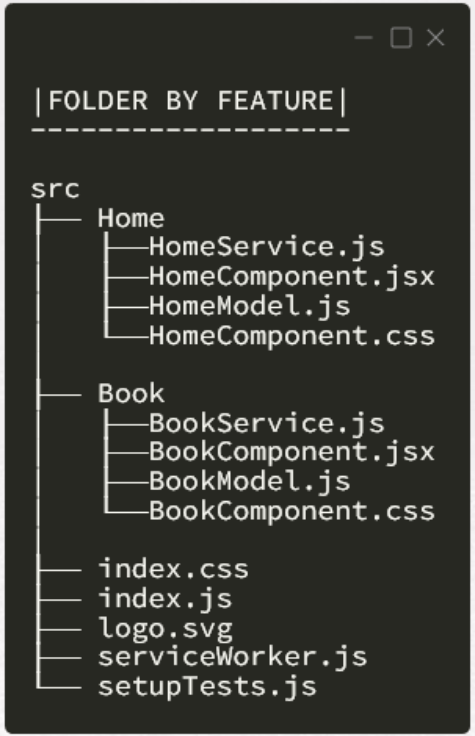
```
|FOLDER BY FEATURE|
-------------------

src
├── Home
│   ├──HomeService.js
│   ├──HomeComponent.jsx
│   ├──HomeModel.js
│   └──HomeComponent.css
│
├── Book
│   ├──BookService.js
│   ├──BookComponent.jsx
│   ├──BookModel.js
│   └──BookComponent.css
│
├── index.css
├── index.js
├── logo.svg
├── serviceWorker.js
└── setupTests.js
```

# Estructurando nuestra aplicación en React




- **Ventajas de Folder-by-type**
  - Mejor separación de responsabilidades
  - Mejor distinción de roles en componentes
  - Una forma fácil de recordar la cascada de props
- **Desventajas de Folder-by-type**
  - Artificioso a la hora de integrar
  - A partir de aprox. 10 entidades resulta caótico

# Estructurando nuestra aplicación en React




- **Ventajas de Folder-by-feature**
  - Sencillo de navegar
  - Más intuitivo respecto al proceso de desarrollo
  - Estructura más reducida, bueno para grandes proyectos
- **Desventajas de Folder-by-type**
  - Es más fácil terminar con un estado descontrolado
  - Puede ser más complejo para un recién llegado

# Creación de los servicios base

```
import { API_URL } from "../config/constants";
import { requestInterceptor, responseInterceptor } from "../config/interceptors"

export const request = (resource, init) => {
    requestInterceptor(resource, init)
    .then(response => { responseInterceptor(response); })
    .catch(error => {
        console.error(`Ha ocurrido un error al pedir el recurso ${resource} con el
        mensaje: ${error.message}`)
    })
}

export const get = (endpoint, params, options={}) => {
    const url = new URL(API_URL + endpoint);
    Object.keys(params).forEach(key => url.searchParams.append(key, params[key]));
    return request(url, {
        method: 'GET',
        headers: 'headers' in options ? options.headers : {},
        ...options
    })
}

export const post = (endpoint, body, options={}) => {
    const url = new URL(API_URL + endpoint);
    return request(url, {
        method: 'POST',
        headers: 'headers' in options ? options.headers : {},
        body: JSON.stringify(body),
        ...options
    })
}
```

# Creación de los servicios base

```
export const request = (resource, init) => {
    requestInterceptor(resource, init)
    .then(response => { responseInterceptor(response); })
    .catch(error => {
        console.error(`Ha ocurrido un error al pedir el recurso ${resource} con el
        mensaje: ${error.message}`)
    })
}
```

- La petición base debe de ser lo suficientemente genérica para ser reusable y lo suficientemente concreta para ser justificable
- Los interceptores nos darán mayor juego a medida que vayamos introduciendo características en la aplicación

# Creación de los servicios base

```
export const requestInterceptor = (resource, init) => {
    return fetch(resource, {
        ...init,
        headers: {
            'content-type': 'application/json',
            ...init.headers
        }
    })
}
export const responseInterceptor = (response) => {
    if (response.ok) {
        return response.json();
    } else {
        return response.json().then((data) => {
            let error = new Error(response.status);
            error.response = data;
            error.status = response.status;
            throw error;
        })
    }
}
```

# Creación de los servicios base

```
export const get = (endpoint, params, options={}) => {
    const url = new URL(API_URL + endpoint);
    Object.keys(params).forEach(key => url.searchParams.append(key, params[key]));
    return request(url, {
        method: 'GET',
        headers: 'headers' in options ? options.headers : {},
        ...options
    })
}
export const post = (endpoint, body, options={}) => {
    const url = new URL(API_URL + endpoint);
    return request(url, {
        method: 'POST',
        headers: 'headers' in options ? options.headers : {},
        body: JSON.stringify(body),
        ...options
    })
}
```

- El objetivo de nuestras funciones post y get es crear una interfaz sencilla para interaccionar con nuestra API en concreto

# Creación de los modelos

```
export class Book {
    constructor(dto) {
        this.title = dto.title || '';
        this.description = dto.description || '';
        this.tags = [
            ...dto.subjects,
            ...dto.subject_places,
            ...dto.subject_people,
            ...dto.subject_times
        ] || [];
        this.authors = dto.authors.map(key => {
            return key.replace('/authors/', '');
        })
        this.coverId = dto.covers.length > 0 ? dto.covers[0] : null
    }
}
```

- Uno de los objetivos del modelo es **simplificar la estructura de la respuesta** para que encaje con la finalidad de la entidad en nuestro Front

# Desarrollo de servicios por entidad

```
import { get } from "."
import { Book } from "../models/Book";

export const getBooks = (page, size) => {
    return get('books', { page, size }).then(response => {
        return response.map(bookDto => Book(bookDto));
    });
}
```

- De esta manera el servicio para pedir libros devuelve un array de objetos **Book** correctamente formados y listos para usar
- Este servicio podrá consumirse en cualquier parte de la aplicación mientras nos abstraemos del funcionamiento interno