React intermedio

Comunicación con el servidor

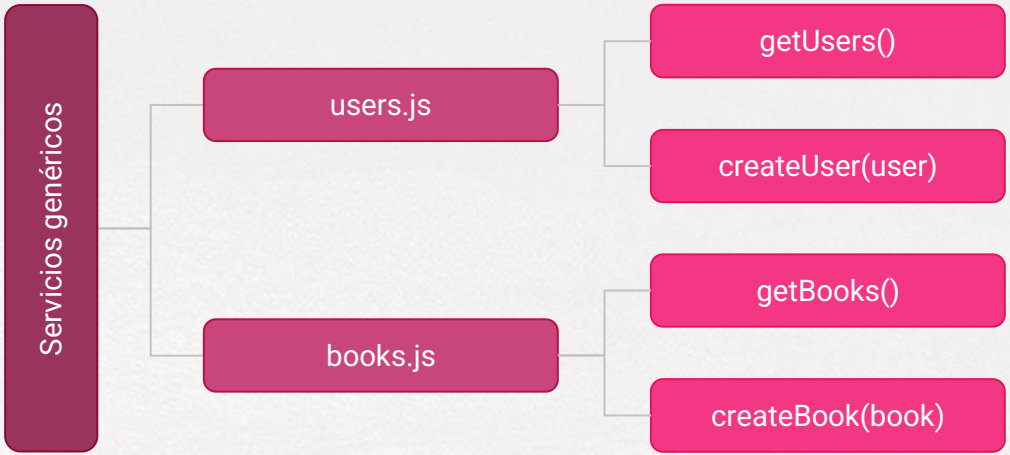
Uso de modelos en servicios

ÍNDICE

- Uso de los modelos en servicios
- Creación de pseudo-interceptores

Uso de los modelos en servicios

- Disponer de modelos y servicios base nos permite organizar nuestro código en torno a conjuntos de servicios “tipados”
- Este tipo de orientación es común en **MVC (Modelo-Vista-Controlador)** o derivados y similares



Uso de los modelos en servicios

```
import { get } from '../services/index';
import User from '../models/user';

export function getUsers(params) {
  return get('users', params).then(response => {
    return response.users.map(userDTO => { return new User(userDTO) });
  });
}
```

- El servicio devuelve una nueva promesa cuya respuesta es ya un array de modelos.

Uso de los modelos en servicios

```
import { post } from '../services/index';
import Book from '../models/book';

export function createPost(book) {
  // Convertimos de nuevo el modelo a su formato de modelo de transporte para el servidor
  const body = book.toDTO();
  return post('books', body).then(response => {
    return new Book(response);
  });
}
```

- El método **toDTO()** del modelo nos proporciona siempre una forma sencilla de transformar nuestro modelo de vuelta al de comunicación del servidor

Creación de pseudo-interceptores

- Un interceptor de una petición es una función que se ejecuta ya sea antes de la petición o a la hora de su resolución
- Los interceptores permiten **generalizar** determinadas lógicas para todas las peticiones salientes
- **Fetch** no proporciona soporte para interceptores de forma nativa, pero hay muchas maneras de emular el mismo comportamiento
- La propuesta aquí es una de las soluciones más simples posibles

```
function responseIntercept(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;
        });
    }
}

export function request(resource, init) {
    return fetch(resource, {
        ...init,
        headers: {
            'content-type': 'application/json',
            'Authorization': `Basic ${TOKEN}`,
            ...init.headers
        }
    }).then(response => {
        responseIntercept(response);
    }).catch(error => {
        console.error(`Ha ocurrido un error al pedir el recurso ${resource} con el
        mensaje: ${error.message}`);
    })
}
```

PARA RESUMIR

- Los modelos de front-end **no tienen porqué** coincidir con los de back-end, y pueden disponer de métodos que nos faciliten su uso
- Un interceptor es una función que **procesa la respuesta o el envío** de una petición y añade funcionalidad global para cualquiera que se realice