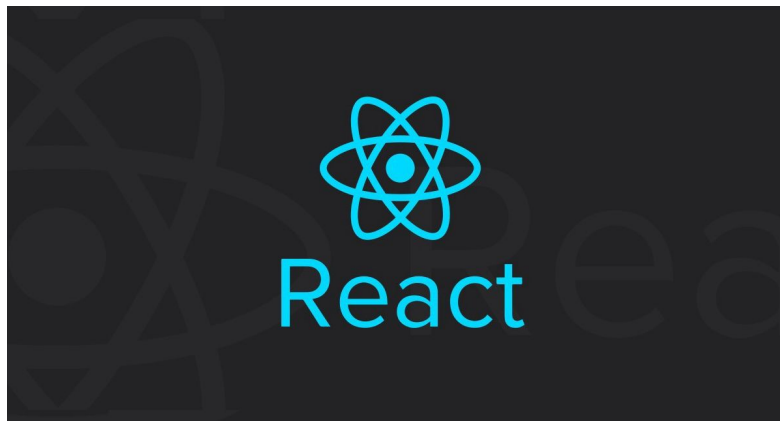
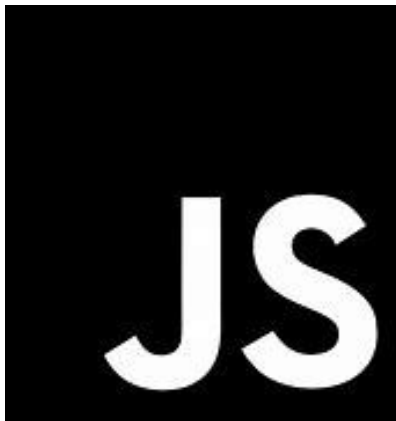


comfama

Software Development Challenge
Ximena Vasco

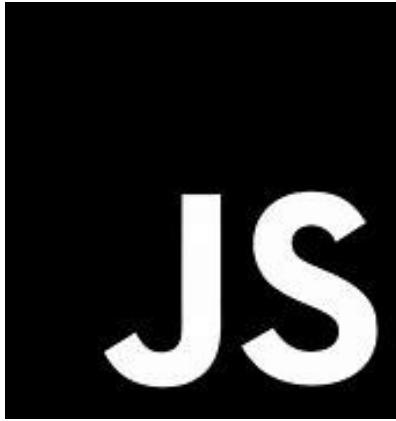
1. Front-end technologies

— — —



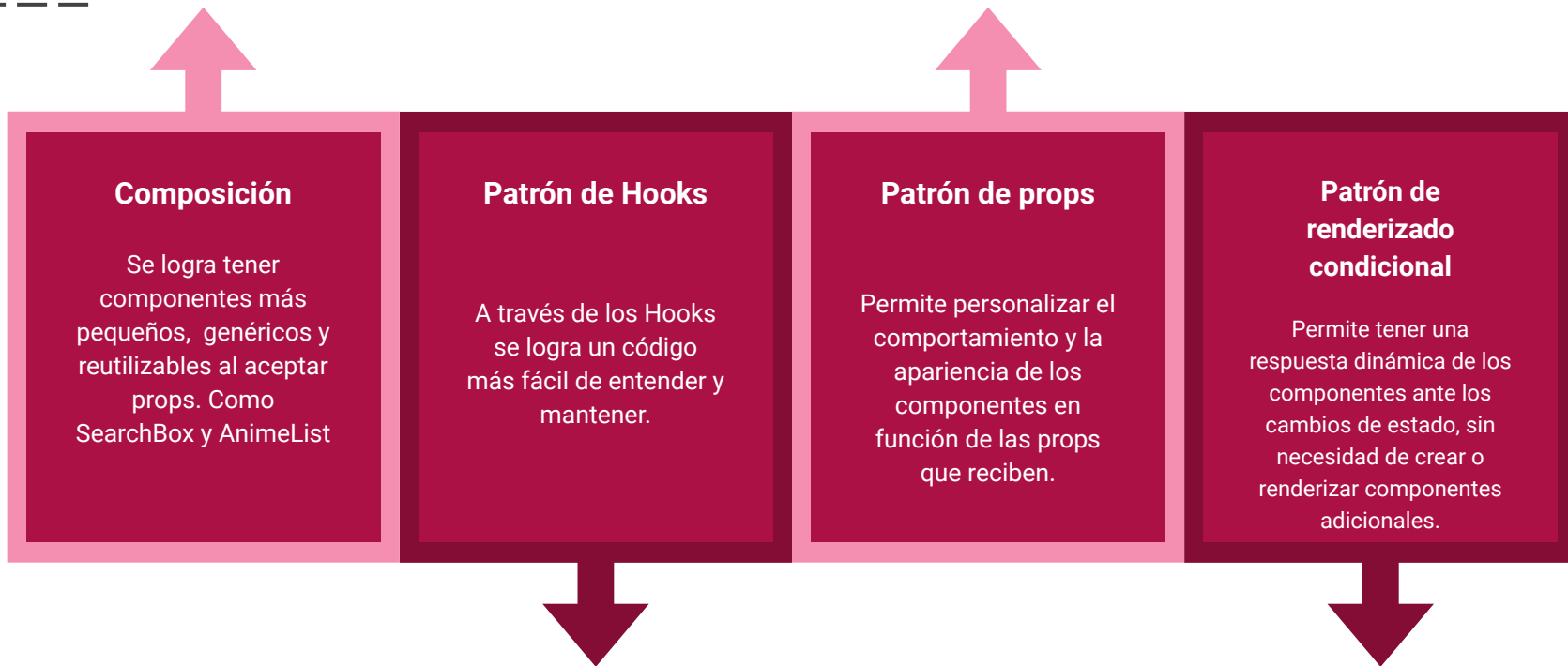
2. Back-end technologies

— — —



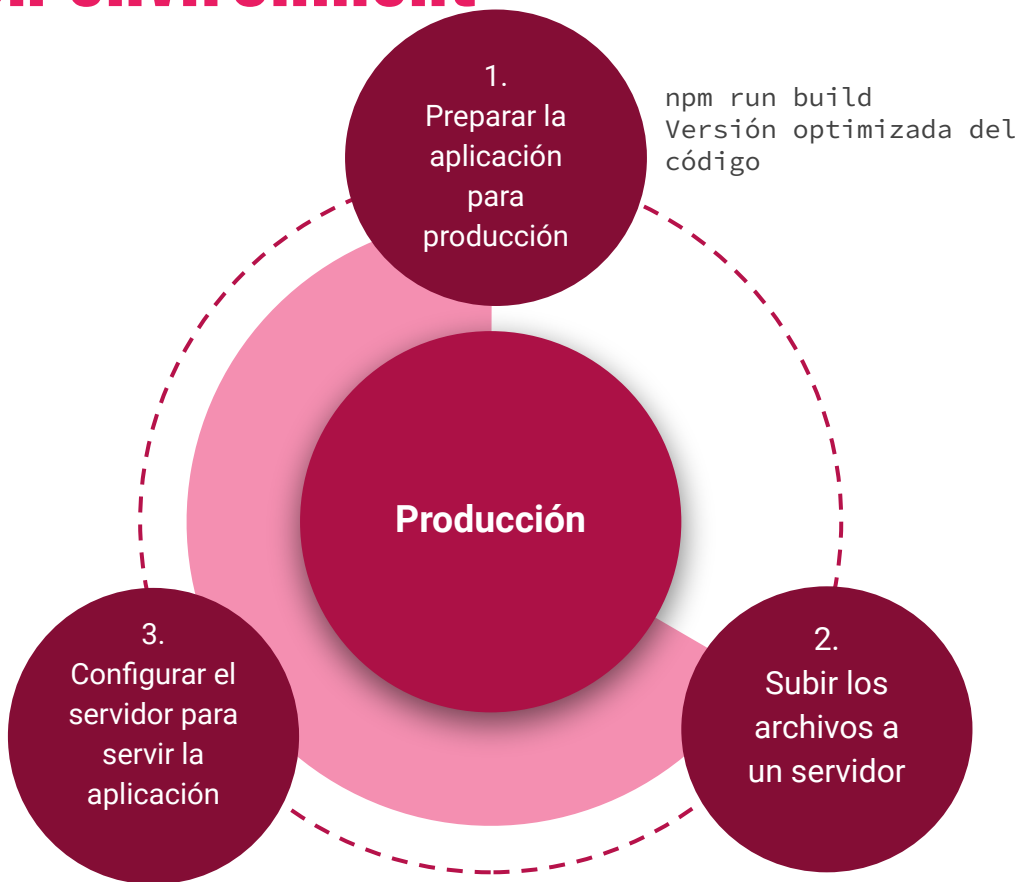
3. Design Pattern

— — —



4. Production environment

— — —



5. Data persistence

— — —

Local Storage

Crear un custom hook y usar localStorage para almacenar la información de la búsqueda actual.

APIs del lado del servidor

Considerar el uso de una base de datos en el servidor.

Comunicar la App con la API creada en el servidor de express para enviar y recuperar datos

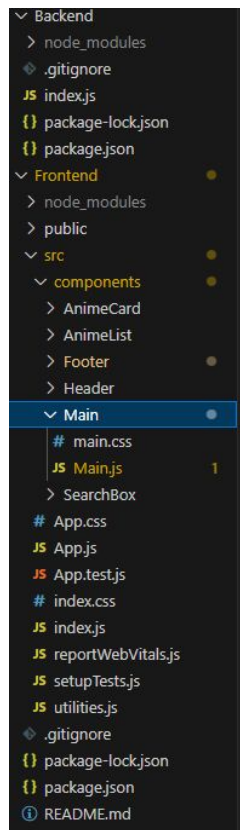
6. Relevant points (1/5)

— — —

Estructura de Carpetas

Dos Directorios principales:

1. **Backend:** contiene un archivo principal, `index.js`, el cual contiene el servidor a través de `express` y los endpoints necesarios.
2. **Frontend:** Dentro del cual se encuentra “src”, que contiene una carpeta “Componentes”, la cual aloja un directorio correspondiente a cada componente usado y éste a su vez contiene el archivo `.js` con la lógica y el `.css` con el estilo. Adicional, hay un archivo “`utilities.js`”, el cual contiene los `fetch`.



6. Relevant points (2/5)

— — —

```
function Main() {  
  const [search, setSearch] = useState("Naruto");  
  const [animeData, setAnimeData] = useState();  
  const [error, setError] = useState(null);  
  const [isLoading, setIsLoading] = useState(false);  
  const [scoreAvg, setScoreAvg] = useState([]);  
}
```

Estados

Dentro del componente Main, se definen los estados de la app:

1. **Search:** criterio de búsqueda definido por el usuario.
2. **animeData:** Información obtenida de la Api externa, ligada al criterio de búsqueda.
3. **scoreAvg:** Información obtenida de la Api interna, ligada a la información del anime.
4. **error:** estado de error durante la carga de la información.
5. **isLoading:** estado de carga de la información.

6. Relevant points (3/5)

— — —

Fetch

1. Petición de la información del anime definido por el usuario a la Api externa. La lógica de la función `getAnimeData` está alojada en `utilities`.
2. Petición a la Api interna del promedio del score de las diferentes temporadas del anime. La Api funciona en el servidor alojado en el puerto 3001

```
const getAnimeDataList = async () => {
  setIsLoading(true);
  try {
    const data = await getAnimeData(search);
    setAnimeData(data);
  } catch (error) {
    setError(error);
  } finally {
    setIsLoading(false);
  }
};

useEffect(() => {
  getAnimeDataList();
}, []);
```

```
useEffect(() => {
  if (animeData) {
    const scoresList = animeData.map((anime) => anime.score);

    getScoresAvg(scoresList)
      .then((data) => {
        setScoreAvg(data);
      })
      .catch((error) => {
        console.log(error);
      });
  }
}, [animeData]);
```

6. Relevant points (4/5)

```
JS Main.js 1, M JS utilities.js M X
Frontend > src > JS utilities.js > [0] getAnimeData
1 { // Fetch to External API
2
3 const getAnimeData = async (search) => {
4   const response = await fetch(
5     `https://api.jikan.moe/v4/anime?q=${search}&sfw&limit=10`
6   );
7
8   if (!response.ok) {
9     throw new Error(`HTTP error! status: ${response.status}`);
10  }
11
12   const data = await response.json();
13   return data.data;
14 };
15
16 //Calculate scores Average in Internal API
17
18 const getScoresAvg = async (scores) => {
19   const response = await fetch("http://localhost:3001/scoresAverage", {
20     method: "POST",
21     headers: {
22       "Content-Type": "application/json",
23     },
24     body: JSON.stringify(scores),
25   });
26
27   if (!response.ok) {
28     throw new Error(`Error al hacer la petición POST: ${response.status}`);
29   }
30
31   const scoresAvg = await response.json();
32   return scoresAvg;
33 };
34
35 export { getAnimeData, getScoresAvg };
36
```

Archivo “utilities.js”

1. Este archivo aloja la lógica de las peticiones realizadas a la Api externa e interna, para el correcto funcionamiento de la App.

6. Relevant points (5/5)

Servidor y endpoints

En el archivo `index.js` del directorio Backend se encuentra:

1. El servidor creado e iniciado en el puerto 3001, a través de express.
2. El middleware que permite parsear la información que llega desde el front.
3. La configuración de cors.
4. el endpoint `/scoresAverage`, que calcula el promedio de los scores de las diferentes temporadas del anime.

```
JS Main.js 1, M  JS utilities.js M  JS index.js X
Backend > JS index.js > ...
1  const express = require("express");
2  const bodyParser = require("body-parser");
3  const cors = require("cors");
4
5  const PORT = process.env.PORT || 3001;
6  const app = express();
7
8  app.use(bodyParser.json());
9  app.use(cors());
10
11 app.get("/", (req, res) => {
12   res.send("Api para mi Anime Browser!");
13 });
14
15 app.post("/scoresAverage", (req, res) => {
16   const scores = req.body;
17   const sum = scores.reduce((acc, curr) => acc + curr, 0);
18   const avg = sum / scores.length;
19   res.json({ avg });
20 });
21
22 app.listen(PORT, () => {
23   console.log("Server running on port 3001");
24 });
25
```

7. Aspects with more difficulties

— — —

Middleware

Congruencia de la información entre el front y el back.

Codigo Modular

Modularizar el código, con el fin de lograr mejor comprensión para el lector y mejor mantenimiento y/o modificación en el futuro.

Errores y estado de carga

Lograr capturar los errores y manejar el estado de carga con el fin de dar una mejor experiencia al usuario.

Gracias!