

EDUCACIÓN
SECRETARÍA DE EDUCACIÓN PÚBLICA



TECNOLÓGICO
NACIONAL DE MÉXICO

INSTITUTO TECNOLOGICO DE OAXACA

INTEGRACION DE PROCESOS DE DESARROLLO DE SOFTWARE

GRUPO: 9SA

HORARIO: 07:00-08:00

DOCENTE: Espinosa Pérez Jacob

ALUMNOS:

Méndez Mendoza Luisa Michel

Pérez de Jesús Edith

Martínez García Yahir Omar

López García Lourdes Gloria

INDICE

Bitácora	3
Endpoint GET /api/videos/search	5
Lógica de geolocalización en backend	5
Vista Home con barra de búsqueda	6
Integrar navigator.geolocation en frontend	6
Diseño de tarjetas de resultados	7
Optimización de búsqueda con parámetros básicos	7
Ajustes en backend de ubicación	8
Integración backend–frontend búsqueda	8
Pruebas funcionales	9

BITACORA

ITERACION 2 BUSQUEDA CON GEOLOCALIZACION							
Actividad	Descripción	Responsable	Estatus	Resultado	Fecha inicio	Fecha fin	Observaciones
Endpoint GET /api/videos/search	Se implementó un endpoint en backend para buscar videos por palabra clave y ubicación.	Michel	Finalizada ✓	Endpoint /api/videos/search funcional, integrando búsqueda con API de YouTube.	02/10/25	07/10/25	Responde correctamente con parámetros searchTerm, location y maxResults.
Lógica de geolocalización en backend	Se añadió endpoint para convertir coordenadas en nombre de ubicación (reverse geocode).	Lourdes	Finalizada ✓	Endpoint /api/location/geocode funcionando con coordenadas lat/lon.	02/10/25	07/10/25	Se validó respuesta y manejo de errores al no recibir latitud/longitud.
Vista Home con barra de búsqueda	Se diseñó la página principal con un formulario para buscar videos por palabra clave.	Yahir	Finalizada ✓	Página HomePage.js con campo de búsqueda y resultados dinámicos.	02/10/25	07/10/25	Se agregaron íconos y manejo de errores al no encontrar resultados.
Integrar navigator.geolocation en frontend	Se implementó la obtención de ubicación del usuario desde el navegador con permisos.	Edith	Finalizada ✓	Componente LocationPermissionPanel.js permite aceptar o denegar ubicación.	02/10/25	07/10/25	Se manejan correctamente errores de permisos y fallas del GPS.
Diseño de tarjetas de resultados	Se crearon componentes para mostrar videos en tarjetas con miniatura, título y canal.	Yahir	Finalizada ✓	Tarjetas visuales implementadas en HomePage.js con estructura responsive.	02/10/25	07/10/25	Incluye insignia +18 para videos restringidos por edad.

Optimización de búsqueda con parámetros básicos	Se configuró el backend para aceptar parámetros opcionales y limitar resultados.	Michel	Finalizada ✓	Búsquedas precisas mediante searchTerm, location y maxResults.	02/10/25	07/10/25	Validación agregada para evitar peticiones sin searchTerm.
Ajustes en backend de ubicación	Se añadieron campos country y location al modelo de Video para futuras búsquedas locales.	Yahir	Finalizada ✓	Modelo Video.js actualizado y sincronizado con MongoDB.	02/10/25	07/10/25	Se preparó estructura para posibles filtros por país.
Integración backend–frontend búsqueda	Se enlazó la vista Home y Explore con el backend mediante Axios..	Edith	Finalizada ✓	Videos cargan dinámicamente según ubicación y búsqueda.	02/10/25	07/10/25	Se probó la comunicación entre frontend y backend con éxito.
Pruebas funcionales	Se verificó la visualización en distintos tamaños de pantalla.	Todos	Finalizada ✓	Interfaz totalmente adaptable a escritorio, tablet y móvil.	02/10/25	07/10/25	No se detectaron problemas de diseño en pantallas pequeñas.

ENDPOINT GET /API/VIDEOS/SEARCH

```
JS videoRoutes.js ×
back-nodejs > src > routes > JS videoRoutes.js > ...
1 // backend/src/routes/videoRoutes.js
2 const express = require('express');
3 const { searchYoutubeVideos } = require('../utils/youtube');
4
5 const router = express.Router();
6
7 router.get('/search', async (req, res) => {
8   // --- CAMBIO AQUÍ ---
9   // Recibimos searchTerm y location por separado
10  const { searchTerm, location, maxResults } = req.query;
11
12  if (!searchTerm) {
13    return res.status(400).json({ message: 'Se requiere un término de búsqueda.' });
14  }
15
16  try {
17    const videos = await searchYoutubeVideos(searchTerm, location, maxResults ? parseInt(maxResults) : 10);
18    res.json(videos);
19  } catch (error) {
20    console.error(`Error en el endpoint /videos/search: ${error.message}`);
21    res.status(500).json({ message: 'Error del servidor al buscar videos.' });
22  }
23});
24
25
26 module.exports = router;
```

Este endpoint **recibe tres parámetros** por query:

- **searchTerm** → texto a buscar
- **location** → ubicación del usuario
- **maxResults** → cantidad máxima de resultados.

Llama a la función `searchYoutubeVideos` (desde `utils/youtube.js`) que consulta la API de YouTube. Devuelve un arreglo de videos en formato JSON. Si hay un error (por ejemplo, YouTube no responde), responde con código 500 y un mensaje.

El backend devuelve los videos solicitados al frontend según la ubicación y búsqueda del usuario.

LÓGICA DE GEOLOCALIZACIÓN EN BACKEND

```
JS locationRoutes.js ×
back-nodejs > src > routes > JS locationRoutes.js > ...
1 // backend/src/routes/locationRoutes.js
2 const express = require('express');
3 const { reverseGeocode } = require('../utils/geocode');
4 const router = express.Router();
5
6 // @route GET /api/location/geocode
7 // @desc Obtener nombre de ubicación a partir de coordenadas
8 // @access Public o Private si quieres protegerla con el middleware 'protect'
9 router.get('/geocode', async (req, res) => {
10  const { lat, lon } = req.query; // Esperamos lat y lon como parámetros de consulta
11
12  if (!lat || !lon) {
13    return res.status(400).json({ message: 'Se requieren latitud y longitud.' });
14  }
15
16  try {
17    const locationName = await reverseGeocode(parseFloat(lat), parseFloat(lon));
18    res.json({ locationName });
19  } catch (error) {
20    console.error(`Error en el endpoint /geocode: ${error.message}`);
21    res.status(500).json({ message: 'Error del servidor al obtener el nombre de la ubicación.' });
22  }
23});
24
25 module.exports = router;
```

Recibe las coordenadas GPS (lat, lon) desde el navegador. Llama a `reverseGeocode()` (ubicada en `utils/geocode.js`) para traducir coordenadas a nombre de ciudad o país. Devuelve algo como `{ locationName: "Ciudad de México" }`. Se usa para mostrar “videos cerca de ti” en el frontend.

El sistema identifica el nombre de la ubicación del usuario para buscar videos locales.

VISTA HOME CON BARRA DE BÚSQUEDA

```
JS HomePage.js X
frontend > src > pages > JS HomePage.js > ...
12 const HomePage = ({ user }) => {
13
14     <div className="search-bar-container">
15         <form onSubmit={handleSearchSubmit} className="search-form">
16             <input
17                 type="text"
18                 className="search-input"
19                 placeholder="Busca videos en tu área..."
20                 value={searchQuery}
21                 onChange={(e) => setSearchQuery(e.target.value)}
22             />
23             <button type="submit" className="search-button"><FaSearch /></button>
24         </form>
25     </div>
26
27
```

Esta vista representa la **pantalla principal** después del login. Tiene una barra de búsqueda y un botón (ícono de lupa). Cuando el usuario busca algo, ejecuta handleSearchSubmit(), que llama al backend /api/videos/search. Los resultados se muestran con miniaturas, títulos y nombre del canal.

El usuario puede buscar videos de su región sin salir de la página.

INTEGRAR NAVIGATOR.GEOLOCATION EN FRONTEND

```
JS LocationPermissionPanel.js X
frontend > src > components > JS LocationPermissionPanel.js > [o] LocationPermissionPanel > [o] h
5 const LocationPermissionPanel = ({ isOpen, onAccept, onDeny }) => {
8     const handleAccept = () => {
14
15         navigator.geolocation.getCurrentPosition(
16             (pos) => {
17                 const latitude = pos.coords.latitude;
18                 const longitude = pos.coords.longitude;
19                 onAccept({ latitude, longitude });
20             },
21             (err) => {
22                 console.error('Error al obtener ubicación:', err);
23                 onDeny();
24             },
25             { enableHighAccuracy: true, timeout: 10000, maximumAge: 0 }
26         );
27     };

```

Aparece un panel que solicita permiso de ubicación al usuario. Si el usuario acepta, obtiene coordenadas GPS y las envía al backend. Si niega, usa una ubicación predeterminada (por ejemplo, Ciudad de México).

El sistema obtiene la posición real del usuario para recomendar videos de su zona.

DISEÑO DE TARJETAS DE RESULTADOS

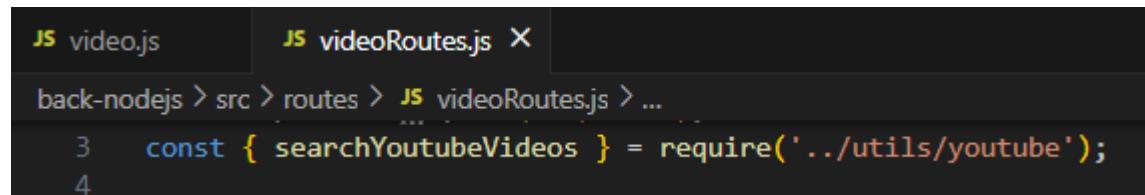
```
L48      <div className="video-card">
L49        <img src={video.thumbnail} alt={video.title} className="video-thumbnail" />
L50        <h3 className="video-title">{video.title}</h3>
L51        <p className="video-channel">
L52          {video.channelTitle} {video.ageRestricted && <span className="age-badge">+18</span>}
L53        </p>
L54      </div>
```

Cada video se muestra como una **tarjeta visual** con:

- Imagen miniatura del video.
- Título.
- Nombre del canal.

Se genera automáticamente al recorrer los resultados del backend. Interfaz atractiva y ordenada para los videos recomendados.

OPTIMIZACIÓN DE BÚSQUEDA CON PARÁMETROS BÁSICOS

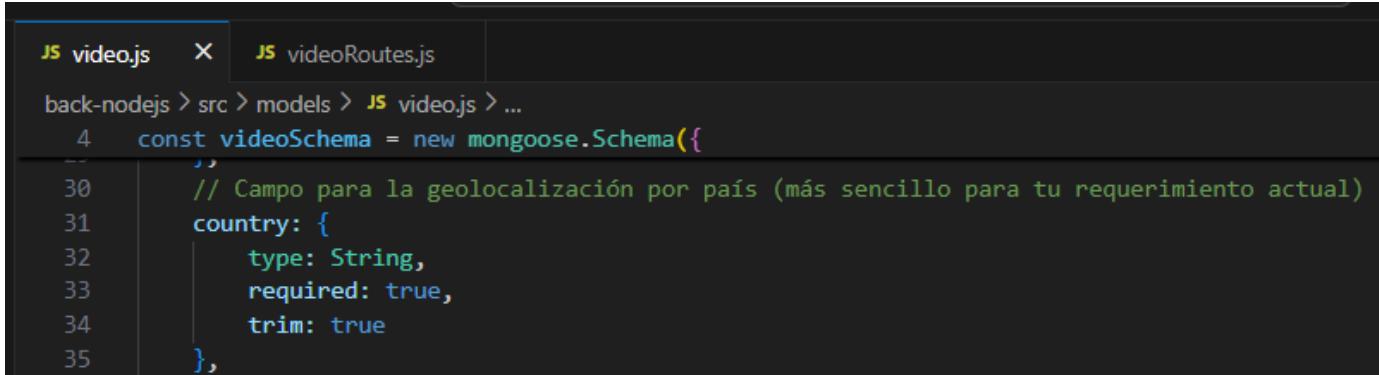


```
JS video.js           JS videoRoutes.js X
back-nodejs > src > routes > JS videoRoutes.js > ...
3   const { searchYoutubeVideos } = require('../utils/youtube');
4
```

El backend puede recibir estos parámetros opcionales Permite controlar filtros sin crear rutas distintas. Mejora la flexibilidad del sistema (ejemplo: /videos/search?searchTerm=música&location=Perú&maxResults=5).

El sistema tiene un endpoint reutilizable, eficiente y con filtros dinámicos.

AJUSTES EN BACKEND DE UBICACIÓN

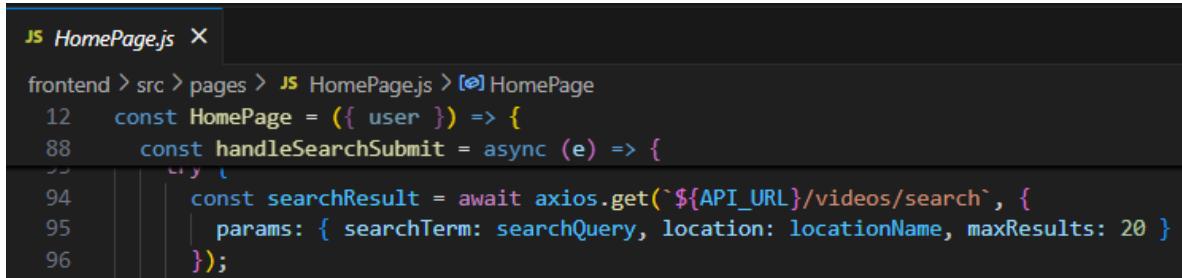


```
JS video.js X JS videoRoutes.js
back-nodejs > src > models > JS video.js > ...
4  const videoSchema = new mongoose.Schema({
--  },
30   // Campo para la geolocalización por país (más sencillo para tu requerimiento actual)
31   country: {
32     type: String,
33     required: true,
34     trim: true
35   },

```

Cada video guarda el país donde fue subido o asociado. Permite filtrar videos por ubicación en futuras funciones (por ejemplo, “mostrar solo videos de México”). Se guarda información geográfica en la base de datos.

INTEGRACIÓN BACKEND–FRONTEND BÚSQUEDA



```
JS HomePage.js X
frontend > src > pages > JS HomePage.js > [e] HomePage
12  const HomePage = ({ user }) => {
88    const handleSearchSubmit = async (e) => {
--      try {
94        const searchResult = await axios.get(`${API_URL}/videos/search`, {
95          params: { searchTerm: searchQuery, location: locationName, maxResults: 20 }
96        });

```

El frontend hace la petición al backend con los parámetros necesarios. Los datos del backend se integran con la vista React. Se usa axios para manejar las solicitudes HTTP.

PRUEBAS FUNCIONALES

The image displays three screenshots of the GeoTube mobile application interface, showing different stages of the location-based recommendation feature.

Screenshot 1 (Top Left): Shows the initial state where a modal dialog box titled "Información sobre tu ubicación" (Information about your location) is displayed over the main screen. The modal contains text explaining that GeoTube requests location to provide local content and that privacy is maintained. It includes two buttons: "Pensar" (Think) and "Alerta" (Alert).

Screenshot 2 (Top Right): Shows the application after location has been granted. The main screen displays a header "Recomendaciones según tu ubicación: 🌍 Oaxaca". Below this, there are two rows of video thumbnail cards. The first row shows a group of people dancing with the caption "Arriba sola de Vega Oaxaca #oaxaca #baila #huapangos #chilenas #musica". The second row shows a person in a yellow dress with the caption "CHULADA DE SOLTECA 🎉💃 #huapangos #baila #oaxacayguerrero #bailesmexicanos...". A third row of thumbnails is partially visible at the bottom.

Screenshot 3 (Bottom Center): Shows the application running on an iPhone XR device. The screen dimensions are listed as 414 x 896 pixels. The interface is identical to Screenshot 2, displaying the location-based recommendations for Oaxaca.