Manual técnico caso de estudio NutriTrack

Aplicaciones Progresivas

Universidad Tecnológica de Querétaro

Alumno: Alfonso Esaú Leyva Pérez

Profesor: Manuel Contreras Castillo

Fecha: 30 de septiembre de 2025

Índice

1. Introducción

1.1. Propósito del Documento

Este documento tiene como objetivo presentar un análisis técnico detallado de la aplicación web progresiva NutriTrack, una solución para el seguimiento y control de la nutrición personal. La aplicación permite a los usuarios registrar sus comidas diarias, escanear códigos de barras de productos alimenticios, consultar información nutricional y mantener un registro de su consumo calórico y de macronutrientes, incluso en condiciones sin conexión a internet.

1.2. Alcance del Sistema

NutriTrack abarca dos componentes principales: una aplicación cliente desarrollada con Vue.js que implementa las características de una PWA (Progressive Web Application) y un servidor backend que proporciona APIs para la gestión de datos nutricionales. El sistema soporta operaciones offline mediante IndexedDB, sincronización automática cuando se recupera la conexión, y utiliza la cámara del dispositivo para escanear códigos de barras de productos alimenticios.

1.3. Convenciones del Documento

En este documento se utilizan las siguientes convenciones:

- **PWA**: Progressive Web Application, aplicación web que funciona como aplicación nativa.
- **Vue.js**: Framework JavaScript progresivo utilizado para construir la interfaz de usuario.
- **IndexedDB**: Base de datos del navegador para almacenamiento local.
- **API**: Application Programming Interface, conjunto de reglas que permiten la comunicación entre aplicaciones.
- **Offline-first**: Enfoque de diseño que prioriza el funcionamiento sin conexión.
- **Repositorio del sistema:** `https://github.com/Emaus025/NutriTrack`

2. Arquitectura del Sistema

2.1. Visión General

NutriTrack implementa una arquitectura cliente-servidor donde el cliente es una PWA desarrollada con Vue.js y el servidor proporciona APIs RESTful para la gestión de datos. La aplicación utiliza un enfoque offline-first, permitiendo a los usuarios interactuar con la aplicación incluso sin conexión a internet, sincronizando los datos cuando la conexión se restablece.

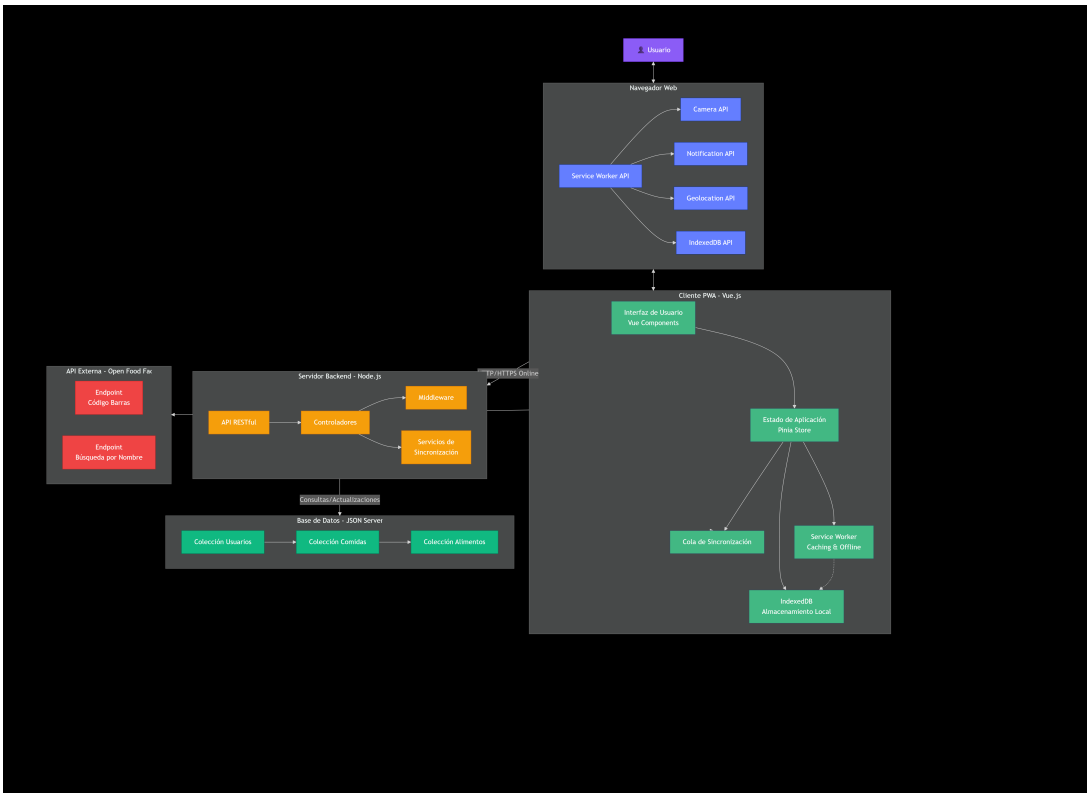

Figura 1: Diagrama de arquitectura de NutriTrack

2.2. Componentes del Sistema

2.2.1. Cliente PWA (Vue.js)

La aplicación cliente implementa las siguientes características:

- Interfaz de usuario responsiva adaptable a diferentes dispositivos
- Almacenamiento local con IndexedDB para operaciones offline
- Service Worker para caching de recursos y funcionamiento offline
- Acceso a la cámara para escaneo de códigos de barras
- Sincronización automática con el servidor cuando hay conexión
- Notificaciones push para recordatorios y actualizaciones

2.2.2. Servidor Backend

El servidor proporciona los siguientes servicios:

- API RESTful para gestión de comidas y alimentos
- Base de datos de productos alimenticios
- Integración con APIs externas para información nutricional
- Sistema de autenticación y gestión de usuarios
- Servicio de notificaciones push

2.2.3. Base de Datos

El sistema utiliza dos tipos de almacenamiento:

- **Servidor**: Base de datos JSON para almacenar información de usuarios, comidas y alimentos
- **Cliente**: IndexedDB para almacenamiento local y funcionamiento offline

2.3. Patrones de Diseño Implementados

El sistema implementa varios patrones de diseño:

- **Patrón MVC**: Separación de la lógica de negocio, datos y presentación.
- **Patrón Observador**: Implementado mediante el sistema reactivo de Vue.js.
- **Patrón Repositorio**: Para abstraer el acceso a datos tanto locales como remotos.
- **Patrón Estrategia**: Para manejar diferentes estrategias de sincronización.
- **Patrón Singleton**: Utilizado para gestionar instancias únicas como el gestor de sincronización.

3. Requisitos del Sistema

3.1. Requisitos Funcionales

- **RF1**: La aplicación debe permitir registrar comidas con fecha, nombre y alimentos.
- **RF2**: La aplicación debe calcular automáticamente calorías y macronutrientes.
- **RF3**: La aplicación debe permitir buscar alimentos por nombre.
- **RF4**: La aplicación debe permitir escanear códigos de barras de productos.
- **RF5**: La aplicación debe funcionar sin conexión a internet.
- **RF6**: La aplicación debe sincronizar datos automáticamente cuando haya conexión.
- **RF7**: La aplicación debe enviar notificaciones para recordatorios.
- **RF8**: La aplicación debe mostrar un historial de comidas por fecha.

3.2. Requisitos No Funcionales

- **RNF1**: La aplicación debe ser instalable como PWA en dispositivos móviles.
- **RNF2**: La aplicación debe tener un tiempo de respuesta menor a 2 segundos.
- **RNF3**: La aplicación debe ser responsiva y adaptarse a diferentes tamaños de pantalla.
- **RNF4**: La aplicación debe ser compatible con los navegadores modernos.
- **RNF5**: La aplicación debe minimizar el uso de datos móviles mediante caching.
- **RNF6**: La aplicación debe proporcionar feedback visual durante operaciones asíncronas.

4. Implementación

4.1. Tecnologías Utilizadas

- **Frontend**:
  - Vue.js 3: Framework JavaScript para la interfaz de usuario
  - Vite: Herramienta de construcción y desarrollo
  - Pinia: Gestión de estado de la aplicación
  - IndexedDB: Almacenamiento local en el navegador
  - Service Workers: Para funcionalidad offline y caching
  - QuaggaJS: Biblioteca para escaneo de códigos de barras
- **Backend**:
  - Node.js: Entorno de ejecución para el servidor
  - JSON Server: API REST simple basada en archivos JSON
  - Axios: Cliente HTTP para comunicación con APIs

4.2. Estructura del Proyecto

La estructura del proyecto sigue una organización modular:

Listing 1: Estructura de directorios del proyecto

```
NutriTrack/
          client/
                public/
                      manifest.json
                      service-worker.js
                src/
                      App.vue
                      main.js
                      router.js
                      registerServiceWorker.js
                      utils/
                            camera.js
                            idb.js
                            productApi.js
                      views/
                          Home.vue
                          Meals.vue
                          Profile.vue
                index.html
                package.json
                vite.config.js
          server/
              db.json
```

4.3. Componentes Principales

4.3.1. Módulo de Registro de Comidas

El componente Meals.vue implementa la funcionalidad para registrar comidas, buscar alimentos y calcular valores nutricionales:

Listing 2: Fragmento de código del componente Meals.vue

```
// Escanear c digo de barras
const scanBarcode = async () => {
  try {
    showBarcodeScanner.value = true;
    const barcode = await openCamera();
    if (barcode) {
      isSearching.value = true;
      const product = await searchProductByBarcode(barcode);
      if (product) {
        addFoodToMeal(product);
      }
      isSearching.value = false;
    }
  } catch (error) {
    console.error('Error al escanear c digo de barras:', error);
    isSearching.value = false;
  } finally {
    showBarcodeScanner.value = false;
  }
};
```

4.3.2. Módulo de Escaneo de Códigos de Barras

El archivo camera.js implementa la funcionalidad para acceder a la cámara y escanear códigos de barras utilizando QuaggaJS:

Listing 3: Fragmento de código del módulo de cámara

```
// Funci n para abrir la c mara y escanear un c digo de barras
export async function openCamera() {
  return new Promise(async (resolve, reject) => {
    try {
      // Comprobar si el navegador soporta la API de MediaDevices
      if (!navigator.mediaDevices || !navigator.mediaDevices.
         getUserMedia) {
        throw new Error('Tu navegador no soporta acceso a la
           c mara ');
      }

      // Inicializar Quagga para el escaneo de c digos de barras
      Quagga.init({
        inputStream: {
          name: "Live",
          type: "LiveStream",
```

```
          target: document.querySelector("#barcode-scanner"),
          constraints: {
            width: 480,
            height: 320,
            facingMode: "environment"
          },
        },
        decoder: {
          readers: [
            "ean_reader",
            "ean_8_reader",
            "code_128_reader",
            "code_39_reader",
            "upc_reader"
          ]
        }
      }, function(err) {
        if (err) {
          reject(err);
          return;
        }
        Quagga.start();
      });

      // Detectar c digos de barras
      Quagga.onDetected((result) => {
        const code = result.codeResult.code;
        Quagga.stop();
        resolve(code);
      });
    } catch (error) {
      reject(error);
    }
  });
}
```

4.3.3. Módulo de Sincronización Offline

El archivo idb.js implementa la funcionalidad para almacenamiento local y sincronización:

Listing 4: Fragmento de código del módulo de sincronización

```
export class SyncManager {
  // Guardar comida offline
  static async saveMealOffline(meal) {
    const db = await openDB('nutritrack', 1, {
      upgrade(db) {
        if (!db.objectStoreNames.contains('meals')) {
          db.createObjectStore('meals', { keyPath: 'id' });
        }
        if (!db.objectStoreNames.contains('sync-queue')) {
```

```
          db.createObjectStore('sync-queue', { keyPath: 'id' });
        }
      }
    });

    // Generar ID temporal
    const tempId = 'temp_' + Date.now();
    meal.id = tempId;

    // Guardar en IndexedDB
    await db.put('meals', meal);

    // A adir a la cola de sincronizaci n
    await db.put('sync-queue', {
      id: tempId,
      operation: 'create',
      data: meal,
      timestamp: Date.now()
    });

    // Registrar un sync event si est  disponible
    if ('serviceWorker' in navigator && 'SyncManager' in window)
       {
      const registration = await navigator.serviceWorker.ready;
      await registration.sync.register('sync-meals');
    }

    return meal;
  }
}
```

5. Características PWA Implementadas

5.1. Service Worker

La aplicación implementa un Service Worker para habilitar la funcionalidad offline y mejorar el rendimiento mediante caching:

Listing 5: Configuración del Service Worker

```
// registerServiceWorker.js
import { register } from 'register-service-worker'

if (process.env.NODE_ENV === 'production') {
  register(`${process.env.BASE_URL}service-worker.js`, {
    ready() {
      console.log('App is being served from cache by a service
         worker.')
    },
    registered() {
      console.log('Service worker has been registered.')
    },
    cached() {
      console.log('Content has been cached for offline use.')
    },
    updatefound() {
      console.log('New content is downloading.')
    },
    updated() {
      console.log('New content is available; please refresh.')
    },
    offline() {
      console.log('No internet connection found. App is running
         in offline mode.')
    },
    error(error) {
      console.error('Error during service worker registration:',
         error)
    }
  })
}
```

5.2. Manifest.json

El archivo manifest.json define las características de la PWA para su instalación:

Listing 6: Archivo manifest.json

```
{
  "name": "NutriTrack",
  "short_name": "NutriTrack",
  "theme_color": "#4CAF50",
  "background_color": "#ffffff",
  "display": "standalone",
```

```
  "orientation": "portrait",
  "scope": "/",
  "start_url": "/",
  "icons": [
    {
      "src": "icons/icon-72x72.png",
      "sizes": "72x72",
      "type": "image/png"
    },
    {
      "src": "icons/icon-96x96.png",
      "sizes": "96x96",
      "type": "image/png"
    },
    {
      "src": "icons/icon-128x128.png",
      "sizes": "128x128",
      "type": "image/png"
    },
    {
      "src": "icons/icon-144x144.png",
      "sizes": "144x144",
      "type": "image/png"
    },
    {
      "src": "icons/icon-152x152.png",
      "sizes": "152x152",
      "type": "image/png"
    },
    {
      "src": "icons/icon-192x192.png",
      "sizes": "192x192",
      "type": "image/png"
    },
    {
      "src": "icons/icon-384x384.png",
      "sizes": "384x384",
      "type": "image/png"
    },
    {
      "src": "icons/icon-512x512.png",
      "sizes": "512x512",
      "type": "image/png"
    }
  ]
}
```

5.3. Notificaciones Push

La aplicación implementa notificaciones push para recordatorios y actualizaciones:

Listing 7: Implementación de notificaciones

```
// Solicitar permiso para notificaciones
async function requestNotificationPermission() {
  if (!('Notification' in window)) {
    console.log('Este navegador no soporta notificaciones');
    return false;
  }

  const permission = await Notification.requestPermission();
  return permission === 'granted';
}

// Enviar notificaci n
async function sendNotification(title, options) {
  if (await requestNotificationPermission()) {
    const registration = await navigator.serviceWorker.ready;
    registration.showNotification(title, options);
  }
}

// Ejemplo de uso para recordatorio
function scheduleReminder() {
  const now = new Date();
  const scheduledTime = new Date(
    now.getFullYear(),
    now.getMonth(),
    now.getDate(),
    12, 0, 0 // 12:00 PM
  );

  const timeUntilReminder = scheduledTime - now;
  if (timeUntilReminder > 0) {
    setTimeout(() => {
      sendNotification('Recordatorio de NutriTrack', {
        body: ' Es  hora de registrar tu comida!',
        icon: '/icons/icon-192x192.png',
        badge: '/icons/badge-72x72.png',
        vibrate: [100, 50, 100],
        data: {
          url: '/meals'
        }
      });
    }, timeUntilReminder);
  }
}
```

6. Uso de Elementos Físicos del Dispositivo

6.1. Acceso a la Cámara

La aplicación utiliza la API MediaDevices para acceder a la cámara del dispositivo y escanear códigos de barras:

Listing 8: Acceso a la cámara del dispositivo

```
// Comprobar si el navegador soporta la API de MediaDevices
if (!navigator.mediaDevices || !navigator.mediaDevices.
   getUserMedia) {
  throw new Error('Tu navegador no soporta acceso a la c mara');
}

// Solicitar acceso a la c mara
const stream = await navigator.mediaDevices.getUserMedia({
  video: {
    facingMode: 'environment', // Usar c mara trasera si est
       disponible
    width: { ideal: 1280 },
    height: { ideal: 720 }
  }
});

// Asignar el stream a un elemento de video
videoElement.srcObject = stream;
```

6.2. Detección de Conectividad

La aplicación detecta cambios en la conectividad para implementar la sincronización automática:

Listing 9: Detección de conectividad

```
// Detectar cambios en la conectividad
window.addEventListener('online', async () => {
  console.log('Conexi n restablecida, sincronizando datos...');
  await SyncManager.syncPendingData();
});

window.addEventListener('offline', () => {
  console.log('Conexi n perdida, cambiando a modo offline');
  // Mostrar notificaci n al usuario
  if (Notification.permission === 'granted') {
    new Notification('NutriTrack', {
      body: 'Conexi n perdida. La app est  funcionando en modo
         offline.'
    });
  }
});
```

6.3. Geolocalización

La aplicación utiliza la API de Geolocalización para sugerir alimentos locales:

Listing 10: Uso de geolocalización

```
// Obtener ubicaci n del usuario
async function getUserLocation() {
  return new Promise((resolve, reject) => {
    if (!navigator.geolocation) {
      reject(new Error('Geolocalizaci n no soportada por este
         navegador'));
      return;
    }

    navigator.geolocation.getCurrentPosition(
      position => {
        resolve({
          latitude: position.coords.latitude,
          longitude: position.coords.longitude
        });
      },
      error => {
        reject(error);
      },
      {
        enableHighAccuracy: true,
        timeout: 5000,
        maximumAge: 0
      }
    );
  });
}

// Sugerir alimentos locales basados en ubicaci n
async function suggestLocalFoods() {
  try {
    const location = await getUserLocation();
    // Aqu  se implementar a la l gica para buscar alimentos
       locales
    // basados en la ubicaci n del usuario
  } catch (error) {
    console.error('Error al obtener ubicaci n:', error);
  }
}
```

7. Gestión de Datos

7.1. Datos Locales

La aplicación utiliza IndexedDB para almacenamiento local:

Listing 11: Implementación de almacenamiento local

```
import { openDB } from 'idb';

// Abrir base de datos
async function openDatabase() {
  return openDB('nutritrack', 1, {
    upgrade(db) {
      // Crear almacenes de objetos
      if (!db.objectStoreNames.contains('meals')) {
        db.createObjectStore('meals', { keyPath: 'id' });
      }
      if (!db.objectStoreNames.contains('foods')) {
        db.createObjectStore('foods', { keyPath: 'id' });
      }
      if (!db.objectStoreNames.contains('sync-queue')) {
        db.createObjectStore('sync-queue', { keyPath: 'id' });
      }
    }
  });
}

// Guardar comida en IndexedDB
async function saveMealLocally(meal) {
  const db = await openDatabase();
  return db.put('meals', meal);
}

// Obtener comidas por fecha
async function getMealsByDate(date) {
  const db = await openDatabase();
  const allMeals = await db.getAll('meals');
  return allMeals.filter(meal => meal.date === date);
}
```

7.2. Datos Remotos

La aplicación se comunica con un servidor backend mediante Axios:

Listing 12: Comunicación con API remota

```
import axios from 'axios';

const API_URL = 'http://localhost:3001';

// Obtener comidas del servidor
async function fetchMealsFromServer(date) {
```

```
  try {
    const response = await axios.get(`${API_URL}/meals?date=${
       date}`);
    return response.data;
  } catch (error) {
    console.error('Error al obtener comidas del servidor:', error
       );
    throw error;
  }
}

// Guardar comida en el servidor
async function saveMealToServer(meal) {
  try {
    const response = await axios.post(`${API_URL}/meals`, meal);
    return response.data;
  } catch (error) {
    console.error('Error al guardar comida en el servidor:',
       error);
    throw error;
  }
}

// Buscar alimentos en la base de datos
async function searchFoodsFromServer(query) {
  try {
    const response = await axios.get(`${API_URL}/foodDatabase?q=$
       {query}`);
    return response.data;
  } catch (error) {
    console.error('Error al buscar alimentos:', error);
    throw error;
  }
}
```

7.3. Sincronización de Datos

La aplicación implementa un sistema de sincronización para mantener la coherencia entre datos locales y remotos:

Listing 13: Sistema de sincronización

```
// Sincronizar datos pendientes
async function syncPendingData() {
  const db = await openDatabase();
  const tx = db.transaction('sync-queue', 'readwrite');
  const store = tx.objectStore('sync-queue');

  const pendingItems = await store.getAll();

  for (const item of pendingItems) {
```

```
    try {
      if (item.operation === 'create') {
        const response = await axios.post(`${API_URL}/meals`,
           item.data);
        // Actualizar ID local con ID del servidor
        const mealTx = db.transaction('meals', 'readwrite');
        const mealStore = mealTx.objectStore('meals');
        const localMeal = await mealStore.get(item.id);
        if (localMeal) {
          await mealStore.delete(item.id);
          localMeal.id = response.data.id;
          await mealStore.put(localMeal);
        }
      } else if (item.operation === 'update') {
        await axios.put(`${API_URL}/meals/${item.data.id}`, item.
           data);
      } else if (item.operation === 'delete') {
        await axios.delete(`${API_URL}/meals/${item.id}`);
        const mealTx = db.transaction('meals', 'readwrite');
        await mealTx.objectStore('meals').delete(item.id);
      }

      // Eliminar de la cola de sincronizaci n
      await store.delete(item.id);
    } catch (error) {
      console.error(`Error al sincronizar item ${item.id}:`,
         error);
      // Mantener en la cola para intentar  m  s  tarde
    }
  }

  await tx.done;
}
```

8. Referencias

1. Vue.js Documentation. https://vuejs.org/guide/introduction.html
2. Progressive Web Apps. https://web.dev/progressive-web-apps/
3. MDN Web Docs: IndexedDB API. https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API
4. QuaggaJS: An advanced barcode-scanner written in JavaScript. https://github.com/serratus/quaggaJS
5. Open Food Facts API Documentation. https://world.openfoodfacts.org/data/data-fields.txt
6. Service Workers: an Introduction. https://developers.google.com/web/fundamentals/primers/service-workers
7. Web App Manifest. https://developer.mozilla.org/en-US/docs/Web/Manifest
8. Workbox: JavaScript Libraries for Progressive Web Apps. https://developers.google.com/web/tools/workbox