BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

OTOÑO 2024

MATERIA: CONTROL DE CALIDAD DE SOFTWARE

PROFESOR: LUIS YAEL MENDEZ SANCHEZ

ALUMNOS: VICTOR HUGO MENDOZA SAINZ – 202138595

FRANCISCO SORIANO ROJAS – 202161530

PRÁCTICA 2: PROPUESTA DE ARQUITECTURA DE SOFTWARE

PROPUESTA DE ARQUITECTURA DE SOFTWARE

Pasos clave para proponer una arquitectura de software:

1. **Recolección de Requisitos**
   Funcionales:
   - La aplicación debe permitir a los profesionales de salud registrar a sus pacientes.
   - La aplicación bebe permitir a los profesionales de salud editar o eliminar los datos del paciente.
   - La aplicación deberá permitir al profesional de salud el registro del régimen alimenticio propuesto al paciente en forma de consumo diario o por tiempos.
   - El paciente podrá consultar su régimen alimenticio.
   - El sistema al final del día concentra los datos en el expediente electrónico del paciente de forma automática.
   - La interfaz web proporcionará estadísticas de los pacientes tanto general como de uno en específico
   - La interfaz web debe permitir al professional de salud hacer sugerencias a los pacientes en tiempo real.
   - La interfaz web permitirá a los pacientes ver sus estadísticas semanales y mensuales incluyendo calorías consumidas, calorías recomendadas, porciones consumidas, porciones faltantes, porciones excedentes y seguimiento del día libre
   - La aplicación debe permitir al usuario el registro de su actividad física diaria, el cual será intensidad baja(), intensidad media() e intensidad alta().
   - El sistema proporcionara un estimado de pérdida o ganancia de peso según su actividad física.
   - Los pacientes podrán reportar en su seguimiento diario los cambios que hayan realizado en cada una de las porciones consumidas diariamente (si hizo un cambio por otro alimento y no está seguro si corresponde a su porción o si no lo consumió).
   - El sistema permitirá al paciente visualizar una tabla de alimentos equivalentes.
   - El sistema deberá hacer el cálculo de lo que representa en el número de porciones y calorías de acuerdo con criterio de alimentos equivalentes.
   - El sistema permitirá al profesional de salud modificar el régimen alimenticio del paciente si este hizo algún cambio en sus porciones consumidas diariamente.
   - El sistema permitirá tanto al paciente como al profesional de salud tener una comunicación rápida por medio de mensajes.

- El sistema mostrará los mensajes recibidos al profesional de salud cuando este inicie sesión o cuando se reciban.

No funcionales:

- El sistema debe garantizar la integridad de los datos almacenados y procesados, asegurando que no se produzcan inconsistencias en la información de los pacientes y sus regímenes alimenticios.
- El sistema debe cumplir con los estándares de seguridad y privacidad para el manejo de información personal sensible, protegiendo los datos de los pacientes y profesionales de salud.
- La interfaz web debe ser completa y fácil de navegar tanto para los profesionales de salud como para los pacientes.

2. **Selección del Estilo de Arquitectura.**
   Selección del estilo de arquitectura: MICROSERVICIOS. La aplicación se divide en servicios pequeños e independientes que se comunican entre sí.

3. **Diseño de Capas**
   El diseño de capas en una aplicación es una arquitectura que organiza el software en capas separadas, cada una con una responsabilidad específica. Este enfoque facilita la gestión, el desarrollo y el mantenimiento de aplicaciones complejas, ya que permite separar las preocupaciones y modularizar el código.

   Interfaz de usuario (Frontend)

   - **HTML (Lenguaje de marcado de hipertexto):** Es el estándar para crear la estructura y el contenido de una página web.
   - **CSS (Hojas de estilo en cascada):** Se utiliza para diseñar y dar estilo a las páginas web, haciendo que sean visualmente atractivas y responsivas.
   - **JavaScript:** Es un lenguaje de programación que permite agregar interactividad a las páginas web.
   - **Frameworks/tailwind/Bootstrap/React:** Facilitan la creación de aplicaciones web dinámicas y eficientes, proporcionando herramientas y componentes reutilizables.
   - En la capa de presentación optamos por usar estas tecnologías ya que son las que más se adaptan al tipo arquitectura de microservicios ya que en cada

Ventajas:

- **Separación de preocupaciones:** La separación entre estructura (HTML), estilo (CSS) y comportamiento (JavaScript) facilita el desarrollo y mantenimiento.
- **Interactividad:** JavaScript y frameworks modernos permiten crear interfaces de usuario altamente interactivas y responsivas.
- **Desarrollo Modular:** Frameworks como React y tailwid promueven la reutilización de componentes, lo que acelera el desarrollo y facilita la escalabilidad.

Lógica de Negocio (Backend)

La capa de lógica de negocio define cómo se gestionan las reglas de negocio y cómo fluyen los datos entre las capas de la aplicación. Tecnologías comunes incluyen:

- **Node.js:** Permite ejecutar JavaScript en el servidor, ofreciendo un entorno eficiente y escalable para aplicaciones de alta concurrencia.

Ventajas:

- **Seguridad:** El backend gestiona la autenticación y autorización, protegiendo los datos y asegurando el acceso adecuado.
- **Escalabilidad:** Frameworks como Node.js está diseñado para manejar grandes volúmenes de tráfico y datos.
- **Gestión de Negocio:** Centraliza la lógica de negocio, lo que facilita el mantenimiento y la evolución de las reglas empresariales.

Persistencia de datos

La capa de persistencia de datos es responsable de almacenar y gestionar los datos de la aplicación. Tecnologías comunes incluyen:

- **MySQL:** Un sistema de gestión de bases de datos relacional ampliamente utilizado, conocido por su confiabilidad y rendimiento.

Ventajas:

- **Integridad de Datos:** Las bases de datos relacionales como MySQL aseguran la integridad y consistencia de los datos.

API/Servicios Web

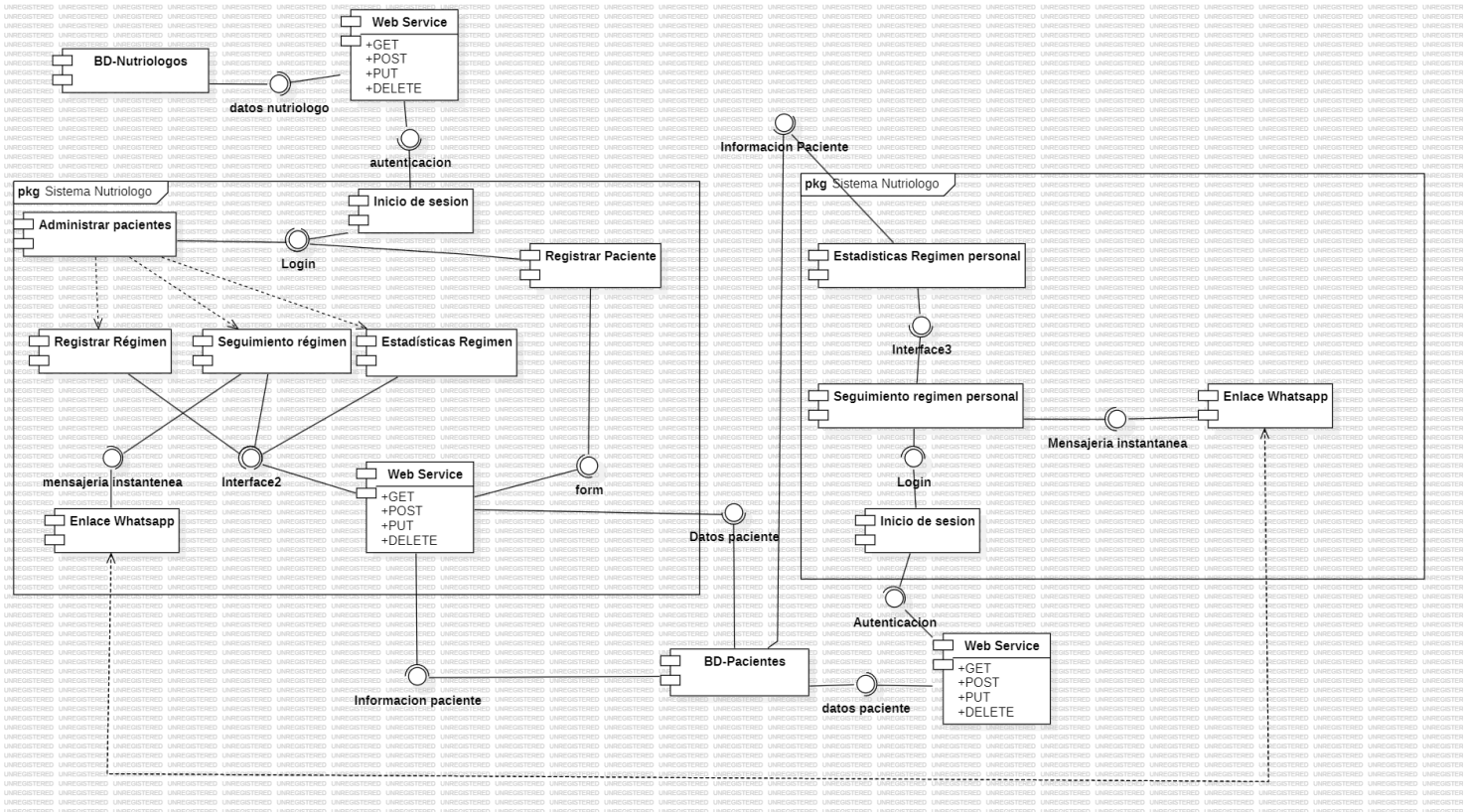
La capa de API/Servicios Web facilita la comunicación entre el frontend y el backend, permitiendo que los datos y la funcionalidad fluyan entre las capas. Tecnologías comunes incluyen:

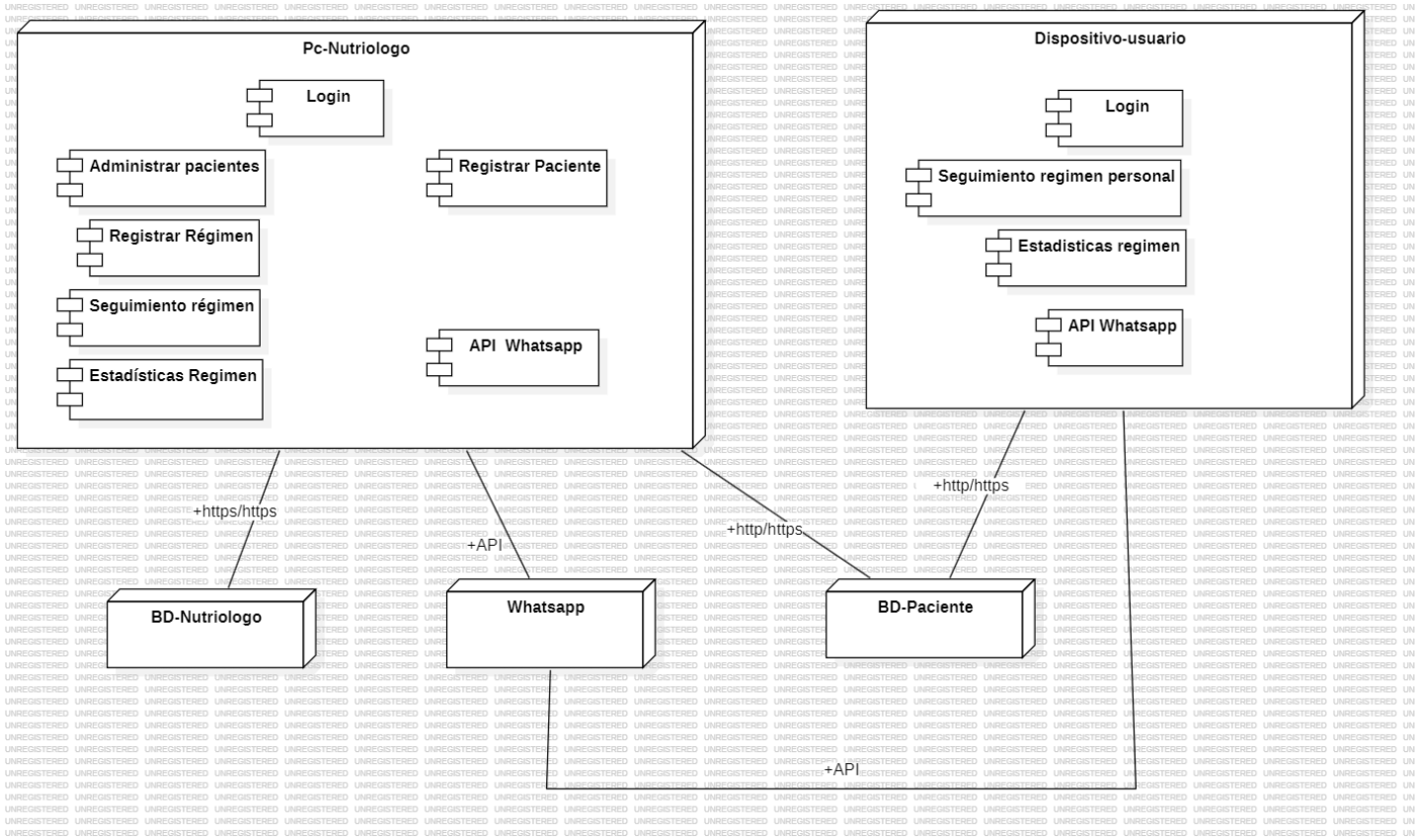
- **APIs REST:** Utilizan el protocolo HTTP para permitir la comunicación entre sistemas, siendo sencillo y ampliamente adoptado.
- **GraphQL:** Un lenguaje de consulta para APIs que permite a los clientes solicitar exactamente los datos que necesitan, reduciendo el tráfico y mejorando la eficiencia.

Ventajas:

- **Interoperabilidad:** Las API permiten que diferentes sistemas y aplicaciones se comuniquen entre sí, facilitando la integración.
- **Flexibilidad:** GraphQL ofrece una mayor flexibilidad en la consulta de datos, permitiendo optimizar las peticiones y mejorar el rendimiento.
- **Escalabilidad:** Las API pueden ser escaladas de forma independiente, permitiendo manejar crecientes volúmenes de tráfico y datos sin afectar a otras partes de la aplicación.

4. Diagrama de componentes





Definir Tecnologías

FRONTEND: Herramientas para la creación de de la interfaz visual del usuario final, utilizaremos las siguientes tecnologías:

- **HTML (Lenguaje de marcado de hipertexto):** Es el estándar para crear la estructura y el contenido de una página web.
- **CSS (Hojas de estilo en cascada):** Se utiliza para diseñar y dar estilo a las páginas web, haciendo que sean visualmente atractivas y responsivas.
- **JavaScript:** Es un lenguaje de programación que permite agregar interactividad a las páginas web.
- **Frameworks/tailwind/Bootstrap/React:** Facilitan la creación de aplicaciones web dinámicas y eficientes, proporcionando herramientas y componentes reutilizables.
- En la capa de presentación optamos por usar estas tecnologías ya que son las que más se adaptan al tipo arquitectura de microservicios ya que en cada

Las ventajas de usar estas tecnologías son que tendremos una mejor estructura de datos ya que cada tecnología se adapta a la necesidad de mantener el proyecto mucho más estructurado por módulos, tendremos interactividad gracias a JavaScript y a los Frameworks modernos que utilizaremos ya que estos nos ayudan a que las interfaces de usuario sean altamente interactivas y responsivas, gracias al desarrollo modular de estas tecnologías promueven la reutilización de componentes, lo que acelera el desarrollo y facilita la estabilidad del proyecto

BACKEND: La capa de lógica de negocio define cómo se gestionan las reglas de negocio y cómo fluyen los datos entre las capas de la aplicación. Tecnologías comunes incluyen:

- **Node.js:** Permite ejecutar JavaScript en el servidor, ofreciendo un entorno eficiente y escalable para aplicaciones de alta concurrencia.

Las ventajas de utilizar Node.js para el backend en nuestro proyecto es que esta tecnología tiene mayor eficiencia, ya que Node.js utiliza el motor v8 de Google ya que lo hace extremadamente rápido para ejecutar JavaScript en el backend.

Al usar Node.js en el backend, puedes desarrollar tanto el frontend como el backend en JavaScript, lo que simplifica la curva de aprendizaje y facilita el intercambio de código entre las capas. Node.js está diseñado para manejar aplicaciones escalables y distribuidas. Gracias a su arquitectura basada en eventos, puede manejar fácilmente grandes cantidades de conexiones concurrentes.

Es adecuado para construir arquitecturas de microservicios, lo que facilita la escalabilidad horizontal y la separación de responsabilidades dentro de un sistema, al igual que tiene una de las bibliotecas de código abiertas más grandes y se activa a través de NPM, lo que permite a los desarrolladores encontrar y reutilizar soluciones para una variedad de problemas comunes. Esto acelera el desarrollo y reduce el tiempo dedicado a reinventar soluciones.

DESARROLLO MOVIL

React Native.

1.Reutilización de Código entre Web y Móvil

React Native y **React.js** comparten la misma arquitectura y muchos componentes, lo que facilita reutilizar gran parte del código entre las aplicaciones web y móviles. Esto reduce significativamente el esfuerzo de desarrollo y mantenimiento, ya que no tienes que escribir dos aplicaciones completamente diferentes.

2. Desarrollo Rápido y Eficiente

- React utiliza el concepto de **componentes reutilizables**, lo que permite desarrollar interfaces de usuario de forma más rápida y organizada. Los componentes pueden usarse tanto en la aplicación móvil como en la web, lo que acelera el proceso de desarrollo.
- Con **React Native**, puedes usar JavaScript y los principios de React para crear aplicaciones móviles nativas que se ejecutan tanto en iOS como en Android. Esto elimina la necesidad de desarrollar aplicaciones nativas separadas para cada plataforma.

3. Gran Ecosistema y Comunidad

- React tiene un ecosistema y una comunidad muy grandes. Esto significa que tienes acceso a muchísimas bibliotecas, herramientas, y soluciones de terceros que te ayudarán a implementar funcionalidades de manera más sencilla, tanto en la web como en móvil.
- La comunidad también ofrece una gran cantidad de recursos, foros, y soluciones a problemas comunes, lo que facilita el soporte y el aprendizaje.

4. Experiencia de Usuario Nativa con React Native

- Aunque React Native utiliza JavaScript para crear interfaces de usuario, compila código nativo en iOS y Android, lo que resulta en aplicaciones con rendimiento cercano al nativo. Esto significa que los usuarios experimentan una aplicación fluida y rápida, algo que es crucial en el desarrollo móvil.
- **Acceso a APIs nativas**: React Native permite integrar fácilmente APIs nativas de los dispositivos, como el acceso a la cámara, ubicación, notificaciones, etc., lo que te da una experiencia móvil completa.

5. Mantenimiento Unificado

6. Rendimiento Aceptable para Aplicaciones Híbridas

8. Actualizaciones en Tiempo Real con Expo o Code Push

- React Native, cuando se utiliza con herramientas como **Expo** o **Microsoft Code Push**, permite realizar actualizaciones en tiempo real sin tener que pasar por las tiendas de aplicaciones (App Store o Google Play), lo que acelera la implementación de correcciones o mejoras.

5. **Consideraciones de seguridad**

1. Autenticación: Implementación JWT. JWT (JSON Web Tokens): Para mantener la autenticación activa, se pueden generar tokens JWT al inicio de sesión. Estos tokens, que contienen la información del usuario encriptada y firmada, se incluyen en cada solicitud posterior. Como ventaja, los JWT permiten un intercambio seguro de información entre el cliente y el servidor sin necesidad de mantener el estado del usuario en el servidor, reduciendo la sobrecarga y mejorando la escalabilidad.

2. Autorización: Control de Acceso Basado en Roles (RBAC). Por qué implementar RBAC: Los profesionales de salud y los pacientes tienen diferentes niveles de acceso. Usar un sistema basado en roles te permitirá restringir funciones según el tipo de usuario. Por ejemplo, los profesionales podrán editar planes de alimentación y ver estadísticas detalladas, mientras que los pacientes solo podrán ver su plan y reportar sus avances.

Recomendación: Implementar verificación de roles tanto en el Front-End (para limitar la visibilidad de ciertas funciones) como en el Back-End (para restringir operaciones no autorizadas).

3. Cifrado de Datos: HTTPS. Hay que asegurar que todas las comunicaciones entre el cliente y el servidor se realicen a través de HTTPS es fundamental. HTTPS asegura que los datos transmitidos estén cifrados, protegiendo la información sensible (como la dieta, los datos de salud, o los mensajes) de ser interceptada.

4. **Protección contra ataques: Inyección SQL**

Inyección SQL: Para evitar ataques de inyección SQL, utiliza ORMs (Object-Relational Mappers) como Sequelize o Prisma, que generan consultas seguras. También puedes usar consultas preparadas y parametrizadas que sanitizan los inputs de los usuarios.

5. **Proteger las APIs: Rate Limiting y CSRF**

Rate Limiting: Evita ataques de denegación de servicio (DoS) mediante la implementación de un límite en el número de solicitudes que un usuario puede realizar en un periodo de tiempo determinado. Librerías como **express-rate-limit** pueden ayudarte a controlar esto en el Back-End.

Protección CSRF: Utiliza tokens CSRF para proteger las solicitudes que modifican datos sensibles. Los formularios o peticiones que hagan cambios deben incluir un token que el servidor valide, asegurando que la solicitud proviene de un origen autorizado.