Índice

Introducción
Aplicación Fitness Personalizada (SaaS)

El presente proyecto consiste en el diseño y desarrollo de una aplicación fitness personalizada bajo el modelo SaaS (Software as a Service). Se busca ofrecer a los usuarios una plataforma digital que les permita acceder a rutinas de entrenamiento adaptadas a sus objetivos personales, realizar un seguimiento de su progreso en tiempo real y fomentar la adherencia a estilos de vida saludables mediante la personalización y la gamificación.

El proyecto se plantea con una arquitectura tecnológica moderna, basada en servicios escalables y en un futuro desplegados en la nube y con un enfoque de calidad del software sustentado en métricas claras, aseguramiento continuo y mejora iterativa.

Entre los objetivos principales se destacan:

- Ofrecer una experiencia personalizada al usuario mediante algoritmos de recomendación.
- Asegurar alta disponibilidad (99.9%), escalabilidad y seguridad en la gestión de datos personales.
- Implementar un flujo de desarrollo automatizado con pruebas unitarias, de integración y seguridad.
- Brindar una interfaz web amigable con el usuario, intuitiva y atractiva, con métricas de usabilidad altas.

En cuanto al plan de calidad de software, se definen métricas como:

- Cobertura de pruebas unitarias ≥ 85%.
- Tiempo de respuesta promedio < 200ms.
- Disponibilidad mensual ≥ 99.9%.
- Vulnerabilidades críticas = 1.
- Nivel de satisfacción de usuario ≥ 85%.

El proyecto también contempla el uso del modelo MoProSoft como marco de procesos, con el fin de estandarizar la gestión de proyectos, el aseguramiento de calidad y la mejora continua.

Finalmente, se incorpora un modelo de negocio Canvas, en el cual se identifican la propuesta de valor (rutinas personalizadas, seguimiento, comunidad y gamificación), los segmentos de clientes (usuarios individuales, entrenadores y gimnasios), las fuentes de ingresos (suscripciones y alianzas) y la estructura de costos (infraestructura cloud, desarrollo y marketing).

Con esta propuesta, el proyecto se plantea como una solución escolarmente viable en su prototipo, pero con visión de escalabilidad profesional, integrando buenas prácticas de ingeniería de software, aseguramiento de calidad y gestión estratégica.

Canvas

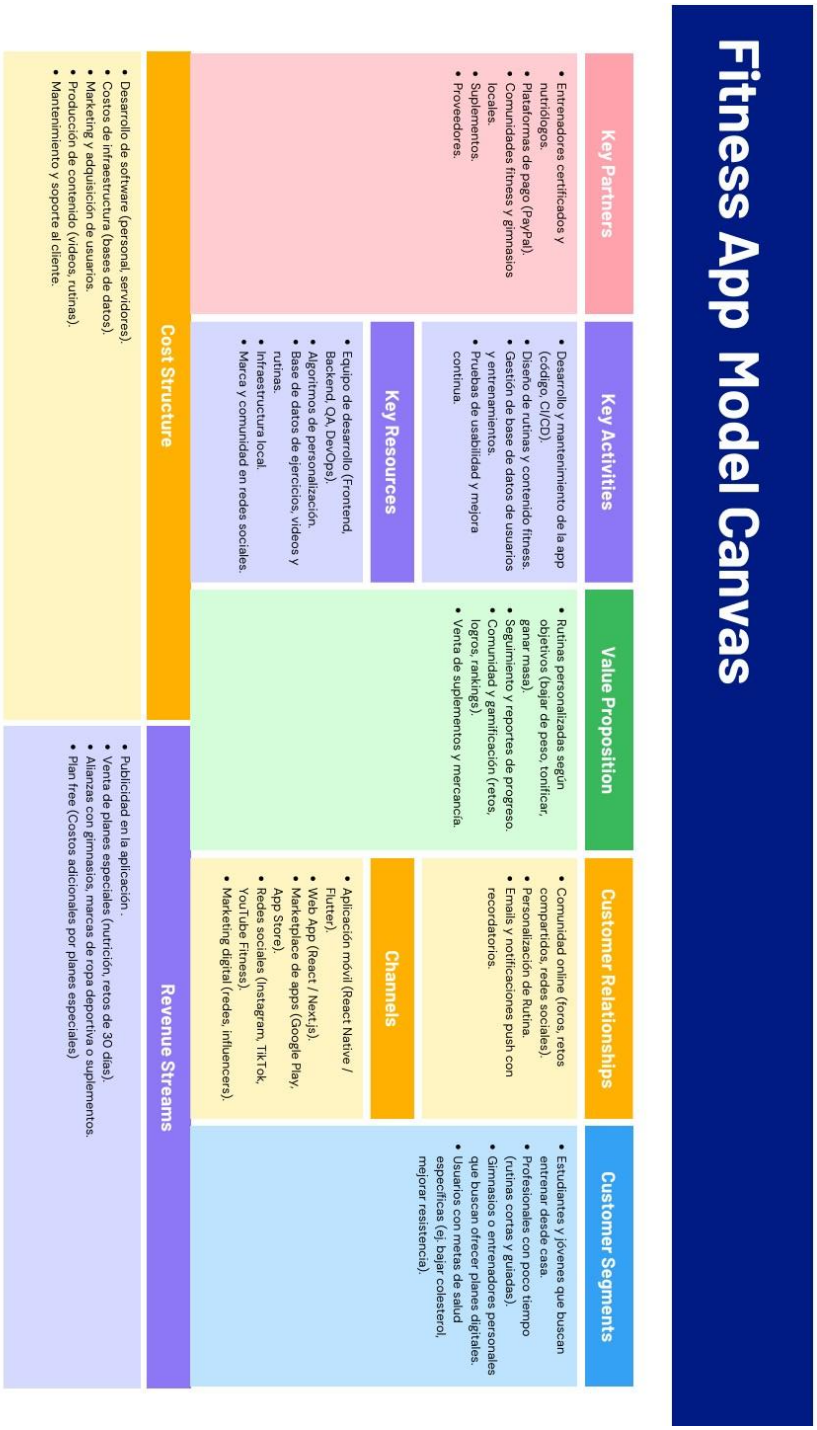
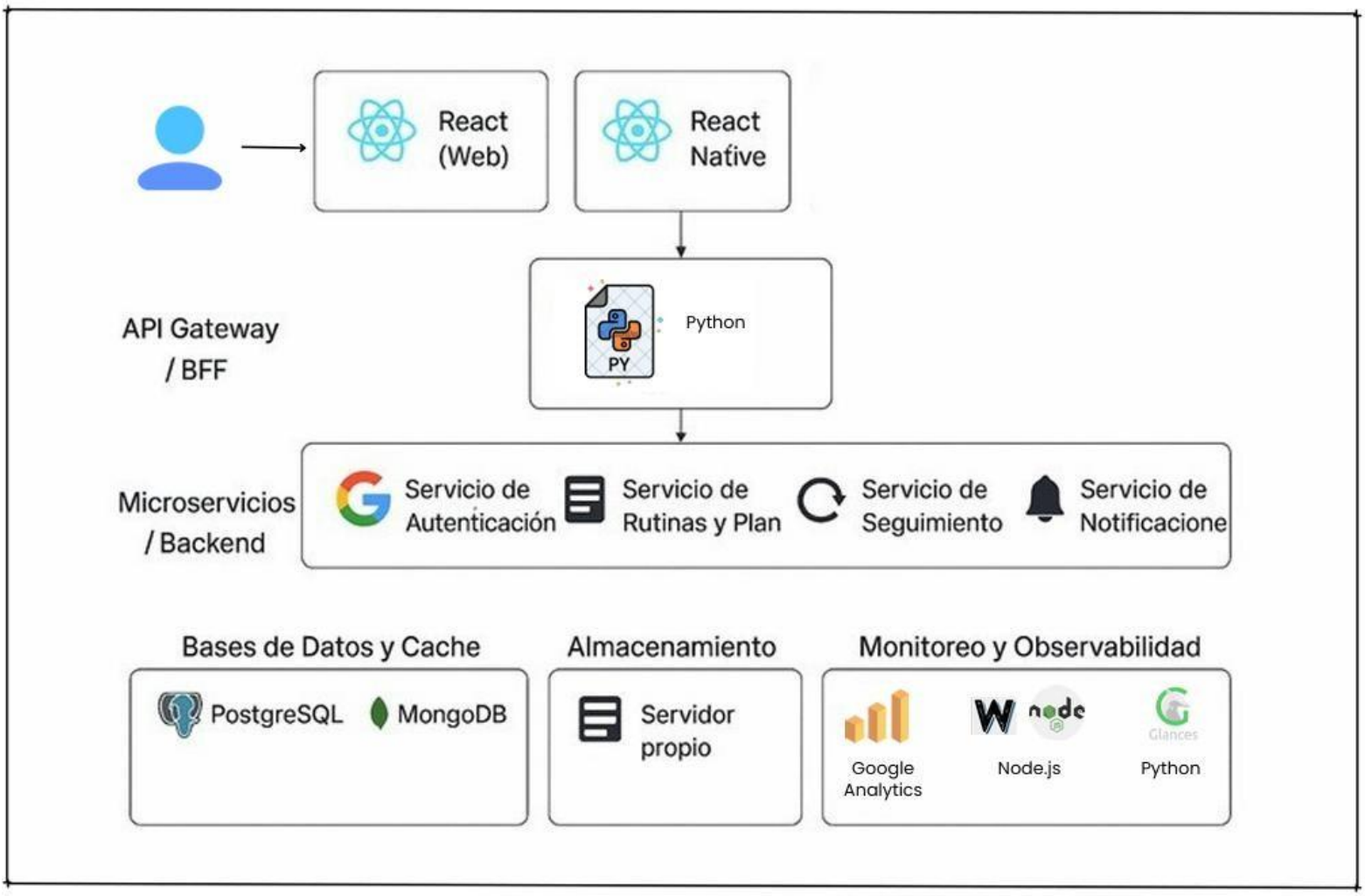

Fitness App Model Canvas
Key Partners
• Entrenadores certificados y nutriólogos.
• Plataformas de pago (PayPal).
• Comunidades fitness y gimnasios locales.
• Suplementos.
• Proveedores.
Key Activities
• Desarrollo y mantenimiento de la app (código, CI/CD).
• Diseño de rutinas y contenido fitness.
• Gestión de base de datos de usuarios y entrenamientos.
• Pruebas de usabilidad y mejora continua.
Key Resources
• Equipo de desarrollo (Frontend, Backend, QA, DevOps).
• Algoritmos de personalización.
• Base de datos de ejercicios, videos y rutinas.
• Infraestructura local.
• Marca y comunidad en redes sociales.
Value Proposition
• Rutinas personalizadas según objetivos (bajar de peso, tonificar, ganar masa).
• Seguimiento y reportes de progreso.
• Comunidad y gamificación (retos, logros, ranking's).
• Venta de suplementos y mercancía.
Customer Relationships
• Comunidad online (foros, retos compartidos, redes sociales).
• Personalización de Rutina.
• Emails y notificaciones push con recordatorios.
Channels
• Aplicación móvil (React Native / Flutter).
• Web App (React / Next.js).
• Marketplace de apps (Google Play, App Store).
• Redes sociales (Instagram, TikTok, YouTube Fitness).
• Marketing digital (redes, influencers).
Customer Segments
• Estudiantes y jóvenes que buscan entrenar desde casa.
• Profesionales con poco tiempo (rutinas cortas y guiadas).
• Gimnasios o entrenadores personales que buscan ofrecer planes digitales.
• Usuarios con metas de salud específicas (ej. bajar colesterol, mejorar resistencia).
Cost Structure
• Desarrollo de software (personal, servidores).
• Costos de infraestructura (bases de datos).
• Marketing y adquisición de usuarios.
• Producción de contenido (videos, rutinas).
• Mantenimiento y soporte al cliente.
Revenue Streams
• Publicidad en la aplicación .
• Venta de planes especiales (nutrición, retos de 30 días).
• Alianzas con gimnasios, marcas de ropa deportiva o suplementos.
• Plan free (Costos adicionales por planes especiales)

Diagrama de Arquitectura



Explicación de Diagrama de Arquitectura

Visión general del sistema

La arquitectura está basada en un modelo SaaS para ofrecer una aplicación fitness accesible desde web (React) y móvil (React Native). El flujo inicia en el frontend, pasa por un API Gateway (Backend For Frontend - BFF) desarrollado en Python, y se conecta con varios microservicios. Cada microservicio está especializado en un dominio: autenticación, rutinas, seguimiento y notificaciones. Los datos se almacenan en bases de datos SQL y NoSQL, y el sistema se monitorea con herramientas de observabilidad modernas.

Frontend – React (Web) y React Native (Móvil)

Funciones principales: - Registro/login de usuarios con OAuth (Google) y credenciales propias. Gestión de perfil y preferencias del usuario. - Visualización y ejecución de rutinas personalizadas. Seguimiento de métricas (peso, repeticiones, tiempo, calorías). - Integración con notificaciones push para recordatorios y logros. - Consumo de APIs versionadas expuestas por el BFF.

API Gateway / BFF (Python)

El Backend For Frontend (BFF) actúa como capa de orquestación entre frontend y microservicios. Adaptación de payloads (respuestas optimizadas para móvil o web). - Autenticación: validación de JWT y manejo de refresh tokens. - Seguridad: protección con rate limiting, CORS, logging de auditoría.

- Orquestación: agrega respuestas de múltiples microservicios en una sola respuesta. Recomendación: usar FastAPI por su rendimiento, validación automática y soporte de OpenAPI.

Microservicios / Backend

Cada microservicio está diseñado para ser independiente, escalable y desplegable en contenedores: Servicio de Autenticación: gestiona usuarios, roles, JWT, OAuth (Google/Facebook), almacenamiento seguro de contraseñas. - Servicio de Rutinas y Plan: CRUD de rutinas, asignación de planes de entrenamiento, recomendaciones basadas en nivel del usuario. - Servicio de Seguimiento: registra métricas de entrenamiento, soporta series temporales, integración con dispositivos externos (ej. smartwatches). - Servicio de Notificaciones: envío de notificaciones push (FCM/APNs), email (SendGrid/SES) y SMS (Twilio), soporta colas de mensajes para procesamiento asíncrono.

Bases de Datos y Cache

- PostgreSQL: usado para datos críticos como usuarios, pagos, suscripciones y roles. Aporta integridad y consistencia. - MongoDB: almacena datos flexibles y semiestructurados como rutinas personalizadas, historiales de entrenamiento y métricas dinámicas. - Redis: usado como cache de

alto rendimiento para sesiones, rate limiting y almacenamiento temporal de tokens. - Timeseries DB (opcional): InfluxDB o TimescaleDB para manejar métricas fisiológicas continuas.
Almacenamiento de archivos
- Archivos multimedia (videos de ejercicios, imágenes) se almacenan en un servidor propio o servicio tipo S3. - Uso recomendado: almacenamiento distribuido (ej. MinIO o AWS S3) con CDN para mejorar la entrega global. - Se recomienda habilitar versionado, encriptación en reposo y políticas de backup automáticas.

Monitoreo y Observabilidad
El sistema incluye diferentes herramientas para mantener visibilidad y garantizar SLA/SLOs: - Google Analytics: métricas de uso y embudos de conversión. - Glances/Prometheus: monitoreo de recursos del sistema. - Grafana: dashboards de métricas y alertas. - ELK/Loki: logs centralizados para auditoría y debugging. - Sentry: captura de errores en frontend y backend. - OpenTelemetry + Jaeger: trazabilidad distribuida de peticiones entre microservicios.

Seguridad
Aspectos clave: - Todo el tráfico se cifra con HTTPS/TLS. - Contraseñas almacenadas con bcrypt/argon2. - Tokens JWT con expiración corta y refresh tokens seguros. - Rate limiting y firewall de aplicaciones web (WAF). - Cifrado en reposo en bases de datos. - Backups cifrados y pruebas periódicas de restauración. - Cumplimiento PCI DSS para procesamiento de pagos.

Escalabilidad y despliegue
- Microservicios stateless para escalar horizontalmente. - Contenerización con Docker. - Orquestación con Kubernetes para alta disponibilidad. - Balanceo de carga y autoescalado. - CI/CD con GitHub Actions o GitLab CI. - Gestión de secretos con Vault o AWS Secrets Manager.

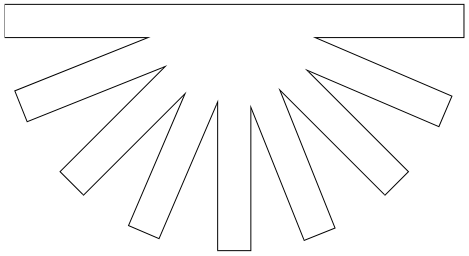
Flujo de ejemplo: registro → compra → plan asignado
1. Usuario se registra (correo o Google) → Servicio de Autenticación genera JWT. 2. Usuario compra suscripción (Stripe/PayPal) → Webhook confirma pago. 3. Microservicio de pagos actualiza estado de la suscripción en PostgreSQL. 4. Servicio de Rutinas asigna un plan inicial. 5. Servicio de Notificaciones envía confirmación por correo y push.

Consideraciones de operación
- Backups diarios de bases de datos y almacenamiento. - Monitoreo en tiempo real con alertas proactivas. - Estrategia de recuperación ante desastres (DRP). - Pruebas periódicas de carga y estrés. - Roadmap de evolución: añadir IA para recomendaciones, integrar wearables, habilitar chatbots.

Documento generado automáticamente – Arquitectura SaaS Fitness

Plan de Calidad
FITNESS

DESARROLLO DE APLICACION



P R E S E N T A D O P O R:

Raúl Barreto Arana
Ernesto Archundia Montiel
Braulio Arturo Moreno Ramírez
Jesús Gabriel Hurtado Mendoza
Jessica Esteban Miguel
Luis Gabriel Acevedo Hernández
Alan Zabdiel Gómez Martínez
Azucena Fátima Martínez Cadena
Vanessa Cruz Correa
Alexander Moreno León
Anel Gómez Vidal

Contexto

Este documento define el Plan de Calidad del Software para el proyecto. El propósito es establecer las estrategias, actividades, estándares y métricas de calidad que aseguren que el software cumpla con los requisitos funcionales, no funcionales y de negocio establecidos.

El plan se alinea con los principios de desarrollo ágil y las mejores prácticas en gestión de calidad de software, contemplando aspectos como:

- Cumplimiento de requisitos funcionales.
- Seguridad y confiabilidad del sistema en la nube.

- Mantenibilidad, escalabilidad y rendimiento.
- Experiencia de usuario (UX) consistente y accesible.

- Pruebas automatizadas y validación continua.
- Cumplimiento de normativas y estándares aplicables (ISO/IEC 25010, ISO/IEC 12207, OWASP, etc.).

Asimismo, este plan será una guía de referencia para todas las etapas del ciclo de vida del software SaaS:

- Planeación y diseño.
- Desarrollo e integración.
- Pruebas y validación.
- Despliegue y operación.
- Mantenimiento y mejora continua.

Alcance y Objetivos de Calidad

El presente Plan de Calidad cubre todas las fases del ciclo de vida del sistema SaaS, incluyendo el diseño, desarrollo, pruebas, despliegue, operación y mantenimiento. Este alcance asegura que la calidad se gestione de forma integral y consistente a lo largo de todo el proceso de desarrollo de software.

Los principales objetivos de calidad definidos para este sistema son:

- Confiabilidad: Garantizar que el sistema funcione correctamente y de forma continua bajo condiciones normales de uso.
- Disponibilidad: Mantener un acuerdo de nivel de servicio (SLA) de al menos 99.9%, asegurando mínima interrupción en la operación.
- Seguridad: Proteger la información y los datos de los usuarios mediante controles robustos de autenticación, autorización y encriptación.
- Rendimiento: Asegurar tiempos de respuesta óptimos y capacidad de procesamiento adecuada para soportar múltiples usuarios concurrentes.
- Usabilidad: Ofrecer una interfaz intuitiva, accesible y fácil de utilizar por parte de diferentes perfiles de usuario.
- Escalabilidad: Permitir la expansión eficiente de la infraestructura y el sistema para atender el crecimiento de la demanda.

Estándares y Referencias

En el desarrollo de la aplicación SaaS de fitness se adoptan estándares reconocidos que garantizan la calidad del software, la seguridad de la información y la eficiencia en el ciclo de vida del proyecto. Estos marcos de referencia permiten estructurar las actividades de cada rol del equipo, mejorar la coordinación y asegurar que el producto final cumpla con las expectativas de los usuarios.

Entre los estándares más relevantes se encuentra MoProSoft (NMX-I-059-NYCE2011), que funge como modelo de procesos para la industria del software y establece fases ordenadas de planeación, desarrollo, pruebas, liberación y mantenimiento. Su alineación con marcos internacionales como ISO/IEC 12207 e ISO/IEC 29110 permite gestionar los proyectos de manera estructurada y reconocida globalmente.

Por otro lado, ISO/IEC 29110 se presenta como un estándar especialmente adecuado para equipos pequeños y medianos, como el de 11 desarrolladores del presente proyecto, definiendo roles, actividades y entregables que favorecen la trazabilidad y el control del ciclo de vida.

Asimismo, la NMX-I-14598 resulta fundamental para la evaluación de productos de software, dado que establece criterios de calidad que permiten comprobar el cumplimiento tanto de requisitos funcionales como no funcionales.
Complementariamente, la NMX-I-15504 (SPICE) se orienta hacia la evaluación y mejora continua de los procesos, contribuyendo al fortalecimiento de la madurez organizacional y a la gestión eficiente de los proyectos.

Finalmente, la ISO/IEC 25000 (SQuaRE) aporta un marco de métricas y criterios de calidad de software, considerando atributos esenciales como fiabilidad, usabilidad, seguridad, mantenibilidad y eficiencia.

En conjunto con estos estándares internacionales, se aplican guías internas como los principios de Clean Code, que buscan la simplicidad y legibilidad del código, y las revisiones de Pull Requests (PR), que aseguran la calidad técnica y fomentan la colaboración entre pares.

Los beneficios de esta adopción se manifiestan en la calidad garantizada mediante criterios definidos, la organización y control gracias a la ISO/IEC 29110, la mejora continua, promovida por SPICE, y la mantenibilidad y seguridad reforzadas con prácticas de desarrollo limpio y alineadas a OWASP.

Versión Técnica

En el desarrollo de la aplicación SaaS de fitness se adoptan estándares que garantizan calidad, seguridad y eficiencia en el ciclo de vida del software.

1.1 Estándares Adoptados

1. MoProSoft (NMX-I-059-NYCE-2011)

*Modelo de procesos para la industria del software.

* Incluye planeación, desarrollo, pruebas, liberación y mantenimiento.

* Alineado con ISO/IEC 12207 e ISO/IEC 29110.

2. ISO/IEC 29110

* Orientado a equipos pequeños y medianos.

* Define roles, actividades y entregables con trazabilidad.

3. NMX-I-14598

* Norma para la evaluación de productos de software.

* Establece criterios de calidad en pruebas (funcionales y no funcionales).

4. NMX-I-15504 (SPICE)

* Marco para la evaluación y mejora continua de procesos.

* Refuerza la madurez del equipo y la gestión del proyecto.

5. ISO/IEC 25000 (SQuaRE)

* Define métricas y criterios de calidad.

* Considera atributos: fiabilidad, usabilidad, seguridad, mantenibilidad y eficiencia.

1.2. Guías Internas

Clean Code: garantiza legibilidad, simplicidad y mantenibilidad del código.

1. Legibilidad, ante todo

- Nombres claros para componentes
  - [WorkoutTracker], [UserProfile], [NutritionDashboard] en lugar de abreviaturas crípticas como UT o ND.

- Props descriptivas
  Evita nombres genéricos:

- <WorkoutCard isLoading={loading} onComplete={handleComplete} /> en lugar de:

<WorkoutCard data1={loading} func={handleComplete} />

2. Simplicidad (KISS & DRY)

- Evita duplicar lógica: extrae hooks reutilizables o utilidades.
- Mantén JSX limpio, sin lógica compleja dentro del render:

3. Consistencia
- Usa herramientas de estilo uniformes.
- Sigue convenciones de carpetas y nombres:
- /components
- Mantén consistencia en props y nombres de funciones

4. Funciones pequeñas y específicas (SRP)
- Cada componente debe tener una sola responsabilidad.
- Divisiones en componentes

5. Pruebas automatizadas
- Escribe unit tests con Jest y React Testing Library.
- Asegura que los componentes funcionen correctamente antes de refactorizar.

6. Refactorización continua
- Extrae hooks o scripts cuando se repite la lógica
- Mejora la estructura sin cambiar el comportamiento visible para el usuario.

Revisiones de Pull Requests (PR): fomentan revisión entre pares, calidad técnica y aprendizaje colaborativo.

1.3. Beneficios de la Adopción

* Calidad garantizada (MoProSoft, NMX-I-14598, ISO/IEC 25000).

* Organización y control (ISO/IEC 29110).

* Mejora continua (NMX-I-15504 / SPICE).

* Seguridad y mantenibilidad reforzadas (Clean Code, PR, OWASP).

4. Estrategia de Aseguramiento de Calidad

1. TDD y BDD
- TDD: antes de implementar componentes clave (ej. formulario de registro, login, selección de rutinas), se escriben pruebas unitarias con Jest + React Testing Library.

- BDD: se definen escenarios de usuario con Cucumber o Cypress, por ejemplo:

o “Dado que un usuario está registrado, cuando inicia sesión, entonces debe ver su dashboard con su progreso semanal”.

2. Integración continua (CI/CD)
• Configurar GitHub Actions para ejecutar automáticamente:

o Pruebas unitarias. o Pruebas de integración. o

Análisis de seguridad (SonarQube o npm audit).

• Deploy automático en un entorno de pruebas (Vercel).

3. Revisiones de código

- Todo nuevo módulo (“Plan de Nutrición” o “Calendario de Rutinas”) pasa por Pull Request + Code Review.
- Reglas: buenas prácticas de React (hooks, props claras, evitar duplicación de código).

4. Pruebas automatizadas •
Unitarias (Jest):

o Validar que el botón de “Iniciar sesión” llama correctamente a la API. o Comprobar que

el gráfico de progreso muestra los datos esperados.

- Integración (React Testing Library + Cypress): o Probar flujo de registro → login → dashboard.
- Regresión: antes de cada release, correr todo el set de pruebas para asegurar que lo nuevo no rompe nada.
- Seguridad (OWASP ZAP / npm audit): o Validar que contraseñas estén cifradas. o Evitar

  inyección de datos en formularios.
- Performance (Lighthouse, Jest performance tests):

o Tiempo de carga del dashboard < 2s.

5. Pruebas manuales

- Exploratorias: QA prueba nuevas funciones buscando fallos (probar rutinas con datos atípicos).
- UAT (User Acceptance Testing): usuarios reales (entrenadores y clientes) validan que la aplicación sea intuitiva y útil.

6. Gestión de bugs

• Todos los errores se documentan en Jira o Trello con: o

Descripción. o Severidad. o Pasos para reproducirlo.

o Estado (pendiente, en progreso, resuelto).

Roles y Responsabilidades

1. QA Lead

Responsabilidades:

- Responsable de la elaboración, gestión y seguimiento del plan de pruebas.
- Coordinar al equipo de QA para garantizar la calidad del producto.
- Priorizar casos de prueba y definir criterios de aceptación.
- Reportar resultados de pruebas y métricas de calidad al Product Owner.

2. Desarrolladores

Responsabilidades:

- Implementar las funcionalidades de la aplicación fitness conforme a los requisitos.
- Diseñar y ejecutar pruebas unitarias de los módulos desarrollados.
- Identificar, documentar y corregir defectos detectados en el código.
- Colaborar con QA para validar la integración de los componentes.

3. Equipo de QA

Responsabilidades:

- Diseñar los casos de prueba funcionales, de integración, rendimiento y usabilidad.
- Ejecutar pruebas manuales y automatizadas.
- Reportar incidencias detectadas durante las pruebas.
- Validar que las correcciones de los desarrolladores cumplan con los criterios de aceptación.

4. DevOps

Responsabilidades:

- Configurar y mantener el entorno de integración y despliegue continuo.

- Gestionar la infraestructura necesaria para el proyecto.
- Monitorear el rendimiento y la seguridad de la aplicación.
- Coordinar los despliegues en los diferentes entornos (pruebas, preproducción y producción).

5. Product Owner

Responsabilidades:

- Ser el enlace directo entre el cliente y el equipo de desarrollo.
- Validar que los entregables cumplan con las necesidades y expectativas de los usuarios.
- Priorizar el backlog del producto y definir los objetivos de cada sprint.
- Asegurar que el producto final aporte valor al cliente.

Métricas y criterios de aceptación

1. Métricas de pruebas y calidad técnica •

Cobertura de pruebas unitarias:

- Criterio de aceptación: Cobertura mínima del 75% en módulos críticos (registro, planes de entrenamiento, historial de progreso).

- Defectos abiertos antes de release: o Criterio de aceptación: < 10% de defectos críticos sin resolver.
- Tiempo medio de corrección de bugs:
  - Criterio de aceptación: Defectos críticos corregidos en máximo 48 horas.

2. Métricas de rendimiento

- Tiempo medio de respuesta de la API:
  - Criterio de aceptación: < 200 ms para consultas básicas (perfil de usuario, rutinas, historial).
- Tiempo de carga de la aplicación (front-end):
  - Criterio de aceptación: < 3 segundos en dispositivos móviles estándar.
- Escalabilidad bajo carga:
  - Criterio de aceptación: soportar al menos 10,000 usuarios concurrentes con degradación <10% en rendimiento.

3. Métricas de disponibilidad y confiabilidad •

Disponibilidad del sistema (SLA):

- Criterio de aceptación: 99.9% mensual.

- Tasa de fallos de sesión o desconexión inesperada:
  - Criterio de aceptación: < 1% en un mes.

4. Métricas de seguridad

- Gestión de datos sensibles (usuarios y salud):
  - Criterio de aceptación: 0 vulnerabilidades críticas detectadas por escáneres (OWASP ZAP, SonarQube).
- Cumplimiento normativo:
  - Criterio de aceptación: manejo de datos personales acorde con GDPR/ISO 27001 (si aplica).
- Pruebas de autenticación/autorización:
  - Criterio de aceptación: 100% de intentos de acceso no autorizado deben ser bloqueados.

5. Métricas de usabilidad

- Satisfacción del usuario (encuestas NPS o CSAT):
  - Criterio de aceptación: puntaje ≥ 80% en usabilidad.
- Tiempo medio de onboarding (registro y primera rutina asignada):
  - Criterio de aceptación: < 2 minutos.
- Tasa de abandono en primeros 7 días:
  - Criterio de aceptación: < 15%.

Herramientas y entornos

1.- Entorno de Desarrollo

- Visual Studio Code: Un editor de código popular y versátil que ofrece características como autocompletado, depuración y gestión de proyectos.
- Node.js: Un entorno de ejecución de JavaScript que permite ejecutar código JavaScript en el servidor.

2.- Herramientas de Desarrollo

- Create React App: Una herramienta que permite crear proyectos de React con una configuración básica y lista para producción.
- React Router: Una biblioteca que permite manejar la navegación y el enrutamiento en aplicaciones de React.
- Redux: Una biblioteca que permite gestionar el estado de la aplicación de manera predecible y escalable.

3.- Herramientas de Pruebas

- Jest: Un framework de pruebas unitarias y de integración que viene incluido con Create React App.
- Cypress: Una herramienta de pruebas de extremo a extremo que permite probar la aplicación de manera exhaustiva.

4.- Herramientas de Diseño

- Figma: Una herramienta de diseño de interfaz de usuario que permite crear prototipos y diseños de alta fidelidad.
- Adobe XD: Una herramienta de diseño de experiencia de usuario que permite crear prototipos y diseños de alta fidelidad.

5.- Herramientas de Despliegue

- GitHub Pages: Un servicio de alojamiento de sitios web estáticos que permite desplegar la aplicación de manera sencilla.
- Netlify: Un servicio de alojamiento de sitios web estáticos que ofrece características como automatización de despliegue y gestión de DNS.

6.- Herramientas de Seguridad

- OAuth: Un protocolo de autenticación que permite a los usuarios autenticarse de manera segura.
- JWT (JSON Web Tokens): Un estándar para la creación y verificación de tokens de autenticación.
- Helmet: Una biblioteca que permite configurar la seguridad de la aplicación mediante la configuración de encabezados HTTP.
- OWASP ZAP: Una herramienta de seguridad que permite identificar vulnerabilidades en la aplicación.
- SonarQube: Una herramienta de análisis de código que permite identificar problemas de seguridad y calidad en el código.

7.- Base de Datos

- Firebase: Una plataforma de desarrollo de aplicaciones móviles y web que ofrece servicios como base de datos en tiempo real y autenticación.
- MongoDB: Una base de datos NoSQL que permite almacenar y gestionar datos de manera flexible y escalable.

8.- Otras Herramientas

- Git: Un sistema de control de versiones que permite gestionar el código de la aplicación. - npm: Un gestor de paquetes que permite instalar y gestionar dependencias en la aplicación. Gestión de riesgos y mejora continua

1.- Se identifica, evalúa y dar tratamiento a posibles eventos que afecten la aplicación:

1. *Riesgos Técnicos* o Fallas en la aplicación: errores en actualizaciones, caídas del servidor. Acción: pruebas de calidad (QA), monitoreo en tiempo real, copias de seguridad. o Compatibilidad:
   problemas en distintos sistemas operativos o dispositivos. Acción: pruebas multiplataforma y versiones beta.
2. **Riesgos de Seguridad** o Robo de datos personales (usuarios, rutinas, datos de salud). Acción: cifrado de datos, autenticación segura, cumplimiento con normativas de protección de datos.
3. **Riesgos de Adopción y Usuarios** o Baja participación de usuarios por falta de motivación o experiencia poco atractiva.
   Acción: encuestas de satisfacción, mejoras en la interfaz, gamificación.
4. **Riesgos Financieros** o Falta de sostenibilidad económica (costos de mantenimiento mayores a ingresos).
   Acción: modelo freemium, publicidad no invasiva, alianzas con gimnasios o marcas.
5. **Riesgos de Reputación** o Mala calificación en tiendas por bugs o mal soporte.
   Acción: canal de soporte activo, respuesta rápida a comentarios, roadmap transparente.

2.- Mejora Continua

Basada en el ciclo PHVA (Planear – Hacer – Verificar – Actuar):

1. **Monitoreo Constante** o Analítica de uso (frecuencia, tiempo de sesión, rutinas completadas).
   - Indicadores de rendimiento de la aplicación (tiempos de carga, estabilidad).
2. **Retroalimentación de Usuarios** o Encuestas dentro de la aplicación.
   - Espacios de reseñas y foros de sugerencias.
   - Beta testers para nuevas funciones.
3. **Optimización Funcional** o Mejorar algoritmos de recomendación de rutinas. o Integración con wearables (smartwatch, bandas de ritmo cardiaco).
   - Personalización de metas y notificaciones inteligentes.
4. **Actualizaciones Periódicas** o Nuevas rutinas, retos semanales o mensuales.
   - Inclusión de programas nutricionales.
   - Corrección de errores reportados en plazos cortos.
5. **Capacitación del Equipo** o Actualización en metodologías ágiles (Scrum/Kanban).
   - Capacitación en tendencias de UX/UI y ciberseguridad.
6. **Innovación** o Incorporar realidad aumentada (para mostrar posturas de ejercicios). o Integrar inteligencia artificial para planes personalizados.

9. Anexos

9.1 Checklist de Revisión de Calidad

Basado en los estándares definidos (MoProSoft, ISO/IEC 25000, Clean Code, PR Reviews) y la estrategia de QA:

1. El código cumple con las guías de Clean Code y pasa el análisis de SonarQube.
2. Todas las funciones críticas cuentan con pruebas unitarias (Jest).
3. Se realizaron revisiones de código vía Pull Requests (PR).
4. Se validaron criterios de seguridad OWASP (uso de JWT, OAuth, Helmet).
5. Los módulos desarrollados pasaron por pruebas de integración y end-to-end (Cypress).
6. Se documentó la trazabilidad de requisitos en Jira/Trello.
7. Se verificó compatibilidad multiplataforma antes de la liberación.

9.2 Ejemplos de Casos de Prueba

En línea con la estrategia de pruebas (unitarias, integración, rendimiento, usabilidad, seguridad):

1. Caso de Prueba TC-001 – Login Correcto
2. Módulo: Autenticación
3. Precondición: Usuario registrado en Firebase/MongoDB
4. Entrada: Email válido + contraseña válida
5. Acción: Ingresar credenciales y presionar “Login”
6. Resultado esperado: Acceso al dashboard en menos de 2s
7. Caso de Prueba TC-002 – Registro de Usuario
8. Módulo: Registro
9. Precondición: No tener cuenta registrada
10. Entrada: Nombre, email válido, contraseña segura (>8 caracteres, mayúscula, número, símbolo)
11. Acción: Completar formulario y enviar

12. Resultado esperado: El sistema guarda al usuario en la BD y envía correo de verificación
13. Caso de Prueba TC-003 – Performance del Dashboard
14. Módulo: Dashboard
15. Entrada: Usuario con historial de 100 rutinas
16. Acción: Acceder al dashboard
17. Resultado esperado: Tiempo de carga < 3s en móvil estándar

9.3 Procedimiento de Rollback
Alineado con la gestión de riesgos y mejora continua (PHVA):

1. Detectar la falla en monitoreo (Prometheus/Grafana) o reporte en producción.
2. Notificar a DevOps y QA.
3. Ejecutar rollback al último build estable en Netlify/GitHub Pages.
4. Validar la recuperación de disponibilidad (SLA 99.9%).
5. Documentar la incidencia en Jira con causa raíz y acciones correctivas.
6. Reprogramar release con fixes validados.

9.4 Ejemplos de métricas

Métrica	Herramienta	Criterio de aceptación
Cobertura de pruebas unitarias	**Jets + SonarQube**	**>= 75 % en módulos críticos**
Tiempo medio de respuesta API	**Lighthouse**	**< 200 ms**

Tiempo de carga del dashboard	**Lighthouse**	**< 3s en móviles**
Disponibilidad del sistema (SLA)	**Prometheus**	**99.9 % mensual**
Defectos críticos abiertos antes del reléase	**Jira/Trello**	**<10 %**
Vulnerabilidades críticas detectadas	**OWASP ZAP**	**0**
Satisfacción del usuario (CSAT/NPS)	**Encuestas in-app**	**>= 80 %**

Objetivos del Proyecto

Objetivos con Metodologia SMART

1. Objetivo General

Diseñar y desarrollar la aplicación fitness personalizada (SaaS) enfocada en ofrecer rutinas personalizadas, seguimiento en tiempo real y gamificación. Los objetivos principales son: Ofrecer una experiencia personalizada, asegurar alta disponibilidad y seguridad, y brindar una interfaz web intuitiva con altas métricas de usabilidad.

Aplicación de la metodología SMART:

- Específico: Crear una aplicación móvil multiplataforma que permita a los usuarios registrar su actividad física diaria, planificar rutinas de entrenamiento y recibir notificaciones de recordatorio personalizadas según sus metas.

- MediblE: Se considerará cumplido el objetivo cuando el sistema permita registrar al menos cinco tipos de actividades físicas, mostrar estadísticas de progreso semanal y generar reportes personalizados del usuario.

- Alcanzable: El desarrollo será realizado por un equipo de tres a cinco desarrolladores utilizando tecnologías disponibles como React Native para el frontend, Node.js y Firebase para el backend, asegurando que los recursos humanos y tecnológicos sean suficientes.

- Relevante: El proyecto contribuye al fomento de hábitos saludables, al bienestar físico y emocional de los usuarios, y fortalece la innovación tecnológica en el ámbito del fitness digital.

- Limitado en el tiempo: El desarrollo se llevará a cabo en un periodo máximo de cuatro meses, distribuidos en fases de análisis, diseño, implementación, pruebas y despliegue final.

2. Objetivo Específico : Registrar y seguir la actividad física del usuario Permitir que los usuarios registren y sigan su actividad física diaria mediante el uso de sensores o ingreso manual de datos.

- Específico: Integrar funciones que permitan registrar pasos, distancia recorrida, tiempo de entrenamiento y calorías quemadas, con posibilidad de agregar comentarios personales.

- Medible Se considerará cumplido cuando el sistema registre al menos cinco tipos de actividades físicas (caminar, correr, nadar, ciclismo y fuerza) y genere un resumen semanal con métricas cuantitativas.

- Alcanzable: La integración se realizará mediante el uso de APIs como Google Fit o Apple HealthKit, permitiendo la compatibilidad con dispositivos móviles actuales.

- Relevante: Facilita el control del progreso físico del usuario y motiva su constancia al mostrar resultados visibles de su desempeño.

- Limitado en el tiempo: Este módulo deberá estar implementado y probado completamente al finalizar el segundo mes del desarrollo.

3. Objetivo Específico 2: Planificar y programar entrenamientos
Desarrollar un módulo de planificación que permita a los usuarios crear rutinas de entrenamiento personalizadas y establecer horarios de práctica.

- Específico: Diseñar una interfaz que permita al usuario crear rutinas con ejercicios personalizados, definir duración, intensidad y frecuencia, y visualizar un calendario de entrenamientos.

- Medible: El sistema deberá permitir al usuario crear al menos tres rutinas diferentes, con un calendario visible que muestre los entrenamientos semanales y un porcentaje de cumplimiento.

- Alcanzable:Se implementará mediante el uso de una base de datos MySQL o Firebase, con una arquitectura que permita almacenamiento, consulta y modificación de rutinas de manera eficiente.

- Relevante: Este módulo mejora la organización del usuario, fomentando la disciplina y el compromiso con sus metas de entrenamiento.

•Limitado en el tiempo: Este objetivo deberá estar completado antes de la semana 10 del proyecto.

4. Objetivo Específico 3: Enviar notificaciones a los usuarios Implementar un sistema de notificaciones push que recuerde al usuario sus entrenamientos programados y metas pendientes.

- Específico: Incorporar un sistema de notificaciones push personalizadas que recuerden al usuario la hora de sus entrenamientos, su progreso y objetivos no cumplidos.

- Medible: Se medirá el cumplimiento mediante la generación de al menos una notificación diaria y la posibilidad de personalización por parte del usuario.

- Alcanzable: Se integrará utilizando Firebase Cloud Messaging (FCM), lo cual permite un envío confiable y adaptable en Android e iOS.

- Relevante: Incrementa la adherencia del usuario al programa de entrenamiento y mejora la interacción continua con la aplicación.

- Limitado en el tiempo: El módulo deberá estar implementado y funcional al cierre de la semana 12.

5. Objetivo Específico 4: Garantizar una interfaz amigable y segura Diseñar una interfaz intuitiva, atractiva y segura que brinde una experiencia de usuario agradable y proteja los datos personales.

- Específico: Implementar una interfaz gráfica moderna e intuitiva basada en principios de usabilidad, junto con medidas de seguridad como HTTPS, cifrado de contraseñas y autenticación segura.

- Medible: Se considerará logrado cuando al menos el 90% de los usuarios en pruebas beta califiquen la interfaz como 'fácil de usar' y no se presenten vulnerabilidades críticas durante las pruebas de seguridad.

- Alcanzable: Se emplearán frameworks como React Native y librerías UI (Material Design, Shadcn/UI) que faciliten un diseño profesional y consistente.

- Relevante: Mejora la satisfacción y confianza del usuario, asegurando la retención y uso continuo de la aplicación.

- Limitado en el tiempo: Este objetivo deberá estar completado antes de la entrega final del proyecto, en la semana 16.

Cronograma
Diagrama de Gantt

TÍTULO DEL PROYECTO

GRÁFICO GANTT

Nombre de la compañía
Responsable del proyecto

Inicio del proyecto: jue, 25/09/2025
Semana para mostrar: 1

TAREA	ASIGNADO A	PROGRESO	INICIO	FIN
PLANEACION				
Definición de objetivos y req.	Nombre	50%	25-9-25	27-9-25
Selección de stack tecnológico "Diagrama de Arquitectura"		60%	27-9-25	28-9-25
Diseño de arquitectura y base de datos		50%	28-9-25	7-10-25
CONFIGURACION INICAL				
Crear repositorio y estructura del proyecto		50%	8-10-25	20-10-25
Configurar entorno virtual y dependencias (FastAPI, React)		50%	20-10-25	28-10-25
Configurar conexión a BD local (PostgreSQL/MongoDB)		40%	28-10-25	7-11-25
DESARROLLO DEL BACKEND Y FRONTEND				
Implementar módulo Auth (registro/login JWT)			8-11-25	11-11-25
Crear módulo Usuarios y Rutinas (CRUD básico)			11-11-25	24-11-25
Agregar Tracking (progreso del usuario)			24-11-25	29-11-25
Crear pantallas Login y Registro			29-11-25	2-12-25
Dashboard principal (rutinas y progreso)			2-12-25	9-12-25
Integración API con frontend			10-12-25	16-12-25
PRUEBAS Y VERIFICACION				
Pruebas de endpoints (Swagger, Postman)			6-1-26	6-1-26
Pruebas de endpoints (Swagger, Postman)			7-1-26	7-1-26
Validar flujo completo usuario ? rutina ? Progreso			8-1-26	8-1-26
Entrega final del proyecto			9-1-26	9-1-26

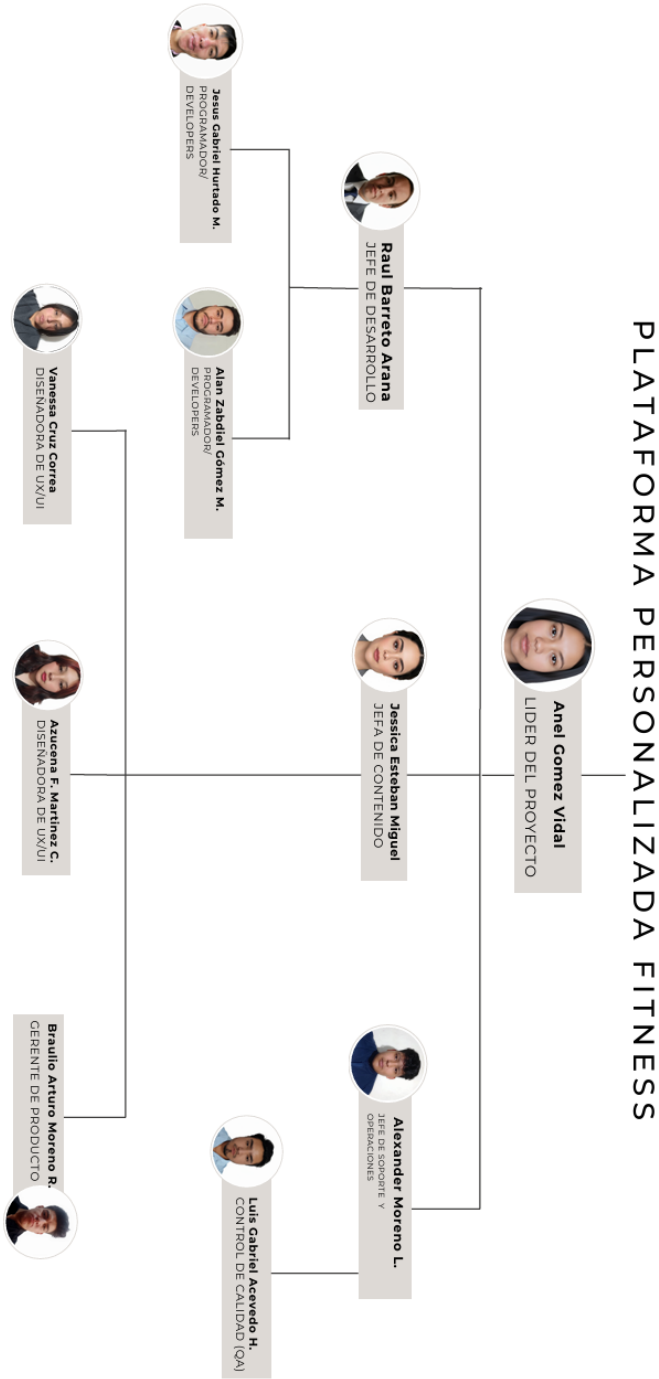
Estimación de Costos

Categoría	Descripción	Costo estimado (MXN)
1. Desarrollo de software	Implementación de backend (FastAPI), frontend (React), base de datos, integración y pruebas.	**$7,000**
2. Diseño e interfaz de usuario (UI/UX)	Diseño de pantallas, flujo de usuario y estilos en CSS.	**$2,500**
3. Infraestructura y servicios en la nube	Uso de hosting o Render gratuito (sin costo), dominio opcional y almacenamiento.	**$800**
4. Herramientas y licencias	Software de desarrollo, control de versiones y herramientas de prueba (VS Code, Postman, GitHub Free).	**$0**
5. Documentación y gestión del proyecto	Elaboración del plan de calidad, MoProSoft, Canvas, diagramas y documentación técnica.	**$1,000**
6. Mantenimiento y soporte	Corrección de errores, ajustes menores y optimización posterior.	**$1,000**

Concepto	Costo total (MXN)
Costo total estimado del proyecto	**$10,000 MXN**
Costo en recursos propios (computadora, conexión, IDE)	Incluido
Costo de despliegue escolar/local	$0 (Render gratuito o localhost)

Fase	Actividades principales	Costo aproximado
Planeación y diseño	Análisis, arquitectura, diseño de BD y UI.	$2,000
Desarrollo Backend	Creación de API, modelos y endpoints.	$3,000
Desarrollo Frontend	Interfaz React, conexión a API.	$2,000
Pruebas e integración	Testing manual y validación funcional.	$1,000
Documentación y presentación	Manual, plan de calidad, evidencia visual.	$2,000

Organigrama


PLATAFORMA PERSONALIZADA FITNESS
Anel Gomez Vidal
LIDER DEL PROYECTO
Raul Barreto Arana
JEFE DE DESARROLLO
Jesus Gabriel Hurtado M.
PROGRAMADOR/ DEVELOPERS
Alan Zabdiel Gómez M.
PROGRAMADOR/ DEVELOPERS
Jessica Esteban Miguel
JEFA DE CONTENIDO
Vanessa Cruz Correa
DISEÑADORA DE UX/UI
Azucena F. Martinez C.
DISEÑADORA DE UX/UI
Alexander Moreno L.
JEFE DE SOPORTE Y OPERACIONES
Luis Gabriel Acevedo H.
CONTROL DE CALIDAD (QA)
Braulio Arturo Moreno R.
GERENTE DE PRODUCTO