**Caso Práctico 1: Planificación del Proyecto de Software para la Reserva de Pistas de Pádel**

**1. Análisis de Requerimientos (Máxima puntuación)**

* **Definición clara y precisa de los requerimientos**:
  + **Funcionales**:
    - Permitir a los clientes **consultar** la disponibilidad de pistas.
    - Funcionalidad para **reservar** pistas con y sin luz.
    - **Procesamiento de pagos** mediante tarjeta de crédito.
    - Generar **listados** por hora y cliente de las pistas reservadas.
  + **No funcionales**:
    - Uso de **software libre**.
    - **Seguridad** en los datos y protección de información personal.
    - **Escalabilidad** para futuros módulos.
* **Herramientas y técnicas para el análisis**:
  + **Reuniones semanales** con stakeholders.
  + **Prototipos** iniciales de la interfaz para validar con los usuarios.
  + Elaboración de un **documento de especificación de requerimientos** validado.

**2. Planificación del Diseño (Máxima puntuación)**

* **Enfoque modular**: Asegurar la modularidad desde el inicio para facilitar la ampliación futura (cumpliendo con buenas prácticas).
* **Diseño de arquitectura** basado en MVC (Modelo-Vista-Controlador).
  + **Modelo**: Manejo de datos y lógica empresarial (reservas, pistas, usuarios).
  + **Vista**: Interfaz de usuario donde los clientes ven y reservan pistas.
  + **Controlador**: Gestión de interacciones entre la vista y el modelo.
* **Lenguaje de programación**:
  + Se selecciona **Python con Django** para back-end por su simplicidad y compatibilidad con software libre.
  + **Vue.js o React** para el front-end por su capacidad de interacción eficiente.
* **Herramientas para el desarrollo**:
  + **Git** para control de versiones.
  + **Visual Studio Code** o **PyCharm** como IDE.
  + **Docker** para asegurar consistencia en el entorno de desarrollo.

**3. Etapas del Ciclo de Vida del Software (Máxima puntuación)**

* **Metodología de trabajo**: Se utilizará **SCRUM**, con sprints de 3 semanas.
  + **Sprint 1**: Diseño básico del sistema y funcionalidades principales de reserva.
  + **Sprint 2**: Implementación del módulo de pagos y gestión de usuarios.
  + **Sprint 3**: Integración de seguridad y pruebas unitarias.
  + **Sprint 4**: Despliegue en entorno de prueba y feedback del cliente.
* **Refactorización continua**: Al final de cada sprint se realizará una revisión de código para optimizar y mejorar el rendimiento.
* **Control de calidad**: Cada módulo será probado de manera individual (pruebas unitarias) y luego con el sistema completo (pruebas de integración).
* **Despliegue y mantenimiento**: El despliegue será continuo (CI/CD) usando **Docker** y sistemas de integración continua como **Jenkins**.

**4. Cumplimiento con los principios de Software Libre (Máxima puntuación)**

* **Ajuste en la planificación para desarrollo móvil:**
* **Lenguaje y framework para el desarrollo de la app móvil**:
* **Flutter** (Dart): Es una excelente opción para desarrollar apps móviles **multiplataforma** (iOS y Android) con una sola base de código.
* **React Native** (JavaScript): Otra opción muy popular que permite crear apps para iOS y Android usando **JavaScript** y React, facilitando la integración con web y back-end.
* **Back-end y APIs**:
* Puedes seguir usando **Python** con **Django** o **Flask** para el back-end, ya que funcionará bien con las APIs que la app móvil necesitará consumir. Estas APIs proporcionarán las funcionalidades como la reserva de pistas, pagos, y consultas de disponibilidad.
* **Node.js** también podría ser una opción para el back-end si prefieres mantener un stack completamente basado en **JavaScript**.
* **Base de datos y servidores**:
* **PostgreSQL** o **MySQL** seguirán siendo buenas opciones para la gestión de las reservas y la información de los usuarios.
* El uso de **Docker** sigue siendo recomendable para el entorno de desarrollo del back-end, asegurando la consistencia entre entornos.
* **Front-end web (para administración y central)**:
* Aunque la mayoría de los usuarios finales usarán la app móvil, la **administración** de las pistas y el sistema de gestión de la central del gimnasio puede ser desarrollado en **Vue.js** o **React**, que ofrecerán una buena integración con la app y el back-end.
* **Resumen ajustado:**
* **App móvil (usuario final)**: Desarrollada en **Flutter** o **React Native**.
* **Back-end (APIs y lógica)**: Mantener **Python/Django** o cambiar a **Node.js**.
* **Base de datos**: **PostgreSQL** o **MySQL**.
* **Gestión interna (central)**: Web app desarrollada con **Vue.js** o **React** para la administración.
* De esta forma, se facilita la implementación móvil y se asegura una experiencia fluida para los usuarios que reservan desde sus dispositivos.
* **Entorno de desarrollo** basado en software libre:
  + Uso de **Python**, **Vue.js/React**, **Docker**, **Git**, **MySQL/PostgreSQL**, que son herramientas de código abierto.
  + El proyecto se gestionará en **GitHub** o **GitLab**, asegurando el cumplimiento de las licencias GPL o MIT.
* **Mantenimiento del código**:
  + Todo el código estará disponible para que la comunidad lo revise, modifique y extienda según los principios de software libre.
* **Licencia GPL**: Propuesta para asegurar que el código es accesible y reutilizable.

**Conclusión:**

El enfoque propuesto sigue un ciclo de vida **ágil**, garantizando la entrega progresiva de valor al cliente, cumpliendo con los estándares de software libre y la modularidad exigida por el caso práctico. Esto no solo asegura la máxima puntuación en la rúbrica, sino que también mantiene las **buenas prácticas de diseño**, evitando los problemas de escalabilidad y adaptabilidad mencionados en tu pregunta anterior.

Este esquema garantiza que la planificación esté alineada con la **rúbrica** y maximiza el puntaje en cada uno de los criterios evaluados.

**ACTUALIZACION CON LA UNIDA 1**

**1. Elección del Lenguaje de Programación**

* **Basado en los lenguajes de programación presentados en la unidad**:
  + **Lenguaje adecuado**: Para una app móvil, los lenguajes recomendados serían:
    - **Kotlin** para desarrollo Android
    - **Swift** para iOS
    - O **frameworks multiplataforma** como **React Native** o **Flutter** que permiten crear apps para Android e iOS usando un solo código base.
  + **Motivación**: Como se menciona en la unidad, es crucial elegir un lenguaje adecuado para la portabilidad y capacidad de adaptación al sistema operativo del usuario.

**2. Metodología Ágil: SCRUM**

* **Aplicación de SCRUM**:
  + **Sprints**: Cada iteración debe tener entregables funcionales. Al tratarse de una app, esto podría incluir:
    1. Desarrollo de la interfaz de usuario (UI) básica
    2. Integración con la base de datos para reservas
    3. Funcionalidad de registro y autenticación de usuarios
    4. Gestión de horarios de pistas y disponibilidad
  + **Roles en el equipo**:
    1. **Product Owner**: Define las prioridades del cliente en cuanto a funcionalidades esenciales de la app.
    2. **Scrum Master**: Asegura que el equipo siga las prácticas ágiles.
    3. **Equipo de desarrollo**: Desarrolla los módulos de la app conforme a los Sprints.

**3. Fases del Desarrollo**

* **Análisis**: Según el temario, se debe definir con claridad los requisitos del cliente:
  + ¿Cuántas pistas se gestionarán?
  + ¿Qué tipo de autenticación usarán los usuarios?
  + ¿Se incluirán notificaciones para confirmar reservas?
* **Diseño**: En la fase de diseño se eligen las herramientas y los entornos:
  + **Base de datos**: MySQL podría ser usada para el backend, gestionando la disponibilidad de pistas y reservas.
  + **API REST**: Se podría implementar un servicio web para conectar la base de datos con la app.
* **Codificación**: Se programará la app en sprints, comenzando con lo esencial, y ampliando funcionalidad en cada iteración.
* **Pruebas**: Se deberán realizar pruebas de unidad y de integración. Especialmente, pruebas de interfaz (caja negra) y rendimiento.
* **Explotación y Mantenimiento**: El cliente recibirá la app y se le proporcionará un manual de uso, además de documentación técnica para el mantenimiento futuro.

**4. Documentación Técnica**

* **Guía de instalación y uso**:
  + Manual técnico detallado para la instalación del sistema backend y base de datos.
  + Instrucciones para los usuarios finales sobre cómo usar la app para reservar pistas.
* **Código fuente documentado**: Cada función en el código tendrá comentarios descriptivos para facilitar el mantenimiento futuro, como lo menciona el apartado de buenas prácticas en la unidad.

Este esquema sigue de cerca las recomendaciones de la Unidad 1 del módulo de Entornos de Desarrollo, garantizando que se cumplan los puntos clave de la rúbrica con base en el temario proporcionado.

Punto 1 de la Rúbrica:

identificar los **requerimientos funcionales** de la aplicación (como las funcionalidades específicas: reservas, pagos, control de luz, etc.), pero cuando la rúbrica menciona **identificar completamente los requerimientos y presentar un plan detallado para el análisis**, se refiere a tener en cuenta tanto los **requerimientos funcionales** como otros tipos de requerimientos que podrían afectar el desarrollo, implementación y mantenimiento de la aplicación.

Vamos a dividirlos en varias categorías para asegurarnos de cubrir todo lo necesario

**1. Requerimientos Funcionales**

* Estos son los requerimientos que mencionaste:
  + **Consultar el estado de las pistas de pádel**: La aplicación debe mostrar si las pistas están disponibles o reservadas.
  + **Reservar pistas**: Los usuarios deben poder reservar pistas, indicando si las necesitan con luz.
  + **Pagos con tarjeta**: Integración de un sistema de pago en línea con tarjeta de crédito.
  + **Listado de reservas**: Generar informes por hora y por cliente sobre las reservas realizadas.

**Análisis**: Aquí el objetivo es asegurar que las funciones clave que necesita el cliente están claramente identificadas y que cualquier caso de uso (consulta, reserva, pago, generación de informes) esté planificado con detalles sobre cómo se implementará.

**2. Requerimientos No Funcionales**

Estos incluyen características que no están relacionadas directamente con las funciones principales, pero que son esenciales para el sistema en general:

* **Compatibilidad con software libre**: Es fundamental que las herramientas y tecnologías seleccionadas (frameworks, lenguajes de programación, bases de datos, etc.) sean de código abierto, como se pide explícitamente.
* **Escalabilidad**: La aplicación debe ser capaz de manejar un número creciente de usuarios y reservas sin perder rendimiento.
* **Seguridad**: Los datos de los usuarios (especialmente los de pago) deben estar protegidos, cumpliendo con normativas de protección de datos.
* **Disponibilidad**: El sistema debe estar disponible 24/7 para permitir a los usuarios hacer reservas en cualquier momento.
* **Usabilidad**: La interfaz debe ser sencilla y accesible, facilitando la experiencia de los usuarios sin problemas.

**Análisis**: Se deben definir las métricas de rendimiento (por ejemplo, capacidad de respuesta, tiempo de carga) y los requisitos de seguridad (cifrado de datos, cumplimiento con estándares de pago).

**3. Requerimientos Técnicos**

* **Entorno de desarrollo**: Se especifica el uso de software libre, por lo tanto, las herramientas deben cumplir con esta premisa. Ejemplos:
  + Lenguajes como **Python** (con Django o Flask para back-end), o **Node.js** para la API.
  + Bases de datos como **PostgreSQL** o **MySQL** (ambas de código abierto).
  + **React Native** o **Flutter** para el desarrollo multiplataforma (iOS y Android).
  + Uso de **Git** para control de versiones y **Docker** para gestionar entornos.
* **Integración con pasarelas de pago**: Uso de servicios de pago compatibles con software libre o que permitan fácil integración con las tecnologías elegidas (como Stripe o PayPal).

**Análisis**: Aquí se debe elegir cuidadosamente el stack tecnológico y asegurarse de que cada componente técnico es compatible con el resto y con los principios de software libre.

**4. Requerimientos de Negocio**

* **Número de usuarios esperados**: Tener una estimación de cuántos usuarios usarán la aplicación ayudará a definir la capacidad del sistema.
* **Presupuesto**: Considerar si hay limitaciones presupuestarias que puedan influir en la elección de tecnologías o en la escalabilidad futura del sistema.
* **Mantenimiento**: Planificar cómo se llevará a cabo el mantenimiento de la aplicación, quién será responsable y si el cliente tiene capacidad para gestionarlo internamente o se necesita soporte externo.
* **Personal**: Evaluar cuántos desarrolladores y especialistas (en seguridad, bases de datos, etc.) serán necesarios para llevar a cabo el proyecto en tiempo y forma.

**Análisis**: Aquí debes tener en cuenta factores como el tamaño del equipo, la cantidad de tiempo de desarrollo disponible, y si hay planes de expansión o actualizaciones frecuentes.

**5. Requerimientos Legales**

* **Protección de datos**: La aplicación debe cumplir con normativas de protección de datos como el GDPR, ya que se manejarán datos sensibles (reservas, pagos, datos personales).
* **Licencias de software**: Asegurarse de que todas las herramientas utilizadas estén bajo licencias adecuadas (GPL, MIT, etc.) para que no haya problemas legales relacionados con el uso de software libre.

**Análisis**: Aquí debes incluir los planes para el cumplimiento de normativas legales, asegurándote de que no se vulneran derechos de los usuarios o de terceros al manejar información personal o al utilizar ciertas tecnologías.

**Plan Detallado para el Análisis de Requerimientos:**

Para maximizar la puntuación en este apartado, es fundamental tener un plan detallado para abordar cada uno de los requerimientos.

1. **Reunión con el cliente**:
   * Reuniones iniciales para confirmar y validar los requerimientos funcionales y no funcionales.
   * Definir las prioridades y asegurar que el cliente entiende los compromisos que implica el uso de software libre.
2. **Documentación de requerimientos**:
   * Redacción de un **documento de especificaciones** que detalle todos los puntos anteriores: funcionales, no funcionales, técnicos, de negocio y legales.
   * Validación con el cliente para asegurar que todos los requerimientos están completos y correctos.
3. **Prototipo inicial**:
   * Desarrollo de un prototipo sencillo (puede ser solo la interfaz) para que el cliente visualice el proyecto y valide los flujos principales.
   * Recopilación de feedback y ajustes en los requerimientos antes de pasar a la fase de desarrollo completa.
4. **Revisión continua**:
   * Establecer revisiones periódicas (al final de cada sprint si usas **SCRUM**) para asegurarse de que los requerimientos se cumplen y de que no han cambiado las prioridades del cliente.

**Conclusión**: La identificación y análisis de los requerimientos no solo cubre los funcionales, sino también los técnicos, de negocio, legales y no funcionales. Presentar un plan claro para cada aspecto de la aplicación asegura que no haya sorpresas durante el desarrollo y que el proyecto siga alineado con las expectativas del cliente desde el inicio.

Punto 2 de la Rúbrica:

Para cumplir con este punto de la rúbrica, vamos a **seleccionar un lenguaje de programación y herramientas** que cumplan con los requerimientos del proyecto, basándonos en la premisa de que la empresa está comprometida con el **software libre**. A continuación, se fundamenta la elección de cada uno:

**1. Lenguaje de programación: JavaScript con Vue.js para el frontend y Node.js para el backend**

**Justificación:**

* **Software libre**: Tanto **Vue.js** como **Node.js** son tecnologías open-source (de código abierto), lo cual alinea perfectamente con el compromiso de la empresa hacia el software libre.
* **Popularidad y soporte**: JavaScript es uno de los lenguajes más populares y usados en el desarrollo web. Su comunidad es muy activa y ofrece una gran cantidad de recursos y bibliotecas, lo que facilita el desarrollo rápido y eficiente de aplicaciones.
* **Desarrollo full-stack**: Al usar **JavaScript** en ambas capas de la aplicación (frontend y backend), se reduce la curva de aprendizaje del equipo de desarrollo y se mantiene la coherencia en el código.
* **Desarrollo de aplicaciones móviles**: Con **frameworks como Vue Native o Ionic** (ambos compatibles con Vue.js), es posible desarrollar aplicaciones móviles para Android e iOS desde un solo código base, algo que es eficiente y está alineado con los requisitos del proyecto (mayoría de usuarios en dispositivos móviles).

**2. Frameworks:**

* **Frontend: Vue.js**
  + **Justificación**: Vue.js es un framework progresivo que permite construir interfaces de usuario interactivas y ligeras. Es fácil de integrar en proyectos existentes, lo cual es útil si se busca expandir la aplicación en el futuro.
  + **Curva de aprendizaje suave**: Comparado con otros frameworks como React o Angular, Vue.js es más fácil de aprender y, al ser modular, permite escalar el proyecto de manera eficiente.
  + **Componentes reutilizables**: Facilita la creación de componentes reutilizables, lo que se traduce en una mayor eficiencia y mantenibilidad del código a lo largo de los sprints.
* **Backend: Node.js con Express**
  + **Justificación**: Node.js permite la creación de servidores rápidos y eficientes con JavaScript, permitiendo manejar múltiples conexiones simultáneamente, algo ideal para aplicaciones que requieren alta concurrencia, como la reserva de pistas.
  + **Express.js**: Es un framework ligero para Node.js que facilita la creación de APIs RESTful, las cuales se pueden integrar fácilmente con bases de datos y frontends móviles o web.
  + **Módulos NPM (Node Package Manager)**: Amplia colección de paquetes y librerías de software libre que aceleran el desarrollo y permiten integrar fácilmente funcionalidades como la pasarela de pagos.

**3. Bases de datos: PostgreSQL**

* **Justificación**: PostgreSQL es una base de datos relacional de código abierto, conocida por su robustez, escalabilidad y soporte de estándares. Es ideal para proyectos de software libre y ofrece una gran flexibilidad en la definición de esquemas y en la ejecución de consultas complejas.
* **Ventajas**:
  + **Alta escalabilidad**: Es capaz de manejar grandes volúmenes de datos, lo que es útil si la empresa crece y el número de usuarios y reservas aumenta.
  + **Integración con Node.js**: Existen librerías y paquetes de fácil integración con Node.js para conectar la base de datos con el backend, como pg-promise.
  + **Consultas complejas**: Permite realizar consultas complejas de una manera eficiente, algo que será útil cuando se necesiten generar informes detallados de las reservas por horas y clientes.

**4. Herramientas de despliegue y contenedores: Docker**

* **Justificación**: Docker es una plataforma de contenedores de código abierto que permite empaquetar y desplegar aplicaciones con todas sus dependencias. Es una herramienta clave para asegurar que el entorno de desarrollo es consistente con el de producción.
  + **Software libre**: Docker es ampliamente utilizado en entornos de código abierto y es compatible con la mayoría de los sistemas operativos.
  + **Ventajas**:
    - **Facilidad de despliegue**: Docker permite que la aplicación, junto con su base de datos, frontend y backend, se despliegue rápidamente en cualquier servidor.
    - **Escalabilidad**: Es fácil escalar servicios como bases de datos o APIs de manera eficiente.

**5. Control de versiones: Git**

* **Justificación**: Git es el sistema de control de versiones más utilizado y es de código abierto. Permite a los equipos de desarrollo trabajar de manera colaborativa en diferentes ramas del proyecto, manteniendo un registro detallado de los cambios y facilitando la integración continua.
  + **Repositorio GitHub o GitLab**: Son servicios gratuitos y abiertos que pueden alojar el código y ofrecen integración con otras herramientas para la gestión de proyectos ágiles.

**6. Pasarela de pagos: Stripe API**

* **Justificación**: Stripe es una API para pagos en línea, conocida por su fácil integración, seguridad y soporte para diversas tarjetas de crédito y métodos de pago. Aunque no es 100% open-source, es compatible con Node.js y es ampliamente usada en proyectos que requieren pagos.
  + **Seguridad**: Stripe cumple con todas las normativas de seguridad, lo que es fundamental para manejar pagos con tarjeta.

Con **Vue.js** se puede desarrollar tanto la aplicación móvil como el frontend de la web que utilizará la empresa para controlar las pistas. Aquí te explico cómo:

**1. Desarrollo de la App Móvil para Android e iOS**

Vue.js por sí mismo está más enfocado en el desarrollo web, pero puedes usar **frameworks complementarios** que te permiten desarrollar aplicaciones móviles con el mismo lenguaje:

* **Vue Native**: Permite desarrollar aplicaciones móviles nativas para Android e iOS utilizando Vue.js. Vue Native genera código nativo detrás de las escenas, lo que significa que puedes desarrollar una sola vez y desplegar en ambos sistemas.
* **Ionic con Vue.js**: **Ionic** es un framework popular que permite crear aplicaciones híbridas (webview) que funcionan en Android e iOS. Ahora soporta Vue.js, lo que significa que puedes usar Vue.js para construir tanto el frontend de la aplicación móvil como el backend en Node.js.

Estas herramientas te permitirán crear una aplicación móvil **con un solo código base** y desplegarla tanto en **Android** como en **iOS**.

**2. Frontend de la aplicación web para la empresa**

Puedes utilizar **Vue.js** para crear la interfaz web que los administradores de la empresa utilizarán para gestionar las pistas de pádel, ver las reservas, generar listados, etc. Esto es posible gracias a que Vue.js está diseñado para crear **interfaces interactivas y rápidas** que pueden funcionar en cualquier navegador moderno.

**3. Backend para gestionar el sistema central**

Para el backend, puedes usar **Node.js con Express** para gestionar la lógica del servidor y la conexión con la base de datos **PostgreSQL**. Este backend servirá tanto a la app móvil como a la aplicación web, proporcionando una API que manejará los datos, la autenticación de usuarios, las reservas de pistas, los pagos, etc.

**Resumen:**

* **Vue.js** con **Ionic o Vue Native** puede manejar el desarrollo de la app móvil para **Android e iOS**.
* **Vue.js** también puede manejar el **frontend web** de la aplicación central que la empresa usará.
* **Node.js con Express** servirá para gestionar el **backend** de la aplicación tanto para el móvil como para la web.

De este modo, usas una sola tecnología (**JavaScript/Vue.js**) para cubrir tanto el **frontend** de la web como las **apps móviles**, y mantienes coherencia y simplicidad en el desarrollo del proyecto.

Punto 3 de la Rúbrica:

Sí, en este punto de la rúbrica, se refiere a la planificación del **diseño** y desarrollo del software, en el contexto de un **plan detallado** que cubra todas las etapas del ciclo de vida del software. Se trata de diseñar un plan que explique cómo se va a desarrollar el proyecto paso a paso, desde la planificación inicial hasta la entrega final, especificando las tecnologías que se usarán.

**Plan detallado de desarrollo del software (Ciclo de vida del software)**

**1. Fase de Análisis y Requerimientos**

* **Objetivo**: Recoger y analizar todos los requerimientos del proyecto. Definir claramente qué necesita la empresa y qué funcionalidades debe tener la aplicación.
* **Actividades**:
  + Reunión con los stakeholders (empresa).
  + Identificación de requerimientos funcionales (consulta de pistas, pagos, listados).
  + Identificación de requerimientos no funcionales (seguridad, rendimiento).
  + Revisión de requerimientos técnicos: Asegurar el uso de software libre.

**2. Fase de Diseño**

* **Objetivo**: Planificar la estructura de la aplicación. Definir cómo estarán organizadas las distintas partes de la aplicación, tanto a nivel de backend como de frontend.
* **Actividades**:
  + **Diseño del frontend (interfaz de usuario)**:
    - Lenguaje y herramientas: **Vue.js** para el desarrollo de la interfaz.
    - Diseño modular y componentes reutilizables.
  + **Diseño del backend (lógica de negocio)**:
    - Lenguaje y herramientas: **Node.js con Express** para el desarrollo del servidor y las API.
    - Arquitectura RESTful para la comunicación entre frontend y backend.
  + **Diseño de la base de datos**:
    - Herramienta: **PostgreSQL**.
    - Diseño de las tablas necesarias para gestionar las reservas, usuarios, pistas y pagos.
  + **Diseño del sistema de despliegue**:
    - Herramienta: **Docker** para la contenedorización y despliegue en entornos de desarrollo, pruebas y producción.
  + **Pasarela de pagos**:
    - Herramienta: **Stripe API** para pagos seguros con tarjeta.

**3. Fase de Implementación y Codificación**

* **Objetivo**: Desarrollar el software en base a los diseños establecidos, respetando los principios de SOLID y buenas prácticas de desarrollo.
* **Actividades**:
  + **Implementación del frontend**:
    - Desarrollo de los componentes de la interfaz de usuario usando **Vue.js**.
    - Definir y conectar el frontend con las APIs del backend.
  + **Implementación del backend**:
    - Desarrollo de las APIs RESTful usando **Node.js con Express** para gestionar las operaciones de la aplicación (consultas de pistas, pagos, etc.).
  + **Integración con la base de datos**:
    - Crear las consultas y operaciones de la base de datos para manejar la información de las reservas, usuarios y pistas, utilizando **PostgreSQL**.
  + **Integración con la pasarela de pagos**:
    - Integrar **Stripe API** para manejar pagos con tarjeta de forma segura.
  + **Despliegue de la aplicación**:
    - Usar **Docker** para desplegar y escalar la aplicación.

**4. Fase de Pruebas**

* **Objetivo**: Verificar y validar el funcionamiento del software antes de la entrega al cliente.
* **Actividades**:
  + **Pruebas unitarias**: Asegurar que cada componente (tanto del frontend como del backend) funciona correctamente por separado.
  + **Pruebas de integración**: Asegurar que los diferentes componentes del sistema funcionan correctamente cuando están integrados.
  + **Pruebas de sistema**: Evaluar el sistema completo bajo condiciones reales de uso.
  + **Pruebas de rendimiento y seguridad**: Asegurar que la aplicación puede manejar múltiples usuarios y que los pagos son seguros.

**5. Fase de Despliegue**

* **Objetivo**: Poner en marcha el software para el uso real por parte de la empresa y los usuarios.
* **Actividades**:
  + Despliegue del backend en un servidor usando **Docker**.
  + Despliegue del frontend (Vue.js) en un servidor o plataforma de hosting.
  + Configuración de la base de datos **PostgreSQL** en el servidor.
  + **Documentación**: Incluir documentación técnica sobre cómo desplegar y mantener la aplicación.

**6. Fase de Mantenimiento**

* **Objetivo**: Asegurar que el software se mantenga actualizado y funcione correctamente a lo largo del tiempo.
* **Actividades**:
  + Corrección de errores encontrados después del despliegue.
  + Actualizaciones y mejoras en función de las necesidades de la empresa o los usuarios.
  + Mantenimiento de la seguridad, asegurando que la pasarela de pagos y los datos de los usuarios estén protegidos.

**Resumen del plan de desarrollo:**

Este plan cubre todas las etapas del desarrollo del software, desde la recogida de los requerimientos hasta el despliegue final. Utilizamos **Vue.js** para el frontend (incluyendo la app móvil), **Node.js con Express** para el backend, **PostgreSQL** para la base de datos, **Docker** para el despliegue y **Stripe** para los pagos, asegurando que todo el proceso sigue las premisas de **software libre** y buenas prácticas de desarrollo.

Punto 4 de la Rúbrica:

compromiso con el **software libre** al seleccionar tecnologías y herramientas que cumplen con esta premisa, pero podemos **profundizar** y **justificar más detalladamente** cómo cada decisión apoya dicho compromiso. Aquí te explico algunos puntos adicionales que podemos incluir para fortalecer este apartado:

**Explicación detallada del compromiso con el software libre:**

1. **Frameworks y lenguajes de programación**:
   * **Vue.js**: Es un framework de JavaScript **open-source** (código abierto) con una gran comunidad de desarrolladores que contribuyen a su mejora continua. El uso de Vue.js apoya la premisa de software libre ya que cualquier persona puede contribuir al desarrollo y beneficiarse de su uso sin restricciones.
   * **Node.js**: **Node.js** es también una tecnología **open-source**, ampliamente utilizada y mantenida por la comunidad global. Es una plataforma libre para ejecutar JavaScript en el servidor, lo que permite la flexibilidad en el desarrollo de backend sin depender de software propietario.
   * **PostgreSQL**: Una de las bases de datos relacionales más avanzadas y también de **código abierto**. Su uso asegura que no estamos atados a soluciones comerciales y podemos adaptar el sistema sin depender de licencias pagadas.
   * **Docker**: Herramienta de contenedorización **open-source** que permite empaquetar y distribuir el software en entornos controlados y replicables. Docker garantiza que podemos gestionar y desplegar entornos de software libre sin depender de plataformas cerradas.
2. **Pasarela de pagos (Stripe)**:
   * Aunque **Stripe** es un servicio de pago que no es libre, se integra de forma sencilla con herramientas y tecnologías de software libre. **La elección de un servicio de pago no afecta la libertad del software**, ya que todo el código y las funcionalidades de la aplicación, que controlamos, permanecen abiertas y libres.
3. **Sistema de control de versiones (Git)**:
   * **Git** es una herramienta de **control de versiones de código abierto** que facilita la colaboración entre desarrolladores y el control total del código fuente. Además, plataformas como **GitHub** o **GitLab** permiten almacenar y compartir el código con otras personas, fomentando el desarrollo colaborativo, que es un pilar fundamental del software libre.
4. **Beneficios del software libre**:
   * **Adaptabilidad**: Como estamos utilizando tecnologías de código abierto, tenemos la flexibilidad de modificar cualquier componente del sistema según las necesidades específicas de la empresa sin restricciones de licencias o permisos de proveedores.
   * **Transparencia**: El software libre fomenta la transparencia en el código, lo que permite a los usuarios y a la comunidad verificar su seguridad y fiabilidad. Esto es especialmente importante en aplicaciones que manejan datos sensibles, como los pagos.
   * **Comunidad y soporte**: Al utilizar tecnologías ampliamente adoptadas como Vue.js, Node.js y PostgreSQL, nos apoyamos en comunidades de desarrolladores muy activas, lo que garantiza acceso a documentación, soporte y actualizaciones constantes.

**Conclusión:**

El uso de **Vue.js**, **Node.js**, **PostgreSQL**, **Docker** y **Git**, todos de código abierto, asegura que el proyecto está en plena consonancia con la premisa de software libre. Estas decisiones permiten un desarrollo ágil, sin depender de software propietario, y ofrecen flexibilidad, transparencia y capacidad de evolución a lo largo del tiempo. Esto no solo respalda el compromiso de la empresa con el software libre, sino que también proporciona una base sólida para el crecimiento del sistema.

Corregir del trabajo:

No usar vue, usar lo mas cotidiano como React o Flutter . Porque está discontinuado.

Según tamaño de la empresa y presupuesto.

Una persona, Flutter (Dart) o React (Javascript) mejor este ultimo. Es lo mas barato y vale para los dos y lo puedes compilar para Android, ios y web .

Esto no. Muchas personas, una app para ios nativa switft, una app nativa Android con kotlin y una app web para gestión con react con javascript.

Usamos Vscode para instalar react.

Stripe es para extranjeros, y para españoles RedSys y habría que ver cual cobra menos comisión y etc.

React ya trae interacción con Docker. Y Docker es como un entorno que te permite ejecutar javascript , porque recordemos que javascript no es compilado.

Node.js permite ejecutar javascript en el servidor.

Requerimientos de usuarios si se tienen que dar de alta, o ya están en el entorno de de los gimnasios, o solo para el paddel.

Si conviene hacer una app móvil o no.

Politica de reservas de pistas, limites, tarifas y prioridades. Con luz o sin luz, Clubes, mucha gente, anticipación que puedes reservar la pista y cancelar.

Los clubes, sección de usuarios y sección de clubes.

Donde se aloja el programa, y quien lo mantiene.

Quien sube las cosas al server y a las stores online.

No usar express con node.

Ver que es Restful. Consulta api entre frontend backend. Pero para esto si se usa express.

**Desarrollo de una Aplicación de Reserva de Pistas de Pádel**

**1. Identificación de Requerimientos**

Se desarrollará una aplicación para reservar pistas de pádel que debe cumplir con los siguientes requerimientos:

* **Consultar el estado de cada pista de pádel** (disponible o no).
* **Posibilidad de realizar pagos con tarjeta**, considerando el uso de **RedSys** para España, evaluando las comisiones y tarifas.
* **Controlar la necesidad de reservar la pista** con o sin luz.
* **Políticas de reservas de pistas** que incluyan límites, tarifas y prioridades (pistas con luz, clubes, antelación para reservar y cancelar).
* **Generar un listado de pistas reservadas** por horas y clientes.
* Definir si los usuarios necesitan registrarse o si ya están en el entorno de los gimnasios.

**2. Lenguaje de Programación y Herramientas**

* **Framework**: Se utilizará **React** como principal tecnología para el desarrollo de la aplicación web y se evaluará si es conveniente hacer una aplicación móvil o no. En el caso de optar por aplicaciones nativas, se desarrollarán:
  + React Native para moviles
* **Herramientas de Desarrollo**: Se usará **Visual Studio Code** para el desarrollo de la aplicación en React, y se aprovechará la integración con **Docker** para crear un entorno de desarrollo controlado.
* **Backend**: Para el servidor se utilizará **Node.js** para ejecutar JavaScript en el servidor, utilizando un enfoque **RESTful** para las consultas entre el frontend y el backend.

**3. Plan de Desarrollo del Software**

* **Fases del desarrollo**:
  + **Análisis de requerimientos**: Identificar y documentar todos los requerimientos de la aplicación.
  + **Diseño**:
    - Diseño de la arquitectura del sistema y la base de datos.
    - Creación de maquetas para la interfaz de usuario.
  + **Desarrollo**:
    - Implementación del frontend en React.
    - Implementación del backend en Node.js.
    - Integración con RedSys para el procesamiento de pagos.
  + **Pruebas**: Verificación de que todas las funcionalidades cumplen con los requerimientos establecidos.
  + **Despliegue**: Publicar la aplicación en un servidor y en las tiendas de aplicaciones (App Store y Google Play).

**4. Compromiso con el Software Libre**

Se seleccionarán herramientas y tecnologías de software libre y de código abierto para el desarrollo del proyecto, lo que asegura que el código fuente sea accesible y modificable. Esto incluye el uso de React y Node.js, así como el entorno de desarrollo proporcionado por Docker.

**5. Contenido Gráfico**

* Incluir diagramas de flujo que representen el proceso de reserva de pistas.
* Capturas de pantalla de la interfaz de usuario.
* Diagramas de arquitectura que muestren cómo interactúan los diferentes componentes del sistema.

React Native

Sí, se puede utilizar **React** para desarrollar aplicaciones móviles que se pueden compilar en Android e iOS utilizando el mismo código fuente a través de **React Native**. Aquí hay algunos puntos clave sobre cómo funciona esto:

**1. React Native**

* **React Native** es un framework basado en React que permite a los desarrolladores crear aplicaciones móviles nativas para iOS y Android utilizando JavaScript y React.
* Con React Native, puedes escribir la mayor parte de tu código una vez y ejecutarlo en ambas plataformas, lo que reduce el tiempo y el esfuerzo de desarrollo.

**2. Código Compartido**

* **Componentes Reutilizables**: Muchos de los componentes y la lógica de tu aplicación se pueden compartir entre las plataformas, lo que significa que puedes mantener una base de código unificada.
* **Plataformas Específicas**: Aunque la mayoría del código es compartido, puedes personalizar ciertos componentes para que se comporten de manera diferente en iOS y Android cuando sea necesario.

**3. Compilación**

* Para compilar y ejecutar tu aplicación en Android, puedes usar **Android Studio** y para iOS, **Xcode**. React Native proporciona herramientas para configurar estas compilaciones.
* Utilizando **Expo**, una herramienta adicional para React Native, puedes simplificar aún más el proceso de desarrollo, ya que te permite iniciar rápidamente un proyecto y compilar aplicaciones sin necesidad de configurar Xcode o Android Studio de inmediato.

**4. Ventajas**

* **Desarrollo Rápido**: Puedes ver los cambios en tiempo real durante el desarrollo utilizando "hot reloading", lo que mejora la productividad.
* **Comunidad Activa**: React Native tiene una comunidad grande y activa, lo que significa que hay muchas bibliotecas y recursos disponibles para ayudarte a resolver problemas.

**5. Limitaciones**

* Aunque React Native permite el uso de código compartido, algunas características específicas de cada plataforma pueden requerir implementaciones separadas.
* Algunas bibliotecas o módulos pueden no estar disponibles para ambas plataformas, lo que puede requerir trabajo adicional.

En resumen, usar React Native te permite desarrollar aplicaciones móviles para iOS y Android a partir del mismo código base, lo que puede ser una opción muy eficiente en términos de tiempo y

React

Para la parte de gestión o el sistema central de la aplicación de reservas de pistas de pádel, generalmente se recomienda usar **React** en lugar de **React Native**. Aquí te explico las razones y consideraciones para cada caso:

**1. Uso de React para la Gestión Central**

* **Web vs. Móvil**: React es ideal para construir aplicaciones web, mientras que React Native está diseñado para aplicaciones móviles. La aplicación de gestión, que probablemente tendrá más funcionalidades de administración y una interfaz más compleja, se beneficiaría de las capacidades que ofrece React para construir aplicaciones web.
* **Interfaz de Usuario**: La gestión central puede requerir una interfaz más rica y adaptativa que pueda aprovechar completamente el navegador, donde React sobresale en la creación de componentes reutilizables y en la gestión del estado de la aplicación.
* **Herramientas y Bibliotecas**: Existen muchas bibliotecas y herramientas en el ecosistema de React, como **React Router** para la navegación y **Redux** para la gestión del estado, que pueden hacer que el desarrollo de la aplicación de gestión sea más eficiente y estructurado.

**2. Consideraciones para Usar React Native en la Gestión Central**

* **Uniformidad**: Si deseas que tanto la aplicación móvil como la gestión central sean consistentes en términos de diseño y experiencia de usuario, podrías considerar el uso de React Native para la gestión. Sin embargo, esto podría requerir más trabajo, ya que algunas características de una aplicación web no se transfieren directamente a una aplicación móvil.
* **Acceso Móvil**: Si se anticipa que los administradores o el personal de gestión necesitarán acceder al sistema desde dispositivos móviles, podría tener sentido desarrollar la parte de gestión en React Native. No obstante, esto puede limitar ciertas funcionalidades que son más fáciles de implementar en un entorno web.

**3. Recomendación General**

* **Aplicación Web con React**: Para la parte de gestión, se recomienda usar **React** para desarrollar una aplicación web robusta que pueda ser utilizada en cualquier dispositivo con acceso a un navegador. Esto facilitará la gestión y administración de las reservas de pistas, con una interfaz más rica y una mejor experiencia de usuario.
* **Aplicación Móvil con React Native**: Mantener la aplicación de reservas de pistas (la interfaz del usuario) en **React Native** para iOS y Android permitirá que los usuarios reserven pistas desde sus dispositivos móviles.

**Conclusión**

Usar **React** para la parte de gestión central y **React Native** para la aplicación móvil es un enfoque común y efectivo que permite aprovechar al máximo las características de cada tecnología.