|  |  |  |
| --- | --- | --- |
|  | |  |
|  | | |
|  | | |
|  |  | |
| alumnO CESUR  Alejandro Muñoz de la Sierra | ENTORNOS DE DESARROLLO 2024/25  CASO PRACTICO 1 | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| CONTENIDO | | | | | | |
| 01 |  | 02 |  | 03 |  | 04 |
| Executive Summary |  | Q1 Highlights |  | Visual Data |  | Q2 Highlights |
|  |  |  |  |  |  |  |
| 05 |  | 06 |  | 07 |  | 08 |
| Q3 Highlights |  | Chapter 2 |  | Chapter 3 |  | Chapter 4 |
|  |  |  |  |  |  |  |
| 09 |  | 10 |  | 11 |  | 12 |
| Chapter 5 |  | Chapter 6 |  | Chapter 7 |  | Chapter 8 |

|  |  |
| --- | --- |
| INTRODUCCION |  |
| En esta práctica, trabajamos para una empresa de desarrollo de software que ha recibido un encargo de una gran compañía de gimnasios, la cual cuenta con más de 100 centros deportivos en toda España. Debido a la creciente demanda de pádel, el cliente requiere una solución digital para gestionar las reservas de sus pistas de pádel. El objetivo del proyecto es desarrollar una aplicación que permita a los usuarios consultar el estado de las pistas, realizar reservas con o sin luz y efectuar pagos con tarjeta, todo ello desde dispositivos móviles, tablets y PCs.  Adoptando la metodología **Scrum**, trabajaremos en equipo siguiendo un enfoque ágil, dividiendo el desarrollo en **sprints** para entregar versiones funcionales del software de forma iterativa. Este enfoque nos permitirá realizar entregas regulares al cliente y ajustar los requerimientos según las necesidades que puedan surgir. La solución se desarrollará utilizando **software libre**, asegurando su fácil implementación y cumpliendo con los compromisos del cliente hacia el uso de tecnologías abiertas. El desarrollo incluirá la creación de aplicaciones móviles para **Android** e **iOS**, además de una versión accesible desde PCs, utilizando tecnologías modernas y eficientes para asegurar un producto final robusto y escalable. |

|  |  |
| --- | --- |
| 01 ESTUDIO DE REQUERIMIENTOS |  |

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.

|  |
| --- |
| 02 Eleccion de software |

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 React para el frontend y Node.js para el backend**

**Justificación**:

* **Software libre**: Tanto **React** como **Node.js** son tecnologías de código abierto, lo que cumple con el compromiso de la empresa hacia el uso de software libre.
* **Popularidad y soporte**: **JavaScript** es uno de los lenguajes más populares y ampliamente utilizado en el desarrollo web y móvil. Tiene una gran comunidad y un ecosistema extenso que proporciona bibliotecas, recursos y soporte.
* **Desarrollo full-stack**: Usando JavaScript tanto en el frontend como en el backend se logra uniformidad en el código y una curva de aprendizaje más baja, lo que mejora la eficiencia del equipo de desarrollo.
* **Desarrollo de aplicaciones móviles**: **React Native** permite crear aplicaciones nativas para **Android** e **iOS** desde un solo código base, lo cual es óptimo considerando que la mayoría de los usuarios utilizarán dispositivos móviles para hacer las reservas.

**2. Frameworks:**

**Frontend para la gestión (app web): React**

* **Justificación**:
  + **Versatilidad**: **React** es uno de los frameworks más populares para construir interfaces de usuario dinámicas y escalables. Es ideal para la aplicación de gestión, que requerirá un entorno web eficiente y rápido.
  + **Componentes reutilizables**: **React** permite crear componentes reutilizables, lo cual agiliza el desarrollo y facilita el mantenimiento a largo plazo. Esto es clave para aplicaciones que pueden crecer en funcionalidad.
  + **Bibliotecas y ecosistema**: React tiene una enorme cantidad de bibliotecas y herramientas que pueden integrarse fácilmente para optimizar el desarrollo de la aplicación de gestión.

**Frontend para apps móviles (Android e iOS): React Native**

* **Justificación**:
  + **Código compartido**: **React Native** permite compartir gran parte del código entre las versiones de Android e iOS, lo que ahorra tiempo y esfuerzo en el desarrollo de las aplicaciones móviles.
  + **Rendimiento nativo**: A diferencia de soluciones híbridas como Ionic, **React Native** genera aplicaciones con rendimiento cercano a las aplicaciones nativas, lo que proporciona una mejor experiencia de usuario.
  + **Soporte extenso**: El ecosistema de React Native es grande y cuenta con numerosas bibliotecas y herramientas compatibles para integrar funcionalidades avanzadas, como pagos móviles y notificaciones push.

**Backend: Node.js con Express**

* **Justificación**:
  + **Alta concurrencia**: **Node.js** permite manejar múltiples solicitudes de forma eficiente, lo que es esencial en aplicaciones como la reserva de pistas que podrían recibir muchas conexiones simultáneamente.
  + **Express.js**: Un framework minimalista para Node.js que facilita la creación de **APIs RESTful**. Estas APIs pueden ser consumidas tanto por la app de gestión web (React) como por las aplicaciones móviles (React Native).
  + **Escalabilidad**: Con **Node.js**, es fácil escalar la aplicación en función de las necesidades de la empresa a medida que el número de usuarios crezca.

**3. Herramientas de desarrollo:**

**Entorno de desarrollo: VSCode (Visual Studio Code)**

* **Justificación**:
  + **Popularidad y extensiones**: **VSCode** es uno de los editores más populares en el desarrollo de software libre y ofrece una amplia variedad de extensiones para JavaScript, React, y React Native, lo que facilita la depuración y el desarrollo en tiempo real.
  + **Integración con Docker**: VSCode permite integrar herramientas como **Docker**, lo que facilita la creación de entornos de desarrollo consistentes para la aplicación.
  + **Soporte para Git**: VSCode tiene integración nativa con **Git**, lo que facilita la gestión de versiones del código durante los diferentes sprints.

**4. Integración con sistemas de pago:**

**RedSys para pagos en España:**

* **Justificación**:
  + **Preferencia local**: Dado que la empresa opera en España, se optará por la integración con **RedSys** para procesar pagos, ya que es la pasarela de pagos más utilizada en el país y permite realizar pagos con tarjeta de forma segura.
  + **Bajas comisiones**: Comparada con plataformas globales como Stripe, RedSys puede tener comisiones más competitivas para transacciones locales en España.

-----------------------------------------------------------------------------------------------------------

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.

-------------------------------------------------------------------------------------------

**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.

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, definiendo qué necesita la empresa y qué funcionalidades debe tener la aplicación.
* **Actividades**:
  + Reunión con los stakeholders (empresa).
  + Identificación de requerimientos funcionales:
    - Consultar el estado de las pistas de pádel.
    - Permitir pagos con tarjeta.
    - Controlar la reserva de pistas con o sin luz.
    - Generar listados de pistas reservadas por horas y clientes.
  + Identificación de requerimientos no funcionales: seguridad, rendimiento, disponibilidad.
  + Revisión de requerimientos técnicos: asegurar el uso de software libre.

**2. Fase de Diseño**

* **Objetivo**: Definir la arquitectura del sistema, el diseño de la interfaz de usuario, la base de datos y la estrategia de despliegue.
* **Actividades**:
  + **Diseño del frontend (interfaz de usuario)**:
    - **Lenguaje y herramientas**: **React** para el desarrollo de la interfaz web de la gestión de pistas y **React Native** para las aplicaciones móviles (iOS y Android).
    - **Diseño modular**: Uso de componentes reutilizables para maximizar la eficiencia en el desarrollo.
  + **Diseño del backend (lógica de negocio)**:
    - **Lenguaje y herramientas**: **Node.js** con **Express** para el desarrollo del servidor y las API.
    - **Arquitectura RESTful**: Comunicación eficiente entre frontend (React y React Native) y backend.
  + **Diseño de la base de datos**:
    - **Herramienta**: **PostgreSQL**.
    - **Diseño de tablas**: 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**: **RedSys** para el procesamiento de pagos seguros en España, preferido frente a Stripe por su compatibilidad con el mercado español.

**3. Fase de Implementación y Codificación**

* **Objetivo**: Desarrollar el software siguiendo el diseño definido y respetando las mejores prácticas de desarrollo (SOLID).
* **Actividades**:
  + **Implementación del frontend**:
    - Desarrollo de la interfaz de usuario para la gestión de pistas utilizando **React**.
    - Desarrollo de las aplicaciones móviles usando **React Native** para Android e iOS.
  + **Implementación del backend**:
    - Desarrollo de las APIs RESTful usando **Node.js** con **Express** para manejar las operaciones del sistema (consulta de pistas, pagos, etc.).
  + **Integración con la base de datos**:
    - Implementación de las consultas y operaciones para gestionar reservas, usuarios y pistas utilizando **PostgreSQL**.
  + **Integración con la pasarela de pagos**:
    - Integración de **RedSys** para procesar pagos con tarjeta de manera segura.
  + **Despliegue de la aplicación**:
    - Usar **Docker** para desplegar y escalar tanto el backend como el frontend de forma eficiente.

**4. Fase de Pruebas**

* **Objetivo**: Verificar que todas las funcionalidades cumplen con los requerimientos antes de la entrega.
* **Actividades**:
  + **Pruebas unitarias**: Asegurar que cada componente funciona correctamente de forma aislada.
  + **Pruebas de integración**: Verificar que los componentes se comunican correctamente entre sí.
  + **Pruebas de sistema**: Evaluar el rendimiento del sistema completo bajo condiciones reales de uso.
  + **Pruebas de rendimiento y seguridad**: Asegurar que la aplicación maneja múltiples usuarios y que los pagos son seguros.

**5. Fase de Despliegue**

* **Objetivo**: Poner en marcha el software para su uso en producción.
* **Actividades**:
  + **Despliegue del backend** en un servidor utilizando **Docker**.
  + **Despliegue del frontend** (React) en un servidor o plataforma de hosting.
  + Configuración de la base de datos **PostgreSQL** en el servidor.
  + Documentación técnica: Incluir instrucciones detalladas sobre cómo desplegar y mantener la aplicación.
  + **Publicación de las apps móviles**: Subir las aplicaciones de React Native a las tiendas (App Store y Google Play).

**6. Fase de Mantenimiento**

* **Objetivo**: Garantizar 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**: Asegurar que los datos de los usuarios y los pagos estén protegidos adecuadamente.

**Resumen del plan de desarrollo:**

Este plan cubre todas las fases del desarrollo, desde la recogida de requerimientos hasta el mantenimiento del software. Se utilizarán **React** para la gestión web, **React Native** para las apps móviles, **Node.js** con **Express** para el backend, **PostgreSQL** para la base de datos y **RedSys** para los pagos. El uso de **Docker** permitirá gestionar el despliegue eficiente en diferentes entornos, cumpliendo así con los requisitos de software libre y las mejores prácticas de desarrollo.

**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.