

Lectura 9. Gestión del desarrollo de software: Product Backlog y Producto Mínimo Viable (MVP)

1. ¿Qué es el Product Backlog?

En el marco de trabajo Scrum, el **Product Backlog** (Pila del Producto) es la fuente única de verdad y el inventario centralizado de todo lo que podría requerir el sistema. No es un listado cerrado ni un manual inamovible; es un artefacto **vivo y dinámico** gestionado primordialmente por el Product Owner, donde los requerimientos cambian, se refinan, se dividen o se descartan conforme se obtiene retroalimentación continua del usuario.

Orden por Valor de Negocio	Detalle Progresivo (Refinamiento)
Se organiza de arriba hacia abajo por utilidad real para el usuario; lo más valioso y crítico se programa primero.	Lo que está en la cima tiene historias detalladas y listas; el fondo conserva ideas generales por madurar.

1.1 ¿Qué es el Valor de Negocio en un desarrollo ágil?

El valor de negocio representa el beneficio directo, impacto funcional o utilidad práctica que una característica aporta al usuario y a los objetivos del proyecto. En proyectos formativos y de impacto social como *Nãtho Digital*, el valor no se mide en ingresos económicos, sino en:

Preservación Cultural	Inclusión y Acceso	Mitigación Técnica
Audios reales y fonética otomí correcta; sin esto la app pierde su propósito.	Navegación intuitiva y directa para toda la comunidad escolar.	Construir primero la arquitectura base antes de detalles cosméticos.

2. ¿Qué es el Producto Mínimo Viable (MVP)?

El **Producto Mínimo Viable (MVP)** es la versión funcional más temprana del sistema construida con las características indispensables para resolver el problema central. No es un software a medias ni defectuoso: es una solución 100% operativa que permite validar el producto con usuarios reales en el menor tiempo posible.

- **Propósito ágil:** Aprender de la retroalimentación directa de la comunidad antes de invertir tiempo y recursos en funciones complejas o decorativas.
- **Línea de corte:** Es el límite formal en el Product Backlog que separa las funcionalidades esenciales de la primera entrega de aquellas mejoras que se posponen para versiones posteriores.

Ejemplo de delimitación de alcance en Ñätho Digital

Nivel de Valor	Funcionalidad / Historia de Usuario	Estado de Entrega	Criterio de Decisión
Crítico	Catálogo básico de términos otomí-español.	Dentro del MVP (v1.0)	Resuelve la necesidad base; sin vocabulario la plataforma no existe.
Crítico	Reproductor de audio con pronunciación de voz nativa.	Dentro del MVP (v1.0)	Indispensable para salvaguardar la fonética exacta.
Medio	Ajustes visuales de interfaz (modo oscuro y tipografía).	Fuera del MVP (v1.1)	Mejora la comodidad, pero no detiene la consulta escolar.
Bajo	Minijuego interactivo de memoria y emparejamiento.	Fuera del MVP (v2.0)	Función lúdica complementaria que excede el tiempo del corte inicial.

Act013. Orden preliminar de 5 Historias de Usuario por valor de negocio + propuesta individual de MVP

Instrucciones:

1. Consulta el catálogo de **Historias de Usuario de Ñätho Digital** trabajado en la sesión anterior y elige **5 historias clave** para el proyecto.
2. Ordénalas del **1 al 5** considerando el **valor de negocio**. Coloca en los puestos **1, 2 y 3** las que sean indispensables para que la aplicación funcione en su versión más básica, por ejemplo, ver palabras o escuchar la voz nativa. En los puestos **4 y 5**, coloca las que sean adornos o funciones secundarias.
3. Dibuja una **línea horizontal gruesa** que separe las historias que formarán parte del **MVP v1.0** de las que se dejarán para después (**Fuera del MVP**).
4. En **3 renglones**, explica por qué no se puede planear el software comenzando por las funciones secundarias y por qué **esta priorización** permite realizar una entrega funcional a tiempo.

Act014. Integración del Expediente Final de Planeación Ágil (Fase 1: Ideación)

Instrucciones: Durante la sesión presencial de cierre y bajo la orientación directa del docente, integra y organiza en tu carpeta académica el compendio completo del Primer Parcial.

Act015. Cuadro de adaptación de SCRUM para Ñätho Digital

Instrucciones: Revisa el material proporcionado sobre **SCRUM** y, con base en lo aprendido, realiza una propuesta de cómo podría aplicarse al proyecto **Ñätho Digital**.

Para elaborar tu propuesta, considera:

- Principios y roles de SCRUM.
- Eventos de SCRUM.
- Duración de los Sprints.
- Tamaño y organización del equipo.
- Funciones del **Product Owner, Scrum Master y equipo de desarrollo**.
- Adaptación de estos elementos a las necesidades de **Ñätho Digital**.

Presenta tus propuestas en un **cuadro de adaptación de SCRUM**, indicando qué es cada elemento y **cómo podría aplicarse al proyecto**.

La propuesta será revisada en la **sesión presencial**, donde se analizarán las diferentes ideas y se tomarán decisiones sobre la organización de SCRUM para Ñätho Digital.

No es necesario definir una propuesta definitiva. La finalidad es llegar a la sesión presencial con una propuesta inicial para trabajar a partir de ella.

Act. 016. Transformación preliminar de 2 Historias de Usuario a criterios Given–When–Then

1. Elige 2 Historias de Usuario prioritarias del Product Backlog de *Ñätho Digital*.
2. Para cada Historia de Usuario, escribe sus criterios de aceptación utilizando la estructura Given–When–Then:
 - **Dado (Given):** ¿En qué situación se encuentra el usuario o el sistema antes de realizar la acción?
 - **Cuando (When):** ¿Qué acción realiza el usuario?
 - **Entonces (Then):** ¿Qué debe suceder después de realizar esa acción?
3. Asegúrate de que cada criterio pueda **comprobarse fácilmente**, de manera que sea posible saber si la función funciona correctamente o no.

Guíate con el siguiente ejemplo resuelto:

Historia de Usuario:

HU-02. Reproducir audio de pronunciación.

- **Dado:** Que el usuario está dentro de la pantalla de “Saludos” y el volumen del dispositivo está activo.
 - **Cuando:** Presiona el botón del altavoz junto a la palabra en otomí.
 - **Entonces:** La aplicación reproduce el audio correspondiente y el botón cambia de color mientras dura el sonido.
4. Realiza lo mismo con las 2 Historias de Usuario que elegiste y presenta tus criterios para su revisión.

No se trata solamente de explicar qué debería hacer la función. Debes indicar qué debe pasar para poder comprobar que funciona correctamente.

Lectura 10. Refinamiento del Backlog, Criterios de Aceptación y DoD

1. Refinamiento del Backlog

Es la revisión continua en la que el equipo y el Product Owner revisan las Historias de Usuario (HU), eliminan duplicidades, corrigen ambigüedades y las ordenan por prioridad. Cada historia debe contar con criterios objetivos para determinar cuándo está completa

Criterios de Aceptación	Definition of Done (DoD)
Condiciones específicas y comprobables que debe cumplir una Historia de Usuario particular para ser aceptada.	Conjunto de criterios de calidad transversales que debe cumplir cualquier entrega para considerarse lista (código limpio, probado y funcional).

2. Formato BDD: Criterios Given – When – Then (Dado – Cuando – Entonces)

Estructura estándar para definir pruebas y comportamientos comprobables:

- ❖ **Given (Dado):** El contexto o precondition inicial del sistema.
- ❖ **When (Cuando):** La acción concreta que realiza el usuario en la aplicación.
- ❖ **Then (Entonces):** La respuesta observable y comprobable del sistema.

Act. 017. Ejercicio preliminar de estimación de velocidad y distribución de Historias de Usuario

Instrucciones:

Realiza un ejercicio preliminar para organizar las Historias de Usuario de Ñätho Digital.

Considera:

- La capacidad de trabajo del equipo.
- Los Story Points de las Historias de Usuario.
- La duración que podría tener el Sprint.
- La distribución de las Historias de Usuario.
- El Sprint Goal que se desea alcanzar.
- Un cronograma ágil para organizar las actividades.

Realiza una propuesta inicial y llévala a la sesión presencial para revisarla y realizar los ajustes necesarios.

Lectura 11. ¿Cómo planear el trabajo en SCRUM sin morir en el intento?

Planear software no es adivinar el futuro: es calcular las fuerzas reales del equipo para entregar código funcional antes de que venza el plazo.



Act018. Exploración de tablero ágil

Instrucciones:

- Revisa el ejemplo de tablero ágil proporcionado y observa cómo se organizan las actividades del proyecto.
- Después, realiza una propuesta preliminar de las columnas que podrían utilizarse para organizar el trabajo de *Nātho Digital*.
- Considera cómo podrían pasar las actividades de una etapa a otra durante el desarrollo del proyecto.
- Lleva tu propuesta a la sesión presencial, donde se revisarán las diferentes propuestas y se decidirá de manera colaborativa cómo quedará organizado el tablero.

Nota: En esta actividad no es necesario configurar el tablero definitivo.

Act. 019. Descomposición Técnica de una Historia de Usuario

Instrucciones: Selecciona una **Historia de Usuario del Sprint 1** de *Ñätho Digital* y analiza qué actividades son necesarias para cumplir con ella. Posteriormente, descompón la Historia de Usuario en un rango de **3 a 5 tareas concretas**.

Para realizar la descomposición, considera:

- ¿Qué actividades son necesarias para completar la Historia de Usuario?
- ¿Qué pasos deben realizarse para cumplir con el objetivo planteado?
- ¿Las tareas propuestas son claras y específicas?
- ¿Las tareas permiten completar la Historia de Usuario en conjunto?

Elabora una tabla que contenga la Historia de Usuario seleccionada, las **3 a 5 tareas** propuestas y una breve descripción de cada tarea..

Lectura 12. De la Historia a la Realidad: Descomposición de Tareas Técnicas

En el marco de trabajo Scrum, el Product Backlog se compone de Historias de Usuario orientadas a resolver necesidades del cliente. Sin embargo, los desarrolladores no programan directamente una historia completa en un solo bloque; primero la descomponen en unidades operativas llamadas **tareas técnicas**.

1. ¿Qué es una Tarea Técnica?

Una tarea técnica es una acción específica, medible y de corta duración que un desarrollador ejecuta para construir una parte concreta de una Historia de Usuario.

- Representa el trabajo real de ingeniería (diseñar una vista, crear un endpoint, escribir una consulta SQL o hacer un test).
- Suele resolverse en cuestión de horas dentro de un mismo Sprint, no en semanas.

3. Diferencia entre Historia de Usuario y Tarea Técnica

Criterio	Historia de Usuario (HU)	Tarea Técnica
Enfoque	En el QUÉ y el PARA QUÉ (Visión del usuario/negocio).	En el CÓMO (Visión técnica del programador).
Redacción	En lenguaje natural: « <i>Como... quiero... para...</i> ».	En verbo de acción técnica: « <i>Diseñar...</i> », « <i>Configurar...</i> ».
Responsable	Pertenece al equipo completo (Scrum Team).	Se asigna a un integrante individual.
Entrega de valor	Entrega valor visible de negocio o funcionalidad al cliente.	Por sí sola no es un entregable final; es un paso constructivo.

2. Capas Técnicas de Desarrollo

Al descomponer una historia, las tareas suelen dividirse según la capa del sistema en la que impactan:

Frontend: Construcción de interfaces gráficas, estilos visuales y captura de eventos en pantalla.

Backend:

Lógica interna del sistema, manejo de archivos de audio y procesamiento de datos.

Base de Datos:

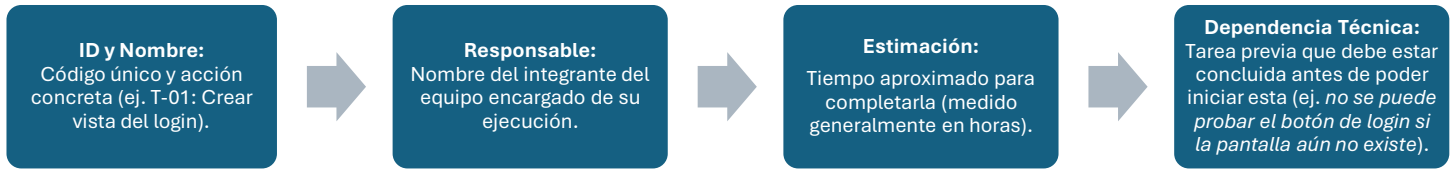
Almacenamiento local o persistente de vocabulario otomí, traducciones y rutas de audio.

Pruebas (QA):

Validación en emuladores y dispositivos reales para asegurar que no existan fallos de ejecución.

4. Elementos Clave de una Tarea Técnica

Para que una tarea esté bien definida en el tablero ágil, debe incluir:



5. Ejemplos de Descomposición Técnica

Ejemplo 1: Reproducción de Audio

Historia de Usuario: «Como estudiante, quiero presionar un botón para escuchar la pronunciación en audio de una palabra y aprender su sonido correcto».

ID	Capa técnica	Tarea propuesta	Responsable	Estimación	Dependencia
T-01	Frontend	Diseñar la interfaz visual con el botón e icono de audio.	Alumno 1	2 h	Ninguna
T-02	Base de Datos / Storage	Almacenar y vincular los archivos de audio en la carpeta local.	Alumno 2	1 h	Ninguna
T-03	Backend	Programar el controlador que reproduce la pista al hacer clic.	Alumno 3	3 h	T-01, T-02
T-04	Pruebas	Probar la reproducción continua y medir tiempos de respuesta.	Alumno 4	2 h	T-03

Ejemplo 2: Catalogo de Lecciones

Historia de Usuario: «Como alumno, quiero ver una lista de lecciones ordenadas por nivel para elegir qué tema estudiar».

ID	Capa técnica	Tarea propuesta	Responsable	Estimación	Dependencia
T-01	Base de Datos	Crear la tabla Lecciones con campos de título, nivel y descripción.	Alumno 2	2 h	Ninguna
T-02	Frontend	Maquetar la pantalla con cuadrícula de tarjetas de lecciones.	Alumno 1	3 h	Ninguna
T-03	Backend	Desarrollar la consulta lógica para filtrar las lecciones por nivel.	Alumno 3	2 h	T-01
T-04	Pruebas	Comprobar que la vista cargue correctamente los datos de la base.	Alumno 4	1 h	T-02, T-03

Act. 020. Propuesta tecnológica preliminar

Instrucciones: Selecciona los componentes tecnológicos que consideres adecuados para el desarrollo de *Ñátho Digital*. Define una propuesta para cada uno de los siguientes elementos:

- **Frontend**
- **Backend**
- **Base de datos**
- **Hosting**
- **Herramientas de desarrollo**

Para cada elemento, indica la tecnología o herramienta propuesta y escribe una justificación breve explicando por qué consideras que es adecuada para el proyecto. Presenta tu propuesta en una tabla que incluya el componente, la tecnología o herramienta seleccionada y su justificación.

Para realizar la propuesta, considera:

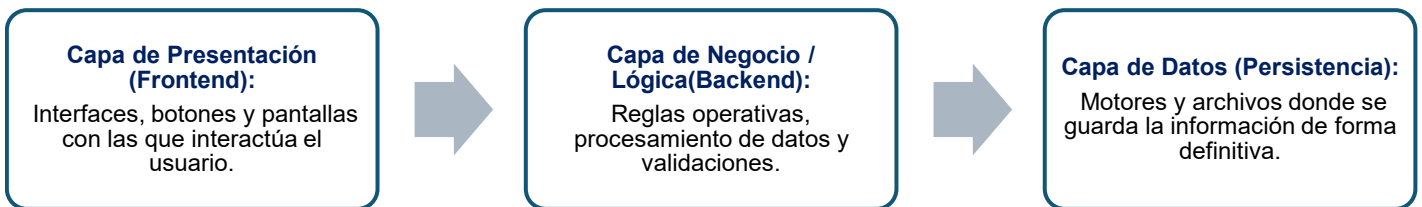
- ¿Qué necesidades del proyecto debe cubrir la tecnología seleccionada?
- ¿La tecnología es adecuada para el tipo de aplicación que se desarrollará?
- ¿Cómo se relaciona con los demás componentes de la arquitectura?
- ¿La herramienta es viable de acuerdo con los recursos y conocimientos disponibles?

Lectura 13. Arquitectura de Software y Selección Tecnológica

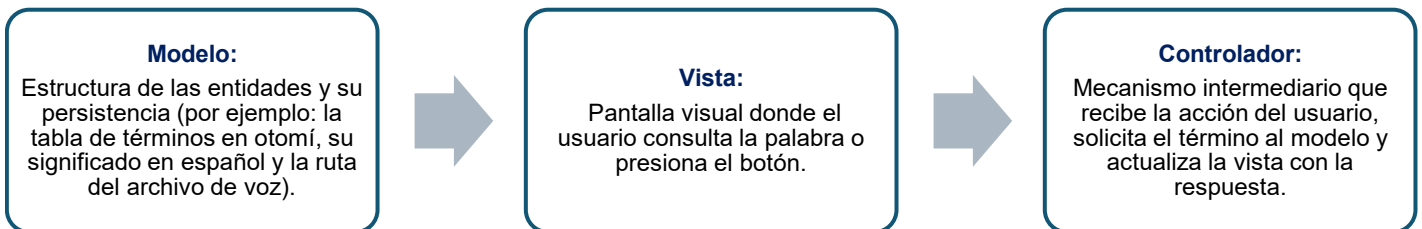
Planear software no consiste únicamente en estimar historias y organizar tableros; requiere definir con exactitud los cimientos técnicos del sistema antes de comenzar la programación. La arquitectura de software establece cómo interactúan los módulos, dónde residen los datos y qué tecnologías garantizan que el Producto Mínimo Viable (MVP) sea funcional, escalable y mantenible..

1. Arquitectura por Capas y Modelo MVC

Arquitectura por capas: Divide el proyecto en niveles independientes. La capa superior consume los servicios de la inferior sin preocuparse por cómo está implementada por dentro:



Modelo - Vista - Controlador (MVC): Patrón que organiza el código para desacoplar la interfaz de la lógica:



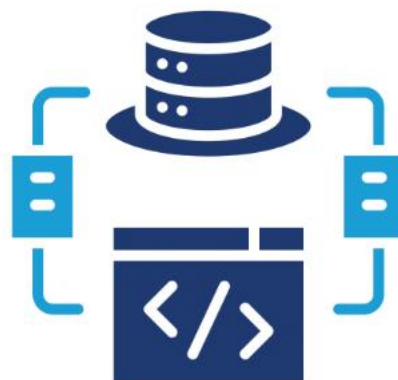
2. Componentes Clave del Sistema

Frontend:

Es la interfaz visual con la que interactúa el usuario de forma directa. Controla el diseño, la responsividad y los eventos como clics o audios.

Backend:

Es el motor lógico que procesa peticiones en el servidor mediante APIs. Valida la seguridad del sistema y coordina el flujo de datos.



Bases de datos:

Almacena y gestiona de forma centralizada toda la información. Usa tablas ordenadas en **SQL** o documentos flexibles tipo JSON en **NoSQL**.

Frameworks

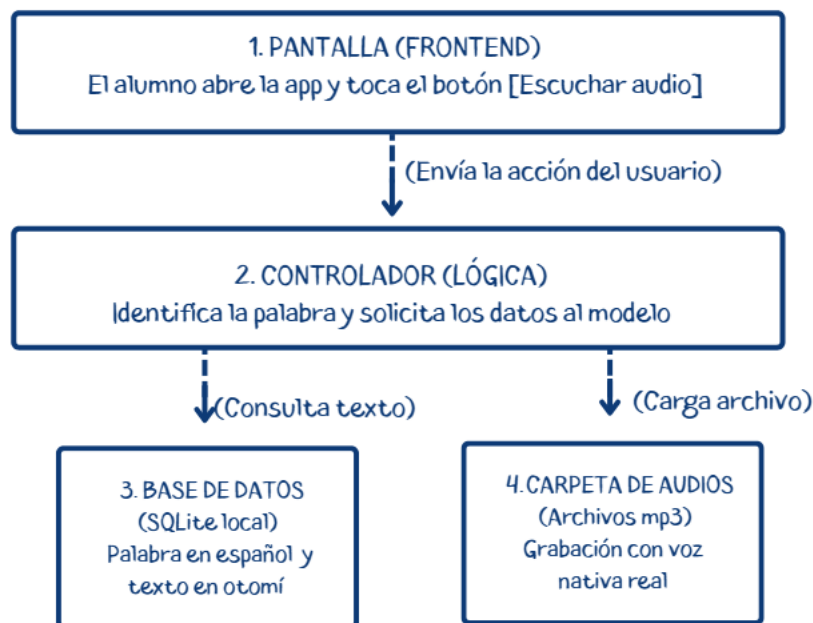
Son estructuras de código prefabricado que agilizan el desarrollo. Ahorran trabajo repetitivo y garantizan buenas prácticas de seguridad.

3. Hosting, Servicios y Despliegue

En la planeación del Sprint debe definirse con antelación el entorno donde vivirá la solución:

- **Hosting:** Espacio en servidor físico o en la nube donde residen los archivos del backend y los servicios web (ej. Render, servidores dedicados o contenedores).
- **Almacenamiento multimedia:** Repositorio optimizado para alojar archivos de peso considerable (audios nativos en formato .mp3 o imágenes de apoyo) sin saturar la base de datos principal.
- **Herramientas de despliegue:** Mecanismos que permiten empaquetar y entregar el software al usuario final, ya sea mediante la compilación directa de un instalador móvil o la publicación en servidores web accesibles en el plantel.

4. Ejemplo Práctico de Arquitectura: *Nãtho Digital*



¿Cómo viaja la información en este esquema?

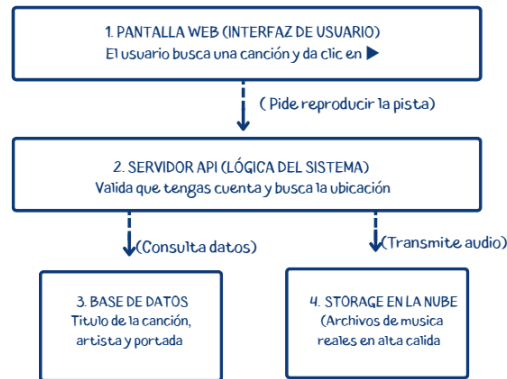
- 1. En la pantalla (Frontend):** El usuario toca el botón de reproducción. La interfaz solo captura el toque y envía la señal.
- 2. En la lógica (Controlador):** El sistema recibe la acción, identifica cuál palabra fue seleccionada y solicita sus recursos correspondientes.
- 3. En la Base de Datos (SQLite):** Entrega el texto escrito con la palabra en otomí y su traducción al español.
- 4. En la Carpeta de Audios:** Suministra el archivo de sonido .mp3 grabado previamente por hablantes nativos.

Resultado: El controlador reproduce el sonido en el dispositivo y actualiza la pantalla, todo de forma local y sin depender de conexión a internet.

Ejemplos Sencillos de Arquitecturas de Aplicaciones Web

1. Plataforma de Streaming de Música

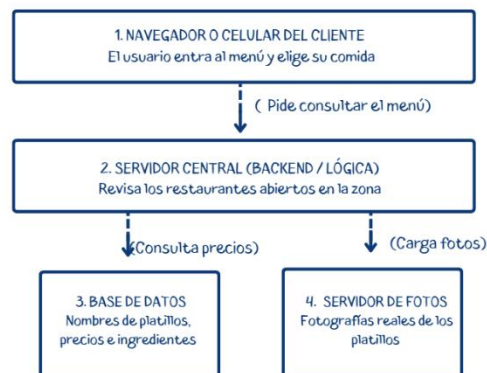
Cuando buscas una canción y le das *play*, la página web nunca se recarga ni parpadea: la música empieza a sonar al instante porque solo viaja el archivo de sonido por internet.



1. **Pantalla:** Registra el clic en el botón de reproducir sin congelar ni reiniciar la página.
2. **Servidor:** Revisa que el usuario tenga acceso y localiza en qué servidor del mundo está guardada la canción.
3. **Base de datos:** Envía el título de la canción, el nombre del artista y la carátula.
4. **Storage en la nube:** Transmite el archivo de audio de alta calidad directo a los audífonos.

2. App Web de Entrega de Comida

Al ordenar una hamburguesa o unos tacos, la pantalla te muestra el platillo con su precio exacto y la fotografía del restaurante sin demoras.



1. **Pantalla del cliente:** Muestra el catálogo visualmente para que el usuario seleccione su pedido.
2. **Servidor central:** Calcula costos de envío, revisa qué comercios están abiertos y coordina los datos.
3. **Base de datos:** Entrega la lista de platillos, precios, ingredientes y comercios disponibles.
4. **Servidor de fotos:** Carga las fotografías reales de los platillos para mostrarlas en el menú.

Act. 021. Lista de verificación para revisión del documento + esquema preliminar del pitch.

1. Revisa tu documento de planificación y verifica que cumpla con las normas de presentación profesional.

Elemento	Regla formal	¿Cómo debe verse?
Márgenes	2.5 cm por lado	Espacio parejo en los 4 bordes de la hoja.
Texto	Arial o Calibri (11 pt)	Párrafos justificados con interlineado a 1.15.
Títulos	Negrita y tamaño gradual	Título principal en 16 pt; subtítulos en 13 pt.
Gráficos	Centrados y rotulados	Cada tabla o figura lleva su nombre (<i>Figura 1</i>).
Páginas	Numeradas	En la esquina inferior derecha (la portada no lleva).

2. Analiza ejemplos de documentos de planificación e identifica los elementos que pueden servir como referencia para mejorar tu trabajo.
3. Revisa que el índice automatizado esté correctamente estructurado y que los títulos y números de página correspondan con el contenido del documento.
4. Elabora una lista de verificación con los aspectos revisados y señala aquellos que requieren alguna corrección.
5. Finalmente, realiza un **esquema preliminar de tu pitch**, identificando los principales puntos que presentarás sobre tu proyecto.
 - **El Problema (Contexto real)**
 - **La Solución (MVP - Producto Mínimo Viable)**
 - **Arquitectura Técnica (Frontend, Backend, Base de Datos y Storage)**
 - **Planificación del Desarrollo (Metodología Scrum y Sprints)**
 - **Cierre e Impacto (Beneficio y preguntas)**

Act022. Guion preliminar del pitch ejecutivo

Instrucciones: Investiga y revisa las principales características de un **pitch ejecutivo**, considerando técnicas de presentación, storytelling y comunicación clara de ideas.

1. Analiza cómo presentar un proyecto de manera breve, ordenada y atractiva, identificando la información más importante que debe comunicarse.
2. Revisa ejemplos de presentaciones profesionales e identifica elementos que puedan servir como referencia para tu propio pitch.
3. Considera posibles preguntas que podrían realizarte durante la presentación y prepara respuestas relacionadas con tu proyecto.
4. Elabora un **guion preliminar** para tu pitch, organizando las ideas que desarrollarás durante la presentación.
5. Distribuye el contenido de manera que la presentación tenga una duración máxima de **10 minutos**.
6. Revisa tu guion y realiza los ajustes necesarios a partir de la retroalimentación recibida.

Lectura 14. Comunicación Ejecutiva, Storytelling y Presentación Ágil

La calidad de un proyecto de software no depende únicamente de que el código funcione; también reside en la capacidad técnica para documentarlo con rigor profesional y defenderlo ante evaluadores o clientes.

1. Normas Básicas de Estilo y Corrección de Documentos

La formalidad editorial garantiza que el lector técnico se concentre en la solución y no en deficiencias visuales:

- ✓ **Consistencia visual:** Emplear una tipografía única y legible (ej. Calibri o Arial de 11 pt) con interlineado de 1.15 y márgenes estándar de 2.5 cm.
- ✓ **Jerarquía clara:** Diferenciar títulos y subtítulos por tamaño y peso visual en negrita, manteniendo el texto de los párrafos siempre justificado.
- ✓ **Control de tecnicismos:** En ingeniería de software es común el uso de anglicismos (*frontend*, *backend*, *database*, *storage*, *sprint*). Es obligatorio configurar el corrector del procesador de textos seleccionando «Omitir todo» para evitar que el documento final muestre líneas onduladas rojas de error ortográfico.

3. Redacción Ejecutiva

La redacción ejecutiva prioriza la precisión sobre el volumen. El evaluador busca datos concretos expresados con economía verbal:

- ✓ **Directo al grano:** Eliminar introducciones vacías o reflexiones informales. Se explica directamente qué problema existe y qué acción ejecuta el software.
- ✓ **Voz activa y datos verificables:** En lugar de redactar «*se intentará hacer que los usuarios puedan ver...*», redactar «*el sistema despliega el catálogo en menos de un segundo*».
- ✓ **Estructuración:** Usar viñetas cortas para listas de requerimientos y tablas para contrastar datos o flujos de trabajo.

2. Elementos de un Pitch Ejecutivo

Un pitch no es una clase teórica, sino una exposición de alto impacto estructurada en cinco momentos esenciales:

Momento	Propósito	Contenido clave
1. El Problema	Contextualizar	La necesidad real y el impacto negativo de no resolverla.
2. La Solución (MVP)	Demostrar valor	Nombre de la herramienta y sus 2 o 3 funciones clave.
3. Arquitectura Técnica	Sustentar viabilidad	Flujo de datos desacoplado (Pantalla, Servidor, BD, Storage).
4. Planificación Ágil	Mostrar control	Historias de usuario, criterios de aceptación y entregas por Sprint.
5. Cierre	Impacto	El beneficio directo para la comunidad y apertura a preguntas.

5. Manejo de Preguntas y Retroalimentación

La sesión de preguntas mide la seguridad técnica y el dominio del trabajo realizado en equipo:

- ✓ **Pausa reflexiva:** Espera a que el evaluador concluya la pregunta completa. Tomar dos segundos antes de hablar demuestra serenidad y evita interrupciones.
- ✓ **Método directo (Hecho → Motivo):** Responde con la decisión técnica tomada y añade inmediatamente el porqué de ingeniería que la justifica.
- ✓ **Recepción profesional de críticas:** Jamás inventes respuestas ni entres en confrontación. Si el jurado señala una falla o función faltante, la respuesta correcta es: «Es una observación muy valiosa; no formó parte del alcance de este MVP, pero queda registrada formalmente en el backlog para la siguiente iteración».

Act023. Entornos de desarrollo, Git/GitHub,

Instrucciones: Revisa el material proporcionado por la docente sobre entornos de desarrollo, Git y GitHub, estructura de carpetas, ramas y guías de estilo. Identifica los conceptos principales y analiza cómo se aplican en un proyecto de desarrollo como *Ñätho Digital*.

Lectura 15: Control de Versiones, Entorno de Desarrollo y Estructura del Proyecto

El desarrollo colaborativo de software requiere herramientas que permitan escribir código de forma eficiente, rastrear cambios históricos sin sobrescribir el trabajo de otros compañeros y mantener los archivos del sistema bajo un orden estandarizado.

1. Entorno de Desarrollo y Control de Versiones

¿Qué es un IDE (Entorno de Desarrollo Integrado)?:

Es una aplicación que reúne en un solo lugar las herramientas esenciales para programar: editor de código fuente con resaltado de sintaxis, autocompletado, depurador de errores (*debugger*) y terminal de comandos integrada (ej. Visual Studio Code). Su función es agilizar la escritura de software sin alternar entre múltiples programas externos.

¿Qué es Git y para qué sirve?:

Git es un **sistema de control de versiones distribuido** que se instala localmente en la computadora. Registra el historial completo de cambios realizados en los archivos de un proyecto a lo largo del tiempo, permitiendo revertir errores, saber quién modificó cada línea de código y trabajar en paralelo sin perder avances.

Diferencia entre Git y GitHub:

Git	GitHub
Software de control de versiones local.	Plataforma en la nube basada en web.
Se ejecuta en la máquina de cada desarrollador.	Se accede mediante navegador web o API.
Registrar versiones, crear ramas y guardar <i>commits</i> .	Alojar repositorios remotos, coordinar equipos y respaldar código.
Funciona 100% sin internet.	Requiere internet para sincronizar datos.

Repositorio y Clonación:

- **Repositorio:** Carpeta raíz administrada por Git que almacena los archivos del proyecto junto con todo el historial de modificaciones en una carpeta oculta (.git)
- **Clonación (git clone):** Acción de descargar una copia idéntica y completa de un repositorio remoto de GitHub hacia la computadora local, incluyendo todas sus ramas e historial.

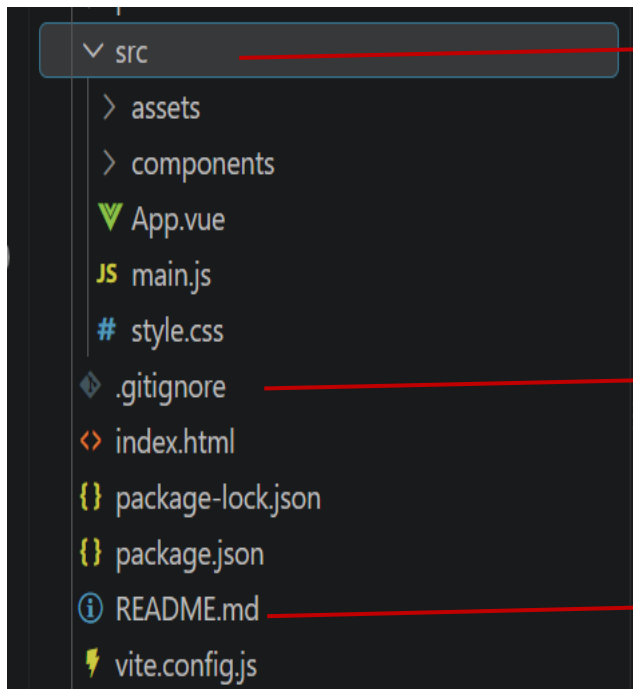
2. Estrategia de Ramas (Git Flow)

Las ramas (*branches*) son líneas de tiempo paralelas que permiten crear funciones nuevas sin alterar la versión estable del software:

- **Rama main (o master):** Contiene exclusivamente el código listo para producción, completamente probado, estable y sin fallas. Nunca se programa directamente sobre ella.
- **Rama develop:** Rama de integración donde se unen las funcionalidades terminadas del Sprint. Es la base sobre la que se preparan las versiones de entrega.
- **Ramas feature:** Ramas temporales creadas para desarrollar una tarea técnica o historia de usuario específica (ej. feature/login-alumno o feature/audio-player). Nacen a partir de develop y se eliminan tras fusionarse (*merge*) al completar la funcionalidad.

3. Estructura Básica de Carpetas de un Proyecto Web

Un proyecto ordenado separa claramente los módulos del sistema, la lógica de negocio y los archivos públicos:



Aquí vive todo el sistema, ordenado en:

- **assets/ (El cajón de fotos y audios):** Todo el material visual o auditivo (imágenes, logos y audios .mp3).
- **components/ (La fachada y muebles - Frontend):** Lo que el usuario ve y toca (botones, barras de menú y tarjetas).

➤ **.gitignore(Lo privado):** Una lista de cosas que no quieres que nadie vea ni se lleve (archivos temporales o cosas muy pesadas)

➤ **README.md (Manual de bienvenida):** La nota en la entrada que explica qué es el proyecto y cómo ponerlo a funcionar paso a paso.

Act. 024. Identificación de pruebas unitarias y refactorización de código

Instrucciones:

1. Selecciona una **Historia de Usuario** de *Ñátho Digital*.
2. Identifica una funcionalidad que pueda comprobarse mediante una **prueba unitaria**.
3. Escribe un ejemplo de **dato de entrada** y el **resultado esperado**.
4. Revisa el código relacionado con esa funcionalidad o escribe un código preliminar.
5. Propón una **mejora al código**, sin cambiar el resultado que debe producir.
6. Explica brevemente por qué tu propuesta mejora el código.

Presenta tu trabajo en una tabla que incluya:

- **Historia de Usuario**
- **Funcionalidad**
- **Dato de entrada**
- **Resultado esperado**
- **Código inicial**
- **Propuesta de mejora y justificación.**

