**Desarrolladores:** Andrea Osorio y Gabriel Jimenez.

**Agenda inteligente de gestor de tiempo**

**Descripción de producto de software**

**Atributos para describir el software**

**Funcionalidades básicas de la app web**

* Mostrar calendario. En el calendario, se podrán visualizar las tareas y eventos registrados ¿Cómo se registra? De la siguiente manera:
* Crear horario (Motor de recomendaciones): El usuario completa el cuestionario y la app **hace** el horario el cual es tentativo.
  + En el horario, están identificados los sprints de trabajo, tomando en cuenta las horas.

**Nota para los desarrolladores:** En principio se trabajará en los días del horario únicamente.

* Hace seguimiento de tareas.
  + Predeterminado: Hace recordatorios. El usuario puede cambiar eso, agregar, modificar o quitar recordatorios si así lo prefiere.
* Evaluar prioridades en el backend.
  + Atributos de las tareas:
    1. Nivel de prioridad (Selección manual): Alta, media y baja.

Se usará una nomenclatura de días, semanas y meses. Años **NO.**

* + 1. Fecha límite.
    2. Plazo de tiempo:

**Frontend:** Habrá selección múltiple. El usuario escogerá por su cuenta MANUALMENTE los plazos de tiempo, que si una semana a meses.

**Backend:** El código guardará la respuesta y lo meterá en un paquete de plazo de tiempo:

* Corto (de horas hasta máximo tres semanas).
* Largo plazo (después de tres semanas a un año).
* Hacer cuestionarios para sacar respuestas.
  + Ofrecer técnicas de gestión de tiempo automáticas basadas en las respuestas del cuestionario.
  + Ofrecer recomendaciones automáticas basándose en los tiempos libres, atributos de las tareas y las fechas límites que seleccione el usuario.

ORDENAR

**Planificación**

**Problemática que busca resolver.**

Muchas personas, tanto profesionales como estudiantes, se enfrentan a la dificultad de gestionar su tiempo de manera eficiente. Esta carencia se traduce en agendas saturadas, plazos incumplidos, aumento del estrés y una baja productividad general. La aplicación busca solucionar este problema mediante una herramienta que centralice la gestión de tareas, eventos y recordatorios, respondiendo a la necesidad de optimizar el tiempo a través de recomendaciones personalizadas y la integración de técnicas de gestión del tiempo basadas en respuestas de un cuestionario. ✔

**Definir los objetivos de la aplicación**

El principal objetivo de la aplicación es facilitar la gestión integral del tiempo, permitiendo que los usuarios organicen sus tareas y eventos de manera automática y personalizada. Entre los objetivos específicos se encuentran:

* Automatización de la planificación: Generar horarios tentativos y sugerencias de sprints de trabajo basados en la prioridad, fecha límite y el plazo de las tareas.
* Optimización de la productividad: Facilitar la organización mediante recordatorios y notificaciones personalizables, asegurando que los usuarios tengan un seguimiento detallado de sus compromisos (tareas).
* Seguridad y usabilidad (login): Garantizar un acceso seguro a la plataforma con una interfaz intuitiva y responsiva, compatible con diversos dispositivos y navegadores.

**Identificar a los usuarios y sus necesidades**

La aplicación está dirigida a:

* Profesionales en entornos laborales dinámicos.
* Estudiantes.
* Emprendedores y freelancers (gente que trabaja por su cuenta y que no tiene un jefe de trabajo).

**Establecer alcance y limitaciones del proyecto**

***Alcance***

* Se desarrollará una versión preliminar de la interfaz primero.
* Se interactuará con la plataforma web durante su desarrollo.
* Funcionalidades básicas de organización: Incluirá la creación, edición y eliminación de tareas y eventos, junto al motor de recomendaciones.
* Cuestionario interactivo: Se implementará un cuestionario para recopilar información sobre el estilo de gestión del tiempo del usuario,
* Seguridad de la información: Se garantizará el almacenamiento seguro y encriptado de las credenciales y datos del usuario.
  + Render: Almacenamiento en la nube.
  + Base de Datos con datos encriptados.

***Limitaciones (Nota para los desarrolladores)***

* No se creará la opción de la personalización de la interfaz de usuario.
* No se incluirá la interactividad con el calendario.
* En principio se trabajará en los días del horario únicamente, excluyendo las horas de los sprints de trabajo.
* Se excluye la configuración detallada de la app: idioma, tamaño de fuente, paleta de colores.
* Se excluye la integración con dispositivos móviles y otros dispositivos externos quedará fuera del alcance de la primera versión.
* No se implementarán las recomendaciones (recordatorios automáticos) basadas en las respuestas del usuario.

**Cronograma del Proyecto**

***Fases del desarrollo***

1. Planificación.
2. Análisis y diseño.
3. Desarrollo.
4. Pruebas.
5. Despliegue.
6. Mantenimiento y actualización.

***Sprints de trabajo***

Periodos definidos para completar tareas específicas del desarrollo.

* **Sesiones de avance:** Reuniones o revisiones periódicas para evaluar el progreso y ajustar estrategias.
  + Días de la semana: Miércoles, Jueves y Sábado.
  + Hora: 8:30am a 11am o en la tarde.
* **Fechas clave:** Plazos estimados para la entrega de cada fase y objetivos importantes.
  + **Planificación:** 26/04 a 10/05/2025.
    - Se definen las funcionalidades básicas de la app, sus objetivos, problemática a resolver, sus requerimientos, diseños preliminares de las interfaces en papel, la estructura general de la arquitectura de software y sus componentes clave.
  + **Análisis y diseño:** 11/05 a 18/05/2025.
    - Definición final de los requerimientos de manera detallada, desarrollo básico de los diagramas de flujo, estructura básica de la app, bocetos digitales y validación de la Arquitectura de Software.
  + **Desarrollo (Codificación):** 19/05 a 21/06/2025 (5 Semanas).
    - Codificar las funcionalidades principales e implementar las interfaces de usuario.
  + **Pruebas:** 22/06 a 28/06 (1 Semana).
    - Elaborar una planilla de verificación de software, ejecutar pruebas funcionales, de integración y usabilidad utilizando la planilla como guía para depurar errores y validar el desempeño de la aplicación previo a su lanzamiento.
  + **Despliegue:** 29/06 a 04/07/2025 (1 Semana).
    - Afinar la transición a producción, realizar ajustes finales y preparar el entorno para el lanzamiento.
  + **Mantenimiento y actualización:** Desde el 05/07/2025 en adelante.
    - Esta fase es continua, para corregir errores iniciales, atender feedback y realizar mejoras conforme se use la aplicación.

**Identificación de Riesgos**

* **Técnicos: Fallos en infraestructura, incompatibilidad de tecnologías.**
  + Incompatibilidad de versiones de Python, Windsurf y Django.

**Soluciones:** Que los desarrolladores instalen las mismas versiones de las tecnologías. Python 3.12 y Django 5.1.7.

* Pérdida de datos si se usa una base de datos SQLite3 en el servidor seleccionado (Render).

**Soluciones:** Usar una base de datos que no tenga riesgo de pérdida de datos. PostgreSQL es la opción recomendada.

* Fallo en el IDE de una laptop, lo que dificulta el proceso de codificación.

**Soluciones:** Usar otro equipo para la codificación, en este caso la PC del desarrollador.

* Falta de organización en el almacenamiento de archivos en la laptop de un desarrollador.

**Soluciones:** Establecer una estructura clara de carpetas en OneDrive y en la laptop, organizando archivos por categoría. O GitHub, mejor, pero igual hay que organizar.

* **Gestión de tiempo: Retrasos en entregas, cambios inesperados en prioridades.**
* Incumplimiento de tareas en la fecha establecida.

**Soluciones:** Se ajusta el progreso en la reunión para evitar retrasos en la ejecución.

* Dependencias entre tareas mal coordinadas, lo que hace que algunos miembros deban esperar a que otros finalicen su parte.

**Soluciones:** **Identificar tareas clave con dependencias** y organizarlas para no generar retrasos innecesarios.

* Falta de planificación realista, causando sobrecarga de trabajo o tiempos insuficientes para completar tareas.

**Soluciones:** Establecer tiempos estimados basados en experiencia previa y evitar comprometer plazos poco realistas.

* **Recursos: Falta de personal, herramientas o presupuesto insuficiente.**
* Falta de laptops suficientes para correr el entorno de desarrollo.

**Soluciones:** Usar la única laptop funcional y la PC del otro desarrollador para codificar la app, aunque esto puede limitar la movilidad en reuniones. Se priorizarán las tareas primordiales para que se realicen la laptop funcional, evitando retrasos en el flujo de trabajo.

* Sobrecarga de trabajo: Pocos miembros en el equipo asumiendo demasiadas responsabilidades, lo que genera agotamiento y errores.

**Soluciones:** Establecer límites en las tareas asumidas por cada miembro.

* Falta de capacitación: El equipo no tiene el conocimiento o habilidades necesarias para ciertas tareas, lo que alarga tiempos de ejecución.

**Soluciones:** Buscar mecanismos de apoyo (Juan Terán) y revisar material con cursos (Youtube).

**Nota:** En caso de incumplimiento de tareas, se ajustará el trabajo en la reunión sobre la base del adelanto previo para minimizar el impacto.

* **Seguridad: Vulnerabilidades en el almacenamiento de datos y acceso a la plataforma.**
* Pérdida de datos: Riesgo de eliminación accidental o falta de respaldo.

**Soluciones:** Guardar archivos en un pendrive como copia de seguridad.

**Fundamentos de Arquitectura de Software**

**Requerimientos funcionales y no funcionales**

***Funcionales***

* Gestión de tareas y eventos:
* Crear, editar y eliminar tareas.
* Asignar fechas límite, plazos de tiempo y niveles de prioridad.
* Visualizar tareas y eventos en un calendario interactivo.

Nota: Se deja pa después lo de las etiquetas.

* Planificación y seguimiento:
* Generar horarios tentativos usando un motor de recomendaciones basado en prioridad y plazo.
* Definir días específicos para trabajar en tareas pendientes, con seguimiento a una hora establecida.
* Monitorear el progreso de los periodos de trabajo (sprints de trabajo) hasta la finalización de la tarea.
* Notificaciones y recordatorios:
* Enviar recordatorios automáticos un día antes y el mismo día del sprint de trabajo.
* Permitir al usuario personalizar sus notificaciones según preferencia.
* Acceso y gestión de usuario:
* Autenticación mediante credenciales almacenadas en la nube.
* Gestión de cuentas de usuario con almacenamiento seguro.
* Interfaz y compatibilidad:
* Diseño responsivo, funcional en computadoras (versión web).
* Visualización clara de horarios, tareas y eventos.
* Cuestionario para técnicas de gestión de tiempo:
* Presentar cuestionarios interactivos que obtengan respuestas sobre el estilo y necesidades del usuario en la gestión del tiempo.
* Utilizar las respuestas para ofrecer técnicas de gestión de tiempo personalizadas.
* Integrar los datos del cuestionario al motor de recomendaciones, considerando tiempos libres, atributos de las tareas y fechas límite seleccionadas por el usuario.

***No funcionales***

* Rendimiento y eficiencia:
* Baja latencia en la carga y actualización de datos.
* Procesamiento optimizado del motor de recomendaciones.
* Escalabilidad y disponibilidad:
* Capacidad para manejar múltiples usuarios con almacenamiento en la nube.
* Funcionamiento estable y recuperación rápida ante fallos.
* Seguridad y privacidad:
* Autenticación segura con estándares mínimos de seguridad en contraseñas (por ejemplo, cumplimiento de NIST SP 800-63B sobre autenticación digital).
* Protección de datos mediante cifrado y almacenamiento seguro.
* Usabilidad y accesibilidad:
* Interfaz intuitiva y adaptable a diferentes dispositivos.
* Compatible con navegadores modernos.

**Preguntas**

Arquitectura de Software

A continuación, se presenta una visión preliminar de la Arquitectura de Software del proyecto de software web: MeAgendo.

**Tecnologías utilizadas**

* **Arquitectura del sistema:**
  + - **Lenguaje de Programación (LP):** Python.
    - **Frontend**
      * **Tecnologías para la interfaz de usuario:** Bootstrap, HTML, CSS, Javascript y Django.
    - **Backend**
      * **Tecnologías para la lógica y procesamiento:** Python, Django.
* **Base de Datos:** PostgreSQL.
* **Servidor:** Render.

**Estructura del sistema**

 **Sección funcional:** Enumera los módulos o funcionalidades que la app ofrece.

**Módulos funcionales**

 **Sección técnica:** Describe los componentes y las capas que usarás para implementar esas funcionalidades. Los componentes técnicos son las tecnologías que ayudan a cumplir las funcionalidades (los módulos funcionales) del sistema

**Capas del sistema (cómo se organiza la arquitectura).**

1. **Capa de presentación (Frontend):**

* Responsable de la interfaz de usuario y la interacción con el sistema.
* Recibe entradas del usuario y envía solicitudes al backend.

1. **Capa de lógica de negocio (Backend):**

* Procesa las reglas del sistema y maneja solicitudes del frontend.
* Realiza cálculos, validaciones y ejecución de algoritmos (por ejemplo, el motor de recomendaciones).

1. **Capa de Datos (Base de Datos):**

* Se encarga del almacenamiento y gestión de la información.
* Almacena datos de usuarios, tareas y preferencias.

1. **Capa de infraestructura (Servidor y Hosting):**

* Garantiza el acceso y despliegue del sistema en la nube.
* Maneja la comunicación entre los componentes y la disponibilidad.

**Componentes técnicos (qué tecnologías se usan en cada capa)**

A continuación, se detallan las tecnologías utilizadas en cada capa:

* **Frontend:** HTML, CSS, JavaScript, Bootstrap.

**¿Por qué se seleccionaron estas tecnologías?**

Porque en una app web siempre se utiliza, Bootstrap se utilizará como herramienta para plantillas y animaciones para la interfaz de usuario.

**Dependencias o herramientas externas**

**Bootstrap:** Framework de diseño utilizado para mejorar la estructura visual y la responsividad del frontend.

* **Backend:** Python, Django.

**¿Por qué se seleccionó Django?**

Porque a los desarrolladores se les asignó así, además de que es un framework de Python sencillo para apps web.

* **Base de Datos:** PostgreSQL.

**¿Por qué se seleccionó PostgreSQL?**

Porque si se usa una una base de datos como SQLite3 en el servidor seleccionado (Render), se corre el riesgo de pérdida de datos. En cambio, PostgreSQL es la opción recomendada para evitar riesgos de pérdida de datos en los servidores de Render.

**Dependencias o herramientas externas.**

**JSON WEB TOKENS (JWT):** Usado para la autenticación y manejo de sesiones, normalmente con una librería como pyjwt (en Python). Guardar tokens de sesión en la BD para que persistan tras una suspensión.

**Cookies o LocalStorage:** Métodos de almacenamiento para los tokens de sesión en el cliente.

* **Servidor:** Render para el despliegue en la nube.

**¿Por qué se seleccionó Render?**

Porque es una opción segura para el almacenamiento de base de datos, además de contar con un plan gratuito viable para el equipo de desarrollo.

**Relación entre componentes**

***ĺndice de componentes a relacionar***

1. **Frontend** (HTML, CSS, JavaScript, Bootstrap)
2. **Backend** (Python, Django)
3. **Base de Datos** (PostgreSQL, con almacenamiento JSONB)
4. **Autenticación** (JWT, almacenamiento en Cookies o LocalStorage)
5. **Servidor** (Render para el despliegue y hospedaje de la aplicación)

Cada uno de estos componentes interactúa con los demás en el flujo de datos y operaciones dentro del sistema. Ahora, siguiendo este índice, aquí está la sección de Relación entre Componentes Técnicos que se estructuraron antes:

***Relación entre componentes técnicos***

1. **Frontend ↔ Backend**
   * **Comunicación:** El Frontend se comunica con el Backend a través de solicitudes del protocolo HTTP siguiendo el enfoque de server-side rendering: enfoque de renderizado de páginas web (SSR). En este modelo, cuando un usuario interactúa con la interfaz (por ejemplo, al iniciar sesión o gestionar tareas), su navegador envía una solicitud HTTP al servidor. El Backend, utilizando Django, procesa dicha solicitud, consulta la base de datos y renderiza la página web completa en el servidor. Una vez generada, la respuesta (HTML completo) se envía de vuelta al Frontend para que el navegador la muestre al usuario.
2. **Backend ↔ Base de Datos**
   * **Interacción de Datos:** Una vez recibido el input del Frontend, el Backend, implementado con Django, interactúa de manera directa con la Base de Datos PostgreSQL mediante la ejecución manual de consultas SQL. Esto implica que, para realizar operaciones CRUD (crear, leer, actualizar y eliminar), se escribirán y ejecutarán directamente las sentencias SQL necesarias. Además, se ha optado por utilizar el tipo de dato JSONB para almacenar información de sesión y otros datos no estructurados, lo que permite una mayor flexibilidad en el manejo y en la realización de consultas complejas, garantizando la persistencia de la información incluso tras reinicios o suspensiones del servicio.
3. **Autenticación (JWT, Cookies o LocalStorage)**
   * **Gestión de Sesiones y Autenticación:** Al iniciar sesión, el Backend genera un token usando JWT (JSON Web Token) mediante librerías como pyjwt en Python. Este token, que contiene datos identificativos (claims) y se firma digitalmente, se envía al Frontend. En el navegador, el token puede almacenarse mediante Cookies (configurables con atributos de seguridad como HttpOnly y Secure) o en LocalStorage, según los requerimientos del proyecto. En cada solicitud subsiguiente, el Frontend incluye el token —ya sea de forma automática (al estar en cookies) o manualmente (leyéndolo de LocalStorage)— para que el Backend verifique su autenticidad y mantenga la sesión activa.

**PeNdIeNtE:** Decidir si trabajar con LocalStorage o Cookies. Tentativo la elección de LocalStorage. Si el nivel de seguridad lo requiere se puede migrar a las cookies.

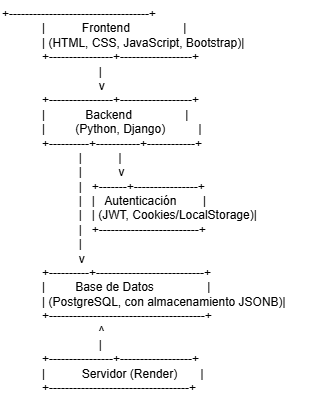
**Glosario**

* **Token:** Fragmento de información usado para autenticar/autorizar.
* **JWT (JSON Web Token):** Tipo de token usado en autenticación web.
* **PyJWT:** Librería en Python para manejar JWT.

1. **Backend y Servidor (Render)**

El backend de la aplicación se despliega en Render, que orquesta la ejecución de la lógica de negocio y el procesamiento de solicitudes. Además, integra mecanismos de autenticación mediante JSON Web Tokens y se conecta de forma óptima a PostgreSQL para la gestión de datos. Esta configuración asegura una infraestructura persistente e íntegra en producción, en contraposición de soluciones locales como SQLite, las cuales presentan limitaciones en escenarios de alta concurrencia.

**Diagrama de la relación de los componentes**



**Modelo de almacenamiento y seguridad**

Pendiente para la próxima fase (Diseño y Implementación).

**Damos por concluida la primera fase del desarrollo del producto de software: MeAgendo.**

**¡¡¡¡¡¡¡¡Ahora largo de nuestras vidas!!!!!!!!!!**