Factu3Dimensions

Índice

Capítulo 1: Introducción

1.1 Introducción del proyecto

La necesidad de control y gestión de tareas, horas y facturación en entornos de trabajo modernos ha crecido exponencialmente en los últimos años, particularmente en sectores donde la externalización de servicios, el trabajo remoto y la flexibilidad operativa se han convertido en pilares esenciales de la productividad. Muchas empresas, especialmente pymes y profesionales autónomos, no cuentan con sistemas informáticos avanzados que les permitan organizar sus operaciones cotidianas, lo que conduce a una pérdida de eficiencia y dificultad en el seguimiento de tareas, tiempos y costes asociados a los proyectos.

Este proyecto nace como respuesta a esta necesidad real detectada en el sector productivo. Se trata del desarrollo de una aplicación web integral, accesible desde cualquier dispositivo, que permita a un administrador asignar clientes a empleados, gestionar las tareas realizadas por estos, llevar un registro preciso de las horas trabajadas y facturar de manera automática las actividades desarrolladas. Esta herramienta se construye sobre tecnologías web modernas y escalables, y se enfoca en ofrecer una experiencia de usuario intuitiva, una estructura de datos robusta y la posibilidad de adaptar el sistema a futuras funcionalidades requeridas por el usuario final.

El presente documento describe en profundidad cada una de las fases del proyecto: desde el análisis del sistema hasta la implementación, pasando por el diseño lógico y físico, las pruebas realizadas y las conclusiones extraídas. La aplicación desarrollada presenta ventajas clave como:

- Acceso desde cualquier lugar y dispositivo con conexión a Internet.
- Centralización y trazabilidad de tareas, clientes, usuarios y facturación.
- Automatización de cálculos de horas y generación de facturas.
- Generación de informes mensuales/anuales exportables.
- Optimización de tiempos de gestión administrativa y mejora de la productividad.

Además, se busca que el sistema sea intuitivo, ligero y adaptable a distintas configuraciones según el tipo de empresa o profesional que lo implemente. La arquitectura cliente-servidor y el uso de una base de datos NoSQL permiten escalar la solución sin perder rendimiento ni estabilidad. En este sentido, la aplicación no solo constituye una solución inmediata a una necesidad específica, sino que también se proyecta como una base para futuros desarrollos más complejos y personalizados.

1.2 Propósito

El propósito fundamental de este proyecto es ofrecer una solución tecnológica eficiente que mejore los procesos de asignación de tareas, control horario y generación de facturación dentro de un entorno empresarial. Este sistema busca servir como una plataforma centralizada donde el administrador pueda organizar a sus empleados, asignarles tareas específicas por cliente y realizar un seguimiento detallado de los tiempos dedicados a cada labor.

La herramienta permitirá también que cada usuario autenticado (empleado o administrador) pueda visualizar el desglose de horas trabajadas, las tareas finalizadas y la correspondiente facturación generada automáticamente según los precios estipulados para cada actividad. La exportación de estos datos en formatos accesibles facilitará la transparencia y la contabilidad tanto interna como de cara a los propios clientes.

Este desarrollo tiene como fin último la creación de una herramienta adaptable, escalable y accesible que reduzca errores administrativos, aumente la eficiencia del personal y proporcione reportes actualizados y precisos, alineándose con las necesidades del sector profesional actual.

1.3 Objetivos del Proyecto

1. Realizar un estudio detallado de los requisitos funcionales y no funcionales del sistema.
2. Diseñar y desarrollar una aplicación web con interfaz moderna, intuitiva y responsive.
3. Implementar un sistema de autenticación y autorización para usuarios diferenciados.
4. Establecer un sistema lógico de gestión de tareas asociadas a clientes y empleados.
5. Automatizar el cálculo de horas y generación de facturas mensuales o anuales.
6. Habilitar exportación de informes y visualización de métricas para el administrador y los usuarios.
7. Aplicar buenas prácticas de desarrollo: separación de capas, control de versiones, uso de patrones de diseño.
8. Redactar la documentación técnica del proyecto (manual de instalación, manual de usuario, memoria técnica).
9. Verificar el cumplimiento de los objetivos mediante pruebas funcionales y estructurales.
10. Presentar el proyecto como una herramienta real, útil y replicable en otros entornos productivos.

1.4 Coste del Proyecto

1.4.1 Costes de Desarrollo

Los costes de desarrollo se dividen en:

- **Costes de Recursos Informáticos:**
  - Ordenador portátil para desarrollo: 800 € (uso ya existente del alumno).
  - Software utilizado: Visual Studio Code, MongoDB, Node.js, Express, librerías JS — todos gratuitos.
  - Licencias: ninguna.
- **Costes de Personal:**

- Tiempo de desarrollo estimado: 300 horas.
- Valor estimado hora: 15 €/hora.
- Total coste personal: 4.500 €.

Coste total de desarrollo estimado: 5.300 €.

1.4.2 Costes de Implantación

- Alojamiento web (servidor Node.js + base de datos MongoDB Atlas): 20 €/mes x 12 meses = 240 €.
- Dominio personalizado (opcional): 12 €/año.
- Soporte y mantenimiento: 300 €/año.

Total implantación anual: aproximadamente 552 €.

Capítulo 2: Análisis del Sistema

2.1 Introducción

El análisis del sistema constituye la base del desarrollo del proyecto, permitiendo identificar las funcionalidades requeridas, las interacciones de los distintos actores y los flujos de información necesarios. Esta fase incluye la definición de los requisitos funcionales y no funcionales, el modelado de los casos de uso y el diseño preliminar de la arquitectura lógica.

La aplicación web desarrollada está diseñada para cumplir con una arquitectura distribuida cliente/servidor, en la cual se identifican distintos perfiles de usuario (administrador, empleados y clientes) que interactúan con el sistema según permisos específicos. Se realiza también un análisis de persistencia de datos mediante una base de datos NoSQL, optando por MongoDB por su flexibilidad y escalabilidad.

2.2 Análisis de Requisitos

Requisitos funcionales:

1. El sistema permitirá el registro y autenticación de usuarios.
2. El administrador podrá asignar empleados a clientes.
3. El administrador podrá crear tareas asociadas a clientes y asignarlas a empleados.
4. Los empleados podrán visualizar sus tareas y registrar horas trabajadas.
5. El sistema calculará automáticamente los importes a facturar por cada cliente en función de las horas registradas y la tarifa establecida.
6. Generación de facturas mensuales/anuales.
7. Exportación de informes.

8. Diferenciación de roles: administrador, empleado, cliente.

Requisitos no funcionales:

1. La aplicación será responsive y accesible desde distintos dispositivos.
2. Se utilizarán estándares de desarrollo web actuales (HTML5, CSS3, ES6+).
3. La base de datos permitirá consultas rápidas y escalabilidad.
4. El sistema tendrá mecanismos de autenticación segura (tokens, sesiones).
5. El código fuente será modular, reutilizable y mantenible.

2.1.- **Casos de uso:** Diagramas y Narrativas de Casos de Uso

Los diagramas de casos de uso reflejarán las funcionalidades que correspondan a cada uno de los ámbitos concretos y más importantes del sistema. En la narrativa de los casos de uso se especifica las funciones, tareas e interacciones que se dan lugar en el sistema, es decir, se describen detalladamente cada uno de los casos de uso. Explica que son estos diagramas y realiza los pertinentes para tu aplicación.

Diagrama de Casos de Uso: Gestión de Tareas y Facturación

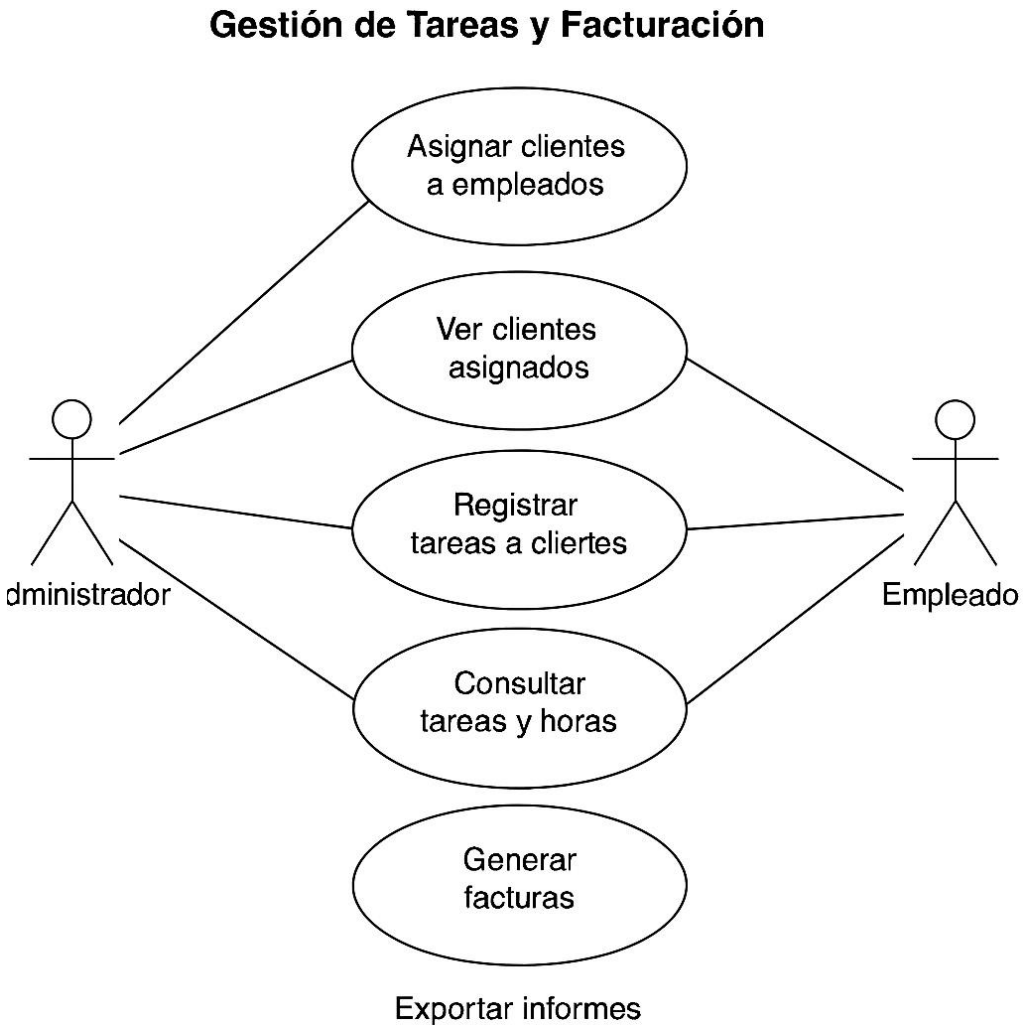
Actores:

- Administrador
- Empleado

Casos de uso principales:

- Asignar clientes a empleados
- Ver clientes asignados
- Registrar tareas a clientes
- Consultar tareas y horas
- Generar facturas
- Exportar informes

DIAGRAMA DE CASO DE USO: GESTIÓN DE DOCUMENTOS



Narrativas de Casos de Uso

Caso de Uso 1: Asignar clientes a empleados

- **Actor Principal:** Administrador
- **Condiciones de entrada:** El administrador debe estar autenticado en el sistema.

- **Flujo de eventos:**
    1. El administrador accede al panel de administración.
    2. Selecciona un empleado.
    3. El sistema muestra una lista de clientes disponibles.
    4. El administrador selecciona uno o varios clientes.
    5. Confirma la asignación.
    6. El sistema registra la relación entre empleado y cliente.
- **Camino alternativo:**

1. 5b. El administrador cancela la acción → vuelve al paso 2.

- **Condición de salida:** El cliente queda correctamente asignado al empleado.

Caso de Uso 2: Registrar tareas a clientes

- **Actor Principal:** Empleado o Administrador
- **Condiciones de entrada:** El actor debe estar autenticado y tener clientes asignados.
- **Flujo de eventos:**
    1. El actor accede al panel de tareas.
    2. Selecciona un cliente de su lista.
    3. Introduce la tarea realizada, fecha, descripción y número de horas.
    4. Confirma el registro.
    5. El sistema guarda la tarea vinculada al cliente y al empleado.
- **Camino alternativo:**
    1. 4b. El usuario cancela el registro → vuelve al paso 2.
- **Condición de salida:** La tarea queda registrada correctamente en la base de datos.

Caso de Uso 3: Generar factura de un cliente

- **Actor Principal:** Empleado o Administrador
- **Condiciones de entrada:** Tener tareas registradas para el cliente.
- **Flujo de eventos:**
    1. El actor accede al panel de facturación.
    2. Selecciona el cliente y el mes.
    3. El sistema muestra todas las tareas registradas y su total.

4. El actor pulsa “Generar factura”.
5. El sistema crea un documento con los datos de la factura y lo muestra.

- **Camino alternativo:**
  1. 4b. El actor cancela la acción → vuelve al paso 2.
- **Condición de salida:** La factura queda generada y almacenada o descargada.

Caso de Uso 4: Exportar informes

- **Actor Principal:** Empleado o Administrador
- **Condiciones de entrada:** Estar autenticado y haber generado tareas o facturas.
- **Flujo de eventos:**
  1. El actor accede al área de informes.
  2. Selecciona el tipo de exportación (mensual/anual).
  3. El sistema genera un documento PDF o Excel.
  4. El archivo se descarga automáticamente o se guarda en el sistema.
- **Condición de salida:** El usuario obtiene un archivo con los datos requeridos.

Caso de Uso 5: Ver clientes asignados

- **Actor Principal:** Empleado o Administrador
- **Flujo de eventos:**
  1. El actor accede a su panel principal.
  2. El sistema muestra la lista de clientes asignados.
- **Condición de salida:** Visualización de todos los clientes asignados al actor.

Capítulo 3: Diseño del Sistema

3.1 Introducción

En esta fase del proyecto se abordan las decisiones técnicas y estructurales que darán forma a la solución implementada. El diseño se orienta a cumplir todos los requisitos funcionales y no funcionales previamente analizados, garantizando una estructura modular, mantenible y escalable.

Este capítulo se divide en tres partes clave:

- Modelado lógico del sistema mediante diagramas de clases.
- Estructuración de la base de datos.
- Diseño visual e interactivo de la interfaz de usuario.

3.2 Diagrama de Clases

El diagrama de clases representa la estructura estática del sistema: define entidades clave, sus atributos y relaciones. Entre las clases principales se encuentran:

- **Usuario**: clase general con atributos comunes (nombre, email, contraseña, rol).
- **Empleado** (hereda de Usuario): puede registrar horas y generar facturas.
- **Administrador** (hereda de Usuario): puede gestionar empleados, clientes y tareas.
- **Cliente**: tiene asignado uno o varios empleados.
- **Tarea**: contiene nombre, descripción, duración, fecha, cliente asociado, y empleado asignado.
- **Factura**: asociada a tareas, generada por administrador o empleado.

Estas clases están relacionadas por asociaciones como "uno a muchos" (un cliente puede tener muchas tareas), herencia (empleado y administrador son usuarios), y composición (una factura contiene tareas).

3.3 Diseño de la Base de Datos

3.3.1 Diseño Lógico

Se definen las **entidades**, sus **atributos**, **claves primarias** y **relaciones**:

- **Usuarios**: ID, nombre, email, contraseña, rol.
- **Clientes**: ID, nombre, email, empresa, etc.
- **Empleados**: ID (referencia a usuario), lista de clientes asignados.
- **Tareas**: ID, descripción, fecha, horas, clienteID, empleadoID.
- **Facturas**: ID, fecha, tareas asociadas, total.

Relaciones:

- Un cliente puede tener muchas tareas.
- Un empleado puede tener muchos clientes y muchas tareas.
- Las facturas agrupan varias tareas.

3.3.2 Diseño Conceptual

Se detallan las **tablas físicas** si se usara SQL, o los **documentos** y relaciones si se usa MongoDB (como en este proyecto). En MongoDB:

- Cada documento de tarea contiene referencias a cliente y empleado.
- Las facturas pueden tener embebidas las tareas o solo sus IDs.
- Validaciones a través de esquemas (por ejemplo, Mongoose).

Tipos de datos: strings, dates, numbers, booleans.
Restricciones: únicos (emails), obligatorios (ID, rol, etc.), validaciones personalizadas.
Triggers: no aplican directamente, pero se simulan con middlewares y validaciones automáticas.

3.4 Diseño de la Interfaz

La interfaz se diseña pensando en la **experiencia de usuario (UX)**: debe ser simple, clara y funcional, adaptable a móviles y escritorio (responsive).

Pantallas clave:

- **Login / Registro**
- **Panel del administrador**: vista de clientes, empleados, tareas, generación de facturas.
- **Panel del empleado**: tareas asignadas, botón para registrar horas, generar factura propia.
- **Vista del cliente** (si aplica): tareas realizadas y facturas.
- **Formulario de creación y edición de tareas**
- **Visualización y exportación de informes**

La interfaz está construida en HTML, CSS (Tailwind) y JavaScript con un framework como React o directamente con plantillas dinámicas renderizadas por el servidor (según arquitectura elegida).

4.1.- CAPÍTULO 4: IMPLEMENTACIÓN.

4.2.- INTRODUCCIÓN

En esta fase, se lleva a cabo la conversión del diseño en código funcional. Se utilizan las tecnologías seleccionadas para desarrollar cada componente del sistema, asegurando que la implementación cumpla con los requisitos establecidos.

El desarrollo se realiza siguiendo una arquitectura basada en **HTML, CSS y JavaScript** para la interfaz de usuario y **MongoDB** para la gestión de la base de datos. Se aplican buenas prácticas de programación, modularidad y separación de responsabilidades para garantizar un código limpio y escalable.

4.3.- ARQUITECTURA CLIENTE/SERVIDOR

La **arquitectura cliente/servidor** es un modelo de comunicación en el que las tareas de procesamiento y almacenamiento de datos se dividen entre dos entidades principales:

1. **Cliente:** Es el dispositivo o aplicación que solicita información o servicios al servidor. En nuestro caso, el cliente es la aplicación web desarrollada con **HTML, CSS y JavaScript**.

2. **Servidor:** Es el sistema que recibe las solicitudes del cliente, las procesa y devuelve la información solicitada. En nuestro caso, el servidor está desarrollado con **Node.js y Express.js**, gestionando la base de datos en **MongoDB**.

4.3 Lenguajes de Programación

Para el desarrollo de la aplicación se han utilizado los siguientes lenguajes de programación:

- **JavaScript**: Es el lenguaje principal utilizado para la lógica del lado del cliente y del servidor. Permite crear una aplicación dinámica e interactiva, manejando la comunicación entre el frontend y el backend.

- **HTML5**: Lenguaje de marcado utilizado para estructurar las páginas web, definiendo los elementos que se mostrarán en la interfaz de usuario.

- **CSS3**: Lenguaje de estilos que se emplea para definir la apariencia visual de la aplicación, incluyendo colores, tipografía, disposición y diseño responsivo.

- **Node.js**: Plataforma que permite ejecutar JavaScript en el servidor, facilitando la creación del backend y la conexión con la base de datos.

- **MongoDB Query Language (MQL)**: Aunque no es un lenguaje de programación tradicional, se utiliza para la manipulación y consulta de documentos en la base de datos NoSQL MongoDB.

4.4 Herramientas de Desarrollo

Para llevar a cabo el desarrollo del proyecto, se han utilizado las siguientes herramientas:

- **Visual Studio Code**: Editor de código fuente con soporte para múltiples extensiones que facilitan la programación en JavaScript, manejo de Git, y depuración.
- **Git y GitHub**: Para el control de versiones y gestión del código fuente, permitiendo mantener un historial de cambios y colaboración remota.
- **Node.js y npm**: Para la gestión de paquetes y ejecución del servidor backend.
- **MongoDB Atlas**: Plataforma de base de datos en la nube utilizada para almacenar y gestionar los datos de la aplicación.
- **Postman**: Herramienta para probar y depurar las APIs REST desarrolladas.
- **Mongoose**: Librería de modelado de objetos MongoDB para Node.js, que facilita la definición de esquemas y la validación de datos.

4.5 Codificación

Todo el código desarrollado durante el proyecto se encuentra disponible para su consulta y evaluación en el medio especificado por el departamento de informática, ya sea a través de un pendrive o en la plataforma Moodle. El repositorio contiene la estructura completa del proyecto, incluyendo el backend, frontend, esquemas de base de datos y archivos de configuración.

5.1.-INTRODUCCIÓN.

Las **pruebas de software** son una parte crucial del proceso de desarrollo de cualquier aplicación. Estas pruebas buscan garantizar que el sistema desarrollado cumpla con los requisitos establecidos, funcione correctamente y esté libre de errores antes de su despliegue y utilización final. El objetivo principal de las pruebas es asegurar la **calidad** del software, verificando su **funcionalidad**, **rendimiento** y **seguridad**.

Las pruebas no solo sirven para detectar y corregir errores, sino que también permiten verificar que la aplicación se comporta de acuerdo con las expectativas del usuario final, y que es capaz de manejar escenarios inesperados de forma eficiente.

5.2.- TÉCNICAS DE PRUEBA.

En el proceso de pruebas de software, existen diferentes técnicas que se utilizan para garantizar la calidad y el buen funcionamiento del sistema. Dos de las técnicas más comunes son las **pruebas de caja blanca** y las **pruebas de caja negra**. A continuación, se detallan ambas técnicas y cómo se aplican en el contexto del desarrollo de una aplicación.

5.2.1.- PRUEBAS DE CAJA BLANCA O ENFOQUE ESTRUCTURAL.

Las pruebas de caja blanca se basan en el conocimiento del código fuente y la estructura interna del sistema. En este tipo de pruebas, el evaluador tiene acceso completo al código del software y realiza las pruebas desde una perspectiva interna, es decir, probando los componentes individuales y el flujo de ejecución dentro del sistema.

Características Principales:

- Se centran en el código interno y la lógica del sistema.
- El tester tiene conocimiento completo del funcionamiento interno del sistema (estructura de datos, algoritmos, código fuente).
- Se realizan pruebas sobre funciones específicas, condiciones de ramas, bucles y secuciones de código.
- Son útiles para verificar la lógica del código y detectar errores a nivel de desarrollo.

Tipos de Pruebas de Caja Blanca:

1. Pruebas de cobertura de código: Se verifica si todas las rutas posibles del código han sido probadas (incluso los casos límite y excepcionales).
2. Pruebas de rutas: Se verifica que todas las rutas posibles en un programa sean evaluadas.

3. Pruebas de flujo de control: Se examina cómo el flujo de control (como los bucles y las decisiones) se comporta en diferentes condiciones.

4. Pruebas de flujo de datos: Se analiza cómo los datos se mueven a través del sistema.

Ejemplo en el Proyecto:
En el caso de una función que calcule el total de las horas trabajadas de un usuario, una prueba de caja blanca podría involucrar la verificación de:

- Si la función maneja correctamente la entrada de horas inválidas (por ejemplo, horas negativas).
- Si la lógica para calcular el total de horas está funcionando bien bajo diferentes condiciones.
- Si el código gestiona adecuadamente los posibles errores o excepciones.

Ventajas de las Pruebas de Caja Blanca:

- Ayuda a detectar errores lógicos en el código, como los bucles infinitos, las ramas no alcanzadas y las variables no utilizadas.
- Proporciona una visión más profunda de la calidad del código.

Desventajas:

- Requiere conocimientos avanzados de programación y comprensión detallada del código.
- Pueden ser más costosas en tiempo y esfuerzo, especialmente cuando se tienen que probar grandes sistemas.

5.2.2.- PRUEBAS DE CAJA NEGRA O ENFOQUE FUNCIONAL.

Las pruebas de **caja negra** se basan en la evaluación del comportamiento del software sin tener en cuenta su código interno. El tester se concentra exclusivamente en las entradas y salidas del sistema, validando si la aplicación funciona correctamente de acuerdo con los requisitos del usuario y las especificaciones sin conocer cómo el sistema produce esos resultados.

Características Principales:

- No se requiere **conocimiento del código** interno del sistema.

- El tester verifica si el software **cumple con los requisitos funcionales** y se comporta como se espera desde la perspectiva del usuario final.

- Se prueban **funcionalidades** del sistema, como formularios, autenticación, navegación, etc.

- Las pruebas se enfocan en **entradas y salidas**, en lugar de detalles internos.

Tipos de Pruebas de Caja Negra:

1. **Pruebas de funcionalidad:** Se verifica que el sistema realice las tareas según lo esperado.

2. **Pruebas de integración:** Se prueba cómo el sistema interactúa con otros sistemas y servicios.

3. **Pruebas de sistema:** Se evalúa si el sistema en su conjunto cumple con los requisitos especificados.

4. **Pruebas de aceptación del usuario:** Se realiza la validación de que el sistema satisface las necesidades del usuario final.

5. **Pruebas de interfaz:** Se verifica que las interfaces de usuario sean fáciles de usar y funcionen correctamente.

Ejemplo en el Proyecto:
En el caso de la funcionalidad de inicio de sesión en la aplicación:

- Se podría realizar una prueba de caja negra proporcionando diferentes combinaciones de correo y contraseña para comprobar que el sistema permite el acceso solo con credenciales válidas, y muestra un mensaje de error en caso contrario, sin necesidad de conocer el código detrás del proceso de autenticación.

Ventajas de las Pruebas de Caja Negra:

- Se enfoca en la experiencia del usuario, por lo que es excelente para probar que el software cumpla con los requisitos desde la perspectiva de uso.

- Es más fácil de realizar, ya que no requiere conocimientos sobre el código fuente.

- Permite detectar problemas en la **interfaz de usuario**, **flujo de trabajo** y **funcionalidades generales**.

Desventajas:

- No se enfoca en el código interno, por lo que no ayuda a detectar errores lógicos o en el código que no se reflejan en las entradas y salidas.

6.1.- CONCLUSIONES

El proyecto desarrollado ha cumplido con los objetivos establecidos al inicio, proporcionando una solución eficiente y funcional para la gestión de tareas y facturación en un entorno de desarrollo web. A través de la implementación de la aplicación, el administrador pueden registrar y gestionar las tareas diarias, visualizar las horas trabajadas y generar facturas de manera sencilla. Este sistema tiene como principal ventaja su accesibilidad en línea y su capacidad para generar informes detallados que facilitan la toma de decisiones tanto para los administradores como para los usuarios.

El desarrollo de este proyecto ha sido una excelente oportunidad para consolidar mis habilidades en desarrollo web, desde el diseño hasta la implementación de la base de datos. He adquirido experiencia en las siguientes áreas:

- **Uso de tecnologías web modernas**: El proyecto permitió un enfoque completo utilizando HTML, CSS, JavaScript, y MongoDB, lo que consolidó mis conocimientos en desarrollo frontend y backend.
- **Desarrollo de bases de datos NoSQL**: A través de MongoDB, aprendí cómo trabajar con bases de datos NoSQL, gestionando colecciones y relaciones entre ellas de forma eficiente.

Coras, mejorando la transparencia y confianza en los procesos administrativos.

6.2.- PROPUESTAS FUTURAS (OPCIONAL).

A. **Integración con otros sistemas de gestión empresarial**
   a) Integrar funciones con **CRM (Customer Relationship Management)** o **ERP (Enterprise Resource Planning)** podría optimizar la gestión de relaciones con los clientes, así como los recursos internos de la empresa.
   b) **Integrar herramientas de facturación**: Vincular la aplicación con sistemas de facturación automática o plataformas como **Stripe** o **PayPal** para permitir pagos directos a través de la plataforma.
   c) **Integrar plataformas de contabilidad**: Sincronizar con herramientas de contabilidad
B. **Funcionalidad de informes más avanzados**
   a) **Informes personalizados**: Permitir a los administradores crear informes personalizados en función de diferentes parámetros (por ejemplo, filtrar por cliente, por proyecto, por fecha).
   b) **Gráficos y visualización de datos**: Implementar representaciones gráficas de las horas trabajadas y los ingresos generados, lo que mejoraría la comprensión de los datos.
C. **Mejoras en la seguridad**
   a) **Autenticación multifactor**: Implementar la autenticación en dos pasos (2FA) para reforzar la seguridad de las cuentas de los usuarios.
   b) **Encriptación avanzada de datos**: Mejorar la seguridad de la base de datos encriptando información sensible, como las contraseñas de los usuarios y los detalles de facturación.
   c) **Autorización de roles**: Establecer permisos más específicos para los diferentes tipos de usuarios, como administradores, empleados o clientes, de modo que se pueda controlar mejor el acceso a los diferentes módulos y funciones. (Cumplido)
D. **Implementación de una aplicación móvil**
   a) **Desarrollo de una app móvil**: Utilizar frameworks como **React Native** o **Flutter** para crear una versión móvil de la aplicación que permita a los usuarios acceder a sus tareas y

facturación desde sus dispositivos móviles, con notificaciones push para recordarles sobre las horas trabajadas.
b) **Optimización responsive**: Mejorar la versión web para que sea completamente responsiva, de modo que los usuarios puedan acceder cómodamente desde cualquier dispositivo sin necesidad de una app adicional.

E. **Soporte multilingüe**
a) **Internacionalización (i18n)**: Implementar una funcionalidad que permita traducir toda la aplicación a varios idiomas, utilizando bibliotecas como **i18next**.
b) **Localización de moneda y formato de fecha**: Dependiendo del idioma, adaptar la moneda, las fechas y las horas al formato local de cada usuario.

F. **Soporte de equipos**
a) **Chat en vivo**: Implementar un sistema de comunicación en tiempo real entre los usuarios y administradores (integración con **WebSockets** o **Socket.io**).
b) **Notificaciones de tareas y cambios**: Configurar alertas automáticas para notificar a los usuarios sobre nuevas tareas asignadas o cambios en sus tareas.

G. **Funciones de Inteligencia Artificial**
a) **Predicción de demanda**: Utilizar **machine learning** para predecir cuántas horas se necesitarán para completar una tarea o proyecto basándose en datos históricos.
b) **Automatización del análisis de datos**: Implementar algoritmos para identificar patrones en las tareas y las horas facturadas, proporcionando recomendaciones de optimización.

H. **Mejora en la experiencia de usuario**
a) **Rediseño de la interfaz**: Mejorar los aspectos visuales y la ergonomía de la plataforma con nuevas tendencias de diseño, utilizando herramientas como **Figma** o **Adobe XD**.
b) **Optimización del flujo de trabajo: Realizar estudios de usabilidad para simplificar los procesos más largos o confusos y asegurarse de que la plataforma sea intuitiva y fácil de usar para todos los usuarios.**

7 Referencias

- MongoDB Docs
- MDN Web Docs
- StackOverflow
- PlantUML para diagramas
- Documentación oficial de Node.js y Express

ANEXO 1: MANUAL DE INSTALACIÓN

Instrucciones para el uso de la app

cuenta de correo para mongoDB: eduromerootero@gmail.com

contraseña correo: proyectoweb2025

Para lanzar el backend (necesario tenerlo simpre ejecutado):
node server.js

Para lanzar el front:
Ejecutar el archivo login.html en explorador

Para lanzar el backend:
entrar en carpeta 'backend' y ejecutar 'npm run dev':
cd backend / npm run dev

datbase: invoices?retryWrites=true&w=majority
url mongoDB: mongodb+srv://eduromerootero:noFEWmGe4YgAY9sa@cluster0.d9sx94x.mongodb.net/

Implementación del Proyecto
Este proyecto consiste en una aplicación de gestión de pedidos de clientes y asignación de tareas a empleados, utilizando una arquitectura tipo REST API con Node.js, Express y MongoDB, y un cliente web construido en HTML/CSS/JavaScript vanilla.

Tecnologías utilizadas
Backend: Node.js, Express, MongoDB, Mongoose

Frontend: HTML5, CSS3, JavaScript

Base de Datos: MongoDB Atlas (o local)

Herramientas adicionales: Nodemon (dev), CORS, dotenv

Estructura del backend
bash
Copiar
Editar
/server
  /models
    client.js
    employee.js
  /routes
    clientRoutes.js

employeeRoutes.js

Instalación y configuración
Clona el repositorio:

git clone https://github.com/Bobino6/ProyectoDAW
cd tu-repositorio
Instala las dependencias del backend:

cd backend
npm install
Crea un archivo .env dentro de la carpeta backend con tus variables:

PORT=5000
MONGO_URI=mongodb+srv://eduromerootero:noFEWmGe4YgAY9sa@cluster0.d9sx94x.mongodb.net/invoices?retryWrites=true&w=majority

Inicia el servidor:
cd backend
npm run dev
Abre el archivo login.html (en la carpeta pages dentro de frontend) en tu navegador para comenzar a interactuar con la interfaz. (editado)

ANEXO 2: MANUAL DE USUARIO

Adjunto vídeo.