Universidad Internacional San Isidro Labrador

Proyecto

Estudiantes:

Andrés Vargas Rodríguez

Cédula: 7-0283-0904,

Curso:

(ISB-32) PROGRAMACIÓN AVANZADA

Proyecto – TaskWise

Profesor:

ESTEFANIA BOZA VILLALOBOS

Nota del Autor:

El presente Proyecto es requisito para optar por la aprobación del curso programación Avanzada del Bachillerato en Ingeniería de Sistemas de la Universidad Internacional San Isidro Labrador.

Para contactar al autor escriba por correo electrónico a la dirección:

andresvr743@gmail.com,

Objetivo general del proyecto

Desarrollar las vistas principales de la aplicación web TaskWise, enfocándome en un diseño visualmente atractivo, ordenado y profesional que sirva como base para futuras funcionalidades interactivas orientadas a la gestión de tareas, auditorías y reportes.

Objetivos específicos

- Diseñar una interfaz coherente y visualmente atractiva utilizando HTML y CSS.
- Simular el flujo completo de navegación de un usuario dentro del sistema.
- Representar pantallas clave como inicio de sesión, panel principal, checklists, reportes y perfil de usuario.
- Garantizar la conexión lógica entre las vistas para una experiencia de usuario fluida.
- Establecer una estructura base para integrar posteriormente funcionalidades dinámicas con JavaScript y conexión a bases de datos.

Estructura del sistema

Alcance

El alcance de este avance se limita al diseño visual y estructural de las vistas principales de *TaskWise*. No se ha incorporado lógica de servidor ni base de datos real, pero se ha simulado su comportamiento básico.

El sistema está compuesto por las siguientes vistas:

- Página de Inicio – index.html
- Inicio de Sesión – login.html
- Panel Principal – dashboard.html
- Checklists – checklists.html
- Administración – administracion.html
- Reportes – reportes.html
- Mi Cuenta – cuenta.html

Requerimientos del sistema

Requerimientos funcionales:

1. Permitir el inicio de sesión básico mediante usuario y contraseña simulados.
2. Visualizar resumen de tareas completadas, pendientes y totales.

3. Mostrar y gestionar listas de verificación (Checklists).
4. Generar y consultar reportes históricos.
5. Administrar usuarios con opciones de edición y eliminación.
6. Modificar los datos personales desde la vista "Mi Cuenta".
7. Simular la navegación entre vistas de forma lógica y coherente.

Requerimientos no funcionales:

1. La interfaz debe tener un diseño moderno, legible y visualmente atractivo.
2. El sistema debe ser intuitivo, con una navegación clara entre vistas.
3. Las vistas deben ser consistentes en diseño (colores, fuentes, botones).
4. El código debe estar organizado, sin uso de plantillas externas.
5. Se debe simular la funcionalidad sin necesidad de backend en este avance.

Diseño inicial de las vistas de sistema

Página de Inicio – index.html

Aquí doy la bienvenida a los usuarios. Mostré el nombre del sistema (TaskWise), un breve mensaje sobre su utilidad, y agregué un botón para ingresar a la plataforma. El diseño tiene colores modernos y agradables, y una imagen de logo central. El logo fue diseñado en Canva, Un poco sencillo pero profesional a mi gusto.

Ilustración 1: Vista de bienvenida

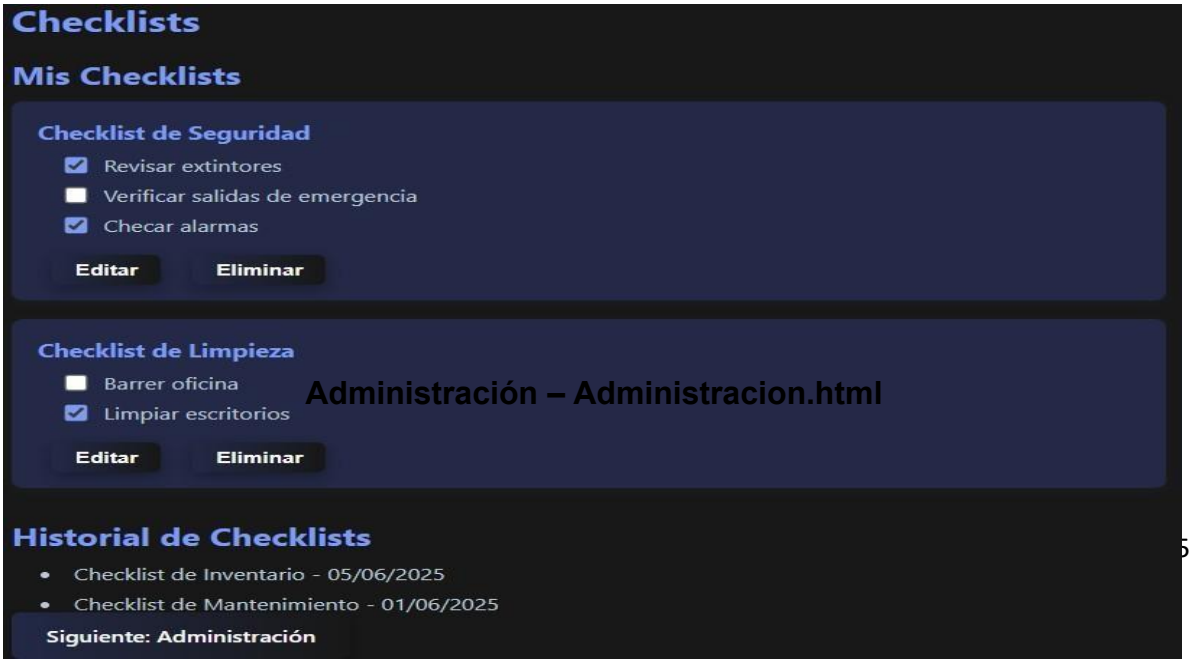
Inicio de Sesión – login.html

Esta es la pantalla donde los usuarios colocan su nombre y contraseña. Si los datos ingresados son correctos (por ejemplo, la contraseña y usuario de prueba ahorita son, 'admin' y '1234'), el sistema lleva al tablero principal. Usé un pequeño bloque de JavaScript para que el formulario funcione de forma simulada. Como lo mencione en el video, ya cuando este la BD los datos para rellenar el formulario de inicio de sesión va a cambiar.

Ilustración 2: Vista de Login

Panel Principal – dashboard.html

Ilustración 3: Vista de Dashboard



Administración – Administracion.html

En esta sección mostré un resumen de tareas: cuántas están completadas, cuántas pendientes y el total general. También incluí una lista con las últimas tareas realizadas, y un botón para continuar a la siguiente vista.

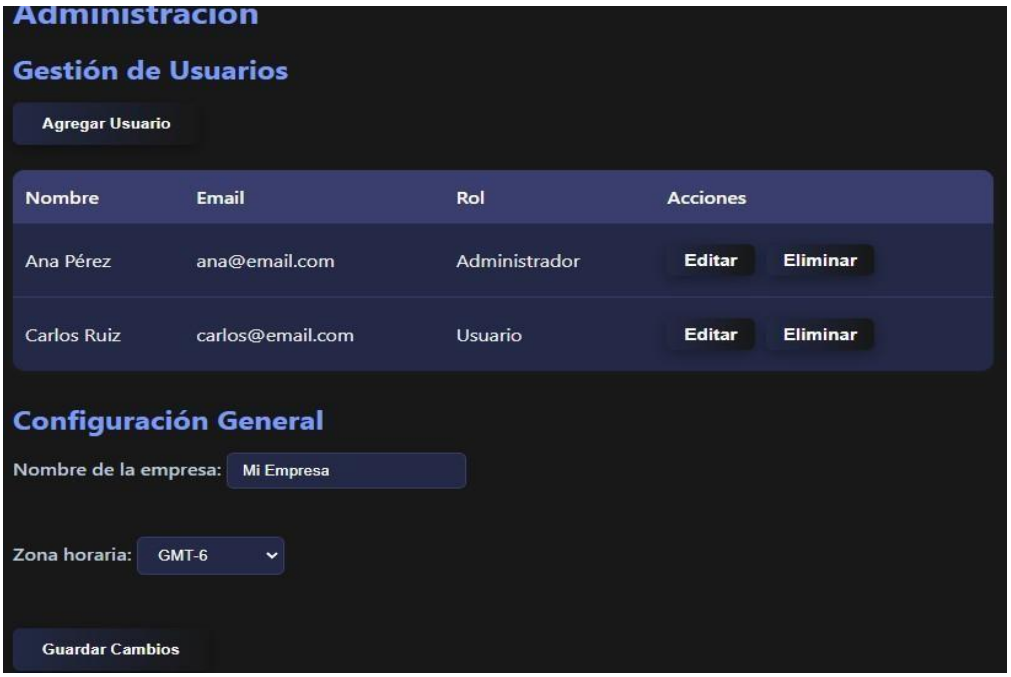
Ilustración 4: Vista de resumen de tareas

Checklists – Checklists.html

Aquí incluí ejemplos de listas de verificación (Checklists), como los que se usan en temas de seguridad o limpieza (son ejemplos, que probablemente los cambie después por unos más del área mía de TI, fue como lo que se me ocurrió en el momento). Los usuarios pueden marcar cada ítem, y también tienen opciones para editar o eliminar las listas. Al final, agregué un historial de Checklists pasados.

En esta vista desarrollé una pequeña tabla de gestión de usuarios. Incluí nombres, correos electrónicos, roles (administrador o usuario) y botones para editar o eliminar. También añadí un formulario con configuraciones generales del sistema, como el nombre de la empresa y la zona horaria.

Ilustración 5: Vista de Checklists



Reportes – reportes.html

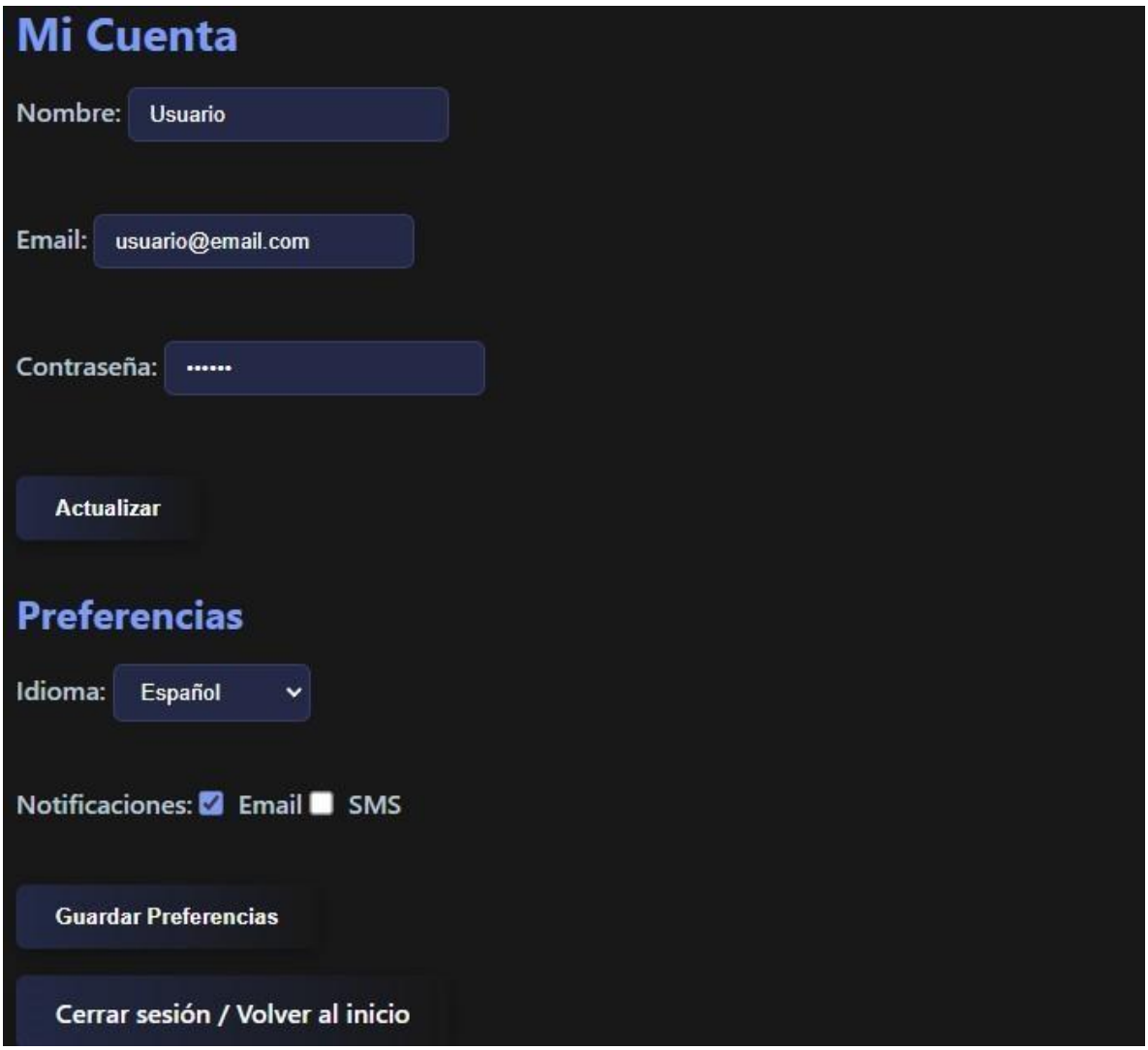
Diseñé un formulario para generar reportes basados en tareas o checklists, con la posibilidad de elegir un rango de fechas. También incluí un historial de reportes previos y una sección de estadísticas simples.

Ilustración 6: Vista de Reportes

Mi Cuenta – cuenta.html

Por último, hice una pantalla para que el usuario pueda modificar sus datos personales (como nombre, correo y contraseña), además de elegir su idioma preferido y para activar notificaciones. También se puede cerrar sesión desde aquí.

Ilustración 7: Vista Mi Cuenta



Diseño y estilo visual

Me aseguré de que todas las pantallas tengan un diseño coherente entre sí. Utilicé colores oscuros con detalles en azul, botones con sombras y bordes redondeados para que la navegación sea agradable. También seleccioné una fuente moderna (Poppins) para mejorar la lectura.

- **¿Qué herramientas utilice?**
  Todo el sistema fue hecho usando HTML para la estructura de las páginas y CSS para los estilos visuales. No usé plantillas ni sistemas externos, lo que me permitió entender mejor cómo organizar y dar forma a cada elemento por mí mismo. Aunque el proyecto aún no está conectado a bases de datos reales, el diseño simula bien cómo

funcionará el sistema final. Si vi varias formas de conectarlo a diseños predeterminado, pero quise hacerlo yo mismo, y que quedara tal y como quedo.

- **Enlaces entre vistas**
  Cada página tiene botones o enlaces que llevan a la siguiente, simulando un recorrido completo por el sistema. El flujo de navegación sigue un orden lógico: desde el inicio, al login, luego al panel, y de ahí a cada función específica.

Información General

- Tipo de sistema: Backend REST API
- Lenguaje principal: JavaScript (Node.js)
- Framework: Express.js
- Base de datos: MySQL (gestionada mediante Sequelize ORM)
- Documentación de API: Swagger

Tecnologías y Herramientas Utilizadas

Tabla 1: Tecnologías

Tecnología	Descripción
Node.js	Plataforma de ejecución para JavaScript del lado del servidor.
Express.js	Framework ligero para construir APIs REST de forma rápida y estructurada.
Sequelize	ORM que permite manipular bases de datos SQL sin escribir consultas SQL.
JWT	Mecanismo de autenticación segura basado en tokens JSON Web Token.
dotenv	Herramienta para manejar variables de entorno desde archivos .env.
Swagger	Framework para documentar, probar y visualizar la API de forma interactiva.

Estructura del Proyecto (src/)

- **controllers/:** Contiene la lógica de negocio. Procesa las peticiones como crear usuarios, obtener tareas, etc.

- **routes/**[1]**:** Define las rutas como /api/users, /api/tasks, etc. Enlaza las rutas con sus respectivos controladores.
- **models/:** Define los modelos Sequelize que representan tablas de la base de datos. Ejemplo: User (email, password, roleId...).
- **middlewares/:** Funciones que se ejecutan antes de responder, como validaciones, autorización o manejo de errores.
- **config/:** Archivos de configuración de la base de datos y parámetros generales del sistema.
- **swagger.js**: Configuración para generar documentación automática con Swagger.
- **.env:** Archivo de variables sensibles (host, puertos, claves). Nunca se debe compartir públicamente.

Flujo de Funcionamiento General

- El cliente realiza una petición HTTP (por ejemplo: POST /api/users).
- La ruta correspondiente definida en /routes/*.js captura la petición.
- Esta ruta invoca el controlador que contiene la lógica del negocio.
- El controlador usa Sequelize para interactuar con la base de datos.
- Si todo es correcto, se devuelve una respuesta adecuada al cliente.

---

[1] Corrección importante en /api/auth/register (Me estaba dando error, a la hora de registrar)

Inicialmente, el endpoint /register requería autenticación como administrador:

```
router.post(
  '/register',   protect,
authorizeRoles(1),   registerValidation,
validationMiddleware,
authController.register
);
```

Para permitir pruebas sin login, se modificó a:

```
router.post(
  '/register',   registerValidation,
validationMiddleware,   authController.register
);
```

Esto permite registrar usuarios desde Swagger sin autenticación previa durante el desarrollo.

- Las rutas protegidas requieren autenticación con JWT (token).

Documentación Swagger (API Interactiva)

- **Ruta de acceso en desarrollo:**
  http://localhost:3000/api-docs

Configuración básica:

```
const options = {
  definition: {     openapi: '3.0.0',     info: {       title: 'TaskWise API',
version: '1.0.0',       description: 'Documentación profesional de la API
TaskWise'
    },
    servers: [{ url: 'http://localhost:3000' }],
components: {       securitySchemes: {
bearerAuth: {          type: 'http',          scheme:
'bearer',          bearerFormat: 'JWT'
       }
     }
    }
  },   apis: ['./src/routes/*.js']
};
```

Swagger permite:

- Navegar visualmente por todos los endpoints disponibles.
- Ejecutar peticiones directamente desde el navegador.
- Probar rutas protegidas autenticándose con JWT.
- Facilitar el trabajo colaborativo entre desarrolladores, actuando como un manual técnico interactivo del backend.

Rutas del sistema

Tabla 2: Autenticación de Usuarios (Auth)

Método	Ruta	Descripción
GET	/api/auth/me	Obtener usuario autenticado
PUT	/api/auth/profile	Actualizar perfil del usuario
PUT	/api/auth/avatar	Subir o actualizar avatar
POST	/api/auth/register	Registrar nuevo usuario
POST	/api/auth/login	Iniciar sesión
POST	/api/auth/refresh-token	Obtener nuevo token JWT

¿Cómo obtener el token JWT?

Ejecutar una petición POST /api/auth/login con el siguiente JSON:

```
{
 "email": "admin@admin.com",
 "password": "123"
}
```

La respuesta incluye un accessToken que puede ser usado para autenticar otras rutas en Swagger.

Tabla 3: Dashboard – Reportes y Métricas

Ruta	Descripción
/api/dashboard/counts	Conteo de tareas por estado
/api/dashboard/completed	Tareas completadas en los últimos 7 días
/api/dashboard/upcoming	Tareas por vencer en próximos 7 días
/api/dashboard/overdue	Tareas vencidas

Tabla 4: Archivos

Ruta	Descripción
/api/files/{filename}	Descargar archivo previamente subido

Tabla 5: Notificaciones

Ruta	Descripción
GET /api/notifications	Obtener todas las notificaciones
PATCH /api/notifications/{id}/read	Marcar como leída
PATCH /api/notifications/read-all	Marcar todas como leídas

Tabla 6: Proyectos

Ruta	Método	Descripción
/api/projects	POST	Crear nuevo proyecto
/api/projects	GET	Obtener todos los proyectos
/api/projects/upload	POST	Crear proyecto con archivo
/api/projects/{id}	GET	Obtener proyecto por ID
/api/projects/{id}	PUT	Actualizar proyecto
/api/projects/{id}	DELETE	Eliminar proyecto

Tabla 7: Roles de Usuario

Ruta	Método	Descripción
/api/roles	POST	Crear nuevo rol
/api/roles	GET	Listar todos los roles
/api/roles/{id}	GET	Obtener rol por ID
/api/roles/{id}	PUT	Actualizar rol
/api/roles/{id}	DELETE	Eliminar rol

Tabla 8: Tareas

Ruta	Método	Descripción
/api/tasks	POST	Crear nueva tarea
/api/tasks	GET	Listar todas las tareas
/api/tasks/{id}	GET	Obtener tarea por ID
/api/tasks/{id}	PUT	Actualizar tarea
/api/tasks/{id}	DELETE	Eliminar tarea (soft delete)
/api/tasks/{id}/upload	POST	Subir archivo adjunto a la tarea

Seguridad

- JWT (JSON Web Token): sistema de autenticación por tokens.
- Middlewares de autenticación y autorización: verifican si el token es válido antes de acceder a rutas protegidas.

Tabla 9: Acciones más comunes en la API

Ruta	Método	Descripción
/api/users	POST	Crear un nuevo usuario
/api/login	POST	Iniciar sesión y obtener token JWT
/api/tasks	GET	Obtener tareas del usuario
/api/tasks	POST	Crear nueva tarea

Reflexión y aprendizaje del desarrollo de backend

Durante este avance, desarrollar el backend fue una experiencia de gran aprendizaje. Me permitió comprender la importancia de estructurar el código de forma ordenada y modular para facilitar su mantenimiento y comprensión. Implementé autenticación mediante tokens JWT para asegurar el acceso al sistema, y separé el backend del frontend, lo que aporta flexibilidad y escalabilidad al proyecto. Además, utilicé Swagger para documentar las funciones, lo que facilita la prueba y detección de errores. Este proyecto no solo cumplió con los objetivos académicos, sino que también me preparó para enfrentar desafíos reales en el desarrollo profesional.

Conclusión del diseño y funcionalidades finales de *TaskWise*

Diseño estructurado en frontend y backend

- El sistema se desarrolló aplicando una arquitectura cliente-servidor, donde el frontend se encarga de la interacción con el usuario y el backend de la lógica de negocio y persistencia de datos.
- Esta separación garantiza un diseño escalable, modular y fácil de mantener.

Backend sólido y seguro

- La capa de backend maneja la lógica de autenticación, gestión de usuarios, tareas y proyectos.

- Incluye control de acceso, validación de datos y organización en servicios, lo que asegura estabilidad y seguridad en la aplicación.
- El diseño de la base de datos permite relacionar usuarios con sus proyectos y tareas, ofreciendo un flujo de trabajo bien definido.

Frontend centrado en la experiencia del usuario (UX)

- El frontend presenta una interfaz intuitiva, con vistas organizadas que facilitan la creación, edición y seguimiento de tareas.
- El diseño visual permite al usuario gestionar de manera rápida sus proyectos y pendientes.

Funcionalidades clave

- Gestión de usuarios: registro, inicio de sesión y administración de cuentas.
- Gestión de proyectos y tareas: creación, edición, asignación y seguimiento de actividades.
- Organización y productividad: categorización de tareas, priorización y administración del tiempo.
- Interactividad: comunicación fluida entre frontend y backend mediante API REST.

Escalabilidad y adaptabilidad

- La arquitectura permite integrar futuras mejoras como notificaciones, integración con calendarios externos o colaboración en tiempo real.
- El diseño modular facilita extender la aplicación hacia dispositivos móviles o integraciones con terceros.

Vídeo del sistema final

https://drive.google.com/file/d/1_6CyCYns6TGULSTQTBYB7r05r27IcLyA/view?usp=sharing