DEPARTAMENTO DE ELECTROTECNIA E INFORMÁTICA

INGENIERÍA EN INFORMÁTICA

UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Desarrollo Front-End de un sistema web para la gestión de inventarios en comedores solidarios.

Diego Hidalgo Carvajal.

diego.hidalgoca@usm.cl

Profesor Guía: Pamela Gatica Caballero.

Resumen: El presente proyecto se enfoca en el desarrollo de un sistema web para la gestión de inventarios en comedores solidarios de la municipalidad de Viña del Mar, una solución destinada a optimizar el control y distribución de recursos alimentarios en beneficio de las comunidades más vulnerables. La problemática inicial se centró en la falta de trazabilidad y el manejo manual de los inventarios, lo que dificultaba la planificación, el seguimiento de los productos y aumentaba el riesgo de desperdicio alimentario.

Para abordar estas deficiencias, se diseñó e implementó StockNow, un sistema integral basado en tecnologías modernas que permiten actualizaciones en tiempo real y una experiencia de usuario optimizada. El sistema cuenta con funcionalidades clave como la visualización y control de stock, alertas automáticas sobre fechas de caducidad, planificación de entregas, gestión de usuarios con roles y permisos, y módulos informativos para la toma de decisiones mediante paneles de recolección de información. Este proyecto representa un avance significativo en la gestión de recursos alimentarios para comedores solidarios, impactando de manera positiva tanto en la comunidad como en la operatividad municipal.

Palabras Clave: Gestión de inventarios, comedores solidarios, angular, socket.IO, trazabilidad, aplicación web.

CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título; ☐ Tesis de Postgrado;

Título del trabajo: Desarrollo Front-End de un sistema web para la gestión de alimentos en comedores solidarios.

Nombre del candidato(a): Diego Maximiliano Hidalgo Carvajal

Carrera / Grado: Ingeniería en Informática.

Campus: Viña del Mar **;** **Departamento**: ELINF

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Pamela Gatica Caballero, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL

☒ El trabajo **NO contiene información que amerite confidencialidad** y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (embargo) por:

☐ 6 meses; ☐ 12 meses; ☐ 2 años; ☐ 3 años; ☐ 5 años; ☐ 10 años

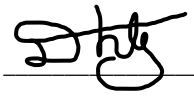
Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha:31/07/2025; **Firma:** 

Estudiante o Candidato(a):

Fecha: 28/07/2025 ; **Firma:** 

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

1. Introducción

1.1. Contexto

A lo largo del tiempo, se ha vuelto esencial para la ciudadanía recibir un apoyo de parte de los municipios y organizaciones en situaciones de crisis socioeconómicas o cuando los problemas de características naturales los afectan. Para las municipalidades cubrir esta problemática es prioridad ya que impacta de manera directa en el bienestar de la comunidad, alterando su vida cotidiana y el funcionamiento normal de la sociedad. Para que esta asistencia sea efectiva, es necesario que los recursos se planifiquen y gestionen de una manera adecuada. El servicio que ofrece la municipalidad de Viña del Mar es el de los comedores solidarios, que gestiona la entrega de suministros básicos a la ciudadanía, funcionando además como un banco de alimentos que recibe, almacena y distribuye productos.

Los comedores solidarios en Viña del Mar son servicios gestionados por la municipalidad, que se encargan de entregar alimentos a la comunidad y de registrar, almacenar y mantener estos productos en buen estado. Sin embargo, las bodegas que se encargan de este almacenamiento se distribuyen en diferentes ubicaciones de la ciudad, lo cual dificulta el control y registro centralizado de los elementos que ingresan y se utilizan en el servicio. La Municipalidad de Viña del Mar necesita implementar un sistema que solucione este problema de control y, al mismo tiempo, aborde los desafíos cotidianos asociados a la gestión de los almacenamientos. Un sistema más eficiente permitirá optimizar la distribución de alimentos, mejorar la trazabilidad y minimizar las pérdidas, garantizando así un mejor servicio a la comunidad.

1.2. Definición del problema

Actualmente, los encargados de los comedores solidarios registran manualmente el ingreso de productos en Excel, lo que limita la trazabilidad y dificulta el control de stock, ya que no se incorporan fechas de caducidad ni se realiza un seguimiento efectivo del estado de los alimentos. Esto incrementa el riesgo de pérdida de productos por caducidad y favorece errores humanos en el registro, lo cual dificulta tanto el análisis como la planificación. Como consecuencia, la comunidad puede verse afectada por la falta de alimentos frescos y variados, generando desconfianza en el sistema. Para resolver estos problemas, es crucial implementar un sistema de gestión de inventarios que permita un registro digital en tiempo real, controle las fechas de caducidad y facilite el análisis de datos, mejorando así la eficiencia operativa y garantizando un mejor servicio a la comunidad.

1.3. Propuesta de solución

Para abordar este desafío de ineficiencia en la gestión de inventarios de la municipalidad, se propone el desarrollo de un sistema web integral que permita a los operadores visualizar el estado de los productos en tiempo real. Esta solución busca optimizar la entrega de alimentos, asegurando un uso más efectivo de los recursos y minimizando el desperdicio alimentario. Este sistema se definirá como StockNow.

Lo que se incluye en este proyecto corresponde a:

- Desarrollo del sistema web para la gestión de inventarios.
- Funcionalidades de visualización y control de stock en tiempo real.
- Implementación de alertas automáticas sobre fechas de caducidad de los alimentos perecibles.
- Herramientas para la planificación de entregas de productos y gestión de bodegas.

- Módulo de gestión de usuarios con roles y permisos.

1.4. Objetivo General y Específicos

Objetivo General:

Desarrollar una interfaz web interactiva y responsiva para la gestión de inventarios en los comedores solidarios para la municipalidad de Viña del Mar, que permita visualizar y controlar en tiempo real el estado de los productos, mejorando la eficiencia en la entrega de alimentos y promoviendo una gestión más sostenible de los recursos.

Los **objetivos específicos** que el sistema web presenta son:

- **Implementar** un sistema de control de inventario con Angular que permita recuperar y presentar los datos almacenados en una base de datos de manera clara y accesible, asegurando una visualización efectiva de los productos disponibles y sus estados.
- **Desarrollar** una funcionalidad de planificación de entregas que permita programar y gestionar distribuciones futuras de productos, basándose en la disponibilidad en bodegas y las fechas de caducidad, para optimizar el uso de recursos y minimizar el desperdicio.
- **Establecer** un módulo de gestión de usuarios que administre el acceso al sistema, creando diferentes roles y permisos que garanticen que solo el personal autorizado pueda acceder a funcionalidades específicas, mejorando la seguridad del sistema.
- **Crear** una interfaz intuitiva para la visualización de bodegas que permita a los usuarios añadir, modificar o eliminar bodegas y espacios, mostrando información crítica sobre el estado de los productos en tiempo real, lo que facilitará la gestión y toma de decisiones.
- **Incorporar** la actualización en tiempo real de los datos del sistema mediante Socket.IO, asegurando que cualquier cambio en el inventario se refleje de inmediato en la base de datos, lo que permitirá a los usuarios tomar decisiones informadas y oportunas sobre la gestión del inventario.
- **Optimizar** la experiencia de usuario mediante el uso de buenas prácticas de diseño y usabilidad, asegurando que la interfaz sea accesible y fácil de usar para todos los operadores, independientemente de su nivel técnico.
- **Implementar** medidas de rendimiento que aseguren tiempos de carga rápidos y una navegación fluida dentro del sistema, mejorando la experiencia general del usuario y fomentando el uso efectivo de la aplicación.

1.5. Justificación de proyecto

Desde el punto de vista técnico, este proyecto es esencial debido a que aborda un problema crítico en la gestión de recursos alimentarios en situaciones de crisis. Al implementar un sistema web para la gestión de inventarios, los usuarios finales se beneficiarán de una solución que les permitirá gestionar los productos de manera más eficiente. La centralización de información, actualmente fragmentada y distribuida, no sólo optimiza el flujo de trabajo, sino que a su vez mejora la trazabilidad y seguimiento de los productos, garantizando una respuesta más rápida y efectiva ante cualquier necesidad de la comunidad.

El uso de tecnologías modernas como Angular o Socket.IO permitirá una construcción eficiente y escalable del desarrollo front-end del sistema. Esto es fundamental para facilitar el acceso y la visualización de datos en un entorno que requiere actualizaciones constantes. Al integrar un sistema de alertas para controlar las fechas de caducidad, se logra minimizar las pérdidas de productos lo cual es un avance significativo en la gestión de inventarios.

Beneficios:

Los beneficios esperados con este desarrollo son:

- **Eficiencia Operativa:** La automatización de procesos de registro y control de stock eliminará errores manuales, optimizando el tiempo de gestión.
- **Usabilidad:** La interfaz será diseñada para ser intuitiva, permitiendo a los administradores con diferentes niveles técnicos navegar sin dificultad, lo que facilitará la adopción del sistema.
- **Experiencia de usuario mejorada:** Al ofrecer visualización en tiempo real y alertas sobre productos, los usuarios podrán tomar decisiones informadas, lo que se traduce en un servicio más confiable para la comunidad.
- **Seguridad:** Con un módulo de gestión de usuarios, se garantiza que sólo el personal autorizado acceda a funciones críticas, lo que aumentaría la seguridad del sistema.

Innovación:

Este proyecto introduce varias innovaciones de características tecnológicas y metodológicas:

- **Framework moderno (Angular):** Utilizar Angular para el desarrollo front-end no sólo garantiza una experiencia de usuario fluida y responsiva, sino que también permite la integración de características avanzadas como la actualización en tiempo real de datos con Socket.IO, mejorando la interactividad del sistema.
- **Diseño progresivo y accesible:** La interfaz se diseñará con principios de accesibilidad, asegurando que sea usable por todos los administradores, independientemente de sus habilidades tecnológicas.
- **Optimización del rendimiento:** Se implementarán técnicas avanzadas para asegurar tiempos de carga rápidos y una navegación fluida, lo que es fundamental para mantener a los usuarios comprometidos y facilitar su trabajo diario.

1.6. Metodología

Para la construcción del front-end del sistema de gestión de inventarios, se ha utilizado la metodología ágil Scrum. Este método de trabajo permite establecer una manera más flexible de distribuir las entregas de resultados, generando una retroalimentación continua lo que es esencial en un entorno dinámico. El desarrollo se organiza en ciclos cortos de trabajo, llamados Sprint, que típicamente duran de dos a cuatro semanas. Cada sprint, gracias a una planificación previa, responde a las necesidades más relevantes para el usuario, priorizando las que aporten mayor valor. Al finalizar cada sprint, se lleva a cabo una revisión que permite proporcionar una retroalimentación inmediata con respecto a las funcionalidades presentadas. Esta revisión, finalmente facilita ajustes rápidos en el diseño y en las prioridades del producto para posibles nuevas iteraciones.

Dada la naturaleza colaborativa y la necesidad de adaptarse rápidamente a los cambios en un proyecto que involucra a múltiples partes interesadas, se concluye que Scrum es el enfoque más adecuado. Este método asegura que el producto evolucione de manera alineada con las expectativas y necesidades de los usuarios. Así, la implementación de Scrum permitirá un desarrollo más ágil y eficiente, garantizando un sistema que realmente responda a las demandas de los comedores solidarios.

El sistema realizado para este proyecto contempla 3 iteraciones, las cuales se contienen lo siguiente:

1. **Primera Iteración: Conexión con el back-end y visualización de productos**

En esta primera fase del desarrollo, el objetivo principal será establecer la conexión entre el back-end y la interfaz web, permitiendo que el sistema pueda visualizar los datos almacenados de productos. Se desarrollarán los siguientes puntos:

- **Conexión entre el back-end y la web:** Se implementará la comunicación entre ambos utilizando Socket.IO para datos en tiempo real, asegurando que la información cargada desde el servidor se muestre correctamente en la interfaz.
- **Visualización de la lista de productos:** Se creará una vista inicial en la cual los usuarios podrán consultar la lista de productos gestionados en el sistema, utilizando la tabla interactiva de PrimeNG para mostrar los datos de forma clara y dinámica.

2. Segunda Iteración: Autenticación y mejora visual

La segunda iteración se enfocará en mejorar la seguridad y la experiencia de usuario, añadiendo las siguientes características:

- **Autenticación y gestión de usuarios:** Se implementará un sistema de autenticación para el acceso al sistema, permitiendo a los usuarios crear cuentas, iniciar sesión y gestionar permisos. Se utilizarán servicios de autenticación y control de acceso para asegurar que los usuarios solo puedan realizar acciones según sus roles asignados.
- **Mejora visual del sistema:** Se trabajará en mejorar la interfaz de usuario, añadiendo estilos y elementos visuales mediante Bootstrap y PrimeNG, con el objetivo de proporcionar una experiencia más atractiva y fácil de usar.
- **Visualización de bodegas:** Se añadirá una vista que permita a los usuarios visualizar las diferentes bodegas del sistema, junto con sus productos y estados.

3. Tercera Iteración: Planificación, alertas, pruebas y validaciones.

La última iteración se centrará en añadir funcionalidades avanzadas para la gestión de inventarios y la finalización del sistema:

- **Sistema de planificación:** Se implementará un módulo de planificación que permitirá a los usuarios programar entregas y gestionar el uso futuro de productos. Este sistema afectará directamente los datos de inventario y será clave para la optimización de recursos.
- **Sistema de alertas:** Se incorporará un sistema de alertas que notificará a los usuarios sobre situaciones críticas como productos cercanos a caducar o niveles bajos de stock. Las alertas se integrarán con la visualización de productos y bodegas.
- **Validaciones y pruebas del producto:** Se realizarán los ajustes finales, incluyendo pruebas exhaustivas, corrección de errores y mejoras en la interfaz de usuario, preparando el sistema para su entrega y uso real en los comedores comunitarios.

1.7. Organización del informe en capítulos

Este informe se estructura de la siguiente manera:

Capítulo 1. Introducción: Presentación del contexto y relevancia del proyecto, definiendo el problema y la propuesta de solución. También se detallan los objetivos generales y específicos del sistema.

Capítulo 2. Marco teórico: Fundamentos del desarrollo front-end, la importancia de construcción del software, arquitectura de software, patrones de diseños que se utilizarán, tecnología y framework utilizado.

Capítulo 3. Diseño e implementación: En este capítulo se explicará la arquitectura y estructura de componentes, desarrollo y diseño de componentes, uso de buenas prácticas, integración con el sistema general, diseño de interfaces, entre otros procesos enfocados en el desarrollo del sistema.

Capítulo 4. Conclusiones.

Capítulo 5. Referencias.

2. Marco Teórico

2.1. Fundamentos del desarrollo front-end

El front-end de una aplicación o sistema constituye la parte visible que interactúa directamente con el usuario, siendo responsable de presentar la interfaz visual y de facilitar la interacción con el software. Según lo señalado en **[1]**, esta capa incluye elementos gráficos y de diseño que conforman la interfaz de usuario (UI), y su optimización busca ofrecer una experiencia de usuario (UX) eficiente y satisfactoria. Tecnologías como HTML, CSS y JavaScript son la base de esta capa, que también actúa como un puente de comunicación con el back-end, encargado del procesamiento de datos y la lógica del sistema.

Desde los inicios de la web, con páginas compuestas principalmente por texto plano en HTML, el front-end ha evolucionado para ofrecer experiencias más dinámicas y atractivas **[2]**. La introducción de CSS enriqueció la presentación visual, mientras que JavaScript revolucionó la interacción al permitir la creación de interfaces dinámicas **[3]**. Frameworks modernos como React, Angular y Vue.js han simplificado el desarrollo, impulsando aplicaciones más robustas y responsivas.

Hoy en día, tecnologías como React Native y Flutter permiten la creación de aplicaciones móviles nativas a partir de un único código base **[4]**, mientras que las Progressive Web Apps (PWAs) combinan lo mejor de las aplicaciones web y móviles **[5]**. Estas herramientas han expandido el alcance del desarrollo front-end hacia nuevos horizontes, adaptándose a diversas plataformas y dispositivos.

Las características clave del desarrollo front-end son la interactividad, responsividad y accesibilidad. La primera permite a los usuarios interactuar dinámicamente con la aplicación, mientras que la responsividad asegura que la interfaz se visualice adecuadamente en una variedad de dispositivos y tamaños de pantalla. La accesibilidad es fundamental para garantizar que todos los usuarios puedan utilizar la aplicación sin dificultades. A pesar de sus numerosas ventajas, como la mejora de la experiencia del usuario y la posibilidad de implementar actualizaciones rápidas, el desarrollo front-end también enfrenta desafíos, incluyendo problemas de rendimiento y la necesidad de pruebas extensivas en diferentes navegadores y dispositivos.

Dentro de la arquitectura de software, el front-end juega un rol crucial como intermediario entre el usuario y el back-end. Se encarga de la presentación de información y la comunicación con el back-end mediante API REST o GraphQL, permitiendo que las solicitudes del usuario sean procesadas y que las respuestas actualicen la interfaz de manera efectiva. Un front-end bien diseñado no sólo facilita la interacción del usuario, sino que también mejora la eficiencia operativa del sistema al garantizar que los datos se presenten de manera clara y que los usuarios puedan realizar acciones de forma intuitiva.

2.2. Arquitectura de software

La arquitectura que se implementará en este proyecto será la arquitectura de capa. La arquitectura en capas es ampliamente utilizada en sistemas de software debido a su capacidad para separar responsabilidades **[6]**. Esta separación permite que cada capa interactúe sólo con sus capas vecinas, reduciendo la complejidad del sistema y mejorando su mantenibilidad y escalabilidad. Las capas más comunes en esta arquitectura, mencionadas por su nivel de dependencia, son: capa de presentación (UI o front-end), capa de lógica de negocio (Aplicación) y capa de datos (Back-end).

La capa de presentación es la encargada de gestionar la interacción con el usuario. Define la estructura visual, la navegación y la forma en que se presentan los datos. Aunque suele estar implementada con tecnologías de interfaz como HTML/CSS, frameworks web o móviles, su verdadero rol es abstraer la lógica de visualización e interacción.

La capa de lógica contiene las reglas del negocio, que define cómo los datos pueden ser creados, almacenados, o manipulados. Esta capa toma decisiones importantes dentro de la aplicación, como la validación de credenciales, gestión de datos, entre otras cosas.

La capa de datos es la encargada de almacenar y recuperar datos. Esta interactúa con la base de datos, ya sea para leer, escribir o actualizar información, proporcionando métodos para que las capas superiores puedan acceder a ella.

2.3. Patrón de diseño

El patrón de diseño seleccionado para este proyecto es el MVC (Modelo-Vista-Controlador). El patrón Modelo-Vista-Controlador (MVC) es una metodología que organiza las aplicaciones en tres componentes principales **[7]**. Esta separación facilita la organización del código y mejora su mantenibilidad.

- **Modelo**: El modelo representa la estructura de los datos y maneja toda la lógica relacionada con su manipulación. Aquí se incluyen las clases que definen las entidades del negocio, como los productos o usuarios, y las operaciones que permiten crear, leer, actualizar y eliminar datos (CRUD). En otras palabras, el modelo es el encargado de gestionar el estado de la aplicación y sus reglas de negocio.
- **Vista**: La vista es la interfaz con la que interactúa el usuario. Su función es presentar los datos del modelo de forma accesible y atractiva, permitiendo una experiencia de usuario (UI) clara y funcional. La vista muestra la información de manera dinámica, actualizándose de acuerdo con los cambios en el modelo.
- **Controlador**: El controlador actúa como intermediario entre el modelo y la vista. Es responsable de recibir las interacciones del usuario a través de la vista, procesar esas acciones, y, según sea necesario, modificar el modelo. Después de actualizar el modelo, el controlador ordena a la vista que muestre la información más reciente, asegurando que la interfaz esté siempre sincronizada con el estado actual de los datos.

Ilustración 1: Patrón de diseño MVC.

Fuente: Elaboración propia.

2.4. Tecnologías y frameworks utilizados

El desarrollo del front-end de este proyecto se ha considerado diversas herramientas, tecnologías y frameworks para garantizar una experiencia de usuario fluida, eficiente y segura.

2.4.1. Framework y Tecnologías principales

- El proyecto utiliza Angular 17, un framework basado en TypeScript (un superconjunto de JavaScript) que facilita la creación de aplicaciones web dinámicas y escalables. Angular presenta una arquitectura modular, lo que permite el desarrollo estructurado y organizado de componentes reutilizables.
- Para el desarrollo se emplea HTML para la estructura de la página, CSS para el diseño y la presentación, y TypeScript para añadir tipado estático y mejorar la mantenibilidad del código.
- Se ha integrado PrimeNG, una biblioteca de componentes UI para Angular que ofrece una variedad de elementos interactivos y personalizables. Bootstrap también está presente para manejar el diseño visual y la presentación, añadiendo iconos a su vez a la interfaz a través de Bootstrap Icons.
- Socket.IO **[8]** es una biblioteca de JavaScript diseñada para gestionar la comunicación en tiempo real entre el front-end y el back-end. Permite que los datos se actualicen instantáneamente en la aplicación, lo cual es esencial para sistemas dinámicos como la gestión de inventarios.

Un socket es un canal de comunicación que conecta dos sistemas, en este caso, el cliente y el servidor. Una vez establecido, este canal permanece abierto, permitiendo el intercambio continuo de información sin necesidad de realizar múltiples solicitudes HTTP, lo que hace que las aplicaciones sean más rápidas y eficientes.

Socket.IO, construido sobre el protocolo WebSocket, añade funcionalidades adicionales que lo hacen más confiable, como soporte para reconexiones automáticas y compatibilidad con navegadores que no soportan WebSocket. Por ello, se considera adecuado utilizar esta herramienta ideal para gestionar flujos de datos bidireccionales y dinámicos en aplicaciones web modernas.

2.4.2. Herramienta de build

- En este caso, Angular 17 presenta una opción de despliegue de aplicaciones, por lo que no se requiere de un elemento externo para la conversión de aplicación a un paquete optimizado para servidores web.

2.4.3. Despliegue del proyecto

- Este proyecto finalmente será desplegado en Nginx **[9]**, un servidor web diseñado con el objetivo de ofrecer un alto rendimiento, eficiencia y capacidad para manejar conexiones de manera simultáneas con un bajo uso de recursos. También, puede actuar como un intermediario entre los usuarios y los servidores back-end, distribuyendo la carga de las peticiones del cliente en diferentes servidores.

3. Diseño del sistema

3.1. Diseño de Componentes

3.1.1. Estudio de distribución de usuarios

En el diseño del sistema general, se evaluó la distribución de los usuarios que se requieren en este, a partir de conversaciones y entrevistas realizadas con funcionarios de la municipalidad. Este análisis permitió identificar la necesidad de desarrollar dos sistemas distintos, un sistema web y un sistema móvil, cada uno diseñado específicamente para satisfacer las necesidades de diferentes tipos de usuarios y con roles claramente definidos.

El sistema web está orientado a los administradores, quienes requieren acceso a funcionalidades avanzadas y herramientas de gestión, mientras que el sistema móvil está destinado a los bodegueros, enfocado en tareas operativas relacionadas con los movimientos de productos e inventarios. Esta separación garantiza que cada sistema esté adaptado de manera óptima a los requerimientos específicos de sus respectivos usuarios.

3.1.2. Arquitectura de los componentes

Este sistema está implementado utilizando el framework Angular, el cual facilita desde el inicio una estructura ordenada y la adopción de buenas prácticas en la organización y almacenamiento del código. Angular permite la creación de componentes de manera eficiente mediante comandos de su CLI (Command Line Interface), generando automáticamente las carpetas y archivos necesarios según las funcionalidades que se deseen implementar.

Cada carpeta generada contiene archivos clave, como TypeScript, HTML y SCSS, que trabajan en conjunto para definir la lógica, la estructura visual y el diseño del componente. Este enfoque modular no solo simplifica el desarrollo y la codificación en herramientas como Visual Studio Code, sino que también asegura que el sistema sea más escalable y fácil de mantener, evitando la centralización del código y mejorando su legibilidad para futuras actualizaciones o nuevas funcionalidades.

En este proyecto, los principales módulos desarrollados incluyen componentes específicos diseñados para abordar distintas necesidades del sistema. Estos módulos están organizados de forma lógica y funcional, optimizando tanto la experiencia del usuario como la eficiencia operativa del sistema.

La definición de la arquitectura del sistema y sus componentes fue producto de un análisis detallado de los requerimientos funcionales obtenidos mediante entrevistas y sesiones de retroalimentación con los funcionarios municipales. A partir de este levantamiento, se identificaron los principales casos de uso (CU), que permitieron establecer las funcionalidades clave y estructurar los módulos del sistema. Entre estos casos de uso destacan:

CU01: Visualizar y filtrar productos del inventario.

CU02: Registrar ingreso de productos a una bodega.

CU03: Gestionar usuarios y asignar roles.

CU04: Planificar entregas semanales a comedores.

CU05: Recibir alertas de productos próximos a vencer.

CU06: Visualizar métricas del sistema para toma de decisiones.

Estos casos de uso guiaron el diseño de los componentes descritos en esta sección, asegurando que cada uno respondiera a una necesidad específica del sistema. De esta manera, la arquitectura en capas no solo obedece a criterios de diseño técnico, sino que refleja una alineación directa con los objetivos funcionales del proyecto y con las necesidades operativas detectadas durante el análisis de requerimientos.

Componentes de Autenticación y Usuario:

3.1.2.1.1. Login:

Este componente gestiona el acceso al sistema general, asegurando la protección de los datos y un control eficiente de los ingresos de usuarios. Su funcionalidad principal se basa en un formulario que recopila las credenciales necesarias para autenticar al usuario.

Además, este componente establece la conexión con el back-end para validar los datos ingresados y permitir el acceso seguro al sistema. Se gestiona la interacción con el back-end mediante el servicio `auth.service.ts`, encargado de realizar las peticiones HTTP necesarias para autenticar al usuario. Específicamente, se envía un formulario al servidor mediante una solicitud HTTP POST, en la cual se transmiten las credenciales del usuario para recibir una respuesta.

Adicionalmente, el componente establece una conexión con el archivo `auth-token.service.ts`, que es responsable de gestionar el token de autenticación. Cuando la petición al servidor es aprobada, el token recibido como respuesta se almacena en el localStorage. Este token no solo se utiliza para autenticar al usuario en futuras solicitudes, sino que también permite la inicialización de diferentes sockets que dependen de este servicio para establecer conexiones seguras y mantener la comunicación en tiempo real con el back-end.

3.1.2.1.2. Gestion-usuarios:

Diseñado principalmente para administradores, este componente permite gestionar los permisos y roles de los usuarios del sistema. A través de una interfaz intuitiva, los administradores pueden asignar diferentes niveles de acceso, así como realizar tareas de creación, edición y eliminación de usuarios. Esto garantiza un control jerárquico y seguro sobre las funcionalidades disponibles para cada usuario.

En esta área se establece la conexión con el servicio `auth.service.ts`, ya que este proporciona acceso a los usuarios registrados en el sistema a través de la misma URL base, utilizando diferentes rutas o endpoints para realizar operaciones específicas. Estas rutas están diseñadas para segmentar funcionalidades, permitiendo una organización clara y eficiente de las solicitudes hacia el servidor.

El servicio `auth.service.ts` incluye los métodos necesarios para realizar las operaciones CRUD (Create, Read, Update, Delete) del sistema de gestión de usuarios, facilitando la creación, consulta, actualización y eliminación de usuarios directamente desde el componente. De esta manera, el componente puede interactuar de forma dinámica con los datos del sistema, enfocándose tanto en aspectos funcionales como en áreas más sociales relacionadas con la gestión de usuarios dentro del sistema.

3.1.2.1.3. Perfil:

El componente de perfil proporciona una vista personalizada donde los usuarios pueden consultar y modificar su información personal registrada en el sistema. Además de ser una funcionalidad informativa, permite a cada usuario identificar su sesión activa y realizar cambios según sea necesario.

Este componente también se conecta con los servicios `auth.service.ts` y `token.service.ts`, ya que gestiona datos específicos del usuario autenticado. Realiza verificaciones constantes para validar la sesión y confirmar que el usuario cuenta con las credenciales apropiadas.

Estas comprobaciones se complementan con el archivo `auth.guard.ts`, el cual protege las rutas del sistema. Su función es asegurar que solo usuarios autenticados accedan a áreas restringidas, trabajando en conjunto con los servicios mencionados para validar el token almacenado y mantener la seguridad en cada interacción.

3.1.2.2. Componentes de Navegación:

3.1.2.2.1. Home:

Este componente representa la pantalla de inicio tras el login. Muestra información clave como la planificación semanal, sus detalles y los productos más recientes del inventario. Actúa como punto de acceso a otros módulos informativos, como el Dashboard.
Desde aquí se establece la conexión con `app.component.ts`, utilizando el token de autenticación para habilitar la comunicación segura con el back-end. En esta fase se inicializan los sockets principales del sistema: inventario (`socket-inventario.service.ts`), planificación (`planificación-socket.service.ts`) y envíos (`envios.component.ts`). Estos servicios permiten la actualización en tiempo real de los datos, una funcionalidad distintiva de *StockNow* frente a sistemas tradicionales que dependen exclusivamente de métodos HTTP como GET y POST.

3.1.2.2.2. Sidebar:

El componente Sidebar es un elemento central de navegación que conecta los diferentes módulos y vistas del sistema. Consiste en una barra de navegación que permite redirigir al usuario a las URLs correspondientes de manera eficiente, asegurando una experiencia de navegación intuitiva y fluida.

Este componente está diseñado para ser adaptable, con dos estados: colapsado o expandido, optimizando el uso del espacio en la interfaz según las preferencias del usuario. Asimismo, el Sidebar incluye funcionalidades críticas como la opción de cerrar sesión, asegurando el control de acceso y la seguridad del sistema.

La relación que este componente establece está directamente vinculada con `app.routes.ts`. En este archivo, se configuran las rutas que permiten la navegación del usuario a través de las diferentes vistas y componentes del sistema. Angular proporciona un sistema de enrutamiento eficiente, que facilita la redirección del usuario mediante las URLs definidas, asegurando que cada ruta esté asociada al componente correspondiente.

Este componente interactúa con el sistema de rutas para mover al usuario entre las distintas secciones del sistema, asegurando una experiencia de navegación fluida y organizada. Además, permite gestionar funcionalidades como rutas protegidas mediante guards y la

definición de parámetros en las rutas para personalizar la navegación en función del contexto o las acciones realizadas por el usuario.

3.1.2.3. Componentes de Inventario:

3.1.2.3.1. Tabla-Inventario:

Este componente es uno de los pilares del sistema, diseñado para proporcionar una visión detallada y organizada de la información relacionada con el inventario. Presenta datos clave como las ubicaciones de los productos, las fechas de caducidad, las fechas de ingreso, y los diferentes lotes registrados en el sistema. En consecuencia, este componente integra información tanto de las entradas realizadas desde la aplicación web como desde la aplicación móvil, garantizando una administración centralizada y eficiente del inventario.

Este componente establece una comunicación constante con el servicio socket-`inventario.service.ts`, el cual se encarga de gestionar la carga de productos y de mantener una escucha activa para eventos provenientes del servidor en tiempo real. Esta conexión permite que las actualizaciones en el inventario, como la incorporación de nuevos productos o cambios en el estado de los existentes, se reflejen de manera inmediata en la interfaz del usuario.

Para operaciones específicas, como modificaciones de datos en el inventario, eliminación de registros o actualizaciones relacionadas con mermas, el componente utiliza funciones dedicadas que envían solicitudes HTTP al back-end.

3.1.2.3.2. Bodegas:

Para soportar la escalabilidad del sistema, se desarrolló un componente específico para la gestión de bodegas. Este módulo permite a los administradores registrar y gestionar nuevos espacios de almacenamiento físico, ampliando las capacidades operativas del sistema de inventario. Además, este componente facilita el control de las ubicaciones donde se encuentran los productos, proporcionando una visión más precisa y consciente del uso del espacio disponible. La funcionalidad incluye la posibilidad de añadir, editar o eliminar bodegas, adaptándose a las necesidades de expansión y optimización del almacenamiento.

Este componente también mantiene una asociación directa con el servicio `socket-inventario.service.ts`, ya que en este se gestionan las variables relacionadas con el registro y almacenamiento de productos, incluyendo las operaciones vinculadas a las bodegas. Este servicio centraliza la lógica para manejar las peticiones relacionadas con las bodegas, permitiendo la consulta, actualización y administración de estas en base a identificadores referenciales (IDs). Adicionalmente, el servicio utiliza conexiones en tiempo real a través de sockets, lo que garantiza que cualquier cambio en las bodegas, como la creación, modificación o eliminación de registros, se refleje de manera inmediata en el sistema.

3.1.2.4. Componentes de Planificación y Envíos:

3.1.2.4.1. Planificación:

El componente de planificación permite organizar y gestionar las futuras entregas a los comedores solidarios, asegurando una distribución eficiente de los productos disponibles. Mediante su conexión con el servicio `planificación-socket.service.ts`, establece

una comunicación en tiempo real con el servidor para crear, modificar y monitorear planificaciones semanales. De igual forma, utiliza `enviar.service.ts` para enviar la información planificada al servidor mediante solicitudes HTTP POST, garantizando la integridad y sincronización de los datos. Este módulo es clave para estructurar productos, establecer prioridades y definir cronogramas, convirtiéndose en la base operativa para el sistema de envíos y la optimización logística.

3.1.2.4.2. Envíos:

Directamente relacionado con el componente de planificación, este módulo permite visualizar los datos de los envíos programados. Mediante una integración con el back-end, este componente ofrece la posibilidad de filtrar los envíos por días específicos, permitiendo a los usuarios monitorear el estado de cada envío en particular y asegurando una trazabilidad completa durante el proceso logístico.

La conexión principal entre los componentes previamente mencionados se establece mediante el servicio `envios.service.ts`. Este servicio actúa como el núcleo para gestionar y sincronizar los datos relacionados con los envíos que se realizan en el sistema. Específicamente, recibe y procesa las respuestas a las solicitudes generadas desde la aplicación móvil utilizada por los bodegueros.

En este servicio se manejan los datos de los envíos en sus diferentes estados: aquellos que están en proceso, en entrega, o que ya han sido finalizados. Toda esta información se gestiona en tiempo real mediante sockets, lo que permite que el sistema actualice continuamente el estado de los envíos. Esto asegura que los administradores puedan monitorear de manera precisa y constante el progreso de las entregas, identificando rápidamente cualquier problema o retraso en el flujo operativo.

3.1.2.4.3. Detalle-Envío:

Este componente amplía la información sobre un envío específico. A través de solicitudes al back-end, recupera datos detallados asociados al envío seleccionado, presentándolos de manera clara y descriptiva para facilitar la comprensión. Esta funcionalidad es clave para ofrecer a los usuarios una visión completa de los productos y detalles logísticos relacionados con cada operación de envío.

Para proporcionar información más detallada sobre un envío específico, se implementó un componente especializado denominado Detalle de Envío. Este componente está diseñado para conectarse con el servicio `envios.service.ts`, permitiendo realizar un filtrado basado en el ID del envío. De esta manera, el componente puede cargar y presentar al usuario una vista completa y detallada de los aspectos relacionados con ese envío en particular.
A través de esta funcionalidad, el componente gestiona la visualización de múltiples datos relevantes, como:

- **Detalles del envío:** Información general del envío, como fechas, solicitantes, autorizadores, y tiempo en envíos.
- **Incidentes registrados**: Reportes de problemas que hayan ocurrido durante el proceso de envío.
- **Movimientos de productos:** Registro de cambios o ajustes realizados en los productos asociados al envío.

- **Entrega a comedores solidarios:** Estado de la distribución y recepción de productos en los comedores asignados.
- **Degradación de productos:** Información sobre pérdidas o daños en los productos dentro de la carga, permitiendo una trazabilidad efectiva.

3.1.2.5. Componentes de Alertas y Solicitudes:

3.1.2.5.1. Solicitud-dialog:

Este componente implementa una alerta interactiva diseñada para los administradores del sistema. Su propósito es notificarles sobre solicitudes críticas enviadas desde el back-end, relacionadas con la aceptación o rechazo de un proceso dentro del sistema de gestión de inventarios. Estas solicitudes suelen corresponder al inicio de una carga física de productos realizada por los encargados de las bodegas, lo que marca un paso esencial en la cadena de recolección y envío de productos.

Dado su carácter prioritario, esta alerta requiere una acción inmediata por parte del administrador para garantizar que el proceso de gestión de inventarios continúe sin interrupciones. Esto asegura que los productos sean procesados y preparados eficientemente para su posterior distribución. La interfaz está diseñada para ser intuitiva, permitiendo al administrador tomar decisiones rápidas y fundamentadas, gracias a la presentación clara de la información relevante.

Para lograr esta funcionalidad, el componente se conecta directamente con el servicio `planificación-socket.service.ts`, el cual es responsable de gestionar en tiempo real la carga y el seguimiento del evento de solicitud que se espera en el sistema. Este servicio permite que la alerta sea emitida y visualizada en cualquier ubicación del apartado web, asegurando que los usuarios administrativos tengan acceso inmediato a la notificación, sin importar la sección en la que se encuentren. La integración del sistema de sockets garantiza la actualización constante y en tiempo real de las solicitudes, brindando una experiencia dinámica y eficiente dentro del sistema.

3.1.2.6. Componentes Informativos:

3.1.2.6.1. Dashboard:

El componente Dashboard cumple un rol esencial al proporcionar una visión general e informativa del sistema, diseñada específicamente para facilitar la toma de decisiones por parte de los usuarios. Este módulo presenta una recopilación de datos clave en forma gráfica, permitiendo establecer relaciones directas con los productos del inventario y otros elementos relevantes del sistema.
Los datos mostrados en el Dashboard se obtienen mediante servicios y conexiones con el back-end, asegurando que la información presentada esté siempre actualizada con el servidor. Este componente utiliza gráficos dinámicos e intuitivos para proporcionar a los usuarios una visualización clara de métricas clave, como el estado del inventario, las tendencias en los envíos, la optimización de recursos y la eficiencia general en la gestión del sistema.

La actualización de los gráficos se logra mediante solicitudes HTTP de tipo GET, configuradas directamente en este componente. Estas solicitudes están asociadas a un rango de tiempo específico, lo que permite al usuario cargar y analizar los datos correspondientes al periodo seleccionado. Este enfoque no solo garantiza que la información sea relevante y precisa, sino que también ofrece flexibilidad al usuario para ajustar el rango temporal y obtener una perspectiva más detallada de las métricas operativas.

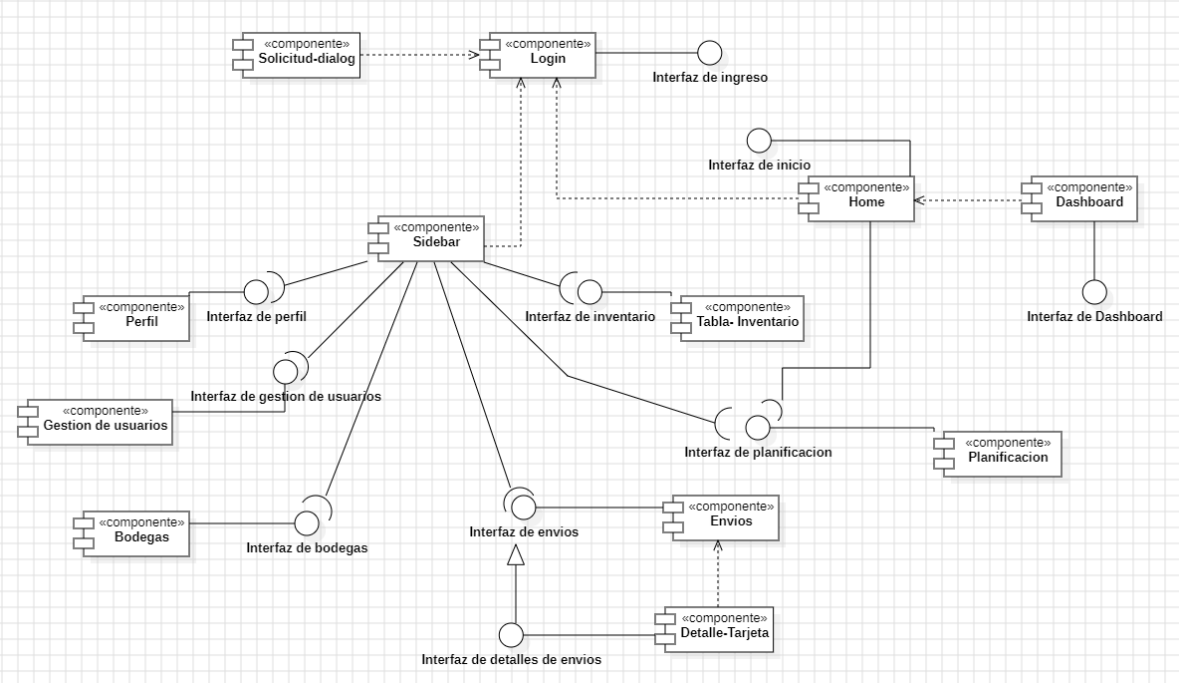


Ilustración 2: Diagrama de componentes de Angular**.**

Fuente: Elaboración propia**.**

3.1.3. Patrones de diseño

Para garantizar la calidad, organización y mantenibilidad del código, esta aplicación Angular adopta la arquitectura MVC (Modelo-Vista-Controlador), segmentando el sistema en tres capas bien definidas que permiten una clara separación de responsabilidades.

3.1.3.1. Modelo:

Esta capa representa la estructura y lógica de los datos, definiendo las clases y objetos que corresponden a las entidades del negocio, como productos o usuarios. En este proyecto, los modelos se implementan en TypeScript, lo que asegura que los datos puedan ser distribuidos y almacenados correctamente junto con la información que se recibe desde el back-end. Estos modelos están organizados en archivos específicos, diseñados para gestionar de manera uniforme y efectiva el uso constante de los datos en toda la aplicación.

A continuación, se muestra una captura de los modelos utilizados en los distintos componentes y servicios, destacando su agrupamiento para mantener una estructura consistente y limpia en el archivo general del proyecto:

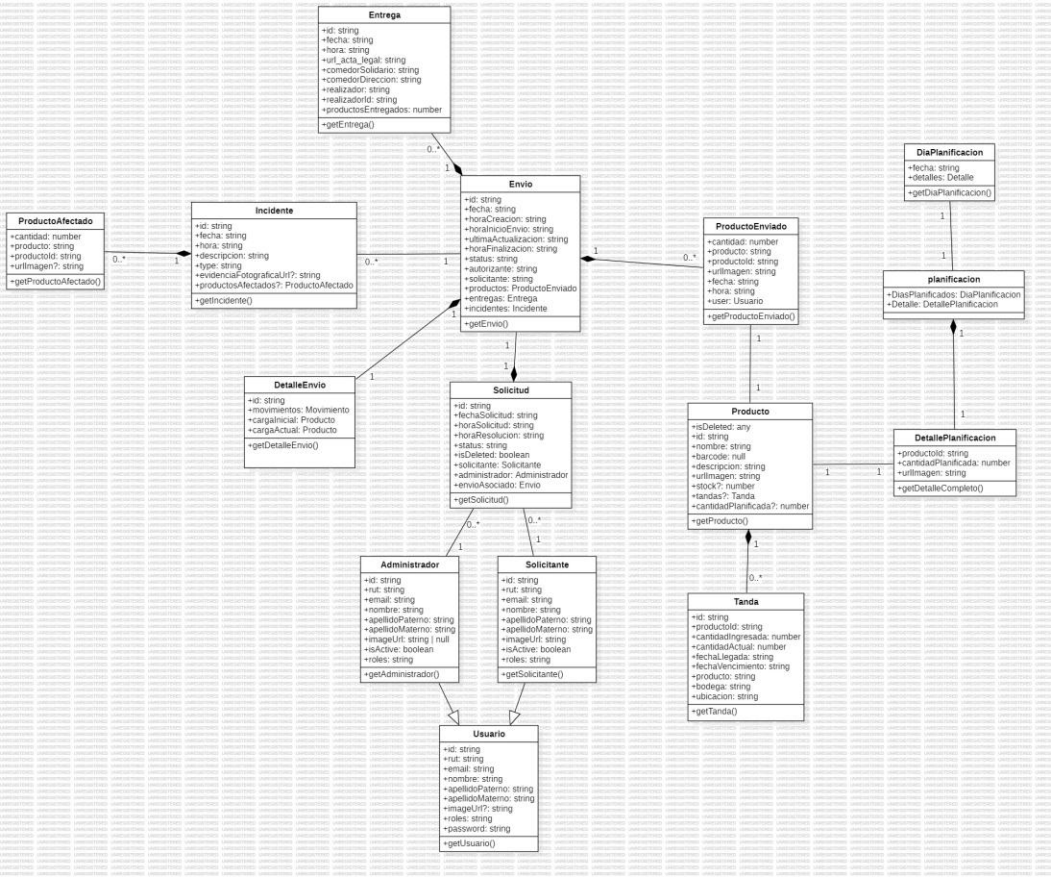


Ilustración 3: Diagrama de clases**.**

Fuente: Elaboración propia**.**

Los modelos mostrados en la imagen representan las distintas clases utilizadas por la aplicación web para procesar y manejar los datos recibidos desde el servidor. Por ejemplo, los modelos tanda y producto se destacan por ser ampliamente utilizados en diversas ubicaciones del proyecto debido a que comparten un formato uniforme de recepción de datos. Este enfoque asegura consistencia y simplicidad en el manejo de información. Sin embargo, no ocurre lo mismo con los modelos como envío, día-envío y detalle-envío, ya que, al recibir múltiples variantes de clases desde el servidor, se optó por crear modelos específicos para cada caso. Esto permite evitar inconsistencias o pérdida de información, destinando cada modelo a vistas concretas con necesidades de datos particulares.

El modelo usuario se relaciona directamente con la gestión de usuarios en el sistema. Este modelo es único, ya que se utiliza tanto para representar al usuario que realiza el ingreso a la aplicación como a aquellos que pueden ser gestionados dentro de la aplicación. La decisión de emplear un solo modelo en este caso asegura un manejo uniforme y claro de la información relacionada con los usuarios.

Por último, el modelo solicitud-data está diseñado específicamente para gestionar la comunicación con el servidor en lo que respecta a solicitudes de diálogo. Este modelo es esencial para manejar

eventos enviados a través de sockets, asegurando que los datos recibidos sean validados y presentados correctamente al usuario en tiempo real.

3.1.3.2. Vista:

Encargada de la interfaz de usuario, esta capa incluye plantillas HTML y SCSS que presentan la información de manera clara y atractiva, capturando las interacciones del usuario. En Angular, las vistas se implementan mediante componentes que combinan estas plantillas con una clase en TypeScript que controla su comportamiento.

Estas clases están directamente vinculadas al modelo y al controlador, asegurando una comunicación eficiente. En este proyecto, la vista juega un papel fundamental al manejar conexiones constantes con el servidor, lo que permite actualizaciones dinámicas y en tiempo real de la interfaz, brindando una experiencia fluida y eficiente al usuario.

Los principales archivos que fueron creados para la vista asociada al usuario se encuentran ubicados en las respectivas carpetas de cada componente, donde se guardan con relación al nombre del componente y su respectivo archivo SCSS.

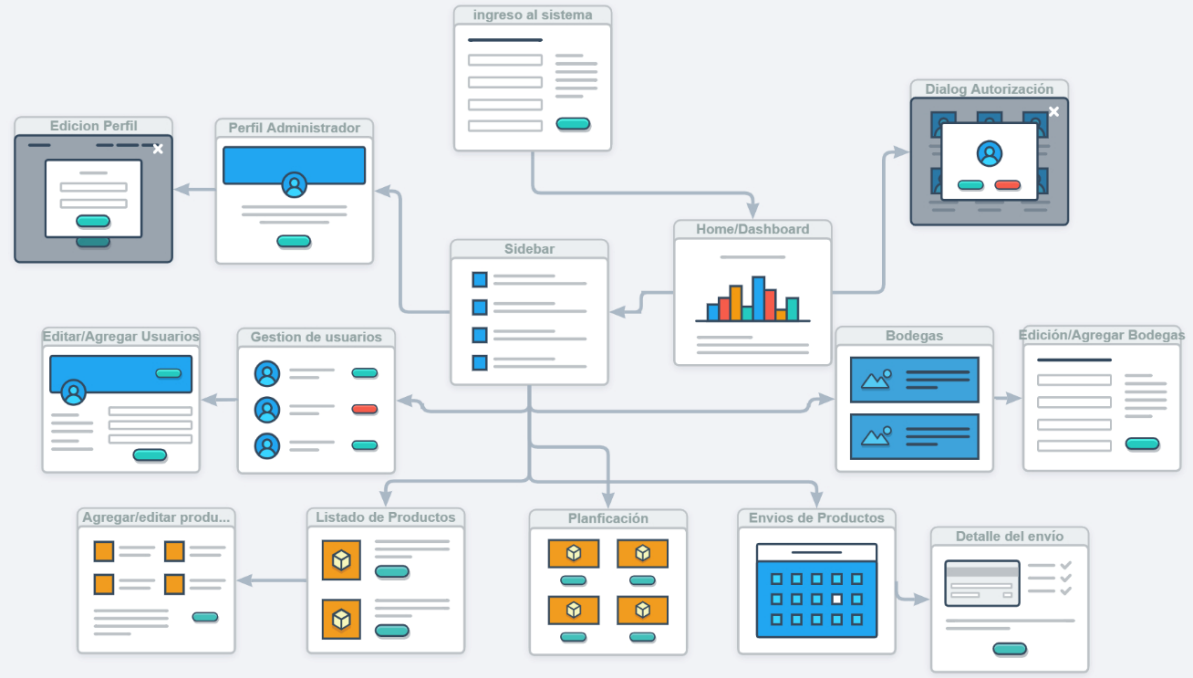


Ilustración 4: Wireflow de vistas de la aplicación.

Fuente: Elaboración propia.

3.1.3.3. Controlador:

Actuando como intermediario entre la vista y el modelo, el controlador gestiona la lógica de negocio y las interacciones del usuario. En Angular, esta función se implementa principalmente mediante servicios y componentes, los cuales procesan eventos desde la vista, acceden al modelo y actualizan la interfaz dinámicamente.

En este proyecto, los controladores están organizados en una carpeta dedicada de servicios en TypeScript, que centralizan las operaciones relacionadas con las peticiones al servidor y la gestión de datos del cliente. Esta estructura permite un flujo de datos eficiente y dinámico, asegurando una comunicación clara entre las distintas capas del sistema.

Cada componente en el sistema cuenta con su archivo TypeScript correspondiente, que se encarga de gestionar y controlar los datos necesarios para su procesamiento. Estos datos son utilizados tanto para enviar solicitudes y recibir respuestas desde el back-end como para actualizar dinámicamente la vista del usuario.

En consecuencia de los componentes, el proyecto incorpora un sistema de archivos dedicado a los servicios, que se encuentran en una ubicación separada dentro de la estructura del proyecto. Estos servicios son responsables de realizar las solicitudes al servidor, manejar las respuestas y procesar los datos necesarios para el funcionamiento del sistema.

Esta arquitectura no solo garantiza una clara separación de responsabilidades, sino que también facilita la reutilización del código, contribuyendo a un diseño más modular y mantenible. Constantemente estos servicios son requeridos por los componentes creados, los que son los que comienzan y habilitan la conexión en tiempo real con el servidor.

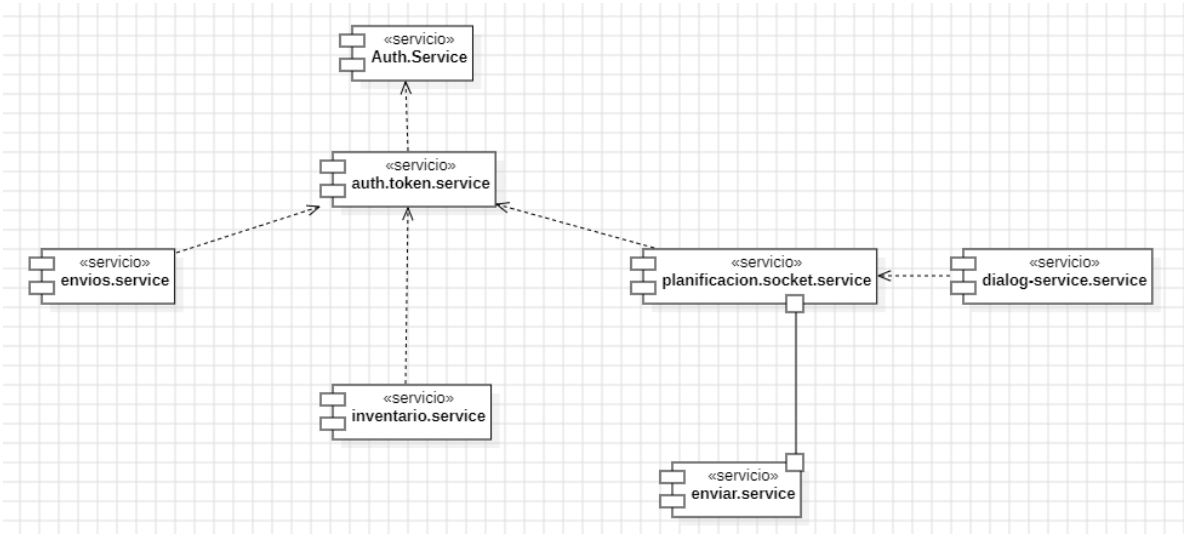


Ilustración 5: Diagrama de componentes (Servicios).

Fuente: Elaboración propia.

Los servicios mostrados en la imagen anterior se dividen en dos categorías principales según su ubicación y función dentro del proyecto. Los archivos dentro de la carpeta Socket corresponden a servicios diseñados para gestionar llamadas al servidor a través de endpoints definidos. Estos servicios establecen la comunicación en tiempo real entre la aplicación y el servidor, facilitando la transmisión constante de datos.

Por otro lado, los archivos TypeScript ubicados en la carpeta principal de services, pero fuera de la carpeta Socket, se definen como servicios locales. Estos servicios dependen de las conexiones

establecidas por los servicios de Socket para recolectar información, procesarla y estructurarla de manera adecuada.

Posteriormente, estos datos se presentan al usuario en respuestas específicas activadas por los componentes. Esta separación funcional asegura una gestión eficiente de los datos, optimiza la organización del código y facilita la interacción entre la lógica del servidor y la experiencia del usuario.

3.1.4. Integración con el sistema general

En este proyecto, el enfoque principal está en establecer conexiones directas y centralizadas exclusivamente en el sistema web mediante la implementación de Socket.IO. Esto permite que el front-end genere una conexión bidireccional con el servidor, facilitando un flujo constante de datos en tiempo real.

A través de eventos asociados a diferentes endpoints, el sistema puede recolectar y procesar datos instantáneamente, adaptándose a los cambios realizados en el servidor. Estos eventos son el medio principal de comunicación entre el servidor y el cliente, actuando como gatillos que activan una serie de acciones y operaciones específicas dentro del sistema.

La conexión con los sockets está diseñada para ser gestionada desde los servicios específicos ubicados en la carpeta services/sockets, lo que centraliza la lógica de interacción con el servidor. Sin embargo, para que estas conexiones sean funcionales, es fundamental inicializarlas al comienzo de la aplicación. Esto se debe a la configuración del servidor, que requiere un token de autenticación válido para permitir el acceso a los diferentes servicios.

Este token se obtiene mediante un método HTTP POST a la URL `http://<API_URL>/api/auth/login`, que valida las credenciales del usuario y devuelve, en caso de éxito, un objeto que incluye el token junto con la información del usuario.

```
// Método de login con actualización del BehaviorSubject
login(email: string, password: string): Observable<any> {
  return this.http.post<any>(`${this.apiUrl}/login`, { email, password }).pipe(
    tap(response => {
      console.log('Respuesta de login:', response); // Para verificar la respuesta completa en consola
      if (response && response.token) {
        this.tokenService.setToken(response.token); // Guarda el token usando `response.token`
        this.setUser(response); // Actualiza el usuario en el BehaviorSubject
      } else {
        console.error('No se recibió un token en la respuesta de login');
      }
    })
  );
}
```

Ilustración 6: Captura de pantalla de código de Aplicación**.**

Fuente: Elaboración Propia.

El token obtenido es esencial para inicializar los diferentes sockets, ya que se incluye como parte de la autenticación en cada conexión. Esto asegura que el sistema sea seguro y que las operaciones de comunicación en tiempo real estén debidamente autorizadas. Esta arquitectura permite al sistema mantener un flujo dinámico y confiable de información entre el cliente y el servidor, garantizando que los datos sean actualizados y gestionados de manera eficiente.

En la imágen mostrada, se evidencia que al realizar una solicitud mediante el método POST, el servidor responde con los datos del usuario junto con el token de autenticación correspondiente. Este token es

procesado y almacenado en el localStorage a través del servicio de gestión de tokens. Este procedimiento garantiza que el sistema pueda acceder al token de manera eficiente, evitando la necesidad de realizar reiteradas llamadas a la función de inicio de sesión (login) para cada operación que requiera autenticación.

Este enfoque facilita la autenticación para otras partes de la aplicación, como la inicialización de sockets o solicitudes protegidas. Al centralizar y automatizar el acceso al token mediante el servicio de tokens, se asegura un flujo más fluido y seguro dentro de la arquitectura del sistema.

Una vez obtenido el token de autenticación, se pueden inicializar los tres servicios principales del sistema que mantienen una comunicación constante con el servidor: Inventario, Planificación y Envíos. Cada uno de estos servicios utiliza un socket de comunicación basado en socket.io para gestionar la conexión bidireccional con el servidor y procesar eventos en tiempo real.

3.1.4.1. Servicio de Inventario:

El servicio de inventario, que se considera el más relevante dentro del sistema, se inicializa a través de un nuevo socket configurado en el archivo `app.config.ts`. Este socket utiliza el endpoint `http://<API_URL>/inventario` y requiere el token almacenado en el localStorage para establecer la conexión con el servidor.

3.1.4.1.1. Configuración del socket:

Se importa la librería socket.io y se establece la URL junto con las credenciales necesarias (token).

3.1.4.1.2. Mensaje de inicio:

El primer mensaje que inicia la conexión con el servidor es `getAllProducts`. Este mensaje, junto con el token, permite al servidor identificar al cliente y procesar la solicitud inicial.

3.1.4.1.3. Eventos de respuesta:

Para manejar las respuestas del servidor, el sistema escucha cuatro eventos principales:

loadAllProducts: Envía todos los productos disponibles en el inventario como respuesta inicial al mensaje `getAllProducts`.
stockProductoChange: Notifica y actualiza cualquier cambio en el stock de productos en tiempo real.
newTandaCreated: Añade nuevos productos al listado del inventario cuando se crea una nueva tanda.
newTandaUpdate: Actualiza los datos de las tandas existentes en el inventario.

A continuación, se presenta una captura desde la herramienta Postman, utilizada para simular y probar la conexión con el servidor a través de los sockets. En esta imagen se ilustran los mensajes de inicio y los eventos de respuesta que maneja el sistema de inventario. Esta herramienta fue clave durante el desarrollo para validar que los eventos como `loadAllProducts`, `stockProductoChange`, `newTandaCreated` y `newTandaUpdate` se recibieran correctamente, permitiendo verificar la lógica del flujo de datos en tiempo real.

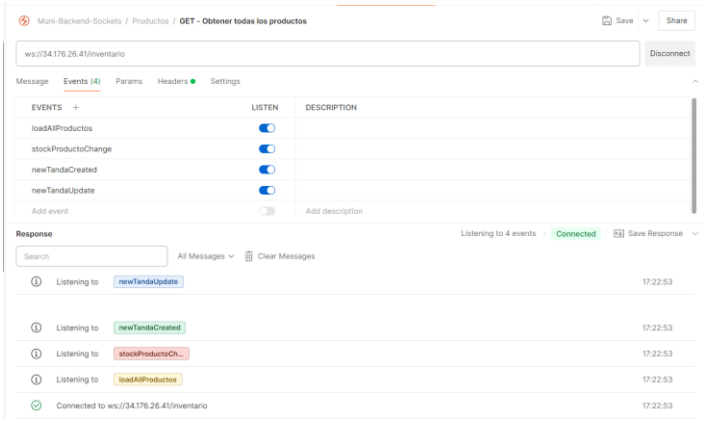


Ilustración 7: Ejemplo visual de **POSTMAN** enfocado en eventos**.**

Fuente: Elaboración propia**.**

La comunicación entre el cliente y el servidor incluye mensajes clave para gestionar datos específicos del sistema en tiempo real. Uno de ellos es `getTandasByIdProducto`, que se envía al servidor junto con un identificador de producto en formato JSON, como {"idProducto": "(id del producto)"}. Este mensaje genera una respuesta dinámica con el evento (idProducto)-tanda, que devuelve todas las tandas o lotes asociados al producto solicitado.

Además, este proceso está vinculado al evento `newTandaCreated`, que permite agregar nuevas tandas al listado de manera dinámica, asegurando la actualización automática en la interfaz web. Por otro lado, para gestionar las bodegas, el mensaje `getAllBodegas` solicita al servidor la lista completa de estas, la cual es entregada mediante el evento `loadAllBodegas`.

De manera similar, el mensaje `getUbicacionesByBodega` permite obtener las ubicaciones específicas de una bodega al enviar un objeto con su identificador, como {"idBodega": "(id de la bodega)"}, y recibir la respuesta a través del evento dinámico (idBodega)-ubicaciones, que contiene los datos requeridos. Estos mensajes y eventos trabajan en conjunto para garantizar un flujo de información eficiente y una experiencia de usuario dinámica y actualizada.

3.1.4.2. Servicio de Planificación:

Este nuevo servicio, se gestiona directamente en dos áreas de la aplicación, la primera en el apartado home para mostrar algunos datos informativos al usuario, mientras que la otra en su respectivo apartado de planificación. Este servicio se conecta a través del endpoint <API_URL>/planificacion, y al igual que el anterior, también requiere del token asociado al usuario para funcionar.

3.1.4.2.1. Configuración del socket:

Se importa la librería socket.io en el archivo de servicio asociado a la planificación y se establece la URL junto con las credenciales necesarias.

3.1.4.2.2. Mensaje de inicio:

El mensaje inicial para gestionar la planificación semanal es `adminPlanificacionManage`, que establece la conexión con el servidor utilizando un token de autenticación y un objeto JSON que define un rango de fechas específico. Estas fechas deben cumplir con la condición de ser únicamente días hábiles, es decir, de lunes a viernes. Por ejemplo, un mensaje válido incluiría {"inicio": "2024-10-21", "fin": "2024-10-25"}. Este mensaje es fundamental para activar la funcionalidad de planificación en el servidor, asegurando que las fechas proporcionadas se ajusten a los parámetros de una semana laboral y permitiendo organizar las operaciones dentro de ese marco.

Complementariamente, el mensaje `getPlanificacion` permite obtener detalles más específicos sobre un día planificado en particular. En este caso, el JSON necesario incluye solo la fecha requerida, por ejemplo, {"fecha": "2024-11-21"}. Este segundo mensaje facilita una visión más detallada de las actividades planificadas para un día específico, fortaleciendo el proceso de planificación semanal.

3.1.4.2.3. Eventos de respuesta:

Para manejar las respuestas del servidor en el proceso de planificación, el sistema escucha tres eventos principales. El primero, `loadAdminPlanificacionManage`, responde al mensaje `adminPlanificacionManage`, enviando los productos asociados a la semana específica que fue solicitada. El segundo, `loadPlanificacion`, está relacionado con el mensaje `getPlanificacion` y envía los productos planificados para el día en particular requerido, proporcionando un detalle más específico.

Por último, el evento `loadSolicitud` es uno de los más relevantes en términos de funcionalidad, ya que se encarga de gestionar la muestra de inicialización de envíos. Este evento es crítico para el flujo del sistema, ya que los usuarios deben estar preparados para recibirlo constantemente en el servicio de planificación y actuar en consecuencia.

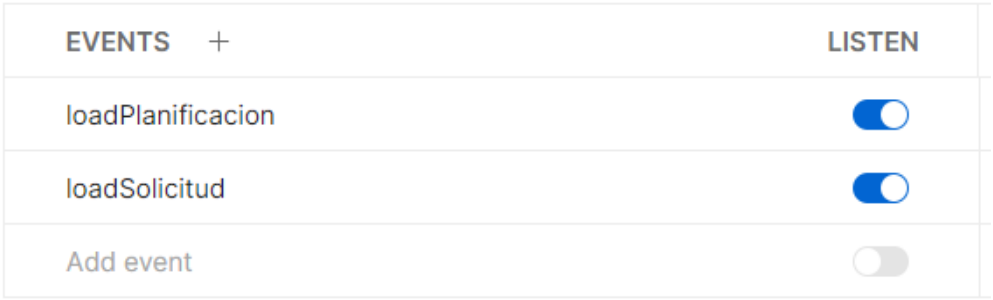


Ilustración 8: Captura de pantalla de **POSTMAN** asociado a eventos de socket**.**

Fuente: Elaboración propia**.**

3.1.4.3. Servicio de envíos:

Este servicio se gestiona en su respectivo apartado de envíos. Este servicio se conecta a través del endpoint http://<API_URL>/logistica/envios, y al igual que el resto, también requiere del token asociado al usuario para funcionar.

3.1.4.3.1. Configuración del socket:

Se importa la librería socket.io en el archivo de servicio asociado a los envíos y se establece la URL junto con las credenciales necesarias.

3.1.4.3.2. Mensaje de inicio:

El mensaje inicial para gestionar los envíos de manera diaria es `getEnviosByFecha`, el cual se envía junto con un JSON que incluye la fecha requerida para cargar los envíos correspondientes como, por ejemplo: {"fecha": "2024-11-14"}. Este mensaje permite al sistema obtener información específica sobre los envíos programados para esa fecha.

Asimismo, existe otro mensaje de inicialización, `getEnvioById`, que se utiliza para solicitar detalles de un envío en particular. Este mensaje incluye un JSON con el identificador del envío: {"idEnvio": "(Id del envío deseado"}, lo que permite recibir información detallada y específica sobre los datos asociados a dicho envío. Ambos mensajes son fundamentales para gestionar y visualizar las operaciones de envío de manera precisa y eficiente.

3.1.4.3.3. Eventos de respuesta:

Los eventos de respuesta asociados a los mensajes de gestión de envíos son `loadEnviosByFecha` y `{idEnvio}-loadEnvioById`. El evento `loadEnviosByFecha` se activa como respuesta al mensaje `getEnviosByFecha`, enviando todos los envíos programados para la fecha solicitada en el JSON.

Por otro lado, el evento dinámico `{idEnvio}-loadEnvioById` responde al mensaje `getEnvioById`, proporcionando los detalles específicos del envío identificado por el idEnvio en el mensaje. Estos eventos garantizan la actualización precisa de la información de envíos, tanto a nivel general como detallado.

3.1.4.4. Métodos HTTP:

Aunque la mayoría de los datos están centralizados en los sockets, los métodos HTTP son esenciales para gestionar la funcionalidad del sistema. Estos métodos permiten realizar operaciones como GET, POST, PATCH, y DELETE para optimizar la interacción con el servidor y garantizar un servicio eficiente. A continuación, se describe cada método y sus principales funcionalidades:

- **Métodos GET:**

http://<API_URL>/api/inventario/infoCharts?fechaInicio=2024-01-01&fechaFin=2024-12-31	Datos del dashboard
http://<API_URL>/api/auth/usuarios	Listar usuarios

- **Métodos POST**

http://<API_URL>/api/inventario/tandas	Crear tandas
http://<API_URL>/api/inventario/productos	Crear productos

http://<API_URL>/api/auth/register	Registrar usuarios
http://<API_URL>/api/auth/token/renew	Renovar token
http://<API_URL>/api/inventario/ubicaciones	Crear ubicaciones
http://<API_URL>/api/inventario/bodegas	Crear bodegas

- **Métodos PATCH**

http://<API_URL>/api/inventario/productos/IdProductoSeleccionado/update	Actualizar productos
http://<API_URL>/api/inventario/tandas/IdTandaSeleccionada/update	Actualizar tandas
http://<API_URL>/api/auth/IdUsuarioSeleccionado/update	Actualizar usuarios
http://<API_URL>/api/inventario/ubicaciones/idUbicacionSeleccionada/update	Actualizar ubicaciones
http://<API_URL>/api/inventario/bodegas/IdBodegaSeleccionada/update	Actualizar bodegas

- **Métodos DELETE**

http://<API_URL>/api/inventario/productos/IdProductoEliminado/delete	Eliminar productos
http://<API_URL>/api/auth/IdUsuarioEliminado/delete	Eliminar usuarios
http://<API_URL>/api/inventario/ubicaciones/IdUbicacionEliminada/delete	Eliminar ubicaciones
http://<API_URL>/api/inventario/bodegas/IdBodegaEliminada/delete	Eliminar bodegas

Estos métodos HTTP complementan la estructura del sistema, permitiendo la gestión eficiente de datos y operaciones clave. La combinación de GET, POST, PATCH, y DELETE asegura que el sistema pueda realizar tareas dinámicas como consultar, crear, actualizar y eliminar recursos de manera efectiva, optimizando la funcionalidad general del servicio.

3.2. Diseño de la Interfaz

En este proyecto, el diseño de la interfaz de usuario (UI) fue desarrollado con un enfoque en la simplicidad, la usabilidad y la adaptabilidad, teniendo en cuenta la importancia de brindar una experiencia de usuario (UX) óptima. Al no contar con un encargado específico de UI/UX en el equipo, el diseño se elaboró de manera colaborativa entre los desarrolladores, utilizando herramientas y metodologías que permitieran estructurar y validar las interfaces antes de su implementación.

3.2.1. Estructura y diseño de la interfaz de usuario (UI)

La estructura de la UI se diseñó siguiendo un enfoque modular y organizado, utilizando componentes reutilizables que no solo reducen la redundancia en el código, sino que también aseguran una consistencia visual y funcional en todo el sistema. Este enfoque permite que los elementos y secciones

compartan un lenguaje de diseño uniforme, facilitando tanto el mantenimiento del proyecto como la experiencia del usuario final.

Cada sección del sistema fue diseñada para cumplir objetivos específicos relacionados con las necesidades del usuario. Por ejemplo:

- **Gestión de inventarios:** Permite a los usuarios ver, agregar y actualizar los productos mediante una tabla dinámica que incluye filtros, búsquedas, y un sistema de alertas que permite indicar al usuario si un producto está próximo a vencer.
- **Visualización de datos en tiempo real:** Incluye dashboards interactivos que muestran gráficos y estadísticas actualizadas constantemente.
- **Navegación intuitiva:** Un menú lateral (sidebar) que se permite adaptar el tamaño de pantalla al usuario entre las diferentes secciones del sistema para brindar mayor visibilidad.

Se optó por un enfoque minimalista en el diseño de la interfaz, priorizando la claridad de los elementos visuales para evitar sobrecargar al usuario con información innecesaria. Las interfaces están estructuradas de manera que las acciones principales sean accesibles con el menor número de clics posible, garantizando un flujo de trabajo eficiente. Los colores y elementos visuales fueron seleccionados para crear una jerarquía visual clara, donde las acciones más importantes destacan sobre las demás.

La paleta de colores seleccionada para el proyecto utiliza tonos de azul como base principal, combinados con colores complementarios como el blanco y un gris claro.

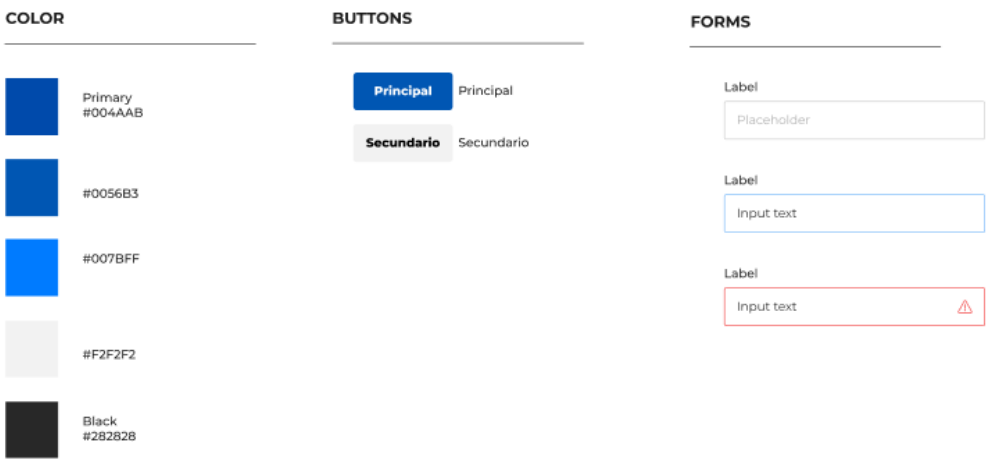


Ilustración 9: Paleta de colores en FIGMA.

Fuente: Elaboración propia.

3.2.2. Proceso de creación de wireframes y prototipos

El proceso de diseño para la interfaz de usuario comenzó con la creación de wireframes y prototipos, herramientas clave para visualizar y planificar la estructura del sistema antes de iniciar el desarrollo. Este enfoque permitió identificar las necesidades funcionales del sistema, optimizar la organización de los elementos y validar el flujo de interacción del usuario.

3.2.2.1. Wireframes

Los wireframes se diseñaron como representaciones preliminares de las pantallas principales del sistema, priorizando la disposición de los elementos y la jerarquía del contenido, antes de enfocarse en los detalles estéticos. Para esta fase inicial, se utilizó una herramienta de dibujo básico, lo que permitió plasmar rápidamente ideas y planteamientos. Cada sección clave, como el menú lateral, los formularios de inventario y los dashboards, fue bosquejada con líneas simples para garantizar una organización lógica y fluida.

La participación del usuario final en esta etapa fue crucial. Se llevaron a cabo sesiones de retroalimentación para ajustar los primeros trazados según las necesidades reales, asegurando que la estructura del sistema respondiera de manera efectiva a las expectativas y objetivos del usuario.

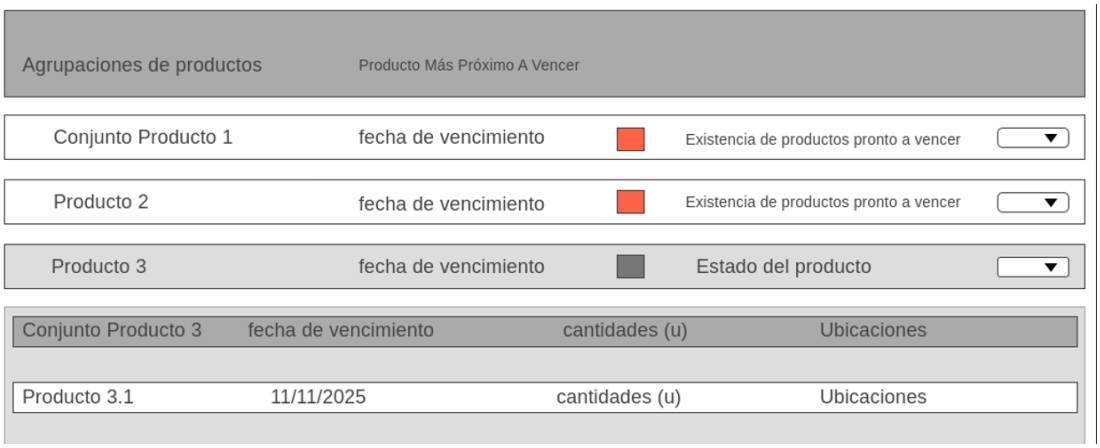


Ilustración 10: Diseño básico de la aplicación (funcionalidad de listado de productos)**.**

Fuente: Elaboración propia.

3.2.2.2. Prototipos

Con las ideas validadas en los wireframes, se avanzó a la creación de prototipos interactivos, utilizando la herramienta Figma **[10]**. Esta plataforma permitió no solo diseñar con un mayor nivel de detalle estético, sino también agregar interactividad que simulaba la experiencia real del usuario.

En esta etapa, se incorporaron elementos clave del diseño visual, como la paleta de colores y los tamaños aproximados de los componentes, lo que facilitó una visualización cercana al aspecto final del sistema. La posibilidad de navegar entre pantallas y simular flujos de interacción permitió realizar ajustes tempranos en la navegación y funcionalidades, optimizando la experiencia del usuario antes de pasar a la implementación final.

Ilustración 11: Prototipo en FIGMA.

Fuente: Elaboración propia.

3.2.3. Responsividad y adaptabilidad a diferentes dispositivos

El diseño del sistema fue creado con un enfoque centrado en la responsividad y adaptabilidad, asegurando una experiencia de usuario consistente y óptima en dispositivos con diferentes tamaños de pantalla, como computadoras de escritorio, tablets y teléfonos móviles.

- **Media Queries (Consultas de Medios):** Se implementaron media queries en el SCSS para adaptar la disposición y tamaño de los elementos a distintos anchos de pantalla.
- **Diseño Fluido**: Se emplearon porcentajes y unidades relativas como em, vh, vw y rem para que los elementos escalaran proporcionalmente. El uso de flexbox y grid permitió una distribución flexible de los elementos.

- **Elementos Reajustables:** Componentes como tablas y gráficos se rediseñaron para ser desplazables horizontalmente o reorganizarse verticalmente en pantallas pequeñas.

Ilustración 12: Imagen de prueba, con 200% de zoom.

Fuente: Elaboración propia.

4. Implementación y validación.

4.1. Detalles de la codificación y desarrollo

Con base en todo lo detallado en el proyecto, el backlog para el desarrollo del Front-End del sistema web de gestión de inventarios en comedores solidarios se basó en tareas organizadas en categorías clave.

4.1.1. Backlog General del Proyecto

- **Configuración inicial**

  Configurar el entorno de desarrollo con Angular 17.

  Establecer la comunicación con el Back-End utilizando Socket.IO.

  Configurar rutas y navegación básica en el proyecto (Angular Router).

  Crear la estructura base del proyecto siguiendo el patrón MVC.

- **Visualización y gestión de inventario**

  Crear la interfaz para la visualización de productos.

  Implementar la carga y actualización de productos en tiempo real mediante sockets.

  Diseñar e implementar filtros y búsquedas para productos.

  Desarrollar alertas visuales para productos próximos a caducar.

- **Gestión de usuarios**

  Implementar la interfaz de autenticación y login (con manejo de tokens).

  Crear el módulo para la gestión de usuarios (CRUD de usuarios).

  Establecer roles y permisos para garantizar la seguridad del sistema.

  Diseñar la vista del perfil de usuario, permitiendo actualizaciones de datos personales.

- **Gestión de bodegas y ubicaciones**

  Crear vistas para el manejo de bodegas (registro, modificación y eliminación).

  Implementar la funcionalidad para visualizar y administrar ubicaciones dentro de las bodegas.

  Integrar la visualización de inventarios por bodega.

- **Planificación y envíos**

  Diseñar e implementar el módulo de planificación:

  Selección de fechas y organización de entregas futuras.

  Visualización de productos asignados a cada planificación.

  Desarrollar el sistema de envíos:

  Mostrar envíos programados, su estado y detalles.

  Permitir modificaciones en las planificaciones y seguimiento de entregas.

- **Dashboard y métricas**

  Implementar un dashboard dinámico con gráficos estadísticos.

  Crear endpoints para cargar datos del dashboard en función de intervalos de tiempo seleccionados.

  Diseñar visualizaciones que incluyan tendencias de uso, caducidad y distribución de productos.

- **Responsividad y diseño UI/UX**

  Implementar un diseño responsivo para adaptarse a dispositivos.

  Aplicar mejoras visuales utilizando PrimeNG y Bootstrap.

  Validar la usabilidad y accesibilidad con usuarios finales.

- **Integración con Back-End**

  Configurar servicios para realizar operaciones HTTP (GET, POST, PATCH, DELETE) para el manejo de datos.

  Sincronizar la información en tiempo real con el servidor mediante sockets.

- **Pruebas y despliegue**

  Realizar pruebas unitarias de componentes y servicios en Angular.

  Validar la integración entre el Front-End y el Back-End en entornos de prueba.

  Optimizar tiempos de carga y navegación.

  Desplegar el sistema en Nginx para producción.

4.1.2. Gestión de Sprints para el Desarrollo del Proyecto

El desarrollo del sistema web se organizó en tres sprints, cada uno con una duración de 4 semanas, lo que permitió dividir el trabajo en fases específicas y avanzar de forma iterativa. Este enfoque facilitó la entrega progresiva de incrementos funcionales, asegurando un progreso constante y bien estructurado.

A lo largo de todo el proceso de desarrollo, la creación del código se realizó de manera continua según las necesidades de cada etapa, priorizando tareas críticas y ajustándose a los objetivos planteados en cada sprint. Además, para garantizar una correcta gestión del código y la colaboración eficiente entre los involucrados, se utilizó un repositorio general como espacio centralizado para almacenar y gestionar todos los archivos del proyecto.

Dentro de este repositorio, se creó un espacio dedicado exclusivamente al apartado web, donde el código relacionado con el Front-End fue registrado y actualizado constantemente. Esto permitió mantener un control de versiones organizado, facilitando la revisión de cambios, la detección de errores y la integración de nuevas funcionalidades de manera ágil y segura.

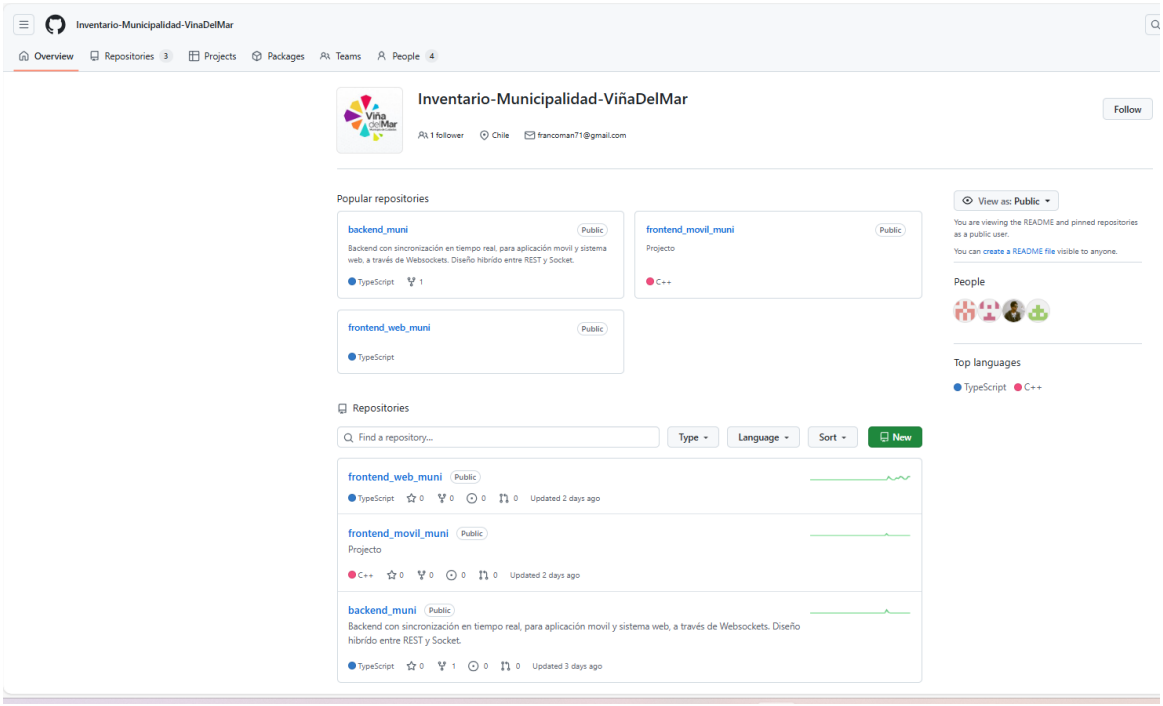


Ilustración 13: Comunidad de **GITHUB** del proyecto**.**

Fuente: Elaboración propia.

A continuación, se detalla el progreso y las tareas realizadas en cada sprint:

4.1.2.1. Primera iteración: Conexión y visualización de productos

En esta fase inicial, el enfoque estuvo en establecer la base del proyecto y garantizar la comunicación fluida entre el Front-End y el Back-End, así como en implementar funcionalidades iniciales para la gestión del inventario.

Tareas realizadas:

- Configuración inicial del proyecto:
  - Se configuró el entorno de desarrollo utilizando Angular 17.
  - Se definió la estructura base del proyecto siguiendo el patrón MVC (Modelo-Vista-Controlador), asegurando una organización clara y escalable del código.
- Conexión con el Back-End:
  - Se estableció la comunicación mediante Socket.IO, configurando sockets para manejar la sincronización de datos en tiempo real.
  - Se implementó un sistema de autenticación inicial para manejar tokens y asegurar una conexión segura.
- Visualización de productos:
  - Se desarrolló la vista principal para listar los productos del inventario.
  - Se utilizó PrimeNG para crear tablas dinámicas con filtros, búsquedas y paginación.
  - Se implementaron los primeros eventos en tiempo real para reflejar actualizaciones en el inventario (como agregar nuevos productos o actualizar el stock).

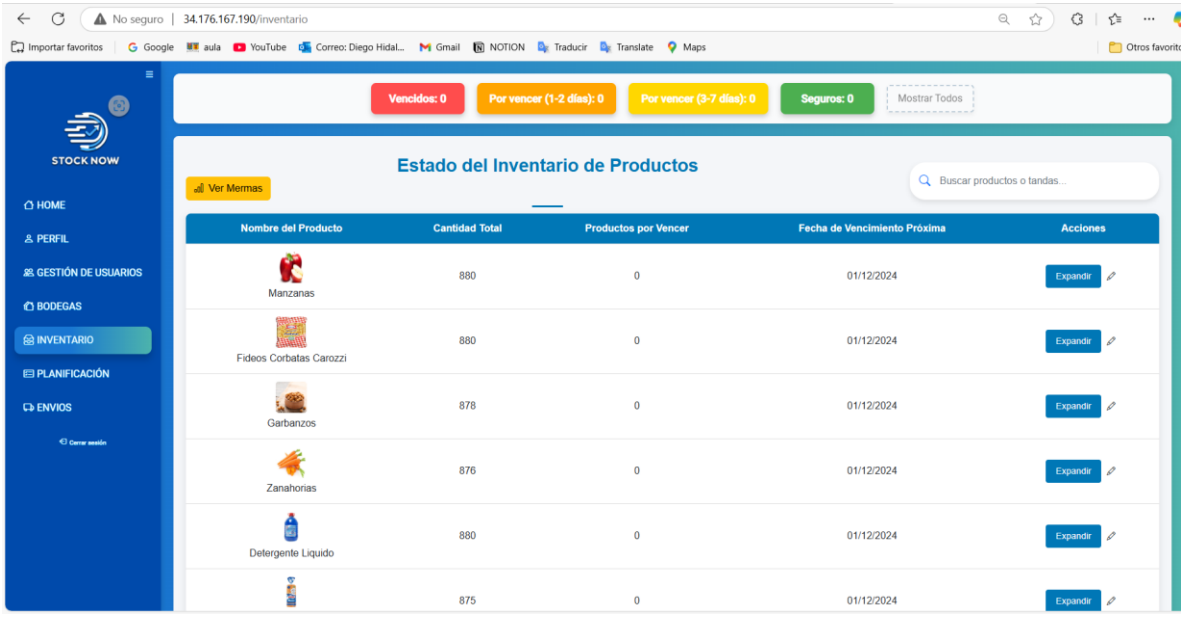


Ilustración 14: Módulo del sistema de inventarios web.

Fuente: Elaboración propia.

4.1.2.2. Segunda Iteración: Autenticación y Gestión de Usuarios

El segundo sprint se enfocó en reforzar la seguridad del sistema y mejorar la experiencia del usuario mediante un diseño más pulido y funcional. También se avanzó en la implementación del módulo de gestión de usuarios.

Tareas realizadas:

- **Sistema de autenticación:**
  - Se creó un sistema de login utilizando tokens para autenticar a los usuarios.
  - Se gestionó el almacenamiento y la validación de tokens mediante un servicio dedicado.
  - Se estableció un guard para proteger rutas del sistema, asegurando que solo los usuarios autorizados accedan a áreas específicas.

- **Gestión de uuarios y roles:**
  - Se implementó un módulo completo para gestionar usuarios, permitiendo a los administradores crear, editar y eliminar usuarios.
  - Se configuraron roles y permisos, asegurando que las acciones críticas del sistema solo estuvieran disponibles para usuarios con las credenciales adecuadas.

- **Mejoras visuales:**
  - Se rediseñó la interfaz general utilizando PrimeNG y Bootstrap, enfocándose en la usabilidad y accesibilidad.
  - Se añadió un menú lateral (sidebar) para facilitar la navegación entre las diferentes secciones del sistema.

Ilustración 15: Módulo de gestión de usuarios web funcionando.

Fuente: Elaboración propia.

4.1.2.3. Tercera Iteración: Planificación, Alertas y Ajustes Finales

En esta última iteración, se completaron las funcionalidades avanzadas del sistema, se implementaron mecanismos para optimizar la gestión de recursos y se preparó el sistema para su despliegue final.

Tareas realizadas:

- **Módulo de planificación y envíos:**
  - Se desarrolló un módulo para planificar entregas futuras, integrando una vista semanal que permite organizar productos según disponibilidad y fechas de caducidad.
  - Se implementó la funcionalidad para generar alertas automáticas sobre productos cercanos a caducar o niveles bajos de stock.
  - Se desarrolló un módulo para recibir y detallar envíos cargados dentro de la base de datos, permitiendo al usuario mostrar en tiempo real, datos relevantes asociados.

- **Dashboard y métricas:**
  - Se diseñó e implementó un dashboard dinámico con gráficos interactivos que muestran métricas clave, como el estado del inventario y estadísticas de entregas.
  - Se configuraron endpoints para cargar datos del dashboard en tiempo real.

- **Pruebas y ajustes:**
  - Se realizaron pruebas de usabilidad con usuarios clave para validar la funcionalidad del sistema.
  - Se optimizaron tiempos de carga y navegación mediante ajustes en el diseño y la lógica de los componentes.
  - Se preparó el sistema para el despliegue final utilizando Nginx como servidor web.

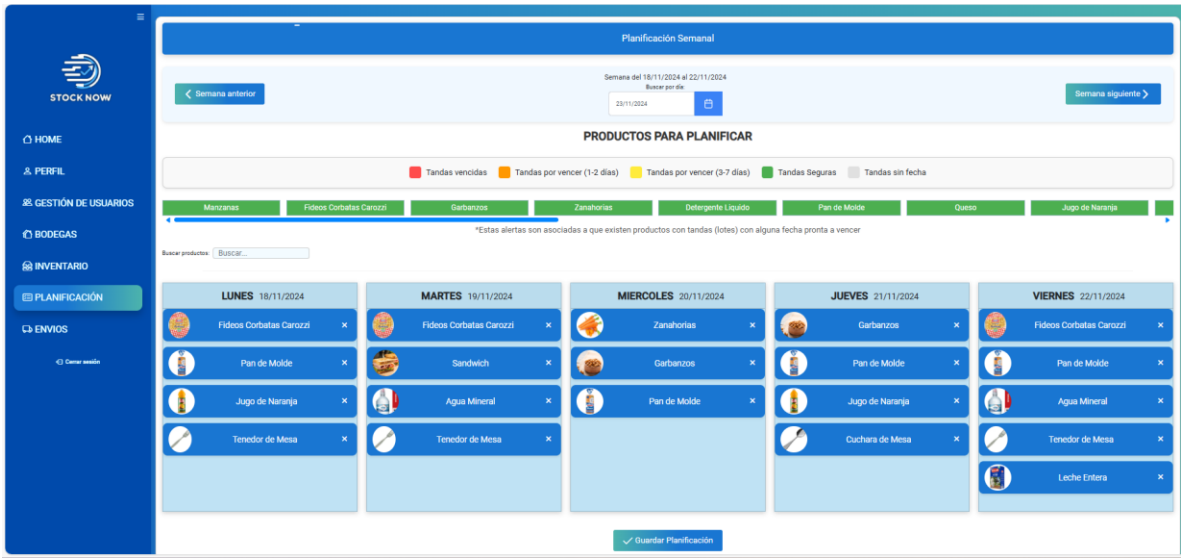


Ilustración 16: Módulo de planificación del proyecto.

Fuente: Elaboración propia.

4.1.3. Control de flujo

Para el control de versiones, el repositorio se gestionó utilizando Git, con todos los cambios integrados directamente en la rama principal (main). Este enfoque fue adoptado para simplificar el flujo de trabajo

y priorizar la rapidez en la integración de nuevas funcionalidades, dado el tiempo limitado y los recursos disponibles para el proyecto.

A lo largo del desarrollo, los commits se realizaron de manera frecuente y descriptiva, asegurando que cada cambio estuviera documentado y que el historial del repositorio reflejara claramente las modificaciones realizadas. Este enfoque permitió mantener un registro continuo de las actualizaciones, facilitando la identificación de versiones funcionales a medida que se avanzaba en el proyecto.

Aunque no se implementó un flujo de ramas convencional, como Git Flow, el enfoque utilizado resultó eficiente para el contexto del proyecto, permitiendo mantener un ritmo constante de desarrollo sin interrupciones.

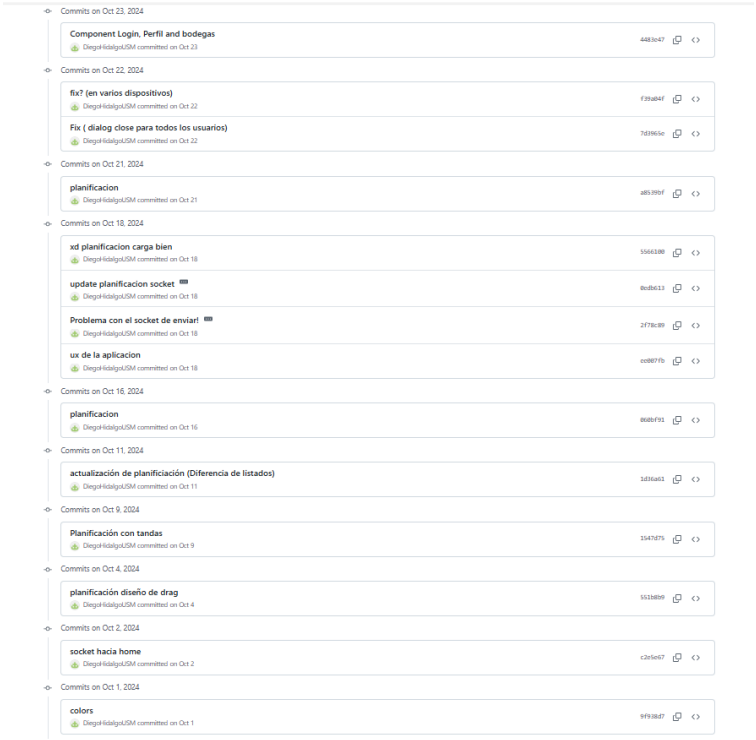


Ilustración 17: Commits realizados en **GITHUB.**

Fuente: Elaboración propia.

En la imagen anterior se observa un extracto del flujo continuo de commits realizados durante el desarrollo, lo que refleja la constancia y agilidad en la integración de actualizaciones en el repositorio de GitHub. Este enfoque, pese a no tener títulos muy informativos, permitió mantener un registro de los avances y asegurar que cada cambio contribuyera al progreso del proyecto de manera estructurada y eficiente.

4.2. Pruebas y Validación

Para garantizar que el front-end cumple con los requerimientos planteados, se implementaron pruebas unitarias específicas para cada componente utilizando el archivo de prueba correspondiente (spec.ts). Estas pruebas fueron diseñadas con el objetivo de verificar que la lógica de negocio, las interacciones

del usuario y la comunicación con los servicios back-end se comporten de acuerdo con las especificaciones del proyecto.

A continuación, se describen los aspectos más relevantes del enfoque de pruebas:

4.2.1. Pruebas Unitarias en Cada Componente

- Cada componente fue testeado de forma independiente para validar su funcionalidad específica. Esto incluyó la verificación de métodos, eventos y el correcto manejo del estado interno.
- Se utilizaron simulaciones (mocks) para los servicios dependientes, lo que permitió evaluar el comportamiento del front-end sin necesidad de conexiones reales al back-end.

4.2.2. Pruebas de Interacción con el Usuario

Durante las pruebas de interacción con los usuarios del sistema web, se realizaron diversos análisis y mediciones para garantizar que las funcionalidades cumplieran con los requerimientos establecidos y se adaptaran a las necesidades operativas de los usuarios. A continuación, se describen los resultados obtenidos:

Evaluación de interacciones:

- Se observaron interacciones clave como clics en botones, selección de elementos en listas desplegables y el envío de formularios. Los usuarios reportaron que estas acciones eran intuitivas y que la interfaz respondía correctamente, mostrando los cambios esperados en la pantalla. Además, se validó que las solicitudes generadas por estas acciones fueran correctamente procesadas por el sistema back-end, asegurando una comunicación fluida y sin errores entre ambas capas.

Cumplimiento de requerimientos:

- Los usuarios confirmaron que las funcionalidades del sistema, especialmente las relacionadas con la gestión de envíos y la generación de alertas de productos, cumplían con los requerimientos previamente definidos. Por ejemplo, las alertas de productos cercanos a agotarse o vencerse se enviaban de manera oportuna, lo que ayudó a evitar incidentes en la operación diaria.

Reducción de tiempo operativo:

- Una de las mejoras más destacadas fue la reducción significativa del tiempo necesario para registrar y procesar datos. Anteriormente, los usuarios reportaron que tomar y registrar los datos requería aproximadamente 30 minutos por operación, mientras que con el nuevo sistema el tiempo se redujo a tan solo 10 minutos, lo que representa un ahorro del 66% en el tiempo de operación.

Resguardo de información:

- La funcionalidad para subir y almacenar actas o archivos directamente en el sistema fue bien recibida por los usuarios. Esto les permitió contar con un resguardo seguro de la información crítica, eliminando la necesidad de depender de registros físicos o soluciones externas de almacenamiento. Según los usuarios, esto incrementó la confianza en la seguridad y accesibilidad de los datos.

Solicitudes de mejora:

- Los usuarios expresaron interés en continuar añadiendo nuevas funcionalidades que mejoren la utilidad del sistema. Una de las sugerencias más destacadas fue la incorporación de un módulo para registrar incidentes en el inventario, lo que facilitaría la identificación y resolución de problemas operativos. Además, se solicitó un cambio en la paleta de colores de la interfaz, específicamente de tonos azules a un esquema más verdoso, con el objetivo de hacer la experiencia visual más agradable.

4.2.3. Verificación de la Comunicación con el Back-end

- Se verificó que las llamadas HTTP y las conexiones en tiempo real mediante sockets estuvieran correctamente configuradas y enviaran los datos esperados.
- Se probaron casos de éxito y fallos en las respuestas del servidor para asegurar que la aplicación manejara errores de forma adecuada.

4.3. Despliegue de la aplicación

El despliegue de la aplicación front-end se realizó utilizando Nginx como servidor web principal. Este proceso implicó varias etapas para garantizar que la aplicación estuviera correctamente configurada, segura y funcional en un entorno de producción.

- Construcción de la Aplicación

El primer paso en el despliegue fue generar un build optimizado de la aplicación Angular. Para esto, se utilizó el comando: ng build –prod. Este comando produjo un conjunto de archivos estáticos que son posteriormente cargados en el servidor que se desea desplegar

Estos archivos fueron preparados para ser servidos por Nginx desde un directorio designado.

5. Conclusiones

El desarrollo del proyecto fue una experiencia enriquecedora que permitió abordar diversos desafíos técnicos y de gestión. A lo largo del proyecto, se implementaron herramientas modernas como Angular, PrimeNG y Bootstrap, junto con tecnologías en tiempo real como Socket.IO. Esto proporcionó una comprensión profunda de la importancia de planificar y estructurar un proyecto, así como de las decisiones que impactan tanto en el desarrollo como en el producto final.

Entre las principales lecciones aprendidas destaca la necesidad de un control de versiones más estructurado. Aunque se optó por realizar commits constantes directamente sobre la rama principal para agilizar el desarrollo, una estrategia más robusta, como Git Flow, habría facilitado la gestión de errores y la integración de nuevas funcionalidades. Asimismo, una documentación inicial más detallada sobre las tecnologías empleadas podría haber reducido el tiempo invertido en solucionar problemas técnicos durante las etapas iniciales.

Uno de los desafíos más complejos fue implementar la sincronización en tiempo real mediante Socket.IO, que requirió una coordinación precisa entre el cliente y el servidor. Además, el diseño de un sistema de autenticación y gestión de roles planteó retos significativos, especialmente al garantizar la seguridad y accesibilidad del sistema.

El diseño de una interfaz intuitiva y responsiva, sin contar con un especialista en UI/UX, también representó un desafío, aunque el uso de bibliotecas como PrimeNG y Bootstrap ayudó a crear una experiencia funcional y accesible para los usuarios.

El proyecto logró importantes avances. La automatización del control de inventarios eliminó la dependencia de hojas de cálculo manuales, reduciendo errores y optimizando los procesos. La integración de actualizaciones en tiempo real mejoró la capacidad de los usuarios para tomar decisiones informadas y oportunas, mientras que el módulo de autenticación aseguró un control estricto sobre las funcionalidades críticas.

Este sistema tiene un impacto social directo al mejorar la eficiencia en la distribución de recursos alimentarios en comedores solidarios, beneficiando a las comunidades más vulnerables.

Desde una perspectiva técnica, este proyecto representa una contribución significativa a la disciplina informática al aplicar tecnologías modernas como la arquitectura modular y la comunicación en tiempo real. Estas herramientas no solo resolvieron un problema crítico en la gestión de inventarios, sino que también demostraron cómo la tecnología puede generar un impacto positivo en la sociedad al abordar problemas prácticos.

En conclusión, este proyecto no solo logró cumplir con los objetivos iniciales, sino que también sentó las bases para futuras mejoras y ampliaciones. Su impacto en los usuarios se traduce en procesos más eficientes y seguros, mientras que, para la comunidad, garantiza una distribución más justa y optimizada de recursos. La experiencia adquirida, los logros alcanzados y las lecciones aprendidas servirán como guía para abordar proyectos similares en el futuro, con un enfoque más estructurado y consciente de las necesidades del cliente y de la sociedad.

6. Referencias

[1] D. Flanagan, *JavaScript: The Definitive Guide*, 7th ed. Sebastopol, CA, USA: O'Reilly Media, 2020.

[2] E. Freeman and E. Robson, *Head First JavaScript Programming*. Sebastopol, CA, USA: O'Reilly Media, 2020.

[3] J. Resig and B. Bibeault, *Secrets of the JavaScript Ninja*, 2nd ed. Shelter Island, NY, USA: Manning Publications, 2012.

[4] R. Meier, *Professional Android Development*, 2nd ed. Indianapolis, IN, USA: Wrox, 2020.

[5] A. Russell, *Progressive Web Apps*. Sebastopol, CA, USA: O'Reilly Media, 2019.

[6] M. Fowler, *Patterns of Enterprise Application Architecture*. Boston, MA, USA: Addison-Wesley, 2003.

[7] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. New York, NY, USA: Wiley, 1996.

[8] Socket.IO, “Socket.IO documentation,” [Online]. Available: https://socket.io/docs/v4/. [Accessed: Nov. 22, 2024].

[9] Nginx, “Documentation,” [Online]. Available: https://nginx.org/en/docs/. [Accessed: Nov. 23, 2024].

[10] Figma, “Prototipo de diseño en Figma,” [Online]. Available: https://www.figma.com/proto/9W8M18K4FN36NcKy2go7f5/Untitled?node-id=1-2&node-type=frame&t=ugwx21EPQl0x9aFR-1&scaling=min-zoom&content-scaling=fixed&page-id=0%3A1. [Accessed: Nov. 22, 2024].