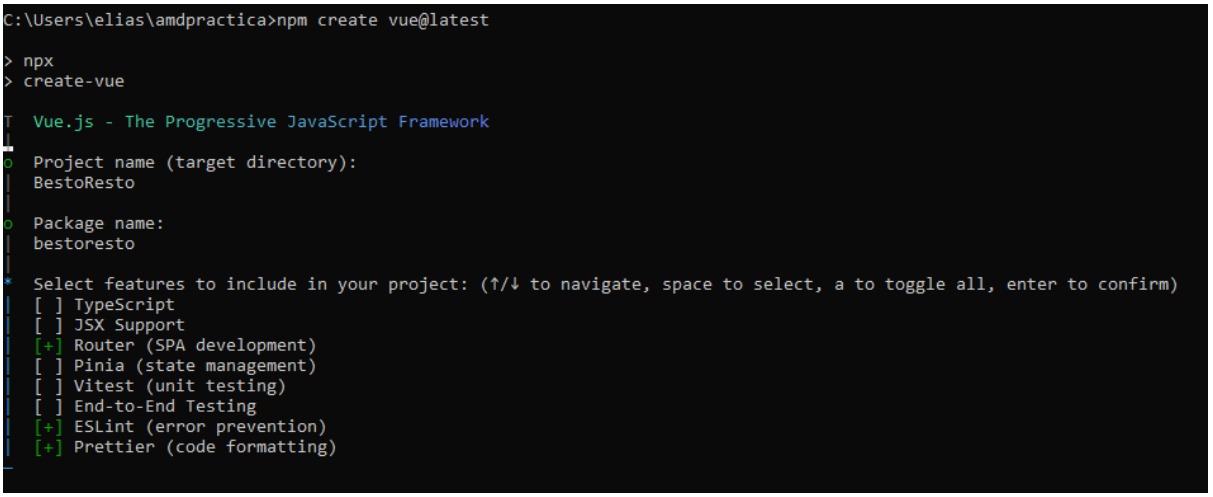
Besto Resto

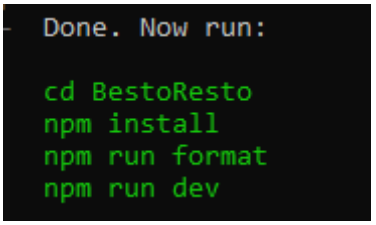
Informe realizado por Diego Asensio y Elias Marolla sobre el proyecto Besto Resto, trabajo realizado en Vue 3 (options)

Inicialización:

Dentro de CMD(simbolo de sistema) creamos una carpeta usando el comando mkdir,
Luego con el comando cd ingresamos a la carpeta y mediante node instalamos vue con el comando npm create vue@latest.
Luego de darle el nombre al proyecto confirmamos las opciones de Router,ESLint y Prettier con la barra espaciadora y damos enter para generar el proyecto.



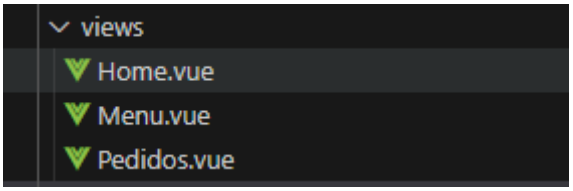
Luego en la misma consola usamos los siguientes comandos:



y de esta manera terminamos de instalar las dependencias necesarias para comenzar el proyecto.

Vistas:

El proyecto cuenta con tres vistas como pide la consigna:

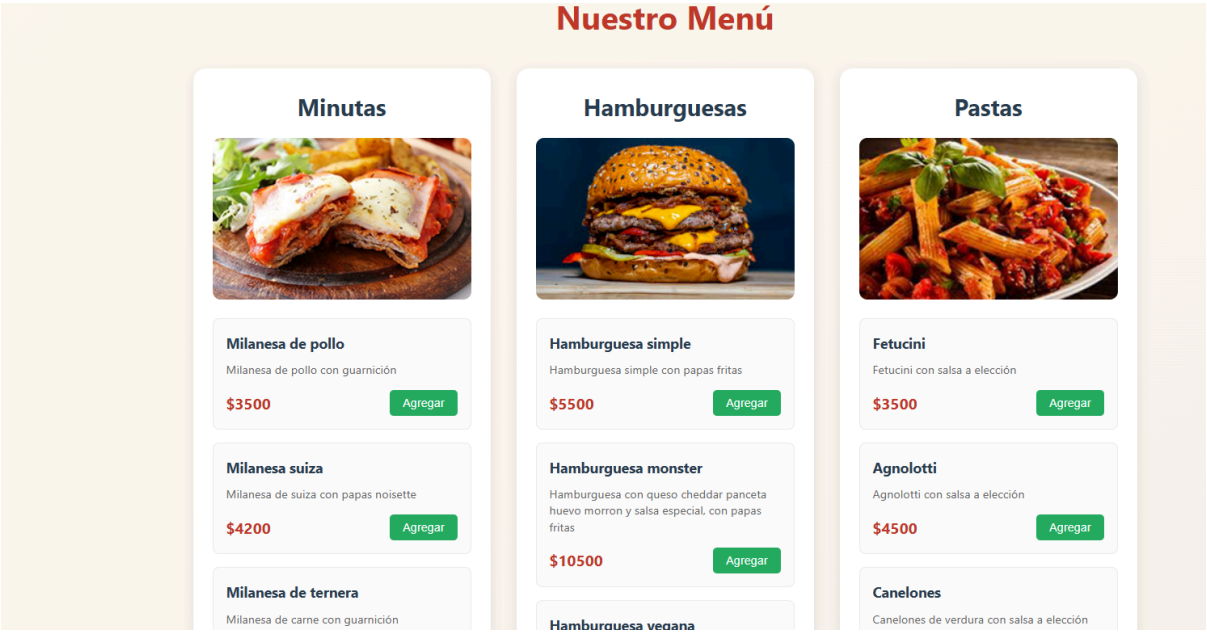


La vista “Home” es la encargada de recibir a los usuarios a la web, en ella se encuentra un componente que muestra la cantidad de platos con la que cuenta el menú y la cantidad

de pedidos que realizó el usuario, y también se encuentra un apartado que nos invita mediante botones a que realicemos nuestro pedido:

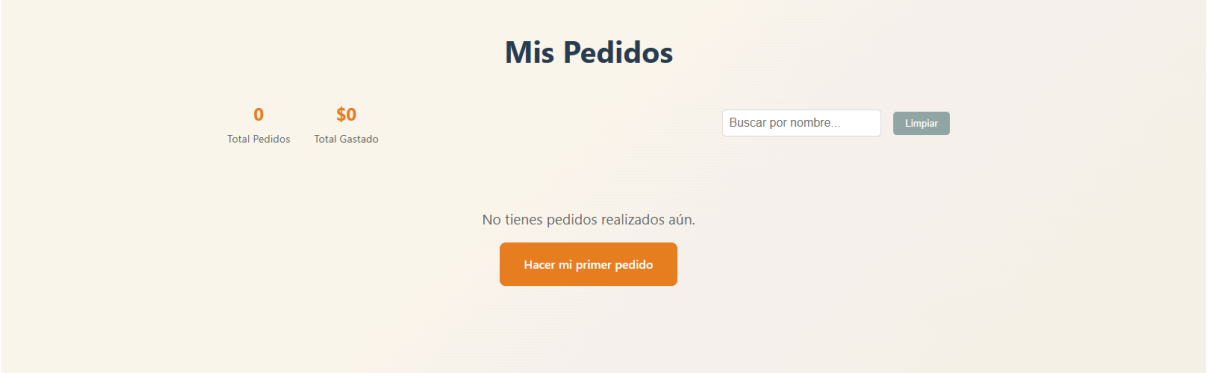
Estos botones cuentan con animaciones integradas mediante animate.css.
Para esto lo instalamos desde la consola utilizando el comando:
npm install animate.css --save

La vista “menu” integra mediante un import los datos almacenados en un archivo javascript y los recorre generando cards (tarjetas) con estilo y funciones propias:



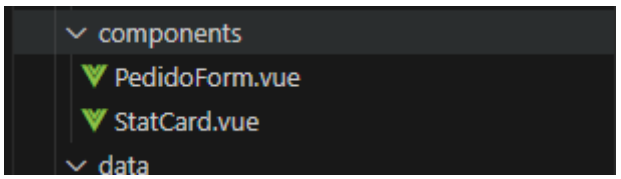
Y por ultimo la vista “pedidos” nos muestra el historial de pedidos realizados que fueron guardados en el localstorage:

En caso de no haber realizado pedidos previamente o tener limpio el historial de compras, aparecera un boton para redireccionarlos a la vista de menu para poder realizar el primer pedido:

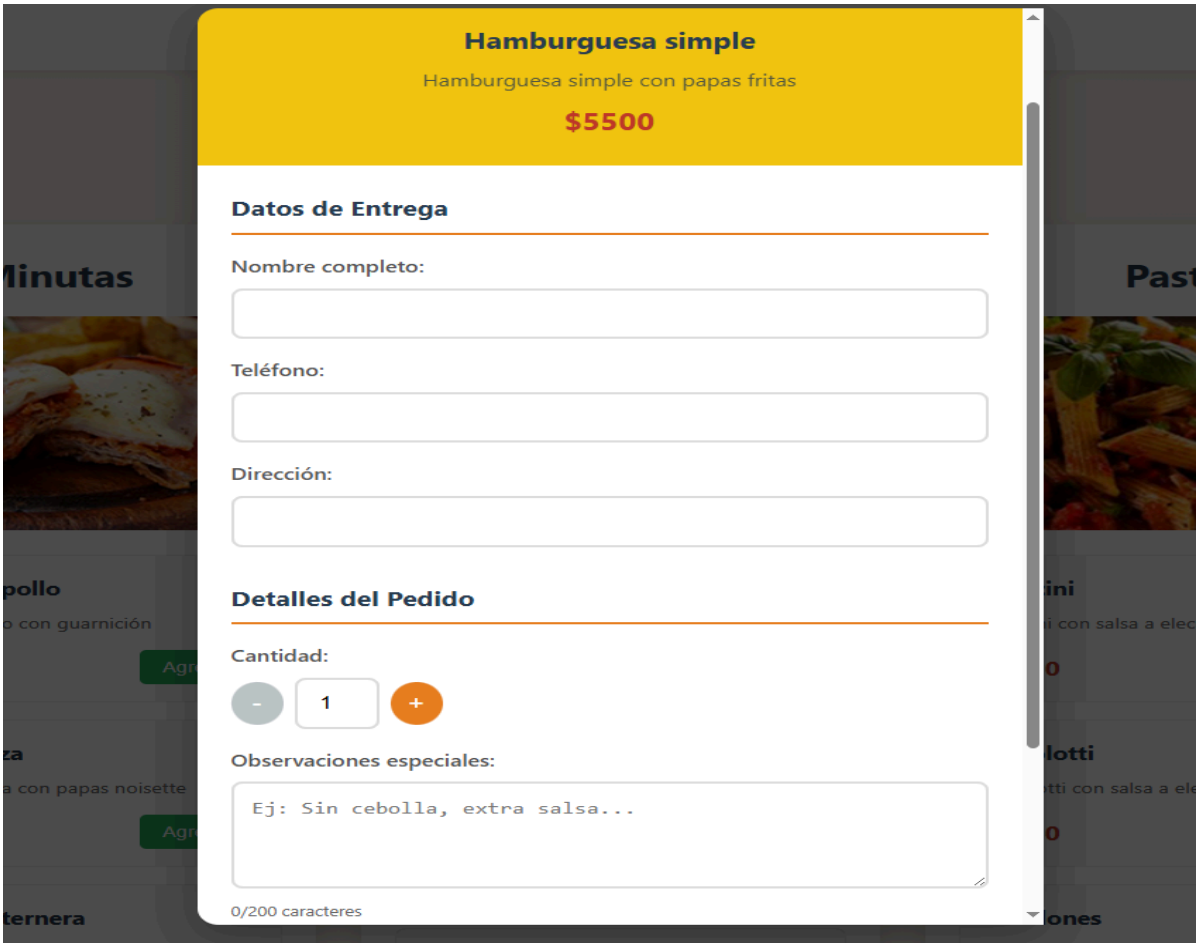


Components:

En el trabajo desarrollado utilizamos únicamente dos componentes:



El primero es “PedidoForm” con el cual pedimos los datos al usuario para registrar el pedido y almacenarlo en el localStorage:



Y el segundo es “Statcard”, un componente que tiene la capacidad de mostrar las cantidades de elementos que hay en el menú y la cantidad de pedidos realizados

al ser dinámicos mediante que el cliente agregue más pedidos este va a incrementar, pero si el cliente borra el los pedidos realizados este va a disminuir hasta llegar a 0.

LocalStorage (ABM):

Dentro de la funcionalidades del componente PedidoForm uno de sus methods es agregar al localStorage el nuevo pedido que se está realizando:

```
procesarPedido() {
  if (!this.validarFormulario()) {
    return
  }

  try {
    // Crear el objeto de pedido
    const nuevoPedido = {
      nombre: this.orden.nombre.trim(),
      telefono: this.orden.telefono.trim(),
      direccion: this.orden.direccion.trim(),
      pedido: [
        {
          opcion: this.opcionSeleccionada.nombre,
          cantidad: this.cantidad,
          observaciones: this.observaciones.trim(),
          valor: this.opcionSeleccionada.precio,
        },
      ],
    }

    // Obtener pedidos existentes
    const ordenesExistentes = JSON.parse(localStorage.getItem('ordenes') || '[]')

    // Agregar el nuevo pedido
    ordenesExistentes.push(nuevoPedido)

    // Guardar en localStorage
    localStorage.setItem('ordenes', JSON.stringify(ordenesExistentes))

    // Emitir evento de éxito
```

Para esto primero revisa si tiene datos guardados en el key de “ordenes” y agrega los nuevos pedidos realizados.

También los pedidos realizados pueden ser eliminados

```
  eliminarPedido(index) {
    if (confirm('¿Estás seguro de que quieres eliminar este pedido?')) {
      this.ordenes.splice(index, 1)
      this.guardarPedidos()
    }
  },
  guardarPedidos() {
    try {
      localStorage.setItem('ordenes', JSON.stringify(this.ordenes))
    } catch (error) {
      console.error('Error al guardar pedidos:', error)
    }
  },
```

De esta manera se puede tener un control sobre los pedidos guardados en el LocalStorage.

Validaciones del formulario:

```
validarFormulario() {
  this.errores.general = []
  this.errores.nombre = false
  this.errores.telefono = false
  this.errores.direccion = false

  if (!this.orden.nombre.trim()) {
    this.errores.nombre = true
    this.errores.general.push('El nombre es obligatorio')
  }

  if (!this.orden.telefono.trim()) {
    this.errores.telefono = true
    this.errores.general.push('El teléfono es obligatorio')
  } else if (!/^\d{8,15}$/.test(this.orden.telefono.replace(/\D/g, ''))) {
    this.errores.telefono = true
    this.errores.general.push('El teléfono debe tener entre 8 y 15 dígitos')
  }

  if (!this.orden.direccion.trim()) {
    this.errores.direccion = true
    this.errores.general.push('La dirección es obligatoria')
  }

  return this.errores.general.length === 0
},
procesarPedido() {
```

El formulario para generar la nueva orden cuenta nos una serie de validaciones de los datos minimos necesarios para que funcione y en caso de que haya un campo sin completar o incorrecto este cuenta con un sistema que genera un mensaje dependiendo del error.

Ciclos de vida:

Agregamos hooks o ciclos de vida para disparar funciones, como por ejemplo en la view de Pedidos:

```
mounted() {
  this.cargarPedidos()
},
methods: {
  cargarPedidos() {
    try {
      const ordenesGuardadas = localStorage.getItem('ordenes')
      this.ordenes = ordenesGuardadas ? JSON.parse(ordenesGuardadas) : []
    } catch (error) {
      console.error('Error al cargar pedidos:', error)
      this.ordenes = []
    }
  },
```

con la finalidad de que al montarse la vista mediante el viewrouter esta realice la funcion de cargar los pedidos preguntando en el localStorage si hay ordenes guardadas.

En la vista de Home podemos ver un proceso similar

```
    mounted() {
      this.cargarPedidosRealizados()
    },
    methods: {
      cargarPedidosRealizados() {
        try {
          const ordenes = JSON.parse(localStorage.getItem('ordenes') || '[]')
          this.pedidosRealizados = ordenes.length
        } catch (error) {
          console.error('Error al cargar pedidos:', error)
          this.pedidosRealizados = 0
        }
      },
    },
  }
/script>
template>
```

donde el componente luego utiliza los datos que trae la función mediante props.