Universidad de Guadalajara

Centro Universitario de Ciencias Exactas e Ingenierías

División de Electrónica y Computación

Departamento de Ciencias Computacionales

Ingeniería en Computación

Programación para Internet

Profesor: Jiménez Rodríguez, Mario

I5909 – D01

Lunes y miércoles 09:00 – 10:55

Proyecto Final: DecoYeso

Ojeda Escobar, César Arley

216306568

Fecha: 22/05/2020

Contenido

Introducción

El proyecto nace a partir de la necesidad de un sistema que ayude a administrar y registrar las ventas en el negocio de mis padres. Ellos, dedicados a la venta de molduras y rosetones de yeso, hacen el registro en una libreta, en donde realmente no guardan todas las ventas, ni tampoco existe un registro del inventario.

Es por todo lo anterior que decidí que mi proyecto estuviera orientado a los trabajadores de la tienda, además de poder almacenar imágenes de las molduras y los rosetones que después pueden ser mostrados a los clientes, sobre todo las que son pintadas.

Por último, hay que mencionar que el negocio tiene el nombre de DecoYeso, y es por esto por lo que la aplicación tiene el nombre de “DecoYeso Manager”.

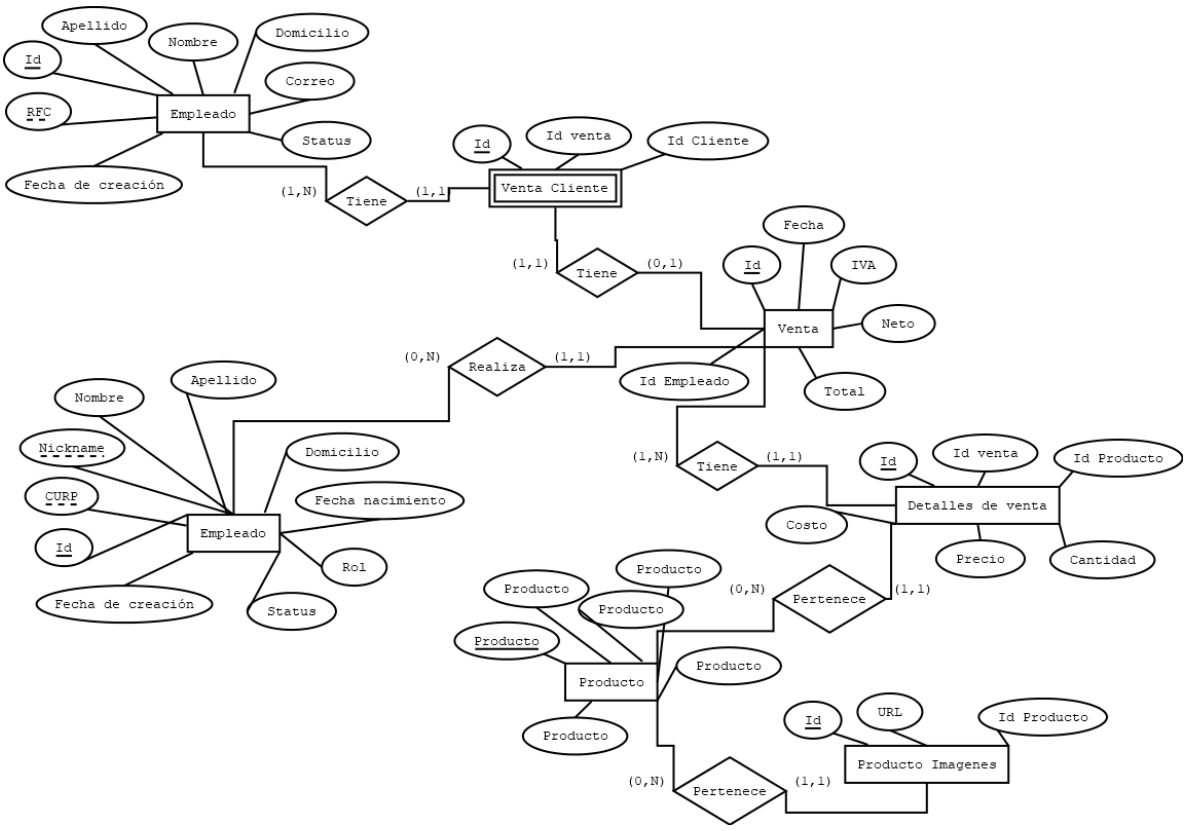
Proyecto

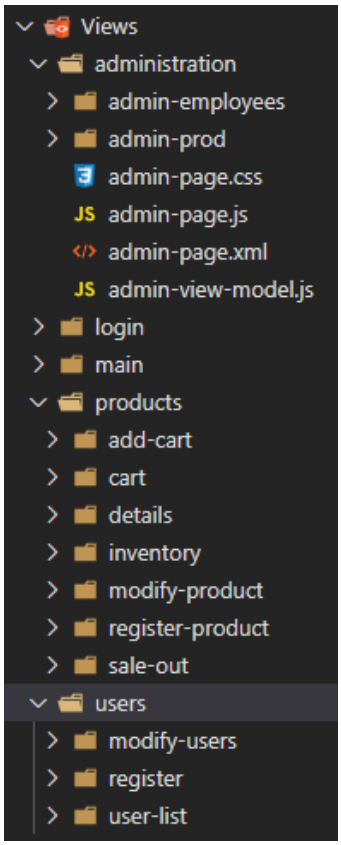
El proyecto esta desarrollado con el modelo cliente-servidor, en donde el cliente hace peticiones a una API, mientras que el servidor realiza las operaciones de autentificación y conexión con la base de datos. En este proyecto se trabajó con el lenguaje PHP para el backend, además de utilizar variables de entorno y Json Web Token.

En el lado del frontend se utilizó la tecnología de Nativescript con JavaScript ES6, y una serie de plugins para realizar las tareas de la app.

Base de datos

Todos los datos de la aplicación son almacenados en una base de datos relacional utilizando MySQL en el servidor, a continuación, se muestra el diagrama entidad-relación de la base de datos.





Aplicación

Como ya se mencionó, la aplicación está desarrollada en Nativescript y JavaScript, por lo que es necesario utilizar NodeJS para poder ejecutar el CLI de este framework.

Ahora bien, pasaremos a analizar la estructura de nuestro proyecto.

Como podemos ver, cada carpeta corresponde a cada una de las vistas de la aplicación, dentro de cada una existen cuatro archivos que corresponden a:

- ViewModel. La carga de los datos en la aplicación.
- XML. Definición de los componentes de la vista.
- page.js. Define las funciones del controlador de la vista.
- Css. Define los estilos de la vista.

Como podemos ver, la carpeta de “products” contiene la mayor cantidad de vistas.

La parte fundamental son los archivos “page.js” y “xml”. En los primeros podemos encontrar una estructura muy similar.

- Carga de librarías.
- Creación de variables globales.
- Carga de la vista.
    - Fetch para validar el token que contiene el rol y la caducidad de la sesión.
    - Fetch de datos iniciales en caso de ser necesarios.
    - Control de excepciones.
- Definición de funciones para eventos.
- Funciones extras en caso de ser necesarias.

A continuación se muestra el código del XML que genera la vista del inventario.

```
<Page xmlns="http://schemas.nativescript.org/tns.xsd" actionBarHidden="true" navigatingTo="onNavigatingTo"
    xmlns:lv="nativescript-ui-listview">

    <StackLayout id="main" class="main">
        <Label id="title" text="Inventario" class="card page-title" visibility="collapsed"/>
        <SearchBar id="searchBar" class="card search" hint="Buscar" clear="onClear" submit="onSubmit" visibility="colla
psed" textFieldHintColor="whitesmoke"/>
        <DockLayout stretchLastChild="true" orientation="vertical" id="inventoriDock" visibility="collapsed">

            <lv:RadListView id="inventoryList" class="card" style="height:1250px" dock="top" itemTap="onItemTap" pullTo
Refresh="true" pullToRefreshInitiated="onPullToRefreshInitiated">
                <lv:RadListView.pullToRefreshStyle>
                    <lv:PullToRefreshStyle indicatorColor="white" indicatorBackgroundColor="blue"/>
                </lv:RadListView.pullToRefreshStyle>
                <lv:RadListView.headerItemTemplate>
                    <FlexboxLayout orientation="horizontal" class="col-titles" dock="top">
                        <Label text="ID" class="col-title" style="width: 8%"/>
                        <Label text="Nombre" class="col-title" style="width: 35%" />
                        <Label text="Estado" class="col-title" style="width: 20%" />
                        <Label text="Cantidad" class="col-title" style="width: 20%"/>
                        <Label text="Costo" class="col-title" style="width: 20%" />
                    </FlexboxLayout>
                </lv:RadListView.headerItemTemplate>
                <lv:RadListView.itemTemplate>
                    <FlexboxLayout flexDirection="row">
                        <Label text="{{ id || 'Cargando...' }}" id="id" class="id" horizontalAlignment="left" style="wi
dth: 10%" />
                        <Label text="{{ name || 'Cargando...' }}" id="name" class="name" horizontalAlignment="left" sty
le="width: 40%" />
                        <Label text="{{ status || 'Cargando...' }}" id="status" class="status" horizontalAlignment="lef
t" style="width: 10%" />
                        <Label text="{{ quantity  || 'Cargando...' }}" class="qnty" verticalAlignment="center" style="w
idth: 15%" />
                        <Label text="{{ value || 'Cargando...'  }}" class="cost" verticalAlignment="center" style="widt
h: 25%" />
```

```
                </FlexboxLayout>
            </lv:RadListView.itemTemplate>
        </lv:RadListView>

    </DockLayout >

</StackLayout>
</Page>
```

Ahora vemos las librerias que utilizamos en la mayoría de los controladores.

```
const dialogs = require("tns-core-modules/ui/dialogs");
const { ObservableArray } = require("tns-core-modules/data/observable-
array");
const appSettings = require("tns-core-modules/application-settings");
const { Frame } = require("tns-core-modules/ui/frame");
```

Explicamos entonces que hacen cada uno.

- dialogs. Permite crear vistas modales con alertas o confirmaciones.
- ObservableArray. Permite crear arrays que actualizan el contenido del elemento al que pertenece.
- appSettings. Una de las mas importantes, ya que nos permite almacenar variables sencillas de forma persistente (incluso si cerramos la aplicación), en esta almacenamos el token de sesión JWT.
- Frame. Permite obtener el frame más reciente y hacer navegación entre ventanas.

El siguiente extracto de código nos permite obtener una animación de carga, mientras se ejecutan los fetch necesarios.

```
  const main = page.getViewById("main");
  const indicator = createActivityIndicator();
  main.addChild(indicator);
```

La siguiente parte es fundamental, ya que pregunta si el token sigue siendo valido y si el usuario tiene permiso para acceder a que componentes de la vista.

```
  const verifiedToken = await verifyToken();
  if (verifiedToken.status) {
    const list = page.getViewById("inventoryList");

    const conectionLink = encodeURI(
      appSettings.getString("backHost") + "lista_inventario.php"
    );

    await fetch(conectionLink, {
```

```
      method: "GET",
      headers: {
        TOKEN: appSettings.getString("token"),
      },
    })
      .then(checkStatus)
      .then(parseJSON)
      .then((json) => {
```

Dificultades

Ahora bien, podría decir que lo más difícil fue entender como hacer funcionar componentes como el carousel, la ventana de selección de imágenes, y las RadListView, sin embargo, una vez entendiéndolos, sobre todo la última, se vuelven fundamentales.

Quizás, las partes más tardadas son cuando aparecen bugs, de los cuales puedo decir que hay de dos tipos según mi experiencia.

Los que aparecen cuando no sabemos como utilizar correctamente ciertos elementos del código, es decir, tenemos que experimentar con prueba y error, hasta lograr completar la funcionalidad, sin embargo, esto consume demasiado tiempo, aun leyendo la documentación.

La segunda son aquellas en donde creemos saber como funciona, pero cometemos errores en la codificación. Esto nos lleva a creer que el problema no somos nosotros, sino cualquier otra cosa, cuando no es así, es por eso por lo que hay que aprender a aceptar los errores, pues el asegurar que nuestro código no contiene errores, cuando en realidad los tiene, puede producir muchas complicaciones y quebraderos de cabeza.

Por último, debó decir que lo más cansado y aburrido es estar repitiendo el código entre ventanas, es decir, si tenemos dos, tres o cuatro ventanas en donde la funcionalidad es parecida, pero debemos hacer algunos cambios, se vuelve tedioso. Personalmente prefiero automatizar ese tipo de situaciones, decantándome más por los momentos en donde estoy creando nuevas funcionalidades.

Orgullo

La vista que más me gustó es en donde se muestran los detalles de los productos, pues me llevo mucho tiempo hacer que funcionara el carousel, además de la funcionalidad agregada que permite abrir la imagen en ventana completa, pues el carousel no tiene esa función.

Backend

El código del backend esta hecho en archivos individuales de PHP, que, aunque me hubiera gustado utilizar algún framework como Laravel para hacer más optimo el código, la verdad es que no tengo mucha experiencia en este lenguaje.

Cada endpoint solicita una cabecera en donde debe ir el token JWT, además de los elementos del GET. Estos endpoints tienen control de excepciones, así como mensajes de respuesta en caso de errores, por lo que se utilizan JSONs para las respuestas.

El backend esta hospedado en la plataforma “000webhost”, la cual permite conexión FTP, haciendo más sencilla la subida de los archivos sin necesidad de estar todo el tiempo en el navegador.

Una funcionalidad extra es la carga de imágenes como URL. La aplicación manda al servidor las imágenes en base64, el servidor las recibe y las manda mediante una API REST a imgur.com, en donde se guarda y retorna la dirección URL, la cual se almacena en la base de datos, esto soluciona la limitación del visor de imágenes en tamaño completa, pues este solo abre imágenes desde URLs y no del dispositivo.

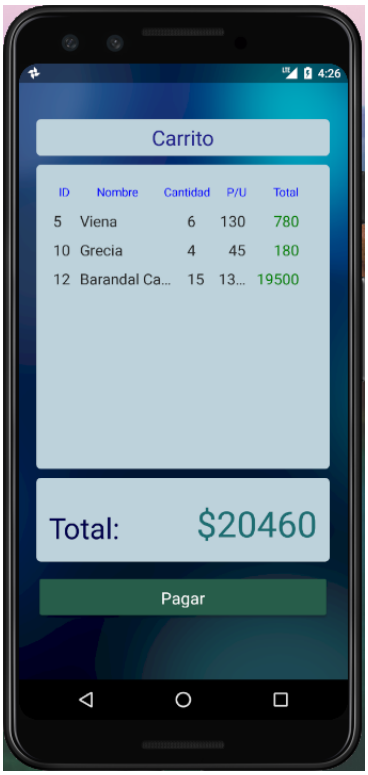
Manual de Usuario

A continuación, se expone un pequeño manual de usuario con imágenes de la aplicación.

Ventana: Inicio de sesión
Entradas: • Usuario • Contraseña
Botones: • Iniciar sesión (1)
Descripción: Es la primera vista de la aplicación (si anteriormente no existió un loggeo, o la sesión expiró). No permite el registro desde aquí, ya que solo los administradores pueden crear usuarios desde adentro de la app.
Lleva a: • (1): Página principal.

Ventana: Página principal
Entradas:
Botones: • Carrito (1) • Inventario (2) • Administrador (3)* • Cerrar sesión (4)
Descripción: Esta es la página principal, siempre será la primera si la sesión es válida, pues se almacena el token. Al cerrar la sesión se borra este token. Muestra la información básica del usuario actual.
Lleva a: • (1) Carrito • (2) Inventario • (3) Administrador • (4) Inicio de sesión
*Siempre y cuando el usuario tenga permisos de administración, en caso contrario, no aparece esta opción.

Ventana: Carrito
Entradas:
Botones: • Pagar (1)
Descripción: Muestra los productos agregados al carrito. Estos productos son almacenados en SQLite y se borran hasta hacer el pago.
Lleva a: • (1): Pagado.



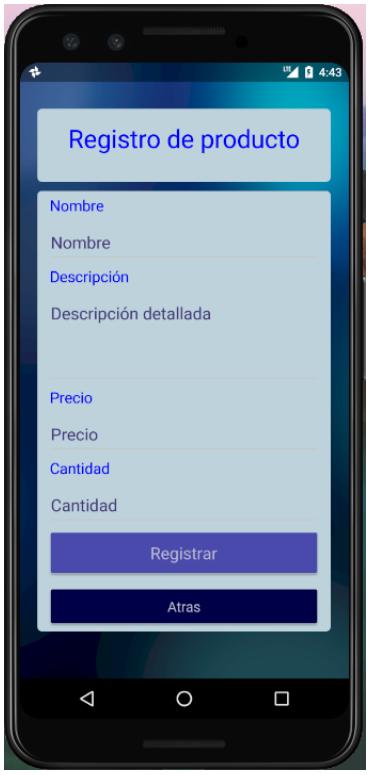
Ventana: Inventario
Entradas: • Buscador
Botones: • Presionar producto (1)
Descripción: Muestra todos los productos, en caso de ser administrador mostrara incluso los eliminados, si se es vendedor, solo mostrara los activos. Si se desplaza la lista hacia abajo, se actualizará. El buscador filtra la lista con cada letra presionada. Se puede buscar por cualquiera de las columnas.
Lleva a: • (1): Detalles de producto.

Ventana: Administrador
Entradas:
Botones: • Administrar Productos (1) • Administrar Empleados (2) • Regresar (3)
Descripción: Permite administrar dos categorías.
Lleva a: • (1): Administrador Productos. • (2): Administrador Empleados. • (3): Página principal.

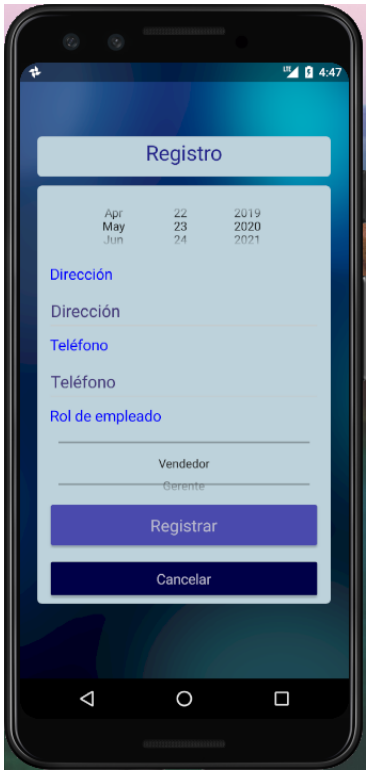
Ventana: Administrador de Productos
Entradas:
Botones: • Registrar nuevo producto (1) • Modificar producto (2) • Regresar (3)
Descripción: Permite ir al registro o modificación de productos. La opción de modificar producto lleva al inventario, en donde se debe buscar el producto a modificar.
Lleva a: • (1): Registrar producto. • (2): Inventario. • (3): Administrador

Ventana: Administrador de Empleados
Entradas:
Botones: • Registrar nuevo empleado (1) • Modificar producto (2) • Regresar (3)
Descripción: Permite ir al registro o modificación de empleado. La opción de modificar empleado lleva a una lista de empleados, en donde se debe buscar el empleado a modificar.
Lleva a: • (1): Registrar empleado. • (2): Modificar empleado. • (3): Administrador

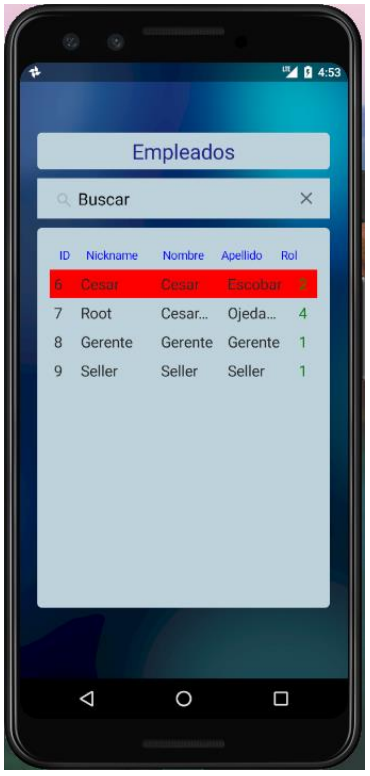
Ventana: Registro de producto
Entradas: • Nombre • Descripción detallada • Precio • Cantidad
Botones: • Registrar (1) • Atrás (2)
Descripción: Crea un nuevo producto en la base de datos. El estado será de Programado hasta que se agregue una imagen al producto, en tal caso, un trigger de la base de datos cambiará el estado a activo.
Lleva a: • (1): Administrador productos • (2): Administrador productos



Ventana: Registro empleados
Entradas: • CURP • Nickname • Nombre • Apellido • Contraseña • Repetir contraseña • Fecha de nacimiento • Dirección • Telefono • Rol
Botones: • Registrar (1) • Cancelar (2)
Descripción: Registra un nuevo empleado en la base de datos. Se valida que la CURP, ni el nickname estén repetidos. Además de que las dos contraseñas coincidan.
Lleva a: • (1): Administrador de Empleados. • (2): Administrador de Empleados.



Ventana: Lista empleados
Entradas: • Buscador
Botones: • Presionar producto (1)
Descripción: Muestra todos los empleados. Si el empleado esta deshabilitado, se muestra en rojo. Si se desplaza la lista hacia abajo, se actualizará. El buscador filtra la lista con cada letra presionada. Se puede buscar por cualquiera de las columnas.
Lleva a: • (1): Modificar empleado.



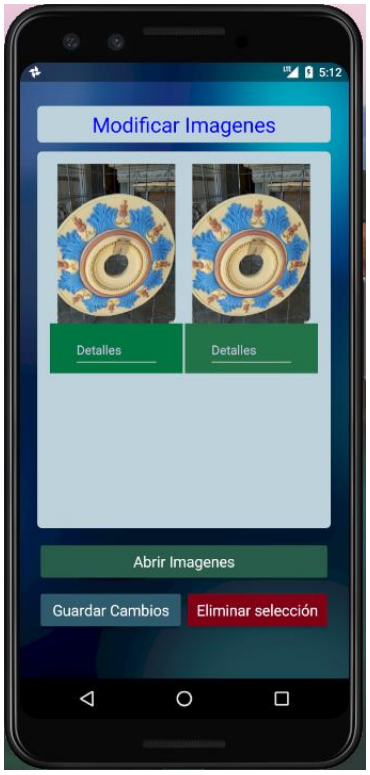
Ventana: Modificar empleado
Entradas: • CURP • Nickname • Nombre • Apellido • Contraseña • Repetir contraseña • Fecha de nacimiento • Dirección • Telefono • Rol • Estado
Botones: • Guardar cambios (1)
Descripción: Todos los campos mostraran la información actual como hint (excepto la contraseña), solo se actualizarán los campos en los que se agregue información. Se valida que la CURP, ni el nickname estén repetidos. Además de que las dos contraseñas coincidan.
Lleva a: • (1): Lista empleados.

Ventana: Detalles de producto
Entradas:
Botones: • Presionar imagen (1) • Agregar al carrito (2) • Modificar (3)*
Descripción: Muestra un carousel, con las imágenes almacenadas del producto, además del título de la foto. Muestra toda la información formateada, de forma que sea agradable la vista. Si se presiona sobre alguna imagen, esta se mostrará en tamaño completo.
Lleva a: • (1): Imagen en tamaño completo. • (2): Selector de cantidad. • (3): Modificar producto.
*Esta opción solo aparecerá para usuarios administradores.

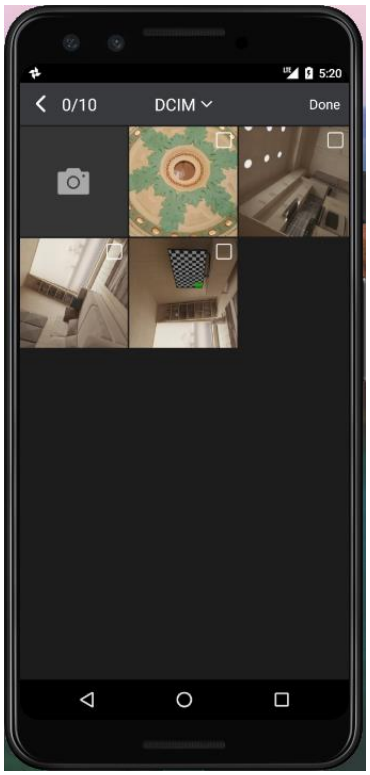
Ventana: Imagen en tamaño completo
Entradas:
Botones: • Administrar Productos (1)
Descripción: Muestra en tamaño todas las imágenes del producto.
Lleva a: • (1): Siguiente imagen.

Ventana: Modificar producto
Entradas: • Nombre • Descripción • Precio • Cantidad • Estado
Botones: • Guardar cambios (1) • Actualizar imágenes (2)
Descripción: Todos los campos mostraran la información actual como hint. Solo se actualizarán los campos en los que se agregue información.
• Modificar imágenes

Ventana: Modificar imágenes
Entradas: • Titulo de cada imagen
Botones: • Abrir imágenes (1) • Guardad cambios (2) • Eliminar selección (3) • Presionar imagen (4) • Gesto: Mantener presionada la imagen*
Descripción: Muestra todas las imágenes de un producto de forma que se pueda modificar el título. La eliminación y carga se lleva a cabo hasta que se de en guardar cambios.
Lleva a: • (1): Galería del teléfono + Cámara** • (2): Modificar producto. • (3): Acción: Borra las imágenes seleccionadas. • (4): Imagen en tamaño completo
*Seleccionará la imagen y producirá una vibración de 50 ms. Se pueden seleccionar varias imágenes. **Muestra las fotos la memoria interna del teléfono, además de dar una opción para abrir la cámara. Solicitara permiso de acceso a las carpetas y a la cámara.



Ventana: Galería del teléfono + Cámara
Entradas:
Botones: • Presionar imagen (1) • Abrir cámara (2) • Marcar imagen (3)
Descripción: Permite ver las imágenes en ventana completa. Permite seleccionar un máximo de 10 imágenes. Permite abrir la cámara del teléfono. Considere que mientras más imágenes carga, mayor será el tiempo de carga.
Lleva a: • (1): Imagen en pantalla completa • (2): Cámara del teléfono • (3): Acción: Selecciona la imagen.



Ventana: Selector de cantidad
Entradas: • Cantidad
Botones: • Agregar al carrito (1)
Descripción: Muestra el total se la cantidad por el valor. Muestra el carousel de imágenes y el nombre. Si la cantidad seleccionada excede a la existente, marca en rojo la cantidad y no permite agregar al carrito. Si la cantidad es 0 o vacío, no se agrega nada al carrito.
Lleva a: • (1): Carrito

Ventana: Pagado
Entradas:
Botones: • Ir a inicio (1)
Descripción: Muestra un mensaje de venta realizara, y se limpia el carrito.
Lleva a: • (1): Pagina principal.

Errores conocidos

- La aplicación tiene algunos errores con la lógica que ya no hubo tiempo de corregir.
- En ocasiones, el servidor da errores “404”, por lo que es algo inestable, sin embargo, se maneja este tipo de excepciones con mensajes de alerta.
- Al agregar varias imágenes, solo se guarda n-(imágenes) de veces la primera imagen seleccionada.
- No esta implementada la función para modificar el carrito, solo se puede agregar elementos y eliminar todos haciendo la venta.
- La modificación del titulo de la imagen no funciona.
- En la ventana “Modificar imágenes”, no se pueden hacer pantalla completa las imágenes que se hayan cargado desde el teléfono y no hayan sido cargadas. (Limitación del plugin para ver las imágenes en tamaño completo).