Módulo 2 - Lectura: Templates personalizadas para Nativescript

Nativescript cuenta con un conjunto de componentes de interfaz de usuario (también conocidos como widgets) que se puede utilizar para crear esa interfaz en una aplicación móvil.

La mayoría de estas vistas o componentes envuelven o adaptan a la implementación nativa correspondiente para cada plataforma, al tiempo que proporcionan una API común para trabajar con ella. Por ejemplo, el componente Button de Nativescript muestra un android.widget.Button en Android y UIButton en iOS.

La definición del diseño de la aplicación también es una parte importante del desarrollo de la aplicación. Para obtener más información sobre los diferentes contenedores de diseño que están disponibles en Nativescript, consulta la siguiente lectura sobre Layouts, pero por ahora nos centraremos en los widgets.

Cuando usamos Nativescript con Angular, a nuestros componentes le indicamos una cadena de texto o archivo como plantilla y, si utilizamos un archivo como plantilla, la convención es que se use la extensión HTML, pero en verdad no estaremos usando un HTML.
Lo que usa Nativescript es un XML, que referencia a componentes nuestros o de librerías y que todos ellos, en algún momento, terminan llegando a componentes hijos que usan alguno de los componentes básicos que aquí veremos.

Acceso a la implementación Nativa:

Se puede acceder al widget nativo subyacente para cada vista en tiempo de ejecución, utilizando la propiedad nativeView:

<view> .nativeView

El acceso a los widgets nativos puede ser útil cuando deseas usar algunas funcionalidades específicas de la plataforma del widget. Puedes encontrar información sobre el componente nativo subyacente para cada vista, a continuación.

Principales Widgets de UI:

Botón: Button
El widget Button proporciona un widget de botón estándar que reacciona a un evento de toque o tap.

Etiqueta: Label
El widget Label proporciona una etiqueta de texto de solo lectura.

Campo de texto: TextField
El widget TextField proporciona un campo de texto de una sola línea editable.

Vista de texto: TextView
El widget TextView proporciona una vista de texto multilínea editable. Puede usarse para mostrar texto de varias líneas e implementar la edición de texto.

Barra de búsqueda: SearchBar
El widget SearchBar proporciona una interfaz de usuario para ingresar consultas de búsqueda y enviar solicitudes al proveedor de búsqueda.

Opción doble: Switch
El widget de Switch proporciona un interruptor de conmutación de dos estados desde el que puedes elegir entre dos opciones.

Deslizador: Slider
El widget Slider proporciona un control deslizante que puedes utilizar para elegir un valor numérico dentro de un rango configurable.

Progreso: Progress
El widget Progress es un indicador en forma de barra que da indicio o permite seleccionar un progreso en una operación. Muestra una barra que representa el progreso actual de la operación.

Indicador de actividad: ActivityIndicator
El widget ActivityIndicator es un indicador de giro visual que muestra que una tarea está en progreso.

Imagen: Image
El widget Image muestra una imagen. Puedes cargarla desde un ImageSource o desde una URL.

Vista de la lista: ListView
El ListView muestra elementos en una lista de desplazamiento vertical. Puedes configurar un itemTemplate para especificar cómo debe mostrarse cada elemento de la lista.

HtmlView
El HtmlView representa una vista con contenido HTML. Usa este componente en lugar de WebView cuando desees mostrar solo contenido HTML estático.

WebView
La vista web muestra páginas web. Puedes cargar una página desde una URL o navegando hacia atrás y hacia adelante.

TabView
Con el control TabView, puedes implementar la navegación por pestañas.

Barra segmentada: SegmentedBar
Con el control de barra segmentada, puedes implementar una selección discreta.

Selector de fechas: DatePicker
Con el control DatePicker, puedes elegir una fecha.

Selector de tiempo: TimePicker
Con el widget TimePicker, puedes elegir una hora.

ListPicker
Con el widget ListPicker, puedes elegir un valor de una lista.

Diálogos: Dialogs
El módulo de diálogos te permite crear y mostrar ventanas de diálogo.

Puedes ver imágenes explicativas de cada uno en la documentación oficial en:
https://docs.nativescript.org/ui/components

Uso interactivo en Playground:

Te invitamos a que pruebes, interactivamente, en el Playground online de Nativescript, cada uno de estos controles, dado que son muy sencillos de disponer sobre un documento con un simple arrastrar y soltar.
Ingresa al playground en: https://play.nativescript.org
Recomiendo que siempre los arrojes sobre un Stack Layout, de esa manera aparecerán uno debajo del otro y podrás verlos sin problemas.
Para ver la aplicación, solamente deberás darle click al botón Preview arriba a la derecha del sitio y escanear el QR que el sitio te muestre, para que luego puedas usar dicha aplicación en su dispositivo.

Componentes propios:

Cuando nosotros desarrollemos nuestros propios componentes, en Angular, al momento de definir la clase Typescript, la misma será decorada con un Decorator/Annotation ydicho decorador recibirá un objeto de configuración, por ejemplo:

```
@Component({
	selector: 'ListadoAlumnos',
	templateUrl: './listado-alumnos.component.html'
})
export class ListadoAlumnosComponent { /* . . . */ }
```

De esta manera, estamos definiendo un selector que será el nuevo nombre del tag que estamos creando para poder utilizar nuestro componente.
Es decir, si otro componente quiere utilizar este listado, deberá escribir:

```
<ListadoAlumnos></ListadoAlumnos>
```

Importante:

Observa que para que todos los widgets sean de Nativescript o definidos por nosotros o, como veremos a futuro, importados de terceros, en todos los casos, tenemos que usar tags de cierre con el nombre completo, es decir:

```
<ListadoAlumnos></ListadoAlumnos>
```

Y no podemos usar:

```
<ListadoAlumnos />
```

Esto se debe a que los tags auto-cerrados no son compatibles con el motor de plantillas de Angular. Es decir que no es por NativeScript, sino que, por el momento, es una limitación de Angular.

Sintaxis de Plantillas en Angular

La aplicación Angular gestiona lo que el usuario ve y puede hacer, logrando esto mediante la interacción de una instancia de clase de componente (el componente) y su plantilla orientada al usuario.

Puedes estar familiarizado con la dualidad de componente / plantilla de tu experiencia con model-view-controller (MVC) o model-view-viewmodel (MVVM). En Angular, el componente reproduce la parte del controlador / modelo de vista, y la plantilla representa la vista.

Interpolación y expresiones de plantilla:

La interpolación te permite incorporar cadenas calculadas en el texto entre las etiquetas de elementos HTML y dentro de las asignaciones de atributos. Las expresiones de plantilla son las que usas para calcular esas cadenas.

La interpolación se refiere a la incorporación de expresiones en el texto de marcado o tags. Por defecto, la interpolación usa como delimitador las llaves dobles, {{y}}. Un ejemplo de interpolación es el siguiente:

```
<h3> Alumno actual: {{ actual.nombre }} </h3>
```

Donde actual es una variable o atributo de nuestro objeto componente en Typescript, y nombre es un atributo de dicho objeto.
Es decir que nuestro componente en Typescript sería como:

```
export class DetalleAlumnoComponent {
        constructor(private actual: Alumno) {}
}
```

Y nuestro modelo Typescript para el Alumno sería como:

```
export class Alumno {
```

```
        constructor(public nombre: string) {}
    }
```

Si bien el texto entre las llaves es a menudo el nombre de una propiedad de componente, también puede ser una variable interna de la plantilla, como verás en futuras lecciones.

Angular reemplaza el nombre con el valor de cadena de la propiedad del componente correspondiente.

Otro ejemplo:

```
<p> {{actual.nombre}} </p>
<div> <img src = "http://miapp.com/fotos/{{actual.nombre}}"> </div>
```

En el ejemplo anterior, Angular evalúa contenido del tag “p” y las propiedades del “img” en este caso la propiedad “src”, y en ambos casos serializa en dichas posiciones el valor del nombre del alumno que está asignado a la propiedad “actual”.

Más generalmente, el texto entre las llaves es una expresión de plantilla que Angular primero evalúa y luego se convierte en una cadena.

Data Binding

Sin un framework, serías responsable de insertar valores de datos en los controles HTML y convertir las respuestas de los usuarios en acciones y actualizaciones de valor.
Escribir tal lógica de empujar y tirar a mano es tedioso, propenso a errores y una pesadilla para leer, como lo puede atestiguar cualquier programador jQuery experimentado.

Angular admite el enlace de datos bidireccional, un mecanismo para coordinar las partes de una plantilla con las partes de un componente. Agrega un marcado de enlace a la plantilla HTML para decirle a Angular cómo conectar ambos lados.

Existen diferentes tipos de Bindings, empecemos con Bindings One Way, es decir, en un solo sentido:

1. <p>{{ valor }}<p> es una interpolación y muestra el valor de la propiedad del componente dentro del elemento <p>.
2. <Alumno [nombre]=”unNombre”></Alumno> el enlace de propiedad pasa el valor de la variable de Typescript llamada “unNombre” (que pertenece al componente de la plantilla) a la propiedad “nombre” del componente que estamos invocando llamado AlumnoComponent. Debes tener en cuenta que unNombre es una variable Typescript, pero no requerimos usar {{ y }}, ya que estamos asignándolo a un atributo con corchetes, es decir [y], y Angular en este caso asume que es código Typescript lo que tenemos en el lado derecho de la asignación.
3. <Button (tap)=”elegido(actual)”></Button> el enlace de eventos llama al método “elegido” en la clase Typescript del componente cuando el usuario hace clic en el Button. Fíjate que en

este caso, como en el punto anterior, también es código Typescript lo que se dispone a la derecha del igual.

El enlace de datos bidireccional (utilizado principalmente en formularios controlados por plantillas) combina el enlace de propiedades y eventos en una sola notación. Aquí un ejemplo:

```
<TextBox [(ngModel)]="nombre"></TextBox >
```

En el enlace de dos vías, un valor de propiedad de datos fluye al cuadro de entrada desde el componente, como con el enlace de propiedad. Los cambios del usuario también vuelven al componente, restableciendo la propiedad al último valor, como con el enlace de eventos.

Angular procesa todos los enlaces de datos una vez para cada ciclo de eventos de JavaScript, desde la raíz del árbol de componentes de la aplicación hasta todos los componentes secundarios.

El enlace de datos juega un papel importante en la comunicación entre una plantilla y su componente, y también es importante para la comunicación entre los componentes principal y secundario.