Taller de desarrollo para dispositivos móviles

Material teórico

Docente: Bruno Díaz
Enero, 2022

Índice

Introducción

Temario

- Introducción al taller.
- Arquitectura de una aplicación móvil.
- Tecnologías a utilizar en el curso.
- Repaso de HTML y JavaScript.
- Desarrollo progresivo de un ejercicio integrador.
    - Análisis.
    - Definición de estructuras.
    - Desarrollo progresivo.
- API REST.
- Protocolo HTTP.
- Fetch Then y Catch.
- Introducción a frameworks de desarrollo móvil.
    - Ionic.
- Build de la aplicación.

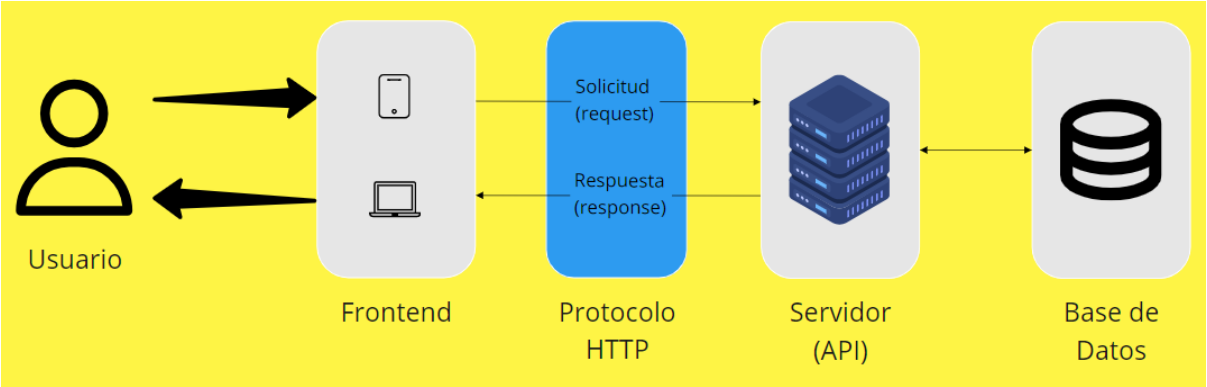
Generalidades del taller

El taller se enfoca en la arquitectura, el diseño y la experimentación de nuevas tecnologías, como el desarrollo de aplicaciones móviles (smartphones y tablets). Realiza un proyecto basado en la aplicación de herramientas integradoras y nuevos entornos de desarrollo.

Arquitectura de una aplicación móvil

A grandes rasgos, podemos decir que en general, tras las aplicaciones móviles (y muchas de las aplicaciones web) encontramos tres grandes partes: frontend, backend y base de datos.
El frontend podemos describirlo como aquella parte con la que el usuario final puede interactuar directamente. En el caso de las aplicaciones móviles, sería la aplicación propiamente dicha.
Por otro lado, requerimos de algo que se encargue de procesar toda la información que el usuario desea ingresar en la aplicación, así como verificar si el mismo tiene permisos para acceder a la información que solicita, etc. Toda esta lógica es la que pondremos en lo que llamamos nuestro backend.
Finalmente, requerimos de un lugar en donde almacenar (por ejemplo) la información que el usuario desea guardar para poder recuperar en un futuro. Aquí es donde entra en juego nuestra base de datos.
Ahora bien, como el frontend y el backend pueden estar desarrollados en diferentes lenguajes, e incluso por diferentes personas, la comunicación entre ambos no se realiza de forma directa, sino que para eso podemos desarrollar una API (Application Programming Interface o Interfaz de Programación de Aplicaciones). La API no es más que un conjunto de subrutinas, funciones y procedimientos creadas para proveer acceso a funciones ya desarrolladas en un determinado software.

La comunicación entre el frontend y la API, se realiza utilizando el protocolo HTTP/S.



Tecnologías a utilizar en el curso

- HTML5.
- JavaScript (ES6).
- Ionic.
  - https://ionicframework.com/
- Capacitor.
  - https://capacitorjs.com/
- Node.js
  - https://nodejs.org/es/
- Android Studio
  - https://developer.android.com/studio

Repaso de HTML y JavaScript

HTML

- HyperText Markup Language (Lenguaje de marcas de hipertexto).
- Es un lenguaje que describe la estructura general del contenido de un documento.
- El lenguaje HTML nos permite mezclar texto (que será visible para los usuarios) con etiquetas de marca (que definirán el tipo de contenido que se quiere mostrar al usuario).
- No es sensible a mayúsculas y tanto el múltiple espaciado como la nueva línea se ignoran si los escribimos en lenguaje natural.
- En un elemento HTML encontramos lo que se conoce por DOM (Document Object Model). Esto hace referencia a la estructura jerárquica de elementos dentro del documento.

Etiquetas

- Las etiquetas HTML son comandos escritos entre signos <>.

- Existen versiones de apertura y de cierre para la mayoría de las etiquetas, pero no para todas.

```
<nombreEtiqueta> Contenido etiqueta </nombreEtiqueta>
```

Etiqueta de apertura. | Contenido (lo que ve el usuario). | Etiqueta de cierre.

- Tanto las etiquetas de apertura como las de cierre usan el mismo comando, pero la etiqueta de cierre agrega una /.

```
<DIV>      </DIV>
"abre"      "cierra"

<IMG src="foto1.jpg" width="50" height="20">
"Se abre y no necesita ser cerrada"
```

- Cada etiqueta puede tener atributos, y si bien en el caso anterior se muestra el src, width y height, existen varios más.
- Etiquetas de la estructura básica de un documento HTML:

```
<!DOCTYPE html>   → versión HTML.
<HTML lang="es">  → lenguaje de la página.
    <HEAD>
    </HEAD>       → Información que se necesita para que la página funcione. No la ve el usuario.
    <BODY>
    </BODY>       → Contenido de la página. Lo que ve el usuario.
</HTML>
```

- Ejemplo básico de página HTML:

```
<!DOCTYPE html>
<html lang="es">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <title>1era p&aacute;gina</title>
    </head>
    <body>
        <h1>Mi p&aacute;gina principal!!!</h1>
        <p>Hola mundo!!!</p>
    </body>
</html>
```

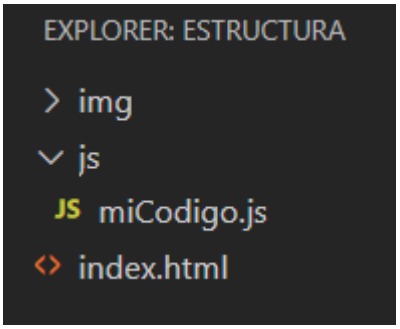
- Existen diversas etiquetas HTML, por lo que nombraremos únicamente algunas de las más utilizadas:
    - h1 … h6

- div
- p
- br
- span
- ul
- ol
- li
- hr
- input
    - type:
        - button
        - checkbox
        - date
        - file
        - number
        - password
        - radio
        - text

JavaScript

Para darle “comportamiento” a nuestra web, es necesario darle instrucciones específicas y decirle cuándo se deben ejecutar (al hacer click en un botón, al pasar el cursor del mouse por encima de un campo, etc.).
Aunque existen otras formas, las instrucciones que le daremos al navegador, estarán escritas en lenguaje JavaScript.
JavaScript es un lenguaje de programación interpretado, por lo que no es necesario compilar los programas para ejecutarlos en un navegador web.

Para poder utilizar JavaScript en nuestra página web, necesitamos hacer uso de la etiqueta script, ya sea para escribir el código dentro de ella, o para vincular un archivos que contenga el código.
Por ejemplo, en la carpeta destinada a nuestro proyecto, al mismo nivel de nuestro .html podremos crear una carpeta con nombre “js” y allí colocar nuestro/s archivo/s JavaScript.



Ahora, nos falta vincular esos archivos para que trabajen juntos y para esto, utilizaremos la etiqueta antes mencionada.

```
<script type=”text/javascript”>
      // Aquí podría escribir código JavaScript
</script>
```

En nuestro caso, esta etiqueta la pondremos dentro del bloque <head> de la página, a fin de que cuando se ejecute en el navegador, el código se cargue en memoria.

```html
<!DOCTYPE html>
<html lang="es">
    <head>
        <title>Título de la página</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <script type="text/javascript">
            alert("Bienvenido al sitio");
        </script>
    </head>
    <body>
        <h1>Encabezado de prueba</h1>
        <label for="inputTexto"></label>
        <input id="inputTexto" type="text">
        <br><br>
        <input id="miBoton1" type="button" value="Botón de prueba">
    </body>
</html>
```

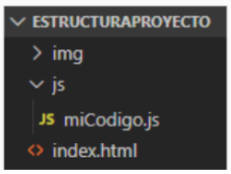
Si bien podemos (como vemos en el ejemplo anterior) poner nuestro código JavaScript directamente en el HTML, por un tema de prolijidad dividiremos el HTML del JavaScript.
Para esto, en la etiqueta <script> agregaremos la información de dónde se encuentra el código que debería ir dentro de la misma. Esto se hace con el atributo src.

```
<script type=”text/javascript” src=”js/miCodigo.js”></script>
```

A esto se le llama JavaScript no intrusivo y es una buena práctica, ya que permite separar completamente el código de la interfaz de usuario (HTML) del comportamiento de la aplicación (JavaScript).

Ejemplo completo:

Estructura



miCodigo.js

```
alert("Bienvenido al sitio");
```

index.html

```
<!DOCTYPE html>
<html lang="es">
    <head>
        <title>Título de la página</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <script type="text/javascript" src="js/miCodigo.js"></script>
    </head>
    <body>
        <h1>Encabezado de prueba</h1>
        <label for="inputTexto"></label>
        <input id="inputTexto" type="text">
        <br><br>
        <input id="miBoton1" type="button" value="Botón de prueba">
    </body>
</html>
```

Para interactuar desde JavaScript con el HTML, es importante (y necesario) que cada elemento HTML esté identificado. Para hacerlo, debemos agregar (por ejemplo) el atributo id a cada una de las etiquetas HTML sobre las cuales podamos querer interactuar.
Por ejemplo, es seguro que vamos a querer interactuar con los botones (para asignarles comportamiento) o con los campos de texto (para poder leer lo que el usuario ingresa).
El valor del atributo id puede ser cualquier combinación de letras y números, siempre que comience por una letra.
Ejemplo: `<input type="text" id="txtNombre">`.

Selectores

Existen varios selectores que podemos utilizar para interactuar con etiquetas del DOM desde nuestro JavaScript.
Veremos algunas de las más importantes:
Selector por tipo de etiqueta:
document.querySelector("tipoEtiqueta")
Ejemplo: document.querySelector("button")
Selector por valor del atributo id:
document.querySelector('#id')
Ejemplo: document.querySelector("#txtNombre")
Selector por valor del atributo class:
document.querySelector(".clase")
Ejemplo: document.querySelector(".textoRojo")
Selector por valor de un atributo igual a un valor dado:
document.querySelector("[atributo='valor']")
Ejemplo: document.querySelector("[name='radioButtonName']")
Los selectores son útiles porque una vez que logramos acceder a un elemento del DOM, podemos aplicar diferentes métodos, que nos sirven (por ejemplo) para leer contenido o darle determinado comportamiento.
document.querySelector("#idElemento").value → Permite obtener el valor del atributo value del elemento que contiene el id indicado.
document.querySelector("#idElemento").innerHTML += parametro → Permite agregar lo que se indique como parámetro dentro del elemento con el id indicado.

Variables y constantes

Las variables las podemos utilizar para almacenar valores y pueden ser de diferentes tipos. Además, podemos sobreescribir una variable en cualquier momento para cambiar así el valor que guarda.
Las constantes, son como variables pero que no cambiarán su valor a lo largo de la ejecución de nuestro programa.
Definiremos nuestras variables con la palabra let y nuestras constantes con la palabra const.

Diferencias entre null y undefined

Undefined

Este valor se muestra cuando algo no está definido o no se ha declarado.

Null

Este valor se muestra cuando una variable tiene asignada un valor nulo.

Estructuras de control

Condicionales

A menudo, necesitamos que ciertas acciones se ejecuten únicamente bajo determinada condición. Para esto, tenemos las estructuras de control condicionales.
Tenemos dos grandes tipos de ellas, y una la podemos dividir en 3 partes.

Switch

Esta estructura se basa en evaluar una variable y ejecutar diversas acciones, dependiendo del valor que tenga la misma.
Imaginemos que tenemos una variable llamada “mes”, la cual de alguna manera captura y guarda el mes actual.
En el ejemplo siguiente, ponemos únicamente el caso en el que se muestra “Enero”, pero podemos agregar un case por cada mes.

```
switch (mes) {
      case "Enero":
            // Código a ejecutar si la variable mes tiene el valor
      "Enero"
             break;
      default:
            // Código a ejecutar si la variable mes tiene cualquier
      valor para el que no haya realizado un case
}
// Código a ejecutarse luego de que termine el bloque del switch
```

If

Es la estructura más sencilla del lenguaje. Se evalúa una condición y en caso de arrojar un resultado verdadero (true), se ejecuta el código encerrado en el bloque if.

```
if (condicion) {
      // Código a ejecutarse si se cumple la condición
}
// Código a ejecutarse luego de que termine el bloque del If
```

if … else

En este caso, tenemos la posibilidad de evaluar una condición y ejecutar un bloque de código en caso de arrojar un resultado verdadero (true) pero con el agregado de poder ejecutar otro código diferente en caso de que la condición arroje un resultado falso (false).

```
if (condicion) {
      // Código a ejecutarse si se cumple la condición
} else {
      // Código a ejecutarse si no se cumple la condición
}
//  Código  a  ejecutarse  luego  de  que  termine  el  bloque
(independientemente de si se entró al if o al else)
```

If … else if

En este último caso, tenemos la posibilidad de evaluar diversas condiciones. Debemos tener en cuenta que (al igual que en el switch) se ejecutará únicamente un bloque.

```
if (condicion1) {
      // Código a ejecutarse si se cumple la condición 1
} else if (condicion2) {
      // Código a ejecutarse si no se cumple la condición 1 y se
cumple la condición 2.
}
//  Código  a  ejecutarse  luego  de  que  termine  el  bloque
(independientemente  de  si  se  cumplió  o  no  alguna  de  las
condiciones)
```

Cabe destacar que podemos agregar tantos else if como se quieran, y que podemos también agregar al final un bloque else para controlar el caso en el que no se cumple ninguna de las condiciones.

Repetitivas

Las estructuras de control repetitivas permiten ejecutar una instrucción o un conjunto de instrucciones varias veces. Una ejecución repetitiva de sentencias se caracteriza por la o las sentencias que se repiten y el test o prueba de condición antes de cada repetición.
Conocemos tres tipos de estructura repetitiva:

While

```
while (condicion) {
      // código a repetir
}
// Código a ejecutarse luego de terminado el while
```

Es importante asegurarse de que la condición se pueda dejar de cumplir en algún momento. Por ejemplo: si tengo una variable que se incrementa (o decrementa) para contar y controlar la cantidad de repeticiones, no debo olvidar realizar el incremento o decremento correspondiente.
El while, ejecutará las sentencias entre llaves cero o más veces.

Do while

```
do {
      // código a repetir
} while (condicion)
// Código a ejecutarse luego de terminado el do while
```

Al igual que en el while, es importante asegurarse de que la condición se pueda dejar de cumplir en algún momento.
A diferencia del while, el do while ejecutará las sentencias entre llaves una o más veces.

For

```
for (let i = 0; i < 10; i++) {
      // Código que se repetirá (en este caso, 10 veces)
}
// Código a ejecutarse luego de terminado el bloque for
```

A diferencia del while y el do while, el for se puede encargar “por sí mismo” de que la ejecución se corte en algún momento, lo cual no quita que para hacerlo, debe estar bien programado.

Protocolo HTTP

HTTP, de sus siglas en inglés “HyperText Transfer Protocol” (Protocolo de Transferencia de Hipertexto), es el nombre de un protocolo el cual nos permite realizar una petición de datos y recursos, como pueden ser documentos HTML.
Es la base de cualquier intercambio de datos en la Web, y un protocolo de estructura cliente-servidor, esto quiere decir que una petición de datos es iniciada por el elemento que recibirá los datos (cliente).
Los mensajes que envía el cliente, normalmente se llaman peticiones, y los mensajes enviados por el servidor se llaman respuestas.
Para poder mostrar una página web, el navegador envía una petición de documento HTML al servidor. Entonces procesa este documento, y envía más peticiones para solicitar scripts,

hojas de estilo, y todos los demás datos que necesite (videos, imágenes, etc). El navegador une todos estos documentos y datos, y compone el resultado final: la página web.
Los scripts, también son ejecutados por el navegador y a su vez, estos pueden generar más peticiones, las cuales serán gestionadas por el navegador para (por ejemplo) actualizar la página web en consecuencia de la respuesta obtenida.
Al otro lado del canal de comunicación, está el servidor, el cual “sirve” los datos que ha pedido el cliente. Un servidor conceptualmente es una única entidad, aunque puede estar formado por varios elementos, que se reparten la carga de peticiones (load balancing), u otros programas.
Una petición está formada por varios campos, entre los que están:

- Method (método)
    - Define la acción que se desea realizar.
    - Existen varios, pero nos quedaremos con los que más utilizaremos: GET, POST y PUT.
- Headers (cabeceras)
    - Se utilizan para enviar información adicional junto a la petición.
    - Un header está compuesto por su nombre seguido de “:”, y a continuación su valor.
- Body (cuerpo)
    - Son los datos que se le pasan al servidor.

Por otro lado, una respuesta también está formada por varios campos, por lo que (al igual que con las peticiones) nos centraremos en los que más nos pueden interesar para el curso.

- Status code (código de estado)
    - Es un número que nos indica si la petición fue exitosa, o no, y debido a que.
    - Podemos diferenciar los códigos de estado en 5 grandes grupos, dependiendo del número con el que comienzan:
        - 1: respuestas informativas.
        - 2: respuestas satisfactorias.
        - 3: Redirecciones.
        - 4: Errores de los clientes.
        - 5: Errores de los servidores.
    - Se puede consultar el listado de códigos en esta web.
- Headers (cabeceras)
    - Al igual que en la petición, se utilizan para enviar información extra.
- Body (cuerpo)
    - También funciona igual que en la petición, contiene los datos que manda el servidor.

Fetch, Then y Catch

Para poder programar aplicaciones que se ejecuten en el cliente, mientras se mantiene la comunicación asíncrona con el servidor en segundo plano, utilizaremos el método fetch.
De esta forma, es posible realizar cambios sobre las páginas sin necesidad de recargarlas, mejorando la interactividad, velocidad y usabilidad en las aplicaciones.
El método antes mencionado puede recibir diferentes parámetros, por lo que nos centraremos en las formas de uso más simples que nos permitan resolver las necesidades que tenemos en el curso.

Ejemplo:

```
function funcionQueLlamaAPI() {
    fetch('Acá va la URL de nuestra API')
    .then(function (response) {response.json()})
    .then(function (data) {
        console.log(data);
        // En data, tenemos la información que nos devuelve la API
    })
    .catch(function (error) {
        console.log(error);
        // Podemos imprimir el error, leerlo y/o dar algún tipo de aviso al usuario
    });
}
```

Frameworks para desarrollo de aplicaciones móviles.

¿Qué es un framework?

Un framework es un entorno o marco de trabajo, un conjunto de prácticas, conceptos y criterios a seguir previamente estandarizados.
Los frameworks nos proporcionan una serie de herramientas (clases y funciones) ya listas para ser usadas. En definitiva, nos ahorran trabajo.

¿Qué nos solucionan los frameworks para desarrollo de aplicaciones móviles?

En primer lugar, facilitan el aprendizaje y el desarrollo de aplicaciones móviles. Con el desarrollo de un código, podemos generar aplicaciones para las plataformas deseadas (celulares, tablets, PCs y, en ocasiones, TV. En lo que refiere a los celulares, el desarrollo suele ser tanto para Android como para iOS.
El desarrollo de aplicaciones híbridas tiene una curva de aprendizaje mucho más suave que el desarrollo nativo, ya que para los mismos, se requiere tener conocimientos de lenguajes de programación como Java, Kotlin, XML, Swift, Objetive-C, entre otros.
Además, el desarrollo nativo implica desarrollar dos interfaces de usuario en paralelo, lo cual agrega la complejidad de tener que lograr que nuestros programas se vean lo más similar posible en ambos sistemas operativos.

Ionic

Es un framework que nos permite desarrollar aplicaciones para iOS nativo, Android y la web, desde una única base de código. Se integra con los principales frameworks de frontend, como Angular, React y Vue, aunque también se puede usar Vanilla JavaScript.