

ADM

Dudas?

The slide contains a collection of 18 cards, each representing a different aspect of the ADM (Architectural Design Methodology) framework. The cards are arranged in a grid-like pattern, some overlapping, and are set against a background of a long wooden pier extending into a body of water under a cloudy sky.

- Card 1:** Descripción General. A general overview of the ADM process.
- Card 2:** Fase de Planificación. The planning phase of the methodology.
- Card 3:** Fase de Diseño. The design phase, involving requirements analysis and system architecture.
- Card 4:** Fase de Implementación. The implementation phase, focusing on system development.
- Card 5:** Fase de Pruebas. The testing phase, ensuring system quality and functionality.
- Card 6:** Fase de Despliegue. Deployment, the final stage where the system is put into production.
- Card 7:** Fase de Mantenimiento. Continuous maintenance and evolution of the system.
- Card 8:** Fase de Evaluación. Evaluation of the system's performance and user satisfaction.
- Card 9:** Fase de Desarrollo. Development phase, often confused with the design phase.
- Card 10:** Fase de Implementación. Another instance of the implementation phase.
- Card 11:** Fase de Pruebas. Another instance of the testing phase.
- Card 12:** Fase de Despliegue. Another instance of the deployment phase.
- Card 13:** Fase de Mantenimiento. Another instance of the maintenance phase.
- Card 14:** Fase de Evaluación. Another instance of the evaluation phase.
- Card 15:** Fase de Desarrollo. Another instance of the development phase.
- Card 16:** Fase de Implementación. Another instance of the implementation phase.
- Card 17:** Fase de Pruebas. Another instance of the testing phase.
- Card 18:** Fase de Despliegue. Another instance of the deployment phase.



Prezi

ADM

Dudas?

The slide contains a collection of 18 cards, each representing a different aspect of the ADM (Architectural Design Methodology) framework. The cards are arranged in a grid-like pattern, some overlapping, and are set against a background of a long wooden pier extending into a body of water under a cloudy sky.

- Card 1:** Descripción General. A general overview of the ADM process.
- Card 2:** Fase de Planificación. The planning phase of the methodology.
- Card 3:** Fase de Diseño. The design phase, involving requirements analysis and system architecture.
- Card 4:** Fase de Implementación. The implementation phase, focusing on system development.
- Card 5:** Fase de Pruebas. The testing phase, ensuring system quality and functionality.
- Card 6:** Fase de Despliegue. Deployment, the final stage where the system is put into production.
- Card 7:** Fase de Mantenimiento. Continuous maintenance and evolution of the system.
- Card 8:** Fase de Evaluación. Evaluation of the system's performance and user satisfaction.
- Card 9:** Fase de Desarrollo. Development phase, often confused with the design phase.
- Card 10:** Fase de Implementación. Another instance of the implementation phase.
- Card 11:** Fase de Pruebas. Another instance of the testing phase.
- Card 12:** Fase de Despliegue. Another instance of the deployment phase.
- Card 13:** Fase de Mantenimiento. Another instance of the maintenance phase.
- Card 14:** Fase de Evaluación. Another instance of the evaluation phase.
- Card 15:** Fase de Desarrollo. Another instance of the development phase.
- Card 16:** Fase de Implementación. Another instance of the implementation phase.
- Card 17:** Fase de Pruebas. Another instance of the testing phase.
- Card 18:** Fase de Despliegue. Another instance of the deployment phase.



Prezi

Representación Condicional

- Podemos representar elementos y contenidos condicionalmente en función de atributos de datos o declaraciones de JavaScript
- Estos incluyen **v-if**, para mostrar un contenedor si una declaración equivale a verdadero, y **v-else**, para mostrar una alternativa.

v-if

- La directiva **v-if** se usa para representar condicionalmente un bloque.
- El bloque solo **se representará** si la expresión de la directiva **devuelve un valor verdadero**.

VueJS

- En la lógica deberíamos tener:

```
const app = Vue.createApp({  
    data() {  
        return {  
            mensajito: "Mi super App en Vue",  
            mostrar:true  
        }  
    }  
});  
app.mount('#contenedor');
```

index.html

```
<p v-if="mostrar">Texto que se mostrará si la variable tiene  
valor TRUE</p>
```

- **v-if** no solo funciona con valores booleanos verdadero / falso. Puede verificar si una propiedad de datos es igual a una cadena específica:

VueJS

index.html

```
<p v-if="selected == 'si' ">Se ve !</p>
```

app.js

```
return {  
    mensajito : "Mi super App en Vue",  
    mostrar : true,  
    selected : "si"  
}  
})
```

- El atributo **v-if** acepta operadores de JavaScript, por lo que puede verificar que no sea igual, mayor o menor que.

VueJS

index.html

```
<p v-if="valor >= 9">El numero es mayor o igual a 9</p>
<p v-if="valor === 9">El numero es igual a 9</p>
<p v-if="valor != 10">El numero distinto a 10</p>
```

app.js

```
return {
  valor : 9
}
```

v-else

- Permite hacer un elemento alternativo basado en lo contrario de la declaración v-if.
- Si eso da como resultado verdadero, se mostrará el primer elemento; de lo contrario, el elemento que contiene v-else lo hará.

VueJS

- **v-else** no tiene valor y se coloca dentro de la etiqueta del elemento.

```
<p v-if="mostrar">Texto que se mostrará si la variable tiene valor TRUE</p>
<p v-else>Texto que se mostrará si la variable tiene valor FALSE</p>
```

- El elemento con **v-else** debe seguir directamente al que contiene **v-if**; de lo contrario, la consola arrojará error.

✖ ▶ [Vue warn]: Error compiling template:

v-else used on element <p> without corresponding v-if.

```
6 |          <p v-if="mostrar">Texto que se mostrará si la variable tiene valor TRUE</p>
7 |          <p>Parrafo</p>
8 |          <p v-else="">Texto que se mostrará si la variable tiene valor FALSE</p>
9 |          ^^^^^^^^^^
10 |        </div>
(found in <Root>)
```

v-else-if

- Se usa para establecer condiciones adicionales que se evalúan sólo si la condición anterior es falsa.
- Debe seguir inmediatamente a un elemento v-if o v-else-if.

app.js

```
return {evaluar: 10}
```

index.html

```
<p v-if="evaluar == 1">Es igual a 1</p>
<p v-else-if="evaluar == 5">Es igual a 5</p>
<p v-else-if="evaluar == 15">Es igual a 15</p>
<p v-else-if="evaluar == 10">Es igual a : {{evaluar}}</p>
<p v-else>No coincide, el valor es : {{evaluar}}</p>
```

Template

- Podemos **englobar código** dentro de un elemento llamado **template**
- Como **v-if** es una directiva, tiene que estar adjunta a un solo elemento del HTML. Qué hacemos si queremos alternar más de un elemento?
- Podemos **utilizar v-if sobre un elemento <template>**, que sirve como un **envoltorio invisible**.
- El **resultado final del renderizado** no incluirá el elemento **<template>**
- Puede usarse además con las **directivas v-else-if y v-else**
- Nos permite **controlar el aspecto semántico del HTML**, ya que no genera etiquetas adicionales en nuestro código

v-if vs v-show

- **v-if** es una renderización condicional “real” donde los elementos son destruidos y recreados durante la alternancia.
- **v-if** también es diferido: si la condición es falsa en la representación inicial, no se hará nada.
- El bloque condicional no se procesará hasta que la condición se convierta en true por primera vez.
- Con **v-show** el elemento siempre se representa independientemente de la condición inicial, con alternancia basada en CSS.

- La documentación sugiere **v-show** para cambiar algo muy a menudo, y elegir **v-if** si es poco probable que la condición cambie en el tiempo de ejecución.

Iterar Arrays

v-for

- Permite representar una lista de elementos basada en una matriz.
- Requiere una sintaxis especial en forma de **item in items**.
- Los **items** son la matriz de datos y el **item** es un alias para el elemento de matriz que se está iterando:

```
<ul>
  <li v-for="item in peliculas">{{item}}</li>
</ul>
```

```
return {
  peliculas:[ "Mortal Kombat",
  "Sherlock Holmes",
  "Kung Fusion" ] }
```

Iterar Array de Objetos

```
juegos:[  
    {nombre: "Super Mario Bros", anio: 1985, calificacion: 9 },  
    {nombre: "Mortal Kombat", anio: 1992, calificacion:10 },  
    {nombre: "Street Fighter", anio :1987, calificacion: 8}  
]
```

```
<ul>  
    <li v-for="item in juegos">{{item.nombre}} : {{item.anio}}</li>  
</ul>
```

- Por cada propiedad del objeto que se quiera mostrar se debe aclarar en la sintaxis del html.

VueJS

- Dentro de los bloques **v-for** tenemos acceso completo a las propiedades del ámbito principal.
- v-for admite un segundo argumento opcional para el **índice** del elemento actual.

```
<ul>
  <li v-for="(item, index) in juegos">Indice: {{index}}, {{item.nombre}} : {{item.anio}}</li>
</ul>
```

- 
- Indice: 0, Super Mario Bros : 1985
 - Indice: 1, Mortal Kombat : 1992
 - Indice: 2, Street Fighter : 1987

VueJS

- Puede usar **of** como delimitador en lugar de **in**, de modo que esté más cerca de la sintaxis de JavaScript para los iteradores:

```
<ul>
  <li v-for="item of juegos">{{item.nombre}} : {{item.anio}}</li>
</ul>
```

Iterar un Objeto

```
mamifero: {
  animal: "gato", raza: "Persa", edad:4
}
```

VueJS

```
<ul>  
  <li v-for="value of obj">{{value}}</li>  
</ul>
```

- gato
- Persa
- 4

- También se puede proporcionar un segundo alias para el nombre de la propiedad (también conocido como **key**).

```
<ul>  
  <li v-for="(value, key) of obj">{{key}} : {{value}}</li>  
</ul>
```

- Clave : animal, Valor: gato
- Clave : raza, Valor: Persa
- Clave : edad, Valor: 4

VueJS

- Además podríamos pasarle un tercer argumento para mostrar el índice del elemento

```
<ul>
  <li v-for="(value, key, index) of obj"> {{index}} --> Clave : {{key}}, Valor: {{value}} </li>
</ul>
```

- 0 --> Clave : animal, Valor: gato
- 1 --> Clave : raza, Valor: Persa
- 2 --> Clave : edad, Valor: 4

- Al iterar sobre un objeto, el orden se basa en el orden de enumeración de claves, tal y como fue escrito en nuestro código.

VueJS

- Podemos sugerirle a Vue un **atributo key único** para cada elemento para que pueda rastrear la identidad de cada nodo y reordenar y reutilizar los elementos existentes.
- Los **valores** que admite el atributo key son de tipo **string o numéricos**
- Tenemos que usar **v-bind** para enlazarlo **v-bind:key**
- Se recomienda que se use el atributo **key** al usar **v-for** para aprovechar el **rendimiento de vue**

```
lista:[  
    {tarea: "Comprar más plantas ", id:1},  
    {tarea: "Ir a la ferretería", id:2},  
    {tarea: "Limpiar la cocina", id: 3},  
    {tarea: "Hacer bici", id:4}  
]
```

VueJS

```
<ul>
  <li v-for="item in lista" :key="item.id" > {{item.tarea}} </li>
</ul>
```

- Comprar más plantas
- Ir a la ferretería
- Lavar la cocina
- Hacer bici

- Con **v-bind** podríamos manipular su estilo visual con CSS en **base al estado true o false de la tarea**. Usando **operador ternario con sintaxis de objeto**.

```
<li v-for="tarea in lista" :key="tarea.id"
:class="{ 'tarea-completada': tarea.estado, 'tarea-pendiente': !tarea.estado }">
  {{ tarea.tarea }}
</li>
```

VueJS

v-for en combinación con v-if

- La documentacion oficial no recomienda usar v-if y v-for en el misma etiqueta.
- Si estuvieran en la misma etiqueta la condición de **v-if no tendrá acceso a las variables del ámbito de v-for y arroja error de consola.**

```
lista:[  
    {tarea: "Comprar más plantas", id:1, estado: false},  
    {tarea: "Ir a la ferretería", id:2, estado: false},  
    {tarea: "Limpiar la cocina", id: 3, estado: true},  
    {tarea: "Hacer bici", id:4, estado: true}  
]
```

VueJS

- Esto se puede solucionar moviendo **v-for** a una etiqueta **<template>** envolvente (lo cual también es más explícito):

```
<ul>
  <template v-for="item in lista" :key="item.id">
    <li v-if="!item.estado"> {{ item.tarea }}</li>
  </template>
</ul>
```

- Esto **solo mostrará las tareas que tengan estado en false**, es decir, que están todavía pendientes de realizar.

VueJS

Iterar estructuras complejas de datos

- Hay ciertas estructuras más complejas para representar la información.
- Podemos tener una lista de tareas que no solo incluyen el nombre de la **tarea** y su **estado**, sino también **detalles** adicionales en formato de Array o en formato Array de objetos.

```
listaDeTareas: [
  { tarea: "Ir al vivero",
    id: 1,
    estado: false,
    detalles: ["suculentas ", "colgantes ", "aromáticas "] },
  {tarea: "Lectura",
    id: 2,
    estado: false,
    detalles: ["Mr.Mercedes ", "Dr.Sueño ", "It"] },
],
]
```

VueJS

```
<li v-for="item in listaDeTareas" :key="item.id" >  
    {{item.tarea}} :  
    <span v-for="x in item.detalles">  
        Detalles: {{x}} -  
    </span>  
</li>
```

Repasemos :

- La directiva **v-for** se utiliza para **iterar sobre cada elemento en listaDeTareas**.
- La propiedad **:key="item.id"** se utiliza para proporcionar una **clave única** a cada elemento de la lista para mejorar la eficiencia de Vue en el **seguimiento de los cambios**.
- Se muestra el **nombre de la tarea** con la interpolación **{{ item.tarea }}**.
- Se usa **2º iteración v-for="x in item.detalles"** para recorrer los detalles de cada tarea.
- Cada detalle se muestra con el alias **{{ x }}**

VueJS

- En el caso de tener una **estructura de objetos dentro de un elemento de tipo array**, el proceso es muy similar.
- Debemos generar 2 iteraciones para poder mostrar la información de forma adecuada.

```
<li v-for="item in listaDeTareas" :key="item.id" >
    {{item.tarea}} :
    <span v-for="x in item.detalles">
        Detalles: {{x.nombre}} - {{item.cantidad}}- {{item.tamanio}}
    </span>
</li>
```

- Nuestro **alias para el iterador debe ir seguido de cada propiedad del objeto** que deseamos visualizar.



Dudas?

ADM

Dudas?

The slide contains a collection of 18 cards, each representing a different aspect of the ADM (Architectural Design Methodology) framework. The cards are arranged in a grid-like pattern, some overlapping, and are set against a background of a long wooden pier extending into a body of water under a cloudy sky.

- Card 1:** Descripción General. A general overview of the ADM process.
- Card 2:** Fase de Planificación. The planning phase of the methodology.
- Card 3:** Fase de Diseño. The design phase, involving requirements analysis and system architecture.
- Card 4:** Fase de Implementación. The implementation phase, focusing on system development.
- Card 5:** Fase de Pruebas. The testing phase, ensuring system quality and functionality.
- Card 6:** Fase de Despliegue. Deployment, the final stage where the system is put into production.
- Card 7:** Fase de Mantenimiento. Continuous maintenance and evolution of the system.
- Card 8:** Fase de Evaluación. Evaluation of the system's performance and user satisfaction.
- Card 9:** Fase de Desarrollo. Development phase, often confused with the design phase.
- Card 10:** Fase de Implementación. Another instance of the implementation phase.
- Card 11:** Fase de Pruebas. Another instance of the testing phase.
- Card 12:** Fase de Despliegue. Another instance of the deployment phase.
- Card 13:** Fase de Mantenimiento. Another instance of the maintenance phase.
- Card 14:** Fase de Evaluación. Another instance of the evaluation phase.
- Card 15:** Fase de Desarrollo. Another instance of the development phase.
- Card 16:** Fase de Implementación. Another instance of the implementation phase.
- Card 17:** Fase de Pruebas. Another instance of the testing phase.
- Card 18:** Fase de Despliegue. Another instance of the deployment phase.



Prezi