



Escuela Politécnica Superior de Alicante

Diseño de aplicaciones web

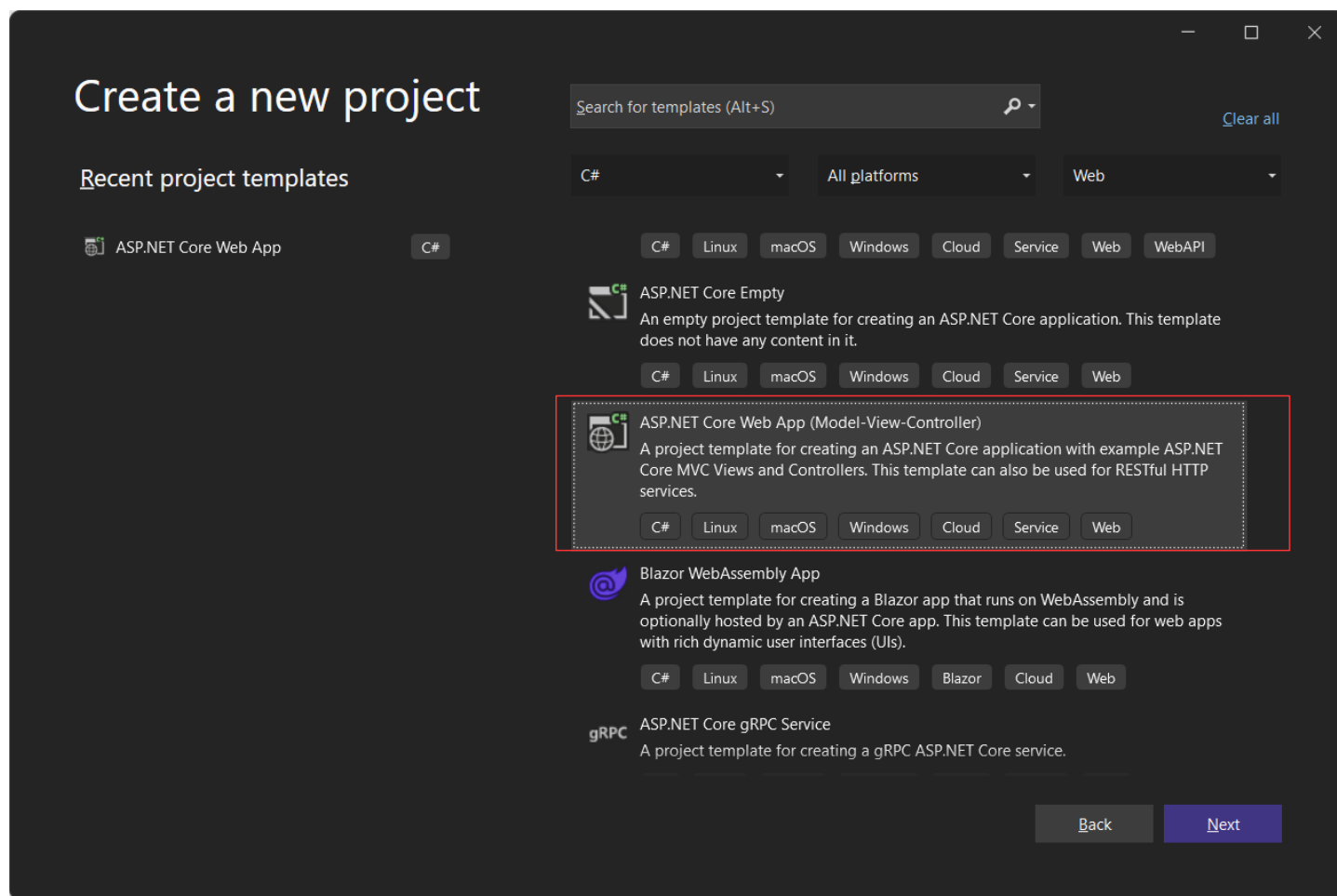
Taller de Vue.js

Contenidos

- Preparando el proyecto
- 'Hola mundo' con vue.js
- App sencilla: gestión de listas
 - Listas: CRUD
 - Tareas: CRUD
 - Componentes, filtros, computed
- Ejercicios
- Referencias y ampliar información

Preparando el proyecto

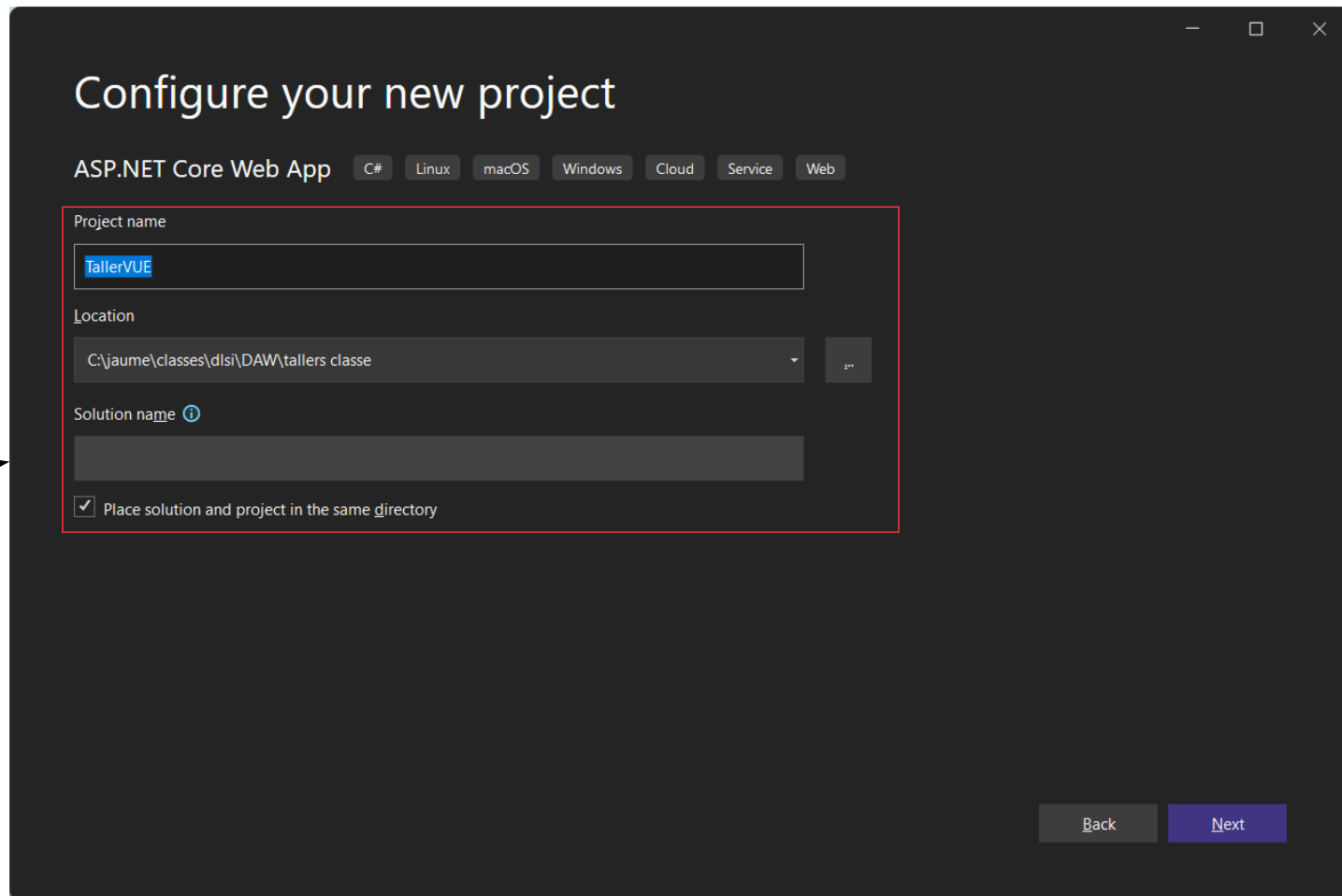
- Crear proyecto:
 - Proyecto ASP.NET CORE Web Application **MVC**



Preparando el proyecto

- Crear proyecto:
 - Nombre: TallerVUE

Seleccionar
nombre y
ubicación



Configure your new project

ASP.NET Core Web App C# Linux macOS Windows Cloud Service Web

Project name

TallerVUE

Location

C:\jaume\classes\dlsi\DAW\tallers classe

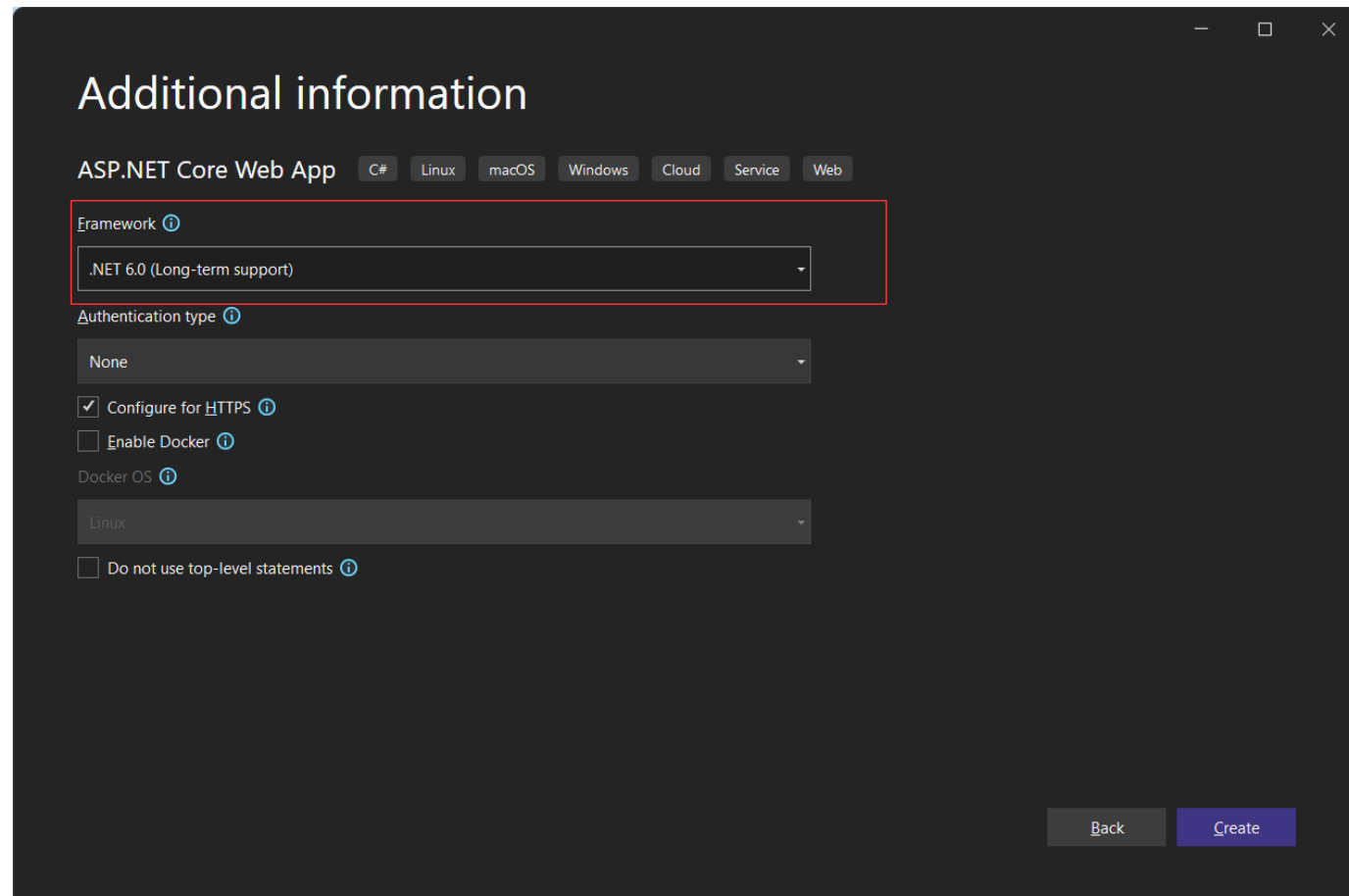
Solution name ⓘ

☒ Place solution and project in the same directory

Back Next

Preparando el proyecto

- Crear proyecto:
 - Framework: 6.0 LTS



Additional information

ASP.NET Core Web App C# Linux macOS Windows Cloud Service Web

Framework ⓘ

.NET 6.0 (Long-term support)

Authentication type ⓘ

None

☒ Configure for HTTPS ⓘ

☐ Enable Docker ⓘ

Docker OS ⓘ

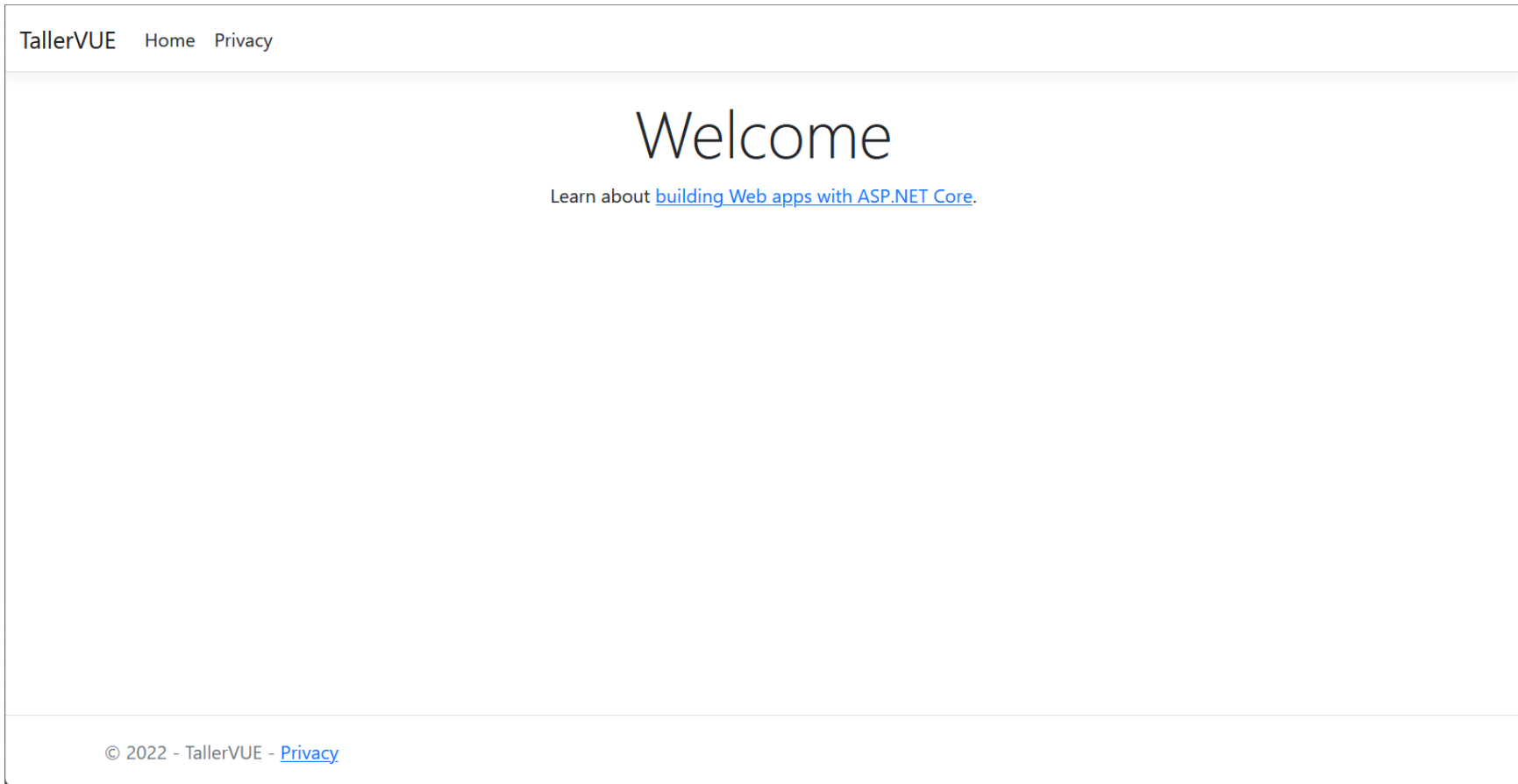
Linux

☐ Do not use top-level statements ⓘ

Back Create

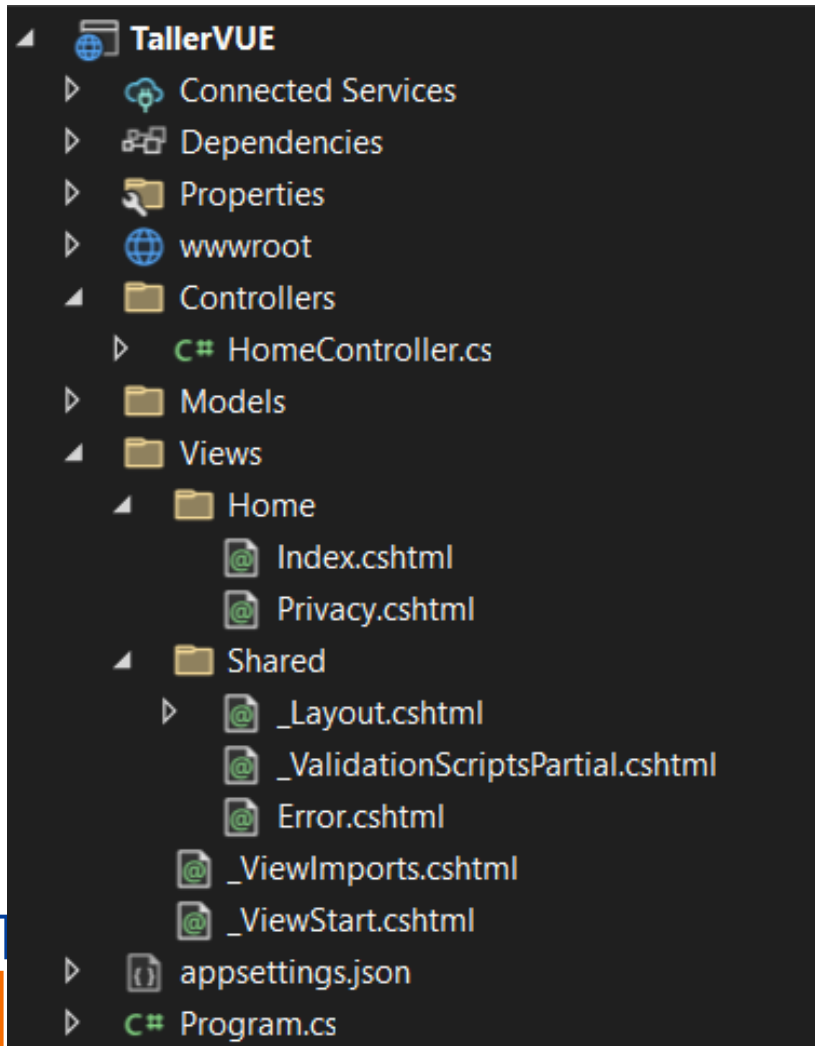
Preparando el proyecto

- Aspecto del proyecto tras 'ejecutarlo' con 'ctrl+F5'.



Preparando el proyecto

- Contenido del proyecto:



- wwwroot: contenido estático: CSS, JS, etc.
- Controllers: controladores
- Models: modelos
- Views: vistas
- Views/Shares: plantilla de vistas del proyecto
- Appsettings.json: configuración del proyecto
- Program.cs: punto de inicio de ejecución, permite incluir, inyectar y declarar componentes y variables al inicio.

Preparando el proyecto

- Vamos a crear un fichero de recursos lingüísticos para internacionalizar el proyecto.
- Crearemos una nueva carpeta: Resources.
- Despues, dentro de la anterior carpeta, crearemos un fichero de recursos que llamaremos 'Textos.resx'.
- Y le añadiremos tres recursos:
 - appName: Mis Listas
 - subtítulo: Gestión de listas de tareas
 - título: Mis Listas
 - menuHome: Inicio
 - menuPrivacy: Privacidad

Finalmente, cambiaremos en el fichero de recursos su acceso para que sea 'público'.

Preparando el proyecto

- Vamos a preparar el layout de la página, para ello modificaremos el fichero Views/Shared/_Layout.cshtml:

```
<div class="container-fluid">
  <a class="navbar-brand" asp-area="" asp-controller="Home" asp-action="Index">@TallerVUE.Resources.Textos.appName</a>
  <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target=".navbar-collapse" aria-control
    aria-expanded="false" aria-label="Toggle navigation">
    <span class="navbar-toggler-icon"></span>
  </button>
  <div class="navbar-collapse collapse d-sm-inline-flex justify-content-between">
    <ul class="navbar-nav flex-grow-1">
      <li class="nav-item">
        <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">
          @TallerVUE.Resources.Textos.menuHome</a>
        </li>
      <li class="nav-item">
        <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Privacy">
          @TallerVUE.Resources.Textos.menuPrivacy</a>
        </li>
      </ul>
    </div>
  </div>
</div>
```

```
<footer class="border-top footer text-muted">
  <div class="container">
    &copy; @DateTime.Now.Year - @TallerVUE.Resources.Textos.appName -
    <a asp-area="" asp-controller="Home" asp-action="Privacy">@TallerVUE.Resources.Textos.menuPrivacy</a>
  </div>
</footer>
```

Preparando el proyecto

- La vista 'Index.cshtml', la modificaremos de esta forma:

```
@{  
    ViewData["Title"] = TallerVUE.Resources.Textos.titulo;  
}  
<h1>@TallerVUE.Resources.Textos.titulo</h1>  
<h2>@TallerVUE.Resources.Textos.subtitulo</h2>
```

Preparando el proyecto

- Instalar Vue.js
 - <https://vuejs.org/v2/guide/installation.html>
- Desde el gestor nuGet:

The screenshot shows the NuGet Package Manager interface in Visual Studio. The left pane displays search results for 'vue.js'. The right pane shows the details for the selected 'vue' package, including a table of installed versions and options to install or uninstall.

NuGet - Solution | Privacy.cshtml | Index.cshtml | HomeController.cs | TallerVUE: Overview

Manage Packages for Solution

Package source: **nuget.org**

Search Results:

- vue.js** by Evan You, **21,7K** downloads | 1.0.22
- Vue.js.Developers.Version** by Evan You, **50,5K** downloads | 2.4.2
NuGet package for the developer's version of Vue.js 2.4.2 which comes with full warnings and debug mode.
- vue** by Evan You, **784K** downloads | 2.6.11
The Progressive JavaScript Framework
- Vue.js.Element.UI** by <http://element.eleme.io/>, **19,4K** downloads | 2.10.0.1
Element, a Vue 2.0 based component library for developers, designers and product managers <http://element.eleme.io/>
- Vue.js.IView.UI** by <https://www.iviewui.com/>, **8K** downloads | 2.14.3.5
A high quality UI Toolkit built on Vue.js. <https://www.talkingdata.com/>
- Vue.js.Vuetify.UI** by Hiro, **2,83K** downloads | 2.1.6
Material Component Framework for Vue.js 2 <https://vuetifyjs.com>

Package Details: vue

versions - 1

	Project	Version	Installed
<input checked="" type="checkbox"/>	Project		
<input checked="" type="checkbox"/>	TallerVUE	2.6.11	2.6.11

Installed: 2.6.11 **Uninstall**

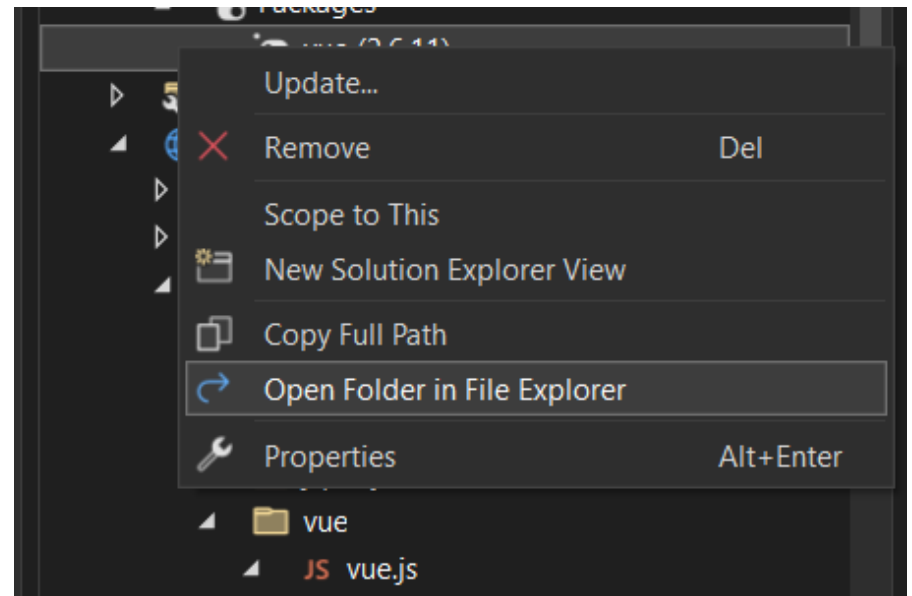
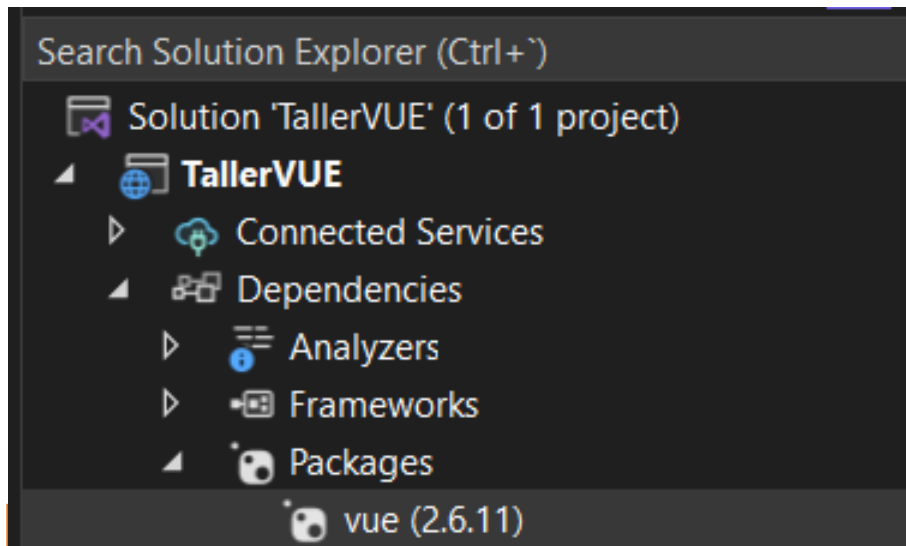
Version: Latest stable 2.6.11 **Install**

Options

Description

Preparando el proyecto

- Tenemos que incluir los ficheros de Vue.js en nuestro proyecto.
- Dentro de 'wwwroot/lib/' crearemos una carpeta 'vue'.
- Accederemos a la carpeta donde esté instalado el paquete 'vue.js'.
- Y copiaremos los dos ficheros de 'content/scripts' en la anterior carpeta 'wwwroot/lib/vue' de nuestro proyecto.



Hola Mundo

- Añadimos el script de vue.js en la vista 'index.cshtml' a partir de la sección 'javascript':

```
@{  
    ViewData["Title"] = TallerVUE.Resources.Textos.titulo;  
}  
  
<h1>@TallerVUE.Resources.Textos.titulo</h1>  
<h2>@TallerVUE.Resources.Textos.subtitulo</h2>  
  
@section Scripts {  
    <script src="~/lib/vue/vue.js"></script>  
}
```

Hola Mundo

- Ahora crearemos una nueva carpeta llamada 'vue' dentro de 'wwwroot' y en ella crearemos un fichero de javascript llamado 'index.vue.js'.
- Su contenido:

```
var app = new Vue({  
  el: '#app',  
  data: {  
    mensaje: 'Hola Mundo!'  
  }  
});
```

Hola Mundo

- Y vamos a modificar 'index.cshtml':
 - Añadiremos el nuevo script 'index.vue.js'.
 - Una capa con id='app' donde se ejecutará el código de vue.js.

```
@{
    ViewData["Title"] = TallerVUE.Resources.Textos.titulo;
}
<h1>@TallerVUE.Resources.Textos.titulo</h1>
<h2>@TallerVUE.Resources.Textos.subtitulo</h2>
<div id="app">
    {{mensaje}}
</div>
@section Scripts {
    <script src="~/lib/vue/vue.js"></script>
    <script src="~/vue/index.vue.js"></script>
}
```


I18n con Vue.js

- Antes de continuar, vamos a instalar el módulo de internacionalización para Vue.js:
 - <http://kazupon.github.io/vue-i18n/installation.html>
- Nos descargamos 'vue-i18n.js' y lo ubicamos en el proyecto, en la carpeta 'wwwroot/lib/vue'.
- A continuación, crearemos un nuevo fichero de tipo javascript dentro de 'wwwroot/vue', que llamaremos 'textos.js'.

I18n con Vue.js

- El contenido del fichero 'textos.js':

```
const textos = {  
  es: {  
    btnBack: 'Volver al listado',  
    add: {  
      btnAdd: 'Añadir lista',  
      title: 'Añadir nueva lista',  
      lblName: 'Nombre: ',  
      ttpName: 'Nombre de la lista',  
      btnAddList: 'Añadir'  
    },  
    list: {  
      title: 'Listado de mis listas de tareas',  
      noLists: 'No tienes listas todavía!'  
    }  
  }  
};
```

I18n con Vue.js

- En el fichero 'index.vue.js' tendremos que añadir el objeto i18n y su inicialización:

```
const i18n = new VueI18n({  
  locale: 'es', // set locale  
  messages: textos  
});  
  
var app = new Vue({  
  i18n,  
  el: '#app',  
  ...  
});
```

I18n con Vue.js

- En la vista 'index.cshtml', añadiremos los includes necesarios:

```
@section javascript {  
    <script src="~/lib/vue/vue.js"></script>  
    <script src="~/lib/vue/vue-i18n.js"></script>  
    <script src="~/vue/textos.js"></script>  
    <script src="~/vue/index.vue.js"></script>  
}
```

Listado y Añadir nueva

- En la vista 'index.cshtml', vamos a incluir
 - un botón que nos servirá para acceder al formulario de añadir nueva lista con una etiqueta localizada usando la función `$t('...')`.
 - Un fieldset con un div de tipo 'card' para cada lista que encontremos (dentro de un v-for)
 - En caso de no tener listas, mostraremos un div de tipo 'alert' con un mensaje adecuado.
 - Añadiremos otro fieldset con el formulario de nueva lista (de momento, solo el campo nombre) y dos botones:
 - para realizar la inserción de la nueva lista
 - y para volver al listado.

Listado y Añadir nueva

- Código de la vista 'index.cshtml':

```
<button v-if="mode!='add'" v-on:click="addMode()"
class="btn btn-primary">{{ $t('add.btnAdd') }}</button>
<div class="container" v-if="mode=='list'">
  <fieldset>
    <legend>{{ $t('list.title') }}</legend>
    <div class="card" v-for="(list, index) in lists"
v-bind:key="list.id">
      <div class="card-body">{{ list.name }}</div>
    </div>
    <div v-if="lists.length==0" class="alert alert-
warning">{{ $t('list.noLists') }}</div>
  </fieldset>
</div>
```

Listado y Añadir nueva

- Continuación del código de la vista 'index.cshtml':

```
<div class="container" v-if="mode=='add'">
  <fieldset><legend>{{t('add.title')}}</legend>
  <form>
    <div class="form-group">
      <label for="addListName">{{t('add.lblName')}}</label>
      <input v-model="newList.name" type="text" class="form-control" id="addListName" v-bind:placeholder="t('add.ttpName')">
    </div> </form>
    <div class="btn-group">
      <button v-on:click="addList()" class="btn btn-primary">{{t('add.btnAddList')}}</button>
      <button v-on:click="listMode()" class="btn btn-danger">{{t('btnBack')}}</button>
    </div> </fieldset>
</div>
```

Listado y Añadir nueva

- En el fichero 'index.vue.js':

```
var app = new Vue({  
  i18n,  
  el: '#app',  
  data: {  
    mode: 'list',  
    lists: [],  
    newList: null  
  },  
  methods: {  
    init() {  
      this.mode = 'list';  
      this.lists = [];  
    },
```


Listado y Añadir nueva

- Continuación del fichero 'index.vue.js':

```
    addMode() {  
      this.newList = { name: '', color: '', visible: true,  
date: '', tasks: [] };  
      this.mode = 'add';  
    },  
    listMode() {  
      this.mode = 'list';  
    },  
    addList() {  
      this.lists.push(this.newList);  
      this.mode = 'list';  
    }  
  }  
});
```

Comprobar campo nombre obligatorio

- Añadiremos en la vista 'index.cshtml' una referencia al 'input text' y un mensaje de error:

```
<input ref="newList_name" v-model="newList.name"
type="text" class="form-control" id="addListName" v-
bind:placeholder="$t('add.ttpName')">
```

```
<span v-if="errList.name" class="small text-
danger">Debe escribir un nombre de la lista.</span>
```

- Añadimos un objeto para gestionar los errores en el formulario de añadir, que pondremos todo a falso al mostrar el formulario.
- En la función de añadir la lista, si el nombre está vacío, mostraremos el mensaje de error y enfocaremos al campo del nombre.

Comprobar campo nombre obligatorio

- En el fichero 'index.vue.js':

```
data: {  
  ...  
  errList: {name: false}  
},  
methods: {  
  ...  
  addMode() {  
    this.newList = { name: '', color: '', visible:  
true, date: '', tasks: [] };  
    this.mode = 'add';  
    this.errList.name = false;  
  },  
}
```

Comprobar campo nombre obligatorio

- En el fichero 'index.vue.js', funcion 'addList':

```
addList() {  
    if (this.newList.name !== "") {  
        this.lists.push(this.newList);  
        this.mode = 'list';  
    } else {  
        this.errList.name = true;  
        this.$refs.newList_name.focus();  
    }  
}
```

Eliminar lista

- Primero, vamos a añadir una librería con estilos de tipo iconos tipográficos.
- Usaremos Font Awesome (<https://fontawesome.com/start>)
- Añadiremos el estilo css en nuestra vista 'index.cshtml':

```
@section css {  
    <link rel="stylesheet"  
href="https://use.fontawesome.com/releases/v5.8.2/css/all.css" integrity="sha384-oS3vJWv+0UjzBfQzYUhtDYW+Pj2yciDJxpsK10YPAYjqT085Qq/1cq5FLXAZQ7Ay" crossorigin="anonymous">  
}
```

Eliminar lista

- Añadiremos un nuevo recurso en 'i18n.js':

```
delete: {  
  ttpBtn: 'Eliminar',  
}
```

- Ahora, modificaremos la caja de tipo 'card-body' para añadir un botón de borrar:

```
<div class="card-body">  
  <h5>{{list.name}}  
    <button v-on:click="deleteList(index)"  
class="btn btn-sm btn-danger" v-bind:title=  
"$t('delete.ttpBtn')"><i class="fas fa-trash"></i>  
    </button>  
  </h5>  
</div>
```

Eliminar lista

- En el fichero del código, añadimos la funcion 'deleteList':

```
deleteList(list) {  
  this.lists.splice(list, 1);  
}
```

Añadir más datos a las listas

- Vamos a ampliar la funcionalidad de nuestra web permitiendo gestionar, para una lista dada:
 - un color de fondo
 - una fecha de alta
 - hacerla visible o invisible

Color de la lista

- Para gestionar un color de fondo para cada lista:
- Añadimos un nuevo recurso de texto localizado:
 - Add.lblColor: "Color de la lista:"
- Luego modificaremos el formulario de añadir una lista:

```
<div class="form-group">  
  <label for="addListColor">  
    {{$t('add.lblColor')}}</label>  
  <input v-model="newList.color" type="color"  
    class="form-control" id="addListColor">  
</div>
```

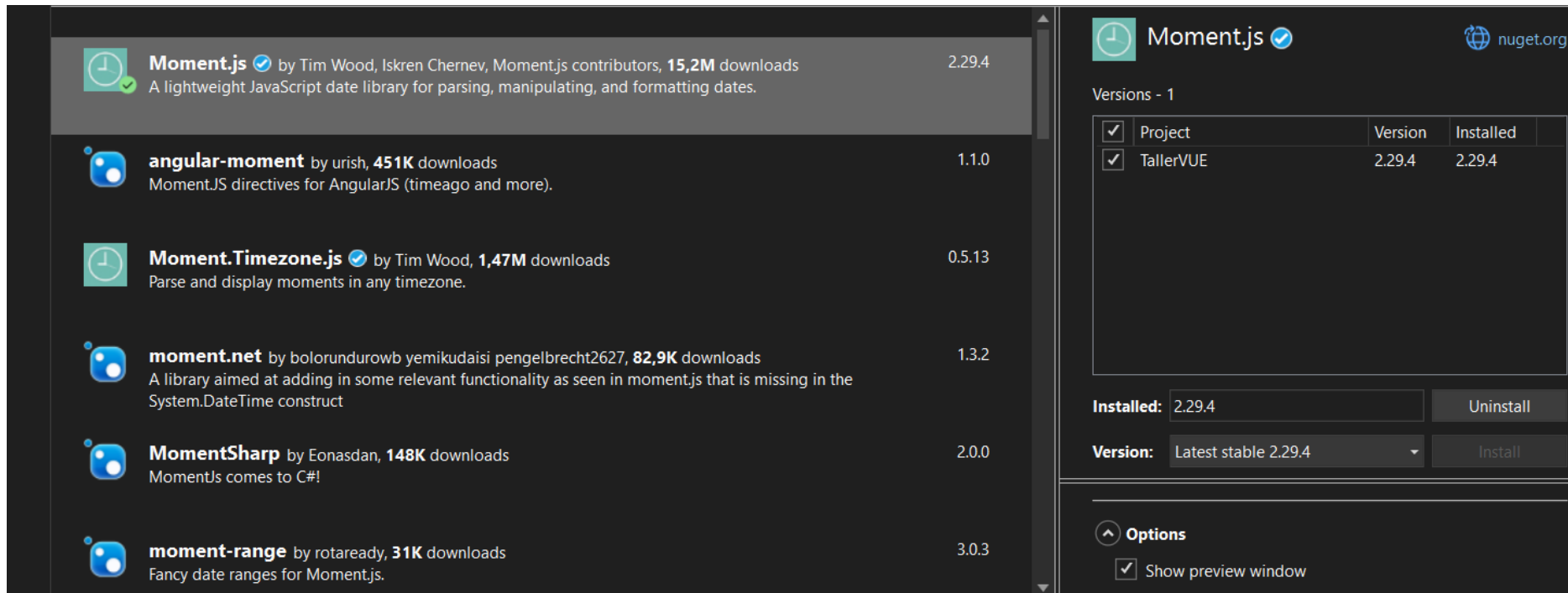
Color de la lista

- Y para terminar modificamos el HTML que muestra una lista, la caja de tipo 'card':

```
<div class="card" v-for="(list, index) in lists"  
v-bind:key="list.id"  
v-bind:style="{ 'background-color': list.color }">
```

Fecha de alta de la lista

- Ahora, gestionaremos la fecha de alta de una lista, para ello, instalaremos la librería moment.js en nuestro proyecto.



The screenshot displays the NuGet package manager interface. On the left, a list of packages is shown, with **Moment.js** selected. The package details for Moment.js are shown on the right, including its version (2.29.4) and a table of installed versions.

Moment.js by Tim Wood, Iskren Chernev, Moment.js contributors, **15,2M** downloads 2.29.4
A lightweight JavaScript date library for parsing, manipulating, and formatting dates.

angular-moment by urish, **451K** downloads 1.1.0
MomentJS directives for AngularJS (timeago and more).

Moment.Timezone.js by Tim Wood, **1,47M** downloads 0.5.13
Parse and display moments in any timezone.

moment.net by bolorundurowb yemikudaisi pengelbrecht2627, **82,9K** downloads 1.3.2
A library aimed at adding in some relevant functionality as seen in moment.js that is missing in the System.DateTime construct

MomentSharp by Eonasdan, **148K** downloads 2.0.0
MomentJs comes to C#!

moment-range by rotaready, **31K** downloads 3.0.3
Fancy date ranges for Moment.js.

Moment.js nuget.org

Versions - 1

	Project	Version	Installed
<input checked="" type="checkbox"/>	TallerVUE	2.29.4	2.29.4

Installed: 2.29.4 Uninstall

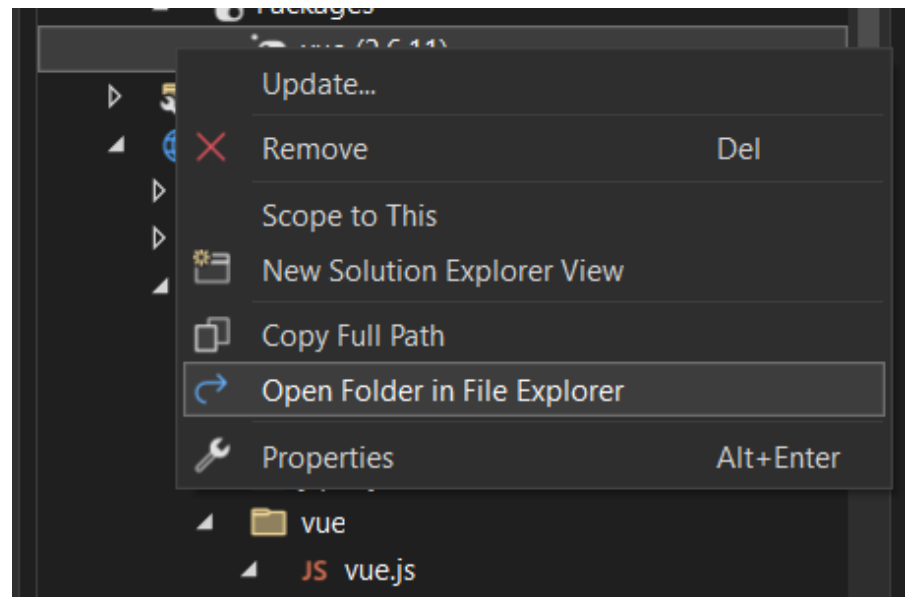
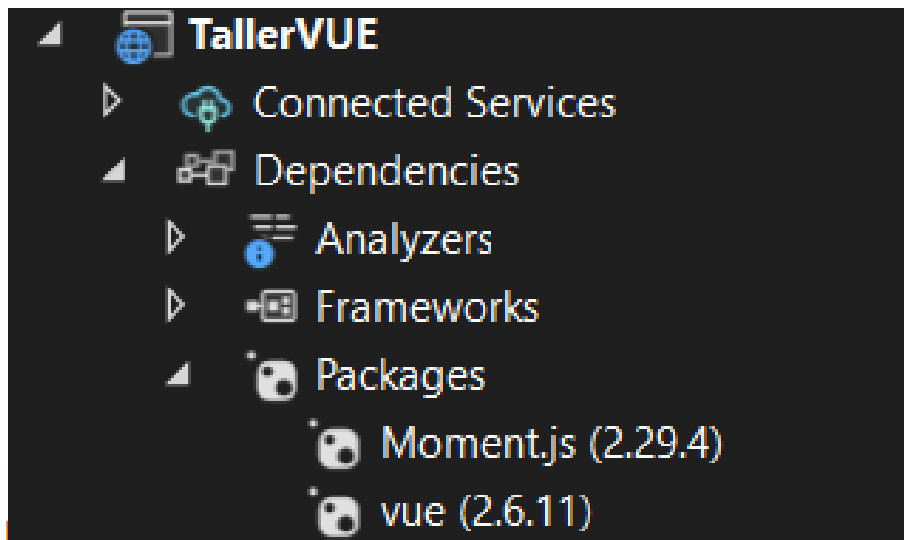
Version: Latest stable 2.29.4 Install

Options

☒ Show preview window

Fecha de alta de la lista

- Tenemos que incluir los ficheros de Moment.js en nuestro proyecto.
- Dentro de 'wwwroot/lib/' crearemos una carpeta 'moment'.
- Accederemos a la carpeta donde esté instalado el paquete 'moment.js'.
- Y copiaremos los ficheros de 'content/scripts' en la anterior carpeta 'wwwroot/lib/moment' de nuestro proyecto.



Fecha de alta de la lista

- Despues, en la vista 'index.cshtml', incluimos la nueva librería:

```
@section javascript {  
    <script src="~/lib/moment/moment.js"></script>  
    <script src="~/lib/vue/vue.js"></script>  
    <script src="~/lib/vue/vue-i18n.js"></script>  
    <script src="~/vue/textos.js"></script>  
    <script src="~/vue/index.vue.js"></script>  
}
```

Fecha de alta de la lista

- En el método de añadir una nueva lista, pondremos la siguiente línea de código adicional:

```
addList() {  
  if (this.newList.name !== "") {  
    this.newList.date = moment();  
    this.lists.push(this.newList);  
    this.mode = 'list';  
  } else {  
    this.errList.name = true;  
    this.$refs.newList_name.focus();  
  }  
},
```

Fecha de alta de la lista

- Añadiremos un filtro para formatear fechas y horas en la instancia:

```
data: {  
  ...  
},  
filters: {  
  formatDate: function (value) {  
    //TODO, verificar que hay algo en value!  
    return value.format("DD/MM/YYYY hh:mm:ss");  
  }  
},  
methods: { ...
```

Fecha de alta de la lista

- Y, finalmente, mostramos la fecha en el listado:

```
<div class="card-body">
  <h5>{{list.name}} <button ...>
</i...></button></h5>
  <small>
    {{list.date | formatDate}}
  </small>
</div>
```


¿Lista visible?

- En la vista, añadimos un botón en el listado vinculado al modelo para poder 'ocultar' la lista, de forma que ahora, sustituimos el 'h5', que quedará así:

```
<div class="listing">
  {{list.name}}
  <div class="btn-group">
    <button ...></i ...></button>
    ...
  </div>
</div>
```

- En el fichero css/site.css añadimos un nuevo estilo:

```
.listing {
  font-size: 1.25rem;
}
```

¿Lista visible?

- Añadiremos dos nuevos botones:

```
<button v-if="list.visible==true"
v-on:click="visibleList(index)"
class="btn btn-sm btn-info"
v-bind:title="$t('list.ttpNoVisible')">
  <i class="fas fa-eye-slash"></i>
</button>
```

```
<button v-if="list.visible==false"
v-on:click="visibleList(index)"
class="btn btn-sm btn-info"
v-bind:title="$t('list.ttpVisible')">
  <i class="fas fa-eye"></i>
</button>
```

¿Lista visible?

- Ahora implementamos el evento 'visibleList'

```
visibleList(index) {  
    this.lists[index].visible =  
(this.lists[index].visible ? false : true);  
}
```

- Añadiremos dos recursos de texto en 'list':
 - ttpNoVisible: 'Hacer no visible'
 - ttpVisible: 'Hacer visible'

¿Lista visible?

- Y modificaremos el listado para que solo muestre listas que están visibles.

```
<template v-for="(list, index) in lists">
  <div class="card" v-bind:key="list.id"
v-bind:style="{ 'background-color': list.color}"
v-if="list.visible">
    <div class="card-body">
      <h5>...</h5>
      <small>{{list.date | formatDate}}</small>
    </div>
  </div>
</template>
```

Patrones de listados

- Ahora vamos a añadir la lógica y controles necesarios para
 - poder ver las listas visibles, no visibles o todas.
 - ordenar las listas por nombre o fecha de creación.

Ver listas visibles/no visibles/todas

- Añadiremos tres recursos de texto en 'list':
 - optTodas: "Ver todas",
 - optVisibles: "Ver visibles",
 - optNOVisibles: "Ver NO visibles"
- Además, necesitaremos una variable nueva en la instancia:

```
data: {  
    ...  
    slctVisible: "  
    },
```

Ver listas visibles/no visibles/todas

- Modificamos la vista y añadimos un select encima del listado con tres opciones:
 - Ver todas (valor: '')
 - Ver visibles (valor: 'V')
 - Ver NO visibles (valor: 'N')

```
<template v-if="lists.length>0">
  <select class="form-control-sm" v-model="slctVisible">
    <option value="" selected>{{ $t('list.optTodas') }}</option>
    <option value="V">{{ $t('list.optVisibles') }}</option>
    <option value="N">{{ $t('list.optNOVisibles') }}</option>
  </select>
  <template v-for="(list, index) in lists">
    <div class="card" ... v-if="getVisibility(index)">
      <div class="card-body">...</div>
    </div>
  </template>
</template>
```

Ver listas visibles/no visibles/todas

- El método 'getVisibility':

```
getVisibility(index) {  
    var res = true;  
    switch (this.slctVisible) {  
        case "V":  
            res = this.lists[index].visible;  
            break;  
        case "N":  
            res = !this.lists[index].visible;  
            break;  
        default:  
            res = true;  
            break;  
    }  
    return res;  
}
```


Ordenar las listas:

- Para ordenar las listas, vamos a añadir otro desplegable con dos opciones:
 - Ordenar por nombre (value: 'N')
 - Ordenar por fecha (value: 'F')
- Añadimos también un nuevo dato a la instancia:
 - **slctOrden: 'N'**
- Y los recursos de texto pertinentes:
 - optOrdenNombre: "ordenar por nombre",
 - optOrdenFecha: "ordenar por fecha",

Ordenar las listas:

- El desplegable de ordenar, en la vista, será:

```
<select class="form-control-sm" v-model="slctOrden">
  <option value="N" selected>
    {{$t('list.optOrdenNombre')}}
  </option>
  <option value="F">
    {{$t('list.optOrdenFecha')}}
  </option>
</select>
```

- El bucle que recorre las listas lo modificamos, tal que así:

```
<template v-for="(list, index) in ordenar(lists)">
```

Ordenar las listas:

- Y, añadimos el método 'ordenar' en el código Vue:

```
ordenar(lists) {  
  var aux = [];  
  switch (this.slctOrden) {  
    case 'F':  
      aux = lists.slice().sort((a, b) => {  
        return ((a.date > b.date) ? 1 : -1);  
      });  
      break;  
    case 'N':  
    default:  
      aux = lists.slice().sort((a, b) => {  
        return ((a.name > b.name) ? 1 : -1);  
      });  
      break;  
  }  
  return aux;  
}
```

Ejercicios

- A partir del proyecto realizado durante el taller:
 - Permitir que sea el propio usuario al añadir una lista, quien aporte la fecha de creación, usad un DatePicker.
 - Añadir la posibilidad de modificar una lista mediante un botón de modificar:
 - que muestre un formulario similar al de añadir, con los datos de la lista a editar
 - que al confirmar la modificación vuelva al listado con los datos cambiados
 - Validar los datos antes de realizar el cambio y mostrar mensajes de error si hiciera falta.

Referencias y más información

- **Vue.js:**
 - <https://v2.vuejs.org/v2/guide/>
- **I18n:**
 - <http://kazupon.github.io/vue-i18n/started.html>