



**Escuela Politécnica Superior de
Alicante**

Diseño de aplicaciones web

Taller ASP.net CORE + Vue.js

Contenidos

- Crear proyecto Vue.js
- Crear proyecto ASP.net CORE MVC
- Integrar ambos proyectos
- Añadir el patrón de i18n
- Preparar/limpiar el código del proyecto
- Primera llamada al back-end
- Ejercicios
- Referencias y ampliar información

Crear proyecto Vue.js

- Dependencias:
 - Node.js versión >12 (recom. 16)
 - NPM versión > 6 (recom. 8)
 - Vue-cli versión > 4.5.15
 - Para instalar: Npm i -g @vue/cli
- Para crear el proyecto:
 - Vue create taller-fe (nombre del proyecto)
 - Seleccionar algún preset o bien 'Manually select features'

Crear proyecto Vue.js

- Creando el proyecto, opciones:

```
npm
Vue CLI v4.5.15
? Please pick a preset: Manually select features
? Check the features needed for your project: Choose Vue version, Babel, Linter
? Choose a version of Vue.js that you want to start the project with 2.x
? Pick a linter / formatter config: Basic
? Pick additional lint features: Lint on save
? Where do you prefer placing config for Babel, ESLint, etc.? In package.json
? Save this as a preset for future projects? (y/N) N
```

- Nos cambiamos a la carpeta del proyecto: cd taller-fe
- Ejecutamos con: npm run serve
- Accedemos en: http://localhost:8080

Crear proyecto Vue.js

- Con 'vue ui', podemos acceder a un entorno gráfico para crear y configurar nuestros proyectos.
 - Permite mayor detalle de configuración

The image shows two screenshots of the 'Administrador de Proyectos de Vue' (Vue Project Manager) interface. The left screenshot shows the initial state with a green header bar containing 'Proyectos', '+ Crear', and 'Importar'. Below it, a message says 'No hay proyectos' with a small icon. The right screenshot shows the same interface after interacting with the '+ Crear' button, which is now highlighted with a purple border. A callout arrow points to this button with the text '← Pinchamos sobre 'Crear''. The bottom right corner of the right screenshot contains a green button with the text '+ Crear un nuevo proyecto aqui'.

← Pinchamos sobre 'Crear'

Administrador de Proyectos de Vue

Proyectos + Crear Importar

No hay proyectos

Administrador de Proyectos de Vue

Proyectos + Crear Importar

D: jaume classes disi DAW tallers classe Taller VUE.js

old

+ Crear un nuevo proyecto aqui

Seleccionamos esta opción →

Crear proyecto Vue.js

- Creando un nuevo proyecto Vue.js:

Crear un proyecto nuevo

Detalles Presets Características Configuración

Directorio del proyecto
Taller-fe
D:/j...lers classe/Taller VUE.js/Taller-fe

Gestor de paquetes
npm

Opciones adicionales
Sobreescribir directorio si ya existe
Esqueleto del proyecto sin instrucciones de principiante

Repositorio git
Inicializar repositorio git (recomendado)
Sobrescribir mensaje de confirmación (opcional)

Preset por defecto [Vue 2] babel, eslint
 Default preset (Vue 3) [Vue 3] babel, eslint
 Manual Seleccionar manualmente las características
 Preset remoto Recupera un preset de un repositorio git

Seleccionamos 'preset' manual.

Crear proyecto Vue.js

- Creando un nuevo proyecto Vue.js, características:

Crear un proyecto nuevo

Detalles Presets Características Configuración

Choose Vue version
Choose a version of Vue.js that you want to start the project with

Babel
Transpile modern JavaScript to older versions (for compatibility) [Más información](#)

TypeScript
Add support for the TypeScript language [Más información](#)

Progressive Web App (PWA) Support
Improve performances with features like Web manifest and Service workers [Más información](#)

Router
Structure the app with dynamic pages [Más información](#)

Vuex
Manage the app state with a centralized store [Más información](#)

CSS Pre-processors
Add support for CSS pre-processors like Sass, Less or Stylus [Más información](#)

Linter / Formatter
Check and enforce code quality with ESLint or Prettier [Más información](#)

Unit Testing
Add a Unit Testing solution like Jest or Mocha [Más información](#)

E2E Testing
Add an End-to-End testing solution to the app like Cypress or Nightwatch [Más información](#)

Usar archivos de configuración
Usar archivos de configuración específicos (como '.babelrc') en vez de usar 'package.json'.



- Escoger versión
- Babel compatibility
- Typescript
- PWA
- Router
- Vuex
- Linter
- Usar package.json

Crear proyecto Vue.js

- Creando un nuevo proyecto Vue.js, pasos finales:

Choose a version of Vue.js that you want to start the project with

2.x (Por defecto) ▾

Use class-style component syntax?

Use the `@Component` decorator on classes. [Más información](#)



Use Babel alongside TypeScript (required for modern mode, auto-detected polyfills, transpiling JSX)?

It will output ES2015 and delegate the rest to Babel for auto polyfill based on browser targets.



Use history mode for router? (Requires proper server setup for index fallback in production)

By using the HTML5 History API, the URLs don't need the '#' character anymore. [Más información](#)



Pick a linter / formatter config:

Checking code errors and enforcing an homogeneous code style is recommended.

ESLint with error prevention only ▾

Pick additional lint features:

Lint on save

Lint and fix on commit

Versión Vue.js 2.0, sintaxis de componentes 'class-style', ESLINT solo con prevención de errores y al guardar.

Crear proyecto Vue.js

- Panel del proyecto donde gestionar complementos, dependencias, configuración y scripts.

The screenshot shows the 'Panel del proyecto' (Project Panel) interface. At the top left is a dropdown menu 'taller-fe'. On the left sidebar, there are several options: 'Panel de control' (selected), 'Complementos', 'Dependencias', 'Configuración', and 'Tareas'. The main area has a green header 'Consejos bienvenidos' with a large 'V' logo. Below it, a message says 'Bienvenido a tu nuevo proyecto'. There are three callout boxes: one about extensions, one about dependencies, and one about the home button. To the right, a 'Matar puerto' (Kill port) section is shown with a 'Preparado para matar' (Ready to kill) status, an input field 'Introduce puerto de red' (Enter port number), and a green 'Matar' (Kill) button.

taller-fe ▾

Panel del proyecto

Consejos bienvenidos

Bienvenido a tu nuevo proyecto

Estás mirando el tablero del proyecto donde puedes añadir herramientas. Utiliza el botón 'Personalizar' para añadir más! Todo será guardado automáticamente.

A la izquierda están las diferentes páginas disponibles. 'Extensiones' te permite añadir nuevas Vue Cli extensiones, 'Dependencias' para administrar los paquetes, 'Configuración' para configurar las herramientas y 'Tareas' para lanzar scripts (por ejemplo webpack).

Volver al administrador de proyecto con el dropdown arriba a la izquierda de la pantalla o el botón home en la barra de estado en el fondo.

Matar puerto

Preparado para matar

Introduce puerto de red

Matar

✓ Entendido

Crear proyecto Vue.js

- Ejecutar el proyecto: tareas > serve > Ejecutar tarea

The screenshot shows the Vuetify CLI interface with the following details:

- Tareas del proyecto (Tasks)** screen.
- serve** task is listed as "En ejecución" (Running), highlighted with a red box.
- Ejecutar tarea** (Run task) button is located next to the task list, also highlighted with a red box.
- Abrir aplicación** (Open application) button is located in the top right, also highlighted with a red box.
- Panel de control** section shows:
 - Estado: Con éxito
 - Errores: 0
 - Advertencias: 1
 - Recursos: 2.2MB (Analizado)
 - Módulos: 764.2kB (Analizado)
 - Dependencias: 726.2kB 95.03%
- Estadísticas de velocidad** section shows performance metrics for various network conditions.
- Vue UI** section displays a green checkmark icon and the message "App ready The build succeeded."

- Acceder al proyecto: <http://localhost:8080> o el botón 'Abrir aplicación'

Crear proyecto Vue.js

- Aspecto del nuevo proyecto creado:

The screenshot shows the initial landing page of a Vue.js application. At the top, there are two green links: "Home" and "About". Below them is the large, stylized green and dark blue "V" logo of the Vue.js framework. The main heading is "Welcome to Your Vue.js + TypeScript App". A descriptive text below it says: "For a guide and recipes on how to configure / customize this project, check out the [vue-cli documentation](#)". Underneath, there's a section titled "Installed CLI Plugins" with links for "pwa", "router", "vuex", "eslint", and "typescript". Further down is an "Essential Links" section with links for "Core Docs", "Forum", "Community Chat", "Twitter", and "News". At the bottom, there's an "Ecosystem" section with links for "vue-router", "vuex", "vue-devtools", "vue-loader", and "awesome-vue".

[Home](#) | [About](#)

Welcome to Your Vue.js + TypeScript App

For a guide and recipes on how to configure / customize this project, check out the [vue-cli documentation](#).

Installed CLI Plugins

[pwa](#) [router](#) [vuex](#) [eslint](#) [typescript](#)

Essential Links

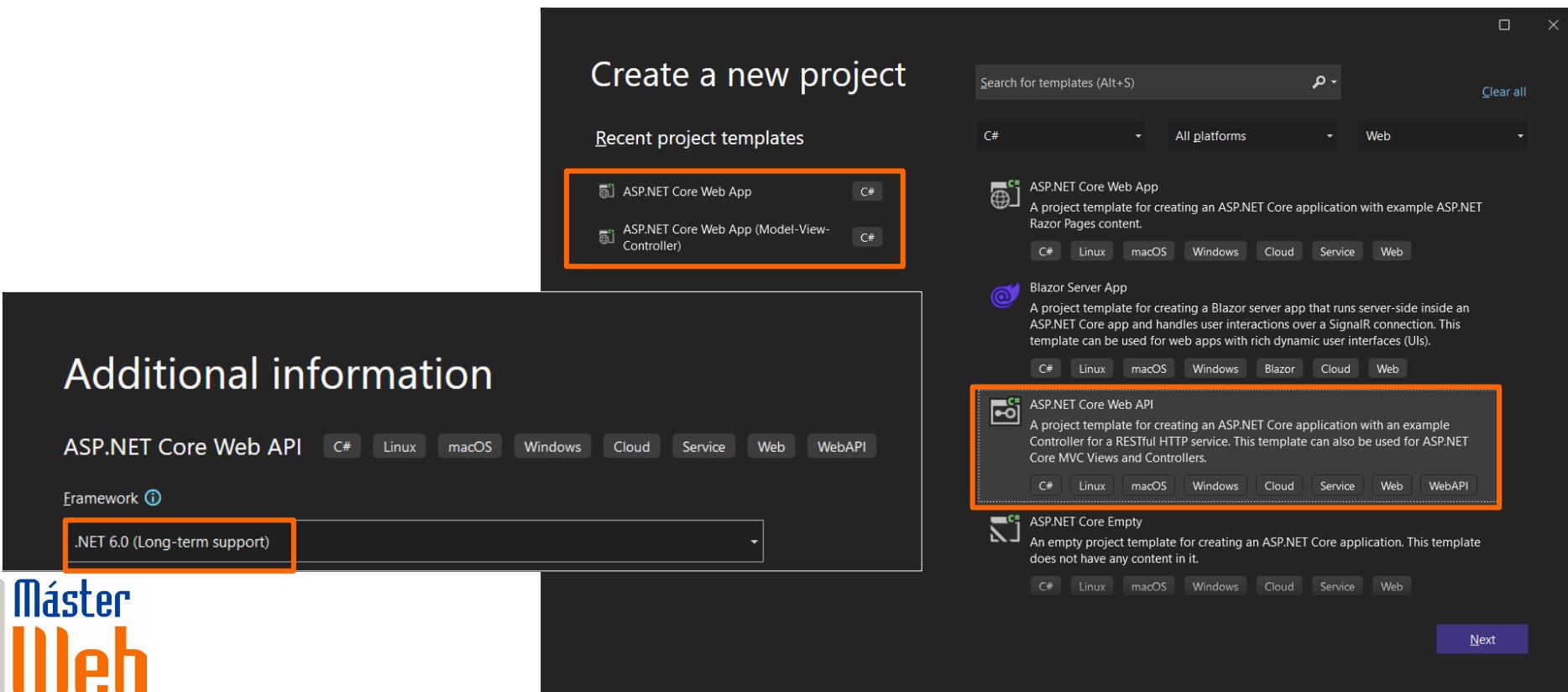
[Core Docs](#) [Forum](#) [Community Chat](#) [Twitter](#) [News](#)

Ecosystem

[vue-router](#) [vuex](#) [vue-devtools](#) [vue-loader](#) [awesome-vue](#)

Crear proyecto ASP.net CORE MVC

- Nuevo proyecto ASP.net CORE: Lo llamamos 'Taller-be' (de back-end)
- Ubicarlo en una carpeta al mismo nivel que el proyecto 'taller-fe' de Vue.js
- Versión 6.0
- Plantilla Aplicación web (MVC) o API, ambas valen.



Crear proyecto ASP.net CORE MVC

- Al iniciar lo podremos ver la pantalla principal del proyecto tal y como lo ha construido el visual studio con la plantilla escogida:

Taller_be Home Privacy

Welcome

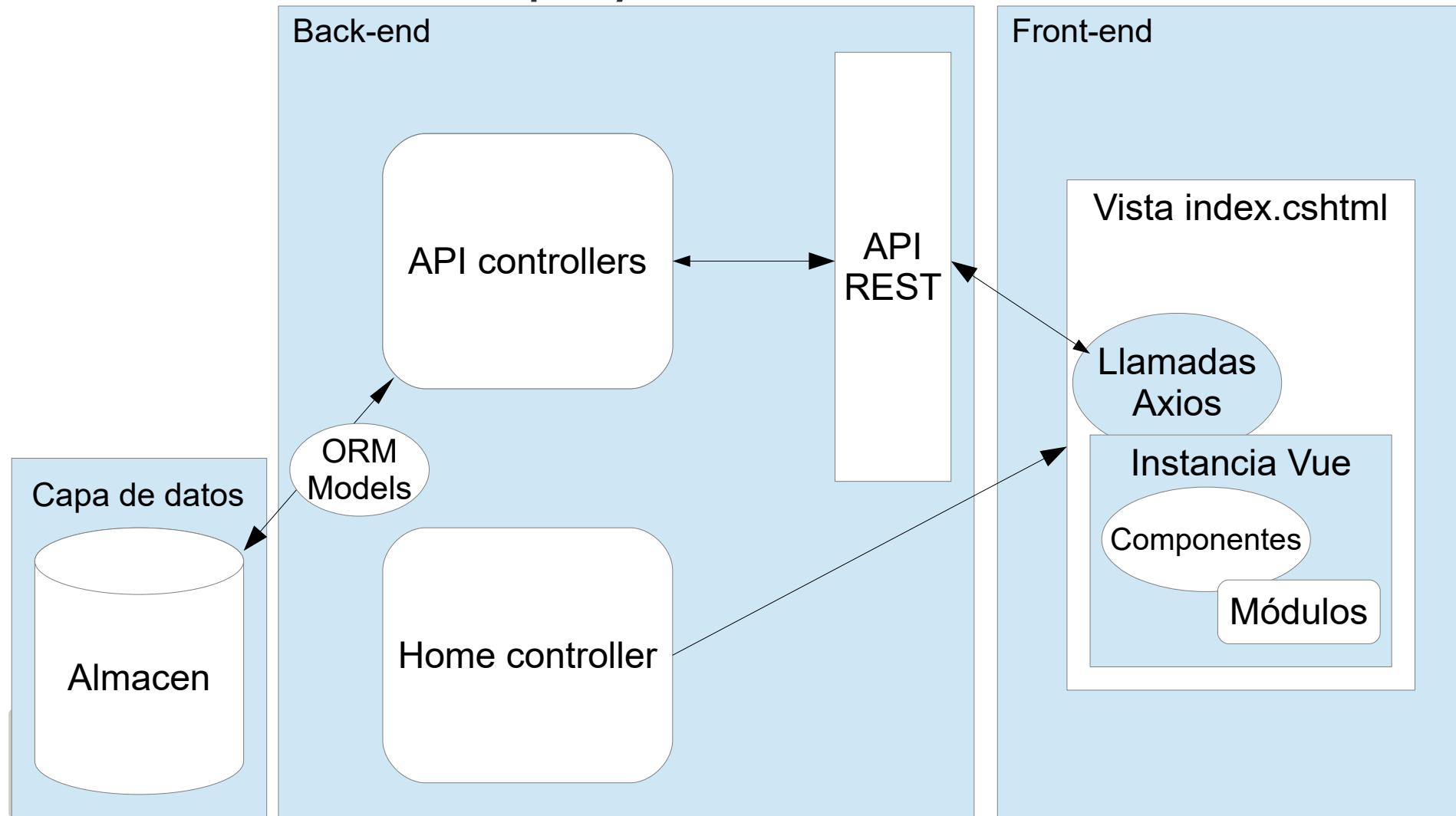
Learn about [building Web apps with ASP.NET Core.](#)

Integrar front-end con back-end

- Llegados a este punto tenemos dos proyectos diferentes:
 - Taller-fe: proyecto del front-end de nuestro sitio web con Vue.js 'empaquetado' o 'compilado' con vue-cli (web-pack).
 - Taller-be: proyecto con el back-end con ASP.net CORE MVC con un controlador (HomeController) y una acción/vista (home/index.cshtml)
- El siguiente paso es integrar ambos proyectos para que trabajen en conjunto y Vue.js sea el front-end de ASP.net Core.

Integrar front-end con back-end

- Estructura del proyecto:



Integrar front-end con back-end

- Cambiamos la configuración del proyecto para que genere los ficheros del front-end dentro el proyecto de back-end (en la carpeta 'wwwroot').

taller-fe ▾

Panel de control

Complementos

Dependencies

Configuración

Tareas

Configuración del proyecto

Vue CLI Configurar tú proyecto

ESLint configuration Verificación de errores & C...

PWA Aplicación Web Progresiva

Configuraciones generales

Ruta pública

La URL base de tu aplicación se desplegará en, por ejemplo '/my-app/'. Utiliza una cadena vacía ('') para que todos los 'assets' se vinculen usando rutas relativas. [Más información](#)

Directorio de salida

El directorio donde se generarán los archivos de compilación de producción. [Más información](#)

Directorio de assets

Un directorio para anidar activos estáticos generados (js, css, img, fuentes). [Más información](#)

Habilitar el compilador en tiempo de ejecución

Esto le permitirá usar la opción de plantilla en los componentes de Vue, pero incurrirá en una carga útil adicional de 10kb para su aplicación. [Más información](#)

Habilitar Mapas de origen en producción

Deshabilitar esto puede acelerar las compilaciones de producción si no necesita mapas de origen en producción [Más información](#)

Compilación en paralelo

Utilizar múltiples procesadores para compilar Babel o Typescript. [Más información](#)

[Más información](#)

Cancelar Guardar cambios

- Accedemos a configuración de Vue CLI
- Ruta pública: en blanco (vincular a rutas relativas)
- Directorio de salida: .../taller-be/wwwroot
- Y guardar cambios.

OFICIAL

Integrar front-end con back-end

- Tras configurar el proyecto Vue, se habrá generado un nuevo fichero 'vue.config.js' con el siguiente contenido:

```
JS vue.config.js U X

JS vue.config.js > [?] <unknown>
1  module.exports = {
2    publicPath: '',
3    outputDir: '../taller-be/wwwroot'
4 }
```

Integrar front-end con back-end

- Crearemos un fichero '.bat' que mueva el fichero index.html (principal del proyecto Vue) a la carpeta 'Views/Home' con el nombre index.cshtml.
- Lo llamaremos: moverindex.bat
- Y lo ubicaremos en la carpeta 'taller-fe'.

```
moverindex.bat
```

```
moverindex.bat
```

```
copy /Y ..\taller-be\wwwroot\index.html ..\taller-be\Views\Home\index.cshtml
```

```
del ..\taller-be\wwwroot\index.html
```

```
3
```

Integrar front-end con back-end

- Ahora vamos a añadir y modificar los scripts del proyecto Vue, ubicados en 'package.json'.
- Inicialmente tenemos tres scripts:
 - Serve: para desarrollar e iniciar el servicio web del proyecto en desarrollo
 - Build: para generar una versión de producción
 - Lint: para depurar/revisar el código

```
package.json > [ ] browserslist
1  {
2    "name": "taller-fe",
3    "version": "0.1.0",
4    "private": true,
5    ▷ Debug
6    "scripts": {
7      "serve": "vue-cli-service serve",
8      "build": "vue-cli-service build",
9      "lint": "vue-cli-service lint"
10     },
11 }
```

Integrar front-end con back-end

- Crearemos los siguientes scripts nuevos:
 - **Build-dev**: genera el proyecto pero en modo desarrollo
 - **Build-watch**: genera el proyecto en modo desarrollo y queda en espera para recompilar si hay cambios
 - **Watch**: macro que genera el proyecto en desarrollo, invoca el '.bat' anterior (para mover el fichero 'index' a las vistas del 'back-end' y se pone en modo observación/desarrollo)
 - **Postbuild**: script que invoca el '.bat' que mueve el fichero 'index'.
 - 'Build' lo renombramos por '**Build-pro**'

Integrar front-end con back-end

- El fichero 'package.json' quedará así:

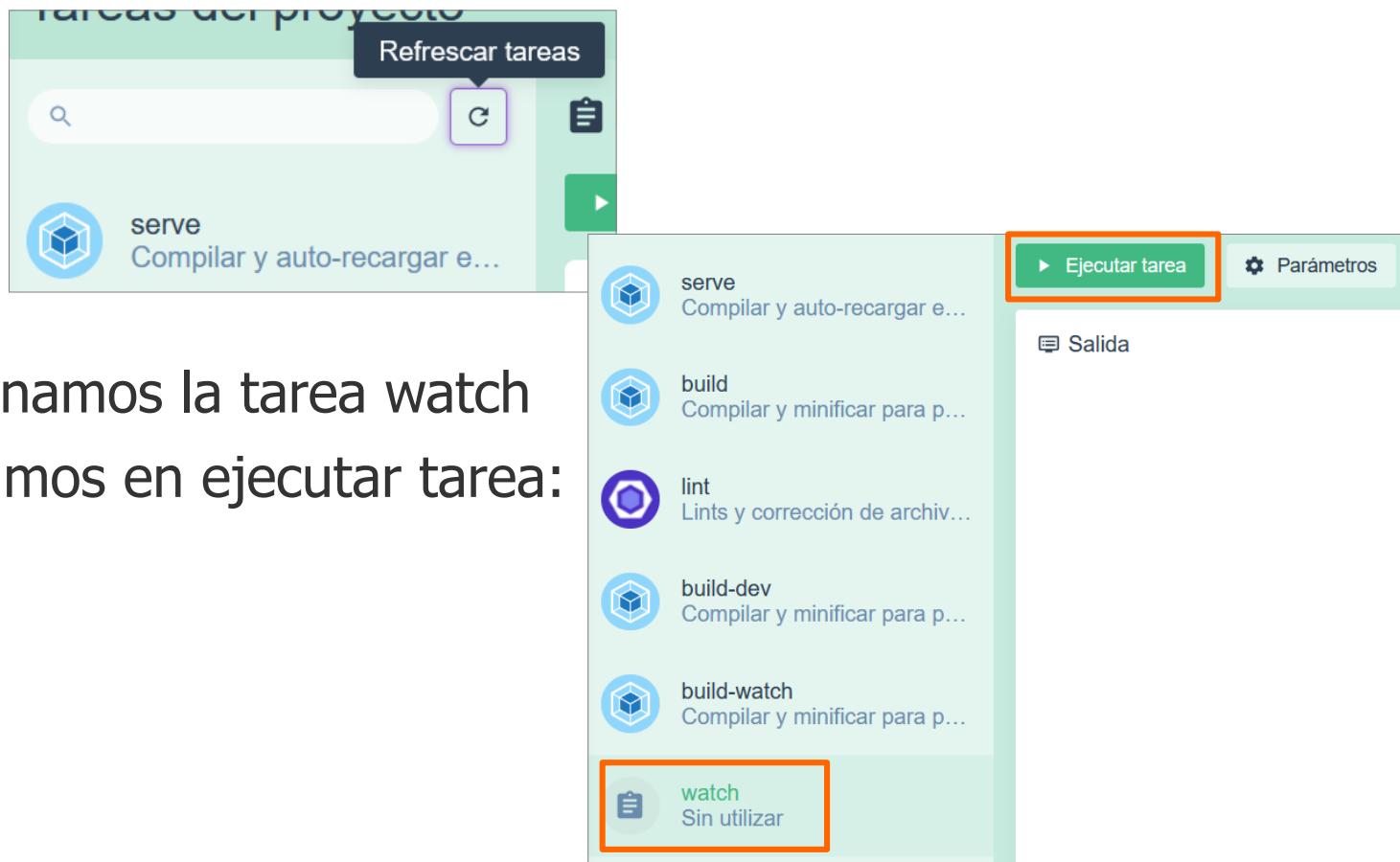
```
{ package.json > ...  
1 {  
2   "name": "taller-fe",  
3   "version": "0.1.0",  
4   "private": true,  
5   ▷ Debug  
6   "scripts": {  
7     "serve": "vue-cli-service serve",  
8     "build-pro": "vue-cli-service build",  
9     "lint": "vue-cli-service lint",  
10    "build-dev": "vue-cli-service build --mode development --no-clean",  
11    "build-watch": "vue-cli-service build --watch --mode development  
--no-clean",  
12    "watch": "npm run build-dev && npm run postbuild && npm run build-watch &  
& npm run postbuild",  
13    "postbuild": "moverindex.bat"  
14  },  
15}
```

Integrar front-end con back-end

- Ahora vamos a arrancar el entorno de desarrollo con los proyectos integrados.
- Primero haremos una copia de la carpeta 'wwwroot' del proyecto 'Taller-be' y tambien de la vista 'Views/Home/index.cshtml'
- Despues iniciaremos el proyecto de back-end sin depurar ('Depurar' → 'Iniciar sin depurar' o Ctrl+F5)
- Finalmente iniciaremos el script 'watch':
 - 'Npm run watch' desde una consola situada en la carpeta del proyecto de front-end.
- Y recargamos el proyecto de back-end para ver que aparece la página del proyecto inicial de Vue.js.

Integrar front-end con back-end

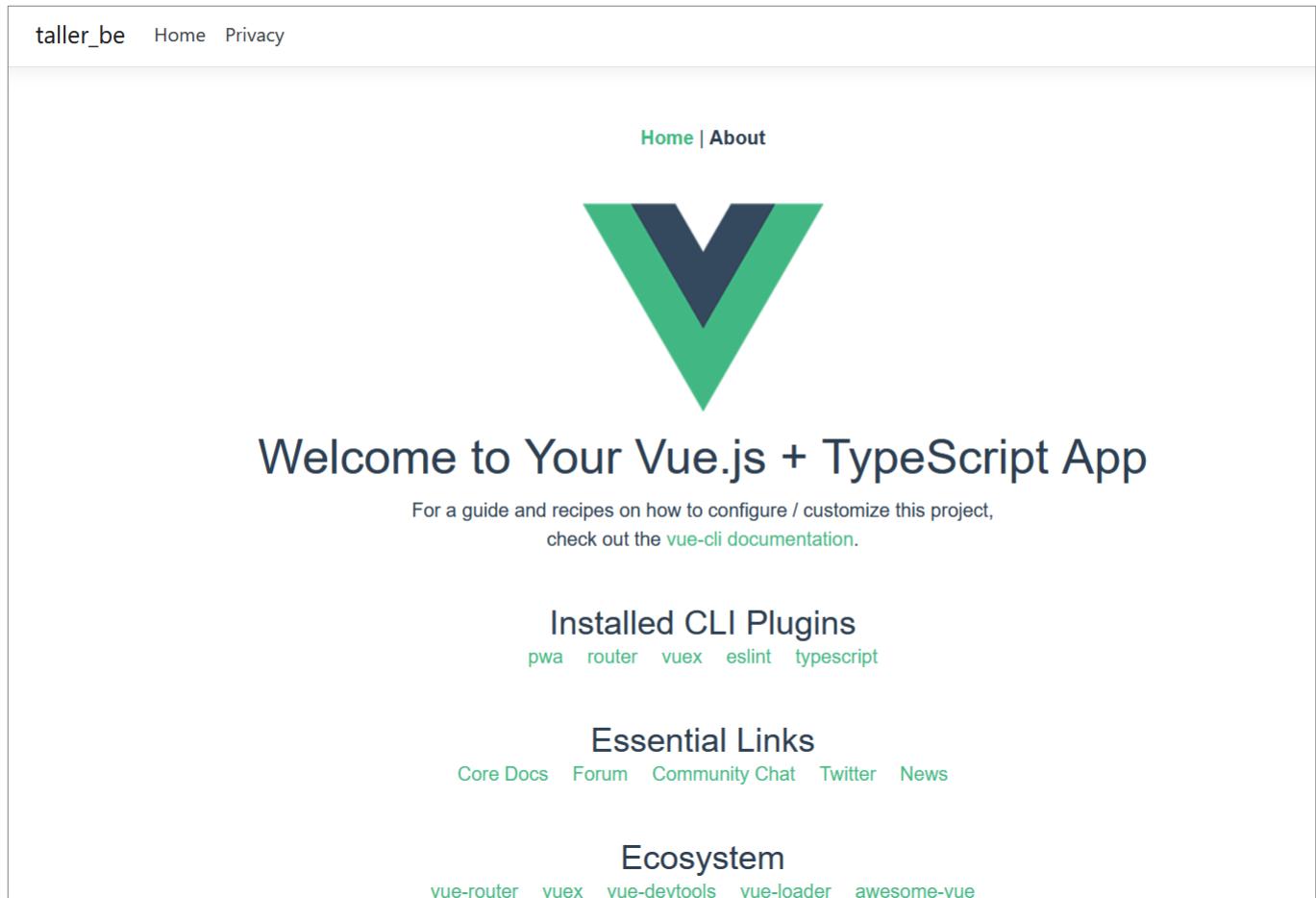
- Tambien podemos iniciar el script 'watch' desde el panel de control de 'vue-cli-service'.
- Accedemos a 'Tareas', actualizamos pinchando en 'refrescar tareas':



- Seleccionamos la tarea watch
y pinchamos en ejecutar tarea:

Integrar front-end con back-end

- Aspecto del proyecto integrado, pero sin adaptar el código:



Internacionalizar el proyecto

- Vamos a crear un fichero de recursos lingüísticos para internacionalizar el proyecto del back-end.
- Crearemos una nueva carpeta: Resources.
- Despues, dentro de la anterior carpeta, crearemos un fichero de recursos que llamaremos 'Textos.resx'.
- Y le añadiremos dos recursos:
 - appName: Mis Listas
 - titulo: Gestión de listas de tareas
- Añadiremos un segundo fichero de recursos que llamaremos Textos.en.resx con el contenido:
 - appName: My Task Lists
 - titulo: Manage my Task Lists
- Finalmente, cambiaremos en ambos ficheros de recursos su tipo de acceso para que sea 'público'.

Internacionalizar el proyecto

- Ahora internacionalizaremos el proyecto de front-end:
 - Instalaremos el módulo de internacionalización para Vue.js:
 - <http://kazupon.github.io/vue-i18n/installation.html>
 - Para instalarlo, desde una consola situada en la carpeta del proyecto 'taller-fe':
npm install vue-i18n
 - Una vez instalado, vamos a crear los ficheros de recursos e integrarlos en nuestro proyecto.
 - Primero crearemos en 'taller-fe/src' una nueva carpeta que llamaremos 'i18n' donde vamos a ubicar los ficheros necesarios.

Internacionalizar el proyecto

- Dentro de la carpeta 'taller-be/src/i18n':
 - Crearemos un fichero llamado 'langs.ts' con el siguiente contenido:

```
c > i18n > TS langs.ts > ...
1   export enum Idiomas {
2     ES = 'es',
3     EN = 'en',
4   }
5
6   export const IDIOMAS = [
7     { value: Idiomas.ES, caption: 'Español' },
8     { value: Idiomas.EN, caption: 'Inglés' }
9   ]
10
```

- Con este código vamos a declarar la estructura de datos necesaria para dar soporte a los idiomas del sitio web.

Internacionalizar el proyecto

- Dentro de la carpeta 'taller-be/src/i18n':
 - Crearemos dos ficheros con contenido Json, uno por cada idioma que soporte nuestro sitio web:
 - Taller-be/src/i18n/es.json

```
{  
  "app": {  
    "titulo": "Mis listas de tareas",  
    "inicio": "Inicio",  
    "acerca_de": "Acerca de"  
  }  
}
```

- Taller-be/src/i18n/en.json

```
{  
  "app": {  
    "titulo": "My task lists",  
    "inicio": "Home",  
    "acerca_de": "About"  
  }  
}
```

Internacionalizar el proyecto

- Dentro de la carpeta 'taller-be/src/i18n':
 - Despues añadiremos un fichero más llamado 'index.ts' que hara el papel de iniciador de nuestro módulo de i18n.
 - En el se importaran los textos de los idiomas declarados.

```
import { Idiomas } from "./langs";

import es from "./es.json";
import en from "./en.json";

export const messages = {
  [Idiomas.ES]: es,
  [Idiomas.EN]: en
};

export const defaultLang = Idiomas.EN;
```

Internacionalizar el proyecto

- En la transparencia anterior los imports de los ficheros '.json' estaban señalados como erróneos.
- Para corregir esto, tenemos que añadir la opción 'resolveJsonModule' en el fichero 'tsconfig.json', dentro de la sección 'compilerOptions':

```
{  
  "compilerOptions": {  
    "target": "es5",  
    "module": "esnext",  
    "strict": true,  
    "jsx": "preserve",  
    "importHelpers": true,  
    "moduleResolution": "node",  
    "resolveJsonModule": true,  
    "experimentalDecorators": true,  
    "skipLibCheck": true,  
    "esModuleInterop": true,  
    "allowSyntheticDefaultImports": true,  
    "sourceMap": true,  
    "baseUrl": ".",  
  },  
}
```

Internacionalizar el proyecto

- Por último, en el módulo principal de nuestro proyecto de front-end, en el fichero 'main.ts', declaramos un objeto tipo 'VueI18n' y lo configurarmos con los datos de los ficheros del módulo de i18n:

```
import store from './store'

import VueI18n from 'vue-i18n'

Vue.use(VueI18n)

import { messages, defaultLang } from "@/i18n";

const i18n = new VueI18n({
  messages,
  locale: defaultLang,
  fallbackLocale: defaultLang
});

Vue.config.productionTip = false

new Vue({
  router,
  store,
  i18n,
  render: h => h(App)
}).$mount('#app')
```

En este código:

- Importamos el módulo de i18n y lo habilitamos
- Importamos nuestros datos de idiomas y textos disponibles
- Creamos un objeto Vuel18n con la configuración importada.
- Incluimos el objeto 'i18n' en la instancia global de Vue de nuestro proyecto.

Adaptar el código del proyecto

- Vamos a unificar el código generado de forma automática por los entornos de desarrollo y adaptarlo para nuestro proyecto:
 - Primero nos ocuparemos del back-end y la plantilla compartida 'shared/_layout.cshtml'.
 - Recomendable **hacer una copia de este fichero!**
 - Despues arreglaremos el front-end modificando los ficheros inciales: 'public/index.html', 'src/main.ts' y 'src/App.vue' y otros.
 - Finalmente, añadiremos algunas prestaciones para completar el proyecto en su inicio.

Adaptar el código del proyecto

- Comenzaremos con el back-end, vamos a limpiar '_layout.cshtml' y enviar el título localizado a la vista:

```
<!DOCTYPE html>
<html lang="en">
  ...
    @RenderBody()
  ...
</html>
```

- Vista plantilla: 'shared/_Layout.cshtml'.
- Solo contendrá la etiqueta 'html' y el helper 'RenderBody()'.
- Todo el código se llevará al proyecto de front-end.

Controlador 'homecontroller.cs', en la acción 'Index', enviaremos el título localizado con el ViewData.

```
public IActionResult Index()
{
    ViewData["Title"] = Resources.Textos.appName;
    return View();
}
```

Adaptar el código del proyecto

- A continuación, vamos a ocuparnos del front-end.
- El fichero 'public/index.html' quedará:

```
<head>
  <meta charset="utf-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
  <link rel="stylesheet" href="~/css/site.css" />
  <link rel="icon" href="<%= BASE_URL %>/favicon.ico">

  <title>@ ViewData["Title"]</title>

</head>
<body>
  <noscript>
    <strong>We're sorry but <%= htmlWebpackPlugin.options.title %> doesn't work without JavaScript</strong>
  </noscript>
  <div id="app"></div>

  <script src="~/lib/jquery/dist/jquery.min.js"></script>
  <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
  <script src="~/js/site.js" asp-append-version="true"></script>

</body>
```

En este fichero integramos los meta de ambos proyectos en la cabecera, y añadimos los scripts del back-end.

En esta caja (div) es donde funcionará todo nuestro proyecto de front-end con Vue.

Adaptar el código del proyecto

- Ahora tenemos que incluir la cabecera, el menú, el cuerpo o contenido y el pie de página desde el front-end. El código lo reutilizaremos desde la vista '`_Layout.cshtml`' (o la copia que hicimos) del proyecto de back-end.
- Para ello vamos a crear dos componentes de Vue nuevos: `head` y `foot`.
- Todo esto lo haremos desde el componente principal del proyecto: `App.vue`.

Adaptar el código del proyecto

- Creamos el componente para el pie de la página:
 - Creamos un nuevo componente llamado 'Foot' que ubicaremos en 'src/components/layout/' cuyo contenido será:

```
<template>
  <footer class="border-top footer text-muted">
    <div class="container">
      &copy; 2022 -
      <router-link to="/about">
        {{$t('app.acerca_de')}} {{$t('app.titulo')}}
      </router-link>
    </div>
  </footer>
</template>

<script lang="ts">
import { Component, Vue } from 'vue-property-decorator';

@Component
export default class Foot extends Vue {

}
</script>
```

Adaptar el código del proyecto

- Incluimos el nuevo componente en el módulo 'App.vue':

```
<template>
  <div id="app">
    ...
    <router-view/>

    <Foot/>
  </div>
</template>

<script lang="ts">
import { Component, Vue } from 'vue-property-decorator';
import Foot from '@/components/layout/Foot.vue';

@Component({
  components: {
    Foot,
  },
})
export default class App extends Vue {}

</script>
```

En el template añadimos la etiqueta del componente 'Foot'.

Debemos añadir también una sección de script para importar e injectar el componente en el módulo principal del proyecto 'App.vue'.

Adaptar el código del proyecto

- Añadiremos también en el módulo principal 'App.vue' el código necesario para formatear el contenido de las vistas/páginas de nuestro proyecto, dentro de la sección template:

```
<template>
  <div id="app">
    <div id="nav">
      <router-link to="/">Home</router-link> |
      <router-link to="/about">About</router-link>
    </div>

    <div class="container">
      <main role="main" class="pb-3">
        <router-view/>
      </main>
    </div>

    <Foot/>
  </div>
</template>
```

Adaptar el código del proyecto

- Creamos el componente para la cabecera de la página, lo llamaremos 'Head' y lo ubicaremos en 'src/components/layout/'
- La sección 'template' tendrá la etiqueta 'header' de la vista '_Layout' del back-end:


```
<template>
  <header>
    <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white border-bottom box-shadow mb-3">
      <div class="container">
        <router-link to="/" class="navbar-brand">Taller FE</router-link>
        <button class="navbar-toggler" type="button" data-toggle="collapse" data-target=".navbar-collapse" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
          <span class="navbar-toggler-icon"></span>
        </button>
        <div class="navbar-collapse collapse d-sm-inline-flex flex-sm-row-reverse">
          <ul class="navbar-nav flex-grow-1">
            <li class="nav-item">
              <router-link to="/" class="nav-link text-dark">{{$t('app.inicio')}}</router-link>
            </li>
            <li class="nav-item">
              <router-link to="/about" class="nav-link text-dark">{{$t('app.acerca_de')}}</router-link>
            </li>
          </ul>
        </div>
      </div>
    </nav>
  </header>
</template>
```
- Cambiamos los enlaces (etiquetas 'a') por el elemento 'router-link' del enrutador de Vue.
- La sección del 'script' es similar al componente 'foot' anterior.

Adaptar el código del proyecto

- Ahora, incluiremos el componente 'Head' en 'App.vue':

```
<template>
  <div id="app">
    <Head />

    <div class="container">
      <main role="main" class="pb-3">
        <router-view/>
      </main>
    </div>

    <Foot/>
  </div>
</template>

<script lang="ts">
import { Component, Vue } from 'vue-property-decorator';
import Head from '@/components/layout/Head.vue';
import Foot from '@/components/layout/Foot.vue';

@Component({
  components: {
    Foot,
    Head
  },
})
```

- Primero importamos el componente referenciando al fichero que lo contiene y la clase principal.
- En el decorador `@Component`, incluimos la clase importada.
- Finalmente, en el template, añadimos la etiqueta para incluir el componente en el código a generar.

Adaptar el código del proyecto

- La vista 'Home' hace referencia a un componente llamado 'Helloworld'. Lo renombramos como 'Mislistas', cambiando:
 - El nombre del fichero del componente
 - El nombre de la clase
 - Las referencias al componente desde Home.vue.
- El código del componente 'MisListas', quedará:

```
<template>
  <div>
    | <h1>{{ msg }}</h1>
  </div>
</template>

<script lang="ts">
import { Component, Prop, Vue } from 'vue-property-decorator';

@Component
export default class Mislistas extends Vue {
  @Prop() private msg!: string;
}
</script>
```

- Limpiamos el template y dejamos solo el h1, por ahora.

Adaptar el código del proyecto

- El componente o vista 'Home.vue' lo cambiamos para que incluya el subcomponente renombrado 'Mislistas':

```
<template>
  <div class="home">
    | <Mislistas v-bind:msg="$t('app.titulo')"/>
    | </div>
</template>

<script lang="ts">
import { Component, Vue } from 'vue-property-decorator';
import Mislistas from '@/components/Mislistas.vue'; // @ is an al

@Component({
  components: {
    Mislistas,
  },
})
export default class Home extends Vue {}

</script>
```

- Como ya no necesitamos la imagen del logo de vue, podemos eliminarla de la carpeta 'assets' y de la carpeta 'wwwroot' del back-end.

Adaptar el código del proyecto

- En todo proyecto, a veces se necesita tener datos globales (accesibles desde cualquier parte del código)
- Para ello se suele añadir un fichero llamado 'env.json', el cual contiene datos generales del proyecto:
 - Versión, URL del back-end, rutas de carpetas especiales, nombres de ficheros, etc.
- En nuestro caso, por ahora, solo vamos a añadir la versión del proyecto como único dato de entorno: 'ver'.

```
{  
  "ver": "0.0.1"  
}
```

Adaptar el código del proyecto

- El fichero 'env.json' y sus datos vamos a usarlos en el componente 'foot' de la siguiente forma:

```
<template>
  <footer class="border-top footer text-muted">
    <div class="container">
      &copy; 2022 -
      <router-link to="/about">
        {{ $t('app.acerca_de') }} {{ $t('app.titulo') }}
      </router-link>
      <small>(Ver. {{ env.ver }})</small>
    </div>
  </footer>
</template>

<script lang="ts">
import { Component, Vue } from 'vue-property-decorator';

import env from "@/env.json";

@Component
export default class Foot extends Vue {
  env: any = env;
}
</script>
```

- Primero, importaremos el fichero 'env.json'
- Lo asignaremos a una variable del componente llamada 'env'
- Finalmente, en el template, mostraremos la versión del proyecto, accesible desde la propiedad 'ver'.

Adaptar el código del proyecto

- Como último detalle, podemos añadir un nuevo texto en los ficheros de i18n: app.appName y usarlo como texto del enlace de la cabecera, antes de las opciones del menú.

```
"app": {  
    "appName": "Mis listas",  
    "titulo": "Mis listas de tareas",  
    "inicio": "Inicio",  
    "acerca_de": "Acerca de"
```

```
<header>  
  <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white  
border-bottom box-shadow mb-3">  
    <div class="container">  
      <router-link to="/" class="navbar-brand">{{t('app.appName')}}</router-link>  
      <button class="navbar-toggler" type="button" data-toggle="collapse"
```

Inicio de la implementación

- Ahora ya tenemos el entorno de desarrollo preparado para comenzar a programar nuestro proyecto de una aplicación sencilla de gestión de listas de tareas.
- Debemos tener en cuenta que trabajaremos realmente con DOS proyectos de implementación:
 - Back-end a partir de un proyecto de ASP.net Core MVC
 - Front-end a partir de un proyecto de Vue.js con vue-CLI
- Además, para desarrollar, tendremos que arrancar ambos proyectos, el de back-end en modo depuración y el de front-end en modo 'watch'.
- Podeis descargar los proyectos tal cual están ahora desde el moodle.

Primera llamada al back-end

- Vamos a obtener la información (un título y un texto) de la página 'Acerca de' de nuestra aplicación web desde el 'back-end' realizando una llamada HTTP.
- En el back-end añadiremos un controlador APIController 'CMSController' y en él, implementaremos un 'end point' (una acción o método del controlador) que responda a la llamada 'API_getAbout' desde el front-end.
- Necesitaremos instalar y configurar la librería de comunicación HTTP Axios.
- También crearemos un servicio inyectable en nuestro proyecto de front-end para gestionar todas las llamadas hacia el back-end.

Primera llamada al back-end

- Primero de todo vamos a crear un objeto dentro de la carpeta 'models' que usaremos para devolver en toda llamada a nuestra APIREST. Lo llamaremos: APIResult.
- Su código:

```
namespace taller_be.Models
{
    public enum APIStatus { ok, err };
    public class APIResult
    {
        public APIStatus status { get; set; }
        public string msg { get; set; }
        public Object data { get; set; }

        public APIResult()
        {
            status = APIStatus.ok;
            msg = "";
            data = null;
        }
    }
}
```

Primera llamada al back-end

- A continuación, también dentro de la carpeta 'models', añadiremos un servicio que nos provea de la lógica de acceso a los contenidos del sitio web, lo llamaremos 'ServiceCMS'.
- Solo contendrá, por ahora, un método: getAbout que accederá a los recursos idiomáticos para obtener el título y el texto de la página de 'Acerca de' del sitio web. Su código:

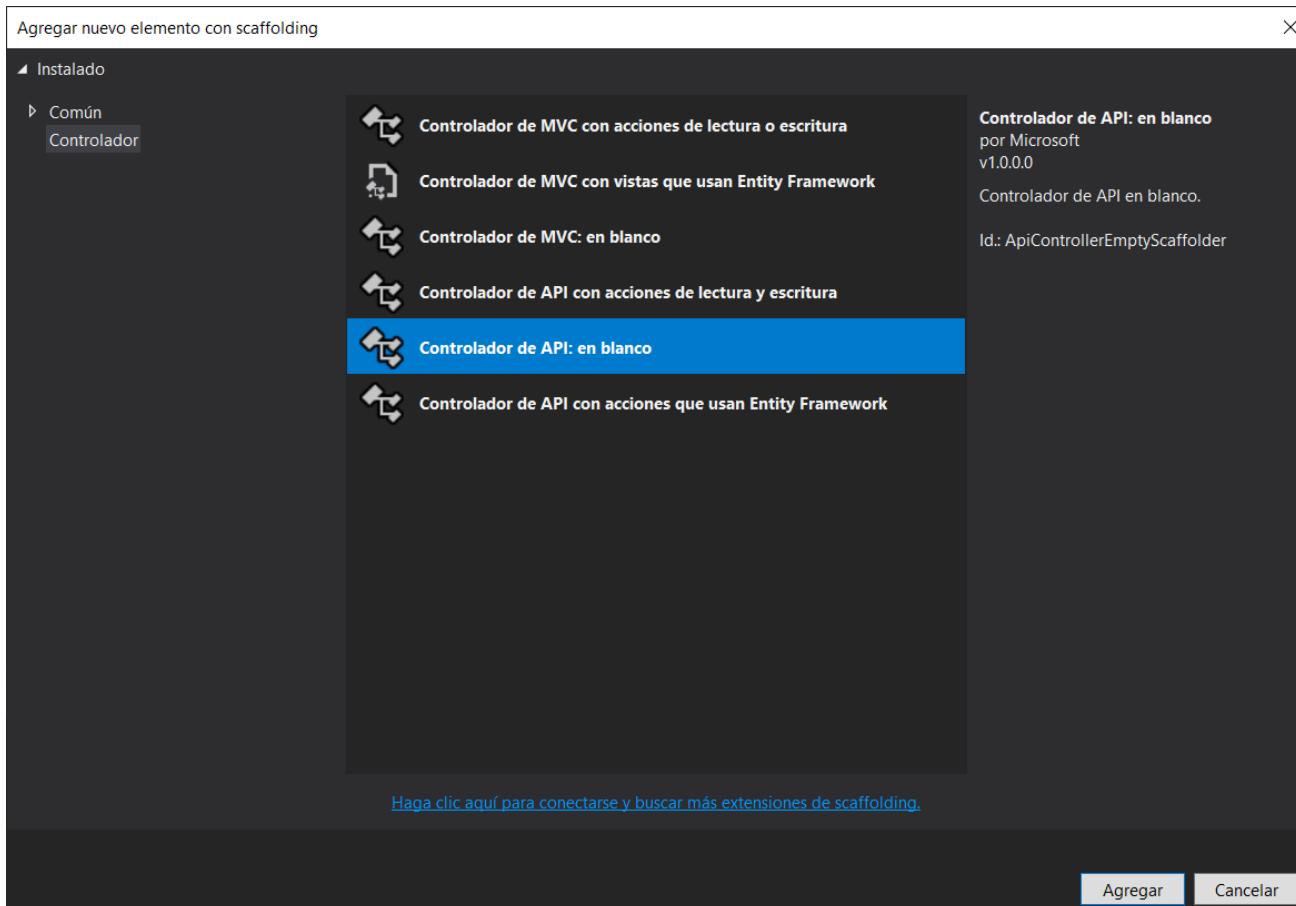
```
namespace taller_be.Models
{
    1 referencia
    public class ServiceCMS
    {
        1 referencia
        public static object getAbout()
        {
            Dictionary<string, string> about = ... new Dictionary<string, string>();

            about.Add("titulo", Resources.Textos.about_titulo);
            about.Add("texto", Resources.Textos.about_texto);

            return about;
        }
    }
}
```

Primera llamada al back-end

- Después crearemos un nuevo controlador llamado CMSController, de tipo 'Controlador API en blanco'.



Primera llamada al back-end

- El controlador CMSController, por ahora, solo tendrá una acción: 'API_getAbout', que responderá a la ruta: '/api/CMS/about' con llamadas tipo GET. Su código:

```
[Route("api/[controller]")]
[ApiController]
0 referencias
public class CMSController : ControllerBase
{
    [HttpGet]
    [Route("about")]
    0 referencias
    public APIResult API_getAbout()
    {
        APIResult res = new APIResult();
        try
        {
            res.status = APIStatus.ok;
            res.data = ServiceCMS.getAbout();
            res.msg = "";
        }
        catch (Exception ex)
        {
            res.status = APIStatus.err;
            res.data = null;
            res.msg = ex.Message;
        }
        return res;
    }
}
```

- Tras crear un objeto de respuesta APIResult, se invocará al servicio 'serviceCMS' para obtener los datos necesarios y devolverlos al Front-end.
- En caso de error, capturaremos la excepción y devolveremos el objeto resultado adaptado consecuentemente.

Primera llamada al back-end

- Axios
 - Cliente HTTP basado en 'promises' para tecnologías de front-end.
 - Web: **<https://github.com/axios/axios>**
 - Instalación: **npm install axios –save**
 - Uso: **import axios from 'axios';**
- Antes que nada, vamos a añadir un nuevo dato en el fichero de entorno 'env.json':

```
{  
  "ver": "0.0.1",  
  "url_api": "https://localhost:44316/api/"  
}
```

Primera llamada al back-end

- Ahora añadiremos un fichero con declaraciones 'src/http-config.ts' y configuraciones necesarias para las llamadas al back-end:
 - Importaremos 'axios' y el fichero 'env'.
 - Creamos un enum con los resultados posibles.
 - Creamos una clase APIResponse con tres datos: status, respuesta y error.
 - Declaramos el objeto 'axios' con la configuración necesaria:
 - baseURL
 - headers

```
import axios from "axios";
import env from "@/env.json";

export enum APIStatus {
    OK = '0',
    ERR = '1',
}

export class APIResponse {
    public respuesta: any;
    public status!: APIStatus;
    public error!: string;
}

export default axios.create({
    baseURL: env.url_api,
    headers: {
        "Content-type": "application/json"
    }
});
```

Primera llamada al back-end

- A continuación, crearemos el fichero 'service.ts' dentro de una nueva carpeta 'services'.
- Este fichero contendrá las llamadas al back-end según las necesitemos.
- En esta primera versión solo implementaremos la función 'get':

```
import http, { APIStatus, APIResponse } from "../http-config";

class APIService {

    async get(ruta:string, id: string="") {
        ...
    }
}

export default new APIService();
```

Primera llamada al back-end

- El código de la función 'get':

```
async get(ruta:string, id: string="") {  
    let dato="";  
    let resp = new APIResponse();  
    if (id!="") {  
        dato = "/" + id;  
    }  
    await http.get(  
        ruta + dato  
    ).then((respuesta) => {  
        resp.status=respuesta.data.status;  
        if (respuesta.data.status==APIStatus.OK) {  
            resp.respuesta=respuesta.data.data;  
        } else {  
            if (respuesta.data.msg!=null && respuesta.data.msg!="") {  
                resp.error=respuesta.data.msg;  
            }  
        }  
    }).catch((error) => {  
        resp.status=APIStatus.ERR;  
        resp.error = error.toString();  
    });  
    return resp;  
}
```

- Tras declarar el objeto de respuesta,
- Invocamos al método 'axios.get' contra la ruta y con un dato opcional.
- La cláusula 'then' se invocará si la respuesta lleva correctamente.
- En caso de error, se invocará el bloque 'catch'.

Primera llamada al back-end

- Por último, modificaremos el componente o vista 'about'.
- La vista o 'template' cambiará para usar variables locales al controlador y añadiremos una caja de error nueva:

```
<template>
  <div class="about">
    <h1>{{titulo}}</h1>
    <p v-html="texto"></p>
    <div class="alert alert-danger" v-if="errorMsg!=''">
      Error al obtener los datos del CMS, causa: {{errorMsg}}
    </div>
  </div>

</template>
```

Primera llamada al back-end

- El código del controlador, la sección 'script' de la página de 'about', será así:

```
<script lang="ts">

import { Component, Vue } from 'vue-property-decorator';
import serviceAPI from '@/services/service'; // @ is an alias to /src
import { APIStatus } from '@/http-config';

@Component
export default class About extends Vue {
    titulo = this.$i18n.t('acerca_de.titulo');
    texto = ""; //this.$i18n.t('acerca_de.tex
    errorMsg = "";

    mounted() {console.log("mounted");
        ...
    }
}
</script>
```

- Inyectamos el servicio 'serviceAPI'
- Declaramos las dos variables de la vista: título y texto y una variable para contener el error.
- Además añadiremos la función 'mounted', la cual se ejecuta automáticamente cuando el componente está listo y cargado en memoria.

Primera llamada al back-end

- El código de la función 'mounted':

```
mounted() {console.log("mounted");
  serviceAPI.get("CMS/about").then(r=> {
    if (r.status==APIStatus.OK) {
      if (r != null && r.respuesta!=null) {
        this.titulo = r.respuesta["titulo"];
        this.texto = r.respuesta["texto"];
      } else {
        this.errorMsg="No se han recibido datos";
      }
    } else {
      this.errorMsg=r.error;
    }
  }).catch(e=> {
    this.errorMsg=e;
  })
}
```

- Invocamos la función 'get' del servicio 'serviceAPI'.
- En caso de error, mostraremos el mensaje adecuado.

Ejercicios

- A partir del proyecto realizado durante el taller:
 - Añade una nueva página al front-end: 'Otros enlaces', que muestre una lista de enlaces de interés (titulo y URL).
 - La añadiremos también al menú para poder invocarla.
 - La lista de enlaces de interés los obtendremos desde el servidor mediante una llamada 'Axios' a un end-point el cual nos devolverá en Json un array de objetos 'link' (titulo y URL, ambos strings). Usad un mockup de datos.
 - Si no se obtienen enlaces del back-end, mostrar un mensaje informativo.
 - Capturar los posibles errores y mostrar un mensaje adecuado.

Referencias y más información

- Vue: <https://vuejs.org/>
- Vue-CLI:
 - <https://cli.vuejs.org/guide/>
 - <https://vuejs.org/v2/guide/typescript.html>
- Vue i18n:
 - <https://kazupon.github.io/vue-i18n/>
 - <https://medium.com/js-dojo/manage-vue-i18n-with-typescript-958b2f69846f>
- Componentes y plugins
 - <https://github.com/axios/axios>
 - <https://www.bezkoder.com/vue-typescript-crud/>