Use la plantilla de proyecto angular con ASP.NET Core

La plantilla de proyecto Angular actualizada proporciona un punto de partida conveniente para las aplicaciones ASP.NET Core que utilizan Angular y la CLI Angular para implementar una interfaz de usuario (UI) enriquecida del lado del cliente.

La plantilla es equivalente a crear un proyecto ASP.NET Core para que actúe como un back-end API y un proyecto CLI angular para que actúe como una IU. La plantilla ofrece la conveniencia de alojar ambos tipos de proyectos en un solo proyecto de aplicación. En consecuencia, el proyecto de la aplicación se puede construir y publicar como una sola unidad.

Crea una nueva aplicación

Si tiene instalado ASP.NET Core 2.1, no es necesario instalar la plantilla de proyecto Angular.

Cree un nuevo proyecto desde un símbolo del sistema utilizando el comando `dotnet new angular`en un directorio vacío. Por ejemplo, los siguientes comandos crean la aplicación en un directorio *my-new-app* y cambian a ese directorio:

CLI de .NET CoreCopiar

```
dotnet new angular -o my-new-app
cd my-new-app
```

Ejecute la aplicación desde Visual Studio o .NET Core CLI:

- Estudio visual
- CLI de .NET Core

  Asegúrese de tener una variable de entorno llamada `ASPNETCORE_Environment`con un valor de `Development`. En Windows (en solicitudes que no sean de PowerShell), ejecute `SET ASPNETCORE_Environment=Development`. En Linux o macOS, ejecute `export ASPNETCORE_Environment=Development`.

  Ejecute dotnet build para verificar que la aplicación se compila correctamente. En la primera ejecución, el proceso de construcción restaura las dependencias npm, lo que puede llevar varios minutos. Las construcciones posteriores son mucho más rápidas.

Ejecute dotnet run para iniciar la aplicación. Se registra un mensaje similar al siguiente:

ConsolaCopiar
```
Now listening on: http://localhost:<port>
```

Navegue a esta URL en un navegador.

Advertencia

La aplicación inicia una instancia del servidor CLI angular en segundo plano. Se registra un mensaje similar al siguiente: *NG Live Development Server está escuchando en localhost: <otherport>, abra un navegador en http: // localhost: <otherport> /* . No haga caso de este mensaje, que es **no** la URL de la aplicación combinada ASP.NET Core y angular de la CLI.

La plantilla del proyecto crea una aplicación ASP.NET Core y una aplicación angular. La aplicación ASP.NET Core está diseñada para usarse para acceso a datos, autorización y otras inquietudes del lado del servidor. La aplicación Angular, que reside en el subdirectorio *ClientApp* , está destinada a ser utilizada para todas las preocupaciones de UI.

Agregue páginas, imágenes, estilos, módulos, etc.

El directorio *ClientApp* contiene una aplicación CLI angular angular estándar. Consulte la documentación oficial de Angular para obtener más información.

Existen ligeras diferencias entre la aplicación Angular creada por esta plantilla y la creada por Angular CLI (vía `ng new`); sin embargo, las capacidades de la aplicación no cambian. La aplicación creada por la plantilla contiene un diseño basado en Bootstrap y un ejemplo de enrutamiento básico.

Ejecutar comandos ng

En un símbolo del sistema, cambie al subdirectorio *ClientApp* :

ConsolaCopiar
```
cd ClientApp
```

Si tiene la `ng`herramienta instalada globalmente, puede ejecutar cualquiera de sus comandos. Por ejemplo, puede ejecutar `ng lint`, `ng test` o cualquiera de los otros comandos de CLI angular . Sin `ng serve` embargo, no hay necesidad de ejecutar , porque su aplicación ASP.NET Core se ocupa de servir tanto las partes

del lado del servidor como del lado del cliente. Internamente, se utiliza `ng serve` en el desarrollo.

Si no tiene la herramienta `ng` instalada, ejecute en su lugar `npm run`. Por ejemplo, puede ejecutar `npm run ng lint` o `npm run ng test`.

Instalar paquetes npm

Para instalar paquetes npm de terceros, use un símbolo del sistema en el subdirectorio *ClientApp* . Por ejemplo:

```
cd ClientApp
npm install --save <package_name>
```

Publicar y desplegar

En desarrollo, la aplicación se ejecuta en un modo optimizado para la conveniencia del desarrollador. Por ejemplo, los paquetes de JavaScript incluyen mapas de origen (para que cuando depure, pueda ver su código TypeScript original). La aplicación observa los cambios de archivos TypeScript, HTML y CSS en el disco y se recompila y recarga automáticamente cuando ve que esos archivos cambian.

En producción, publique una versión de su aplicación que esté optimizada para el rendimiento. Esto está configurado para suceder automáticamente. Cuando publica, la configuración de compilación emite una compilación minima, compilada por adelantado (AoT) de su código del lado del cliente. A diferencia de la compilación de desarrollo, la compilación de producción no requiere que Node.js esté instalado en el servidor (a menos que haya habilitado la representación del lado del servidor (SSR)).

Puede usar los métodos estándar de alojamiento e implementación de ASP.NET Core .

Ejecute "ng serve" de forma independiente

El proyecto está configurado para iniciar su propia instancia del servidor Angular CLI en segundo plano cuando la aplicación ASP.NET Core se inicia en modo de desarrollo. Esto es conveniente porque no tiene que ejecutar un servidor separado manualmente.

Hay una desventaja en esta configuración predeterminada. Cada vez que modifica su código C # y su aplicación ASP.NET Core necesita reiniciarse, el servidor Angular CLI se reinicia. Se requieren alrededor de 10 segundos para

iniciar la copia de seguridad. Si realiza ediciones frecuentes del código C # y no desea esperar a que se reinicie la CLI angular, ejecute el servidor de CLI angular externamente, independientemente del proceso de ASP.NET Core. Para hacerlo:

1. En un símbolo del sistema, cambie al subdirectorio *ClientApp* e inicie el servidor de desarrollo de CLI angular:

   ```
   cd ClientApp
   npm start
   ```

   Importante

   Use `npm start`para iniciar el servidor de desarrollo de CLI angular, no `ng serve`, para que se respete la configuración en *package.json* . Para pasar parámetros adicionales al servidor de CLI angular, agréguelos a la `scripts`línea correspondiente en su archivo *package.json* .

2. Modifique su aplicación ASP.NET Core para usar la instancia externa de CLI angular en lugar de lanzar una propia. En su clase de *Inicio* , reemplace la `spa.UseAngularCliServer`invocación con lo siguiente:

   C#
   ```
   spa.UseProxyToSpaDevelopmentServer("http://localhost:4200");
   ```

Cuando inicia su aplicación ASP.NET Core, no iniciará un servidor CLI angular. En su lugar, se usa la instancia que inició manualmente. Esto le permite iniciar y reiniciar más rápido. Ya no está esperando que Angular CLI reconstruya su aplicación cliente cada vez.

Pase datos del código .NET al código TypeScript

Durante SSR, es posible que desee pasar datos por solicitud de su aplicación ASP.NET Core a su aplicación Angular. Por ejemplo, puede pasar información de cookies o algo leído de una base de datos. Para hacer esto, edite su clase de *inicio* . En la devolución de llamada para `UseSpaPrerendering`, establezca un valor `options.SupplyData`como el siguiente:

C#
```
options.SupplyData = (context, data) =>
{
    // Creates a new value called isHttpsRequest that's passed to TypeScript
code
    data["isHttpsRequest"] = context.Request.IsHttps;
};
```

La `SupplyData`devolución de llamada le permite pasar datos arbitrarios, por solicitud, serializables JSON (por ejemplo, cadenas, booleanos o números). Su código *main.server.ts* recibe esto como `params.data`. Por ejemplo, el ejemplo de código anterior pasa un valor booleano como `params.data.isHttpsRequest`en la `createServerRenderer`devolución de llamada. Puede pasar esto a otras partes de su aplicación de cualquier forma admitida por Angular. Por ejemplo, vea cómo *main.server.ts* pasa el `BASE_URL`valor a cualquier componente cuyo constructor se declare para recibirlo.

Inconvenientes de SSR

No todas las aplicaciones se benefician de SSR. El beneficio principal es el rendimiento percibido. Los visitantes que llegan a su aplicación a través de una conexión de red lenta o en dispositivos móviles lentos ven la IU inicial rápidamente, incluso si lleva un tiempo recuperar o analizar los paquetes de JavaScript. Sin embargo, muchos SPA se utilizan principalmente en redes internas rápidas de la empresa en computadoras rápidas donde la aplicación aparece casi al instante.

Al mismo tiempo, existen inconvenientes significativos para habilitar SSR. Agrega complejidad a su proceso de desarrollo. Su código debe ejecutarse en dos entornos diferentes: del lado del cliente y del lado del servidor (en un entorno Node.js invocado desde ASP.NET Core). Aquí hay algunas cosas a tener en cuenta:

- SSR requiere una instalación de Node.js en sus servidores de producción. Este es automáticamente el caso para algunos escenarios de implementación, como Azure App Services, pero no para otros, como Azure Service Fabric.
- Al habilitar el indicador `BuildServerSideRenderer` de compilación, el directorio *node_modules se* publica. Esta carpeta contiene más de 20,000 archivos, lo que aumenta el tiempo de implementación.
- Para ejecutar su código en un entorno Node.js, no puede confiar en la existencia de API de JavaScript específicas del navegador como `window` o `localStorage`. Si su código (o alguna biblioteca de terceros a la que hace referencia) intenta usar estas API, recibirá un error durante la SSR. Por ejemplo, no use jQuery porque hace referencia a API específicas del navegador en muchos lugares. Para evitar errores, debe evitar SSR o evitar las API o bibliotecas específicas del navegador. Puede ajustar cualquier llamada a dichas API en cheques para asegurarse de que no se invoquen durante la SSR. Por ejemplo, use una verificación como la siguiente en código JavaScript o TypeScript:

JavaScriptCopiar

```
if (typeof window !== 'undefined') {
    // Call browser-specific APIs here
}
```