Elecciones de diseño

Servicio RESTful

Se opto por una arquitectura MVC 5, para ahorrar tiempos de desarrollo y poder reutilizar la propuesta del framework en cuanto a la exposición de los métodos.
Si bien la plantilla generada cuenta con todas las clases y abstracciones necesarias para poder desarrollar todo bajo el mismo proyecto, se separo la solución en dos proyectos distintos, a fin tal de independizar en la mayor medida posible al servicio, asignándole principalmente el CRUD mas la auditoria, y un mínimo de lógica. Esta es la razón por la que se cuenta con dos modelos. Uno para cada proyecto.

Aplicación y back end

De la misma forma que para el servicio, se opto por una plantilla MVC 5. De donde se eliminaron archivos y dependencias no utilizadas en este Challenge.
Se genero un Modelo mediante Entity Framework, y se añadieron métodos customs para la obtención de datos asociados a las tareas.
La clase Tasks para moldear las tareas retornadas por el servicio se definió por fuera del ORM. Para evitar ante actualizaciones del modelo perder funcionalidad agregada.

Front end

Se utilizaron los propuestos por el framework al generar las plantillas. Bootstrap, MDB, Jquery. Mas una librería de JavaScript que se detalla mas adelante.
Se empleo tecnología HTML5/CSS3. No se utilizo Sass.
Se añadieron algunas funciones JavaScript para la invocación de los métodos de la controller y para el manejo dinámico de una barra de progreso.
Si bien la barra de progreso no era un requerimiento, se opto por agregarla para aportar funcionalidad.

Esquema de base de datos

Se definen algunas tablas y campos pensando en una cierta escalabilidad de la aplicación. En cuanto a tener algo más de funcionalidad que el CRUD. Se asume que una administración de tareas puede contener típicamente el manejo de estados tales como: pendientes, completadas, asignadas, reasignadas, rechazadas etc.
Toda esta funcionalidad no se encuentra actualmente programada, aunque se alcanzaron a imputar los estados Pendientes, Completadas, Eliminadas.

obs: Como caso ideal, la auditoría debería volcarse a otra base de datos independiente de la app. De esta forma, al generar archivos.bak y restores, se estaría ante volúmenes de datos considerablemente menores. Se adoptó para el challenge una única base, a modo tal de simplificar el caso y no lidiar entre el ORM y las vistas.

Librerías

Web Api Client. Se utilizó para configurar desde la aplicación los parámetros generales del servicio, y reducir código en la llamada a los métodos. Versión 5.2.7
JS Alertify. Librería Javascript para el manejo de notificaciones/cuadros de diálogos, con estilos ya resueltos. Versión 1.11.4

Consideraciones para el debug

URL GIT del proyecto: https://github.com/Patricio-Montes/TaskManager
Ubicación de la solución: TaskManager\TaskManager\TaskManager.sln
Ubicación de la base de datos: TaskManager\SQL\Base de datos\TaskManager_DEV.bak

Configuraciones:

Puertos IIS

TaskManager
https://localhost:44307/

TaskManagerAPI
https://localhost:44349/

Si esta url no coincide con el puerto donde se efectúe el debug, debe accederse dentro del proyecto a la ruta: TaskManager\TaskManager\WebApiClient\BaseAddress.cs

y reemplazar la url por lo que corresponda.

```
WebApiClient.BaseAddress = new Uri("https://localhost:44349/api/");
```

Web.config TaskManager

```
<connectionStrings>
<add name="TaskBackEnd" connectionString="data source=DESKTOP-T4S8AIP\SQLEXPRESS;initial
catalog=TaskManager_DEV;Integrated Security=True;" providerName="System.Data.SqlClient" />
```

<add name="TaskManagerEntities"
connectionString="metadata=res://*/Models.TaskManagerModel.csdl|res://*/Models.TaskManagerModel.ssdl|res://*/Models.TaskManagerModel.msl;provider=System.Data.SqlClient;provider connection string="data source=**DESKTOP-T4S8AIP\SQLEXPRESS**;initial catalog=TaskManager_DEV;persist security info=True;user id=sa;password=admin;MultipleActiveResultSets=True;App=EntityFramework""
providerName="System.Data.EntityClient" /></connectionStrings>

Web.config TaskManagerAPI

<connectionStrings><add name="TaskManagerApiEntities"
connectionString="metadata=res://*/Models.TaskManagerApiModel.csdl|res://*/Models.TaskManagerApiModel.ssdl|res://*/Models.TaskManagerApiModel.msl;provider=System.Data.SqlClient;provider connection string="data source=**DESKTOP-T4S8AIP\SQLEXPRESS**;initial catalog=TaskManager_DEV;persist security info=True;user id=sa;password=admin;MultipleActiveResultSets=True;App=EntityFramework""
providerName="System.Data.EntityClient" /></connectionStrings>

Algunas aclaraciones

En el método GetTasks() de la API, destinado a retornar todas las tareas, se añade una query para esta búsqueda debido a que las tareas eliminadas se encuentran en la tabla de auditoría. Esto podría haberse resuelto mediante un retorno de la forma "return db.tasks;" directamente, si en vez de eliminar las tareas y auditarlas, se las establece en baja mediante algún campo.

El usuario se establece siempre hardcode en "1", dado un caso real este valor debería tomarse de sesión por usuario logueado, e incluirlo en la firma de los métodos que lo utilizan. Al realizarse de la manera mencionada, la aplicación ya se encuentra preparada para recuperar tareas por usuarios.

Versión de Entity Framework: 6
IDE: Visual Studio 2019. Creación del proyecto en Net Framework 4.5 por incompatibilidad presentada al iniciar el desarrollo entre las plantillas MVC con el Framework 4. (Esto pudo deberse también a la versión de Visual con la que se trabajó).

Pendientes

1. Resolver un inconveniente con los Data Anotation y jqueryvalidate, que se ejecutan en el onload de las vistas parciales, en vez de en el submit de la tarea.
2. Investigar un mejor esquema para resolver la auditoría. Estimo que existe un esquema mucho más elegante.
3. Desarrollo de un login para capturar el usuario en sesión en vez de asumir arbitrariamente su valor, y poder incluir filtros.