**CAPÍTULO 7**

# **IMPLEMENTACIÓN**

Ahora vamos a pasar a la fase de implementación del proyecto. En esta fase van a intervenir varias herramientas, las cuales son: Visual Studio (como se ha mencionado anteriormente), SQL Manager y Postman.

Para la implementación se ha dividido el proyecto en dos partes: la parte de Front-End y la parte de Back-End. Cada una de estas partes tiene un lenguaje determinado. La parte del Back-End estará escrita en C# y para la parte del Front-End, se utilizará el Framework React, el cual está basado en Type-Script, que es un lenguaje similar a Java Script, pero con la imposición de tipos en las variables.

Vamos a proceder a analizar primero la parte del Back-End. En este caso para hacer un buen desarrollo del Back-End nos guiamos de (2 pág. 64) para desarrollar una arquitectura en N capas.

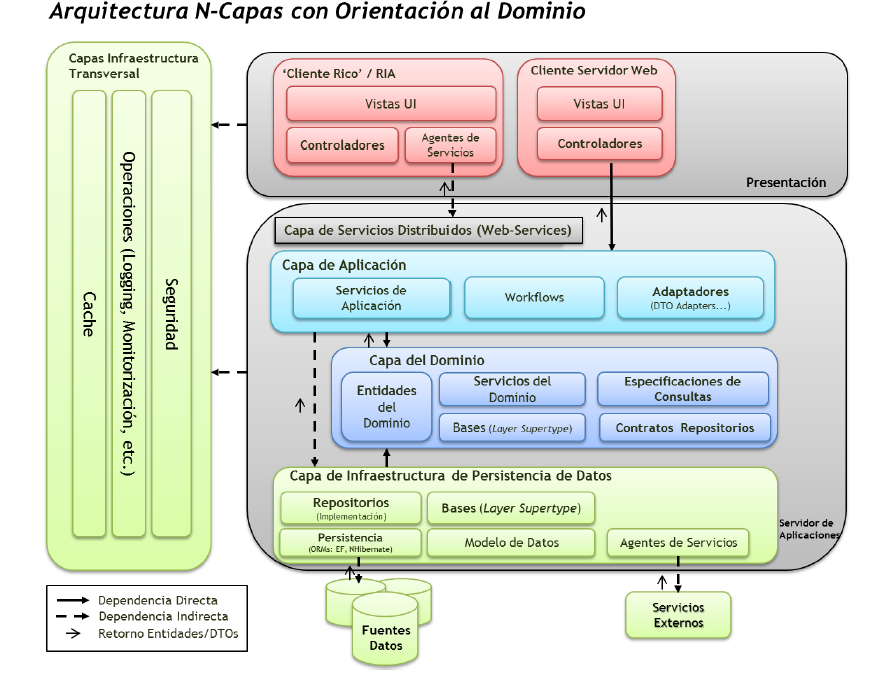


Ilustración 20. Arquitectura en N-Capas con orientación al Dominio, pág. 64.

En este caso, nuestro Back-End constará de una capa DAL (Data Access Layer), la cual se encargará exclusivamente de realizar las consultas en la base de datos. Una capa Business, la cual se encargará de llevar a cabo la lógica en el servidor.

Finalmente, en el Back-End, tendremos una API Rest, la cual se comunicará con la capa de negocio para realizar los cambios en la base de datos. A esta API se accederá mediante las URLs desde las distintas pestañas del Front-End.

También, en este apartado tenemos una funcionalidad que no depende de ninguna llamada por parte del Front-End. Esta funcionalidad se trata del envío de notificaciones, tal y como se indica en el caso de uso **CU-34**, todos los días deberá de enviarse una notificación a todos los usuarios que tengan las notificaciones activadas y que celebren un evento de prioridad alta ese mismo día. Esto se consigue gracias a la implantación de un demonio en la capa de negocio que, una vez lanzado repetirá periódicamente una vez al día las tareas de consulta de los usuarios mediante los criterios citados anteriormente y el envío de mensajes mediante correo electrónico gracias a una API externa de carácter comercial llamada Send Grid. La decisión de utilizar una API externa para el envío de mensajes viene de las limitaciones de los servidores de Azure de correos semanales, todo ello con la finalidad de evitar el SPAM.

Gracias a esta aplicación garantizo no solo la seguridad en el envío de mensajes, sino también una posible ampliación del flujo de mensajes semanales en el caso en que esta aplicación creciera en número de usuarios, únicamente mediante el cambio de suscripción en Send Grid.

Ahora vamos a abordar la parte del Front-End. En esta parte, como he mencionado anteriormente, utilizaremos el Framework React. He elegido este Framework, en primer lugar, por su integración con Visual Studio, y en segundo lugar por su tratamiento con Ajax, ya que, en este caso, React integra a Ajax de forma nativa. Con esto y el uso de las variables de estado de la plantilla, se consigue un efecto de página estática en la que se están produciendo llamadas a la API y continuas modificaciones en los datos sin que el usuario perciba que la página se recarga. Para las plantillas de estilo de la página se ha optado por la opción de Boostrap, debido a su sencillez visual y a su característica responsiva. Es muy importante esta característica porque necesitamos que esta página sea accesible por todos los dispositivos posibles (ordenadores, tabletas, teléfonos móviles…) y es necesario que la página se redimensione correctamente en función del dispositivo que la esté utilizando para garantizar la mayor accesibilidad posible.

Uno de los inconvenientes que se han encontrado a la hora de implementar es la necesidad de depuración en local de la aplicación. Gracias a la herramienta Visual Studio, no necesitamos hacer subidas constantes de la aplicación para probar las nuevas funcionalidades implementadas, a pesar de tener la base de datos en el servidor. Para ayudar a la depuración se han establecido puertos fijos a la ejecución de Back-End y el Front-End.

|  |  |
| --- | --- |
| Ilustración 21. Configuración del Back-End (localhost:11111) | Ilustración 22. Configuración del Back-End (localhost:44444) |

Gracias a esto podemos establecer las direcciones de la API en un fichero a parte y cambiar entre localhost y la dirección de Azure una vez queramos depurar o hacer un despliegue.

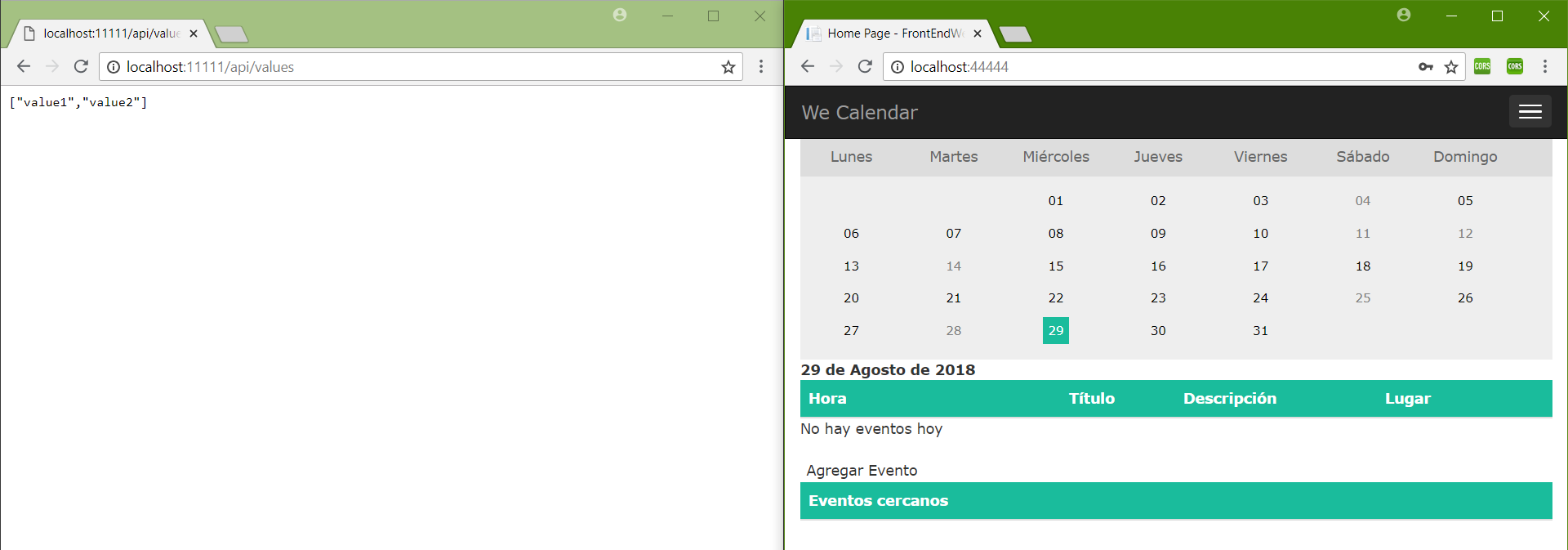


Ilustración 23. Back-End y Front-End ejecutándose a la vez en local.

En la depuración es donde encontramos un problema a la hora de hacer peticiones a la API. El problema es el control de acceso HTTP CORS (3), en este caso es necesario añadir una cabecera específica a nuestras rutas de la API ([EnableCors("AllowSpecificOrigin")]) para permitir que se realicen llamadas desde la aplicación en depuración. También es necesario agregar al navegador (Chrome en este caso ya que es el navegador que utiliza Visual Studio para la depuración) un complemento denominado Allow-Control-Allow-Origin para que el propio navegador también permita este acceso.

Para realizar las peticiones desde el Front-End a la Api vamos a ayudarnos de la librería de Axios, librería compatible con los paquetes de React que nos ayudará a realizar peticiones a la API y recibir las respuestas con ayuda de las “Promesas”. Gracias a las promesas en las peticiones podemos especificar el formato en el que deseamos que se nos proporcione la respuesta. Esto hace que esa petición no acepte un formato distinto al que le proporcionemos, mitigando los errores en la transferencia de datos.

## **7.1 SEGURIDAD**

Los factores que se han tenido en cuenta en este apartado solos factores relacionados con los datos de seguridad del usuario. A continuación, hablaremos del uso de las claves cifradas para guardar las contraseñas, las consultas mediante Token para garantizar anonimato y el hecho de que el código del Back-End ha sido compilado en el servidor, por lo que lo único que encontraremos al acceder a él, serán los archivos ejecutables del mismo.

CIFRADO DE CONTRASEÑAS

En este apartado hablaremos de la necesidad del cifrado de las contraseñas. Todas las contraseñas de los usuarios guardadas en la base de datos están cifradas de manera simétrica, estableciéndole a todas las mismas dimensiones de caracteres con un cifrado de 16bits, haciendo muy difícil las violaciones de seguridad mediante ataques de fuerza bruta.

TOKEN

Todos los usuarios, como se ha visto anteriormente, tienen almacenados una variable llamada Token, la cual nos servirá para realizar consultas. Esta variable es única y gracias a una función de C# (4) se puede generar de una forma sencilla.

La dinámica del Token consiste en que cada vez que el usuario inicie sesión, el sistema tras validar sus credenciales asignará un nuevo Token distinto al usuario, haciendo que cada vez que éste inicie sesión, el campo de búsqueda cambie evitando así posibles ataques de suplantación de identidad.

COMPILACIÓN DEL BACK-END

Aunque esta característica se ha planteado antes como una desventaja a la hora de elegir C# como lenguaje, hay que destacar que es una protección contra posibles hackeos al servidor, ya que, si una persona lograse entrar en nuestro servidor de Azure, lo único que encontraría sería un ejecutable, dificultando así la obtención de datos del código fuente.

## **7.2 USABILIDAD**

Al tratarse este proyecto como uno principalmente orientado al usuario final, es necesario pensar y orientar todas las acciones que se realizan en la página para proporcionarle al usuario una experiencia fácil y cómoda.

Para ello se ha tratado de reducir lo máximo posible el número de pulsaciones que un usuario debe realizar en la página para terminar la acción deseada. Se han implementado buscadores para filtrar y reducir lo máximo posible el desplazamiento por la pantalla, agilizando las búsquedas.

## **7.3 PERSISTENCIA**

USO DEL SERVIDOR DE AZURE Y QUE HACE PARA ASEGURAR LA PERSISTENCIA