Instituto Politécnico do Cávado e do Ave
Engenharia de sistemas Informáticos

Relatório do Trabalho Prático II

Integração de Sistemas de Integração

Servidor de Base de Dados, API e Aplicação cliente para uma Agenda personalizada

Bernardo Lima, nº 14885

Tatiana Maia, nº 14887

ESI Diurno, 3º ano

Índice

1. Introdução

Este trabalho foi-nos proposto pela unidade curricular de Integração de Sistemas de Informação com o objetivo de aplicarmos e consolidarmos os conhecimentos adquiridos nas aulas relativamente a Serviços, como os criar e utilizar, e como publicá-los numa *Cloud.*

2. Tema do trabalho

O nosso trabalho consistiu no desenvolvimento de uma agenda digital que permite ao Utilizador registar-se e fazer o login com os seus dados. Após o login, o Utilizador é deparado com o menu, onde se encontra a previsão meteorológica dos próximos cinco dias para a cidade escolhida pelo Utilizador no momento de registo, bem como uma *DataGrid* que conterá a informação de todos os eventos criados por ele (Utilizador) e ainda três botões que permitem ao Utilizador adicionar um novo evento, bem como alterá-lo e removê-lo. Os dados relativos à previsão meteorológica são provenientes da API do Instituto Português do Mar e da Atmosfera (IPMA), cujo link estará disponibilizado na bibliografia.

3. Arquitetura do trabalho (client e web api e um textinho e os serviços desenvolvidos (só rest))

Inicialmente a arquitetura do nosso trabalho consistia num projeto do Visual Studio do tipo "*WCF Services*". Mas mais tarde, decidimos que seria mais lógico utilizar uma "*Web API*" para implementar os nossos serviços. Portanto o nosso trabalho prático consiste em dois projetos: um onde criamos a nossa *Web API,* onde os serviços estão implementados e outro que mostra a aplicação desenvolvida.

É de referenciar que tanto como a nossa base de dados como a nossa API estão *hosted* na plataforma Azure da Microsoft.

Para além disso, é necessário referenciar que apenas foram desenvolvidos serviços RESTful.

4. Métodos desenvolvidos

No total foram implementados 6 métodos, mais os 3 criados para retirar informação da API do IPMA.

Para o Utilizador foram criados os métodos:

-"`[HttpGet("Login/{email}/{password}")]`
`public Aux Login(string email, string password)`"

Este método GET recebe como parâmetros o email e a password introduzidos na aplicação. O serviço , através do método "`public DataTable ReturnUsersEmailPassword(string email, string password)`" verifica na nossa base de dados por um utilizador com os dados introduzidos e devolve um objeto do tipo DataTable. Caso não exista ninguém com os mesmos dados, o serviço atribui o valor "null" à propriedade Json do objeto do objeto "aux" do tipo "Aux" que irá ser retornado. Caso exista alguém, os dados desse utilizador são atribuídos a um novo objeto "newUser" do tipo "Utilizador" que irá ser serializado para uma string, string esta que será a propriedade "Json" do objeto "aux". No final, o objeto "aux" é retornado.

-"`[HttpPost("Registo/{jsonString}")]`
`public ActionResult<bool> Registo(string jsonString)`"

Este método recebe POST como parâmetro uma string no formato JSON que irá conter a informação do novo utilizador a ser adicionado à nossa base de dados. Este método é do tipo "`ActionResult<bool>`" para retornar "`HttpStatusCodes`", neste caso, "200 OK" caso o registo seja efetuado com sucesso, ou "400 Bad Request" caso já exista um utilizador com o email introduzido.

De modo a saber qual mensagem a retornar, executa o método "`public bool Registo(string jsonString)`", que inicialmente irá desserializar "jsonString" para um novo objeto do tipo "Utilizador". Seguidamente o serviço irá pesquisar na base de dados por utilizadores que tenham o mesmo e-mail que aquele que foi introduzido. Caso já exista algum, é retornado false. Caso contrário, o programa, para fazer a atribuição do ID ao utilizador, pesquisa por todos os utilizadores na base de dados e soma 1 à propriedade "Rows.Count" da "DataTable" que contém todos os utilizadores. Por fim, adiciona o novo utilizador à base de dados e retorna true.

Para os Eventos foram criados os métodos:

-"`[HttpPost("AddEvento/{jsonString}")]`
`public ActionResult<bool> AddEvento(string jsonString)`"

Similarmente ao método de Registo para o utilizador, também este método POST retorna "HttpStatusCodes", também "200 OK" e "400 Bad Request", mas neste caso, para o sucesso ao adicionar o evento e para caso já exista um evento com os mesmos dados, respetivamente.

De modo a saber qual mensagem a retornar, executa o método "`public bool AddEvento(string jsonString)`", que inicialmente irá desserializar "jsonString" para um novo objeto do tipo "Evento". Seguidamente o serviço procura na base de dados por eventos com os mesmos dados. Caso já exista retorna false. Caso contrário executa o processo de atribuição do id similarmente ao método de registo, mas neste caso pesquisa por todos os eventos da base de dados. Por fim adiciona o novo evento à base de dados e retorna true.

-"
```
[HttpGet("GetEventos/{id_utilizador}")]
  public Aux GetEventos(string id_utilizador)
```
"

Este método GET é requisitado pela aplicação quando o form de Menu é mostrado, para carregar todos os eventos do utilizador para a *DataGrid.* O método pesquisa na base de dados por todos os eventos do utilizador. Caso o mesmo não tenha eventos previamente criados a propriedade "Json" do objeto "aux" do tipo "Aux" é inicializada como "null". Caso contrário, para cada linha da DataTable que contém os eventos do utilizador, é criado um novo evento (objeto do tipo "Evento") e adicionado a uma lista de eventos que será serializada para uma string que será atribuída à propriedade "Json" de "aux". Por fim, o objeto "aux" é retornado.

-"
```
[HttpDelete("DeleteEvento/{jsonString}")]
  public bool DeleteEvento(string jsonString)
```
"

Este método DELETE recebe como parâmetro uma string em formato JSON que irá conter a informação do evento a ser removido. Inicialmente, o serviço desserializa o conteúdo da string "jsonString" para um objeto do tipo "Evento". Seguidamente, é efetuada uma pesquisa na base de dados pelo evento que possui o mesmo id do que aquele que se deseja remover e, quando o mesmo é encontrado, é também removido da base de dados.

-"
```
[HttpPut("UpdateEvento/{jsonString}")]
  public bool UpdateEvento(string jsonString)
```
"

Este método UPDATE recebe como parâmetro uma string em formato JSON que irá conter a informação já alterada do evento a ser alterado. Inicialmente, o serviço desserializa o conteúdo da string "jsonString" para um objeto do tipo "Evento". Seguidamente, é efetuada uma pesquisa na base de dados pelo evento que possui o mesmo id do que aquele que se deseja alterar e, quando o mesmo é encontrado, é também alterado na base de dados.

Os métodos criados para o IPMA foram simples requests à API do IPMA para retirar informação sobre a previsão meteorológica dos próximos cinco dias, bem como a descrição dos vários tipos de tempo.

5. Links

-API documentada na plataforma "Swagger": https://app.swaggerhub.com/apis-docs/ISI_TP2/ISI_API/1-oas3#/

-Link da nossa API: https://isitp2-apim.azure-api.net/

6. Problemas encontrados

Um dos maiores problemas encontrados foi na Base de Dados, visto que as primary keys de cada tabela foram definidas como sendo "SERIAL", o que implicaria que não seria necessário atribuir um valor aos id's de evento e utilizador. No entanto tal não se verificou, o que gera um problema na base de dados quando, por exemplo, existem 3 eventos (id=1, id=2, id=3) e o utilizador deseja apagar um que não seja o último. Quando quiser introduzir um novo evento o mesmo não é adicionado por causa da atribuição do id. O outro problema que não foi possível resolver foi do lado do cliente, visto que na aplicação o utilizador só pode adicionar um evento de cada vez que usa a aplicação, sendo este um erro do Windows Forms que não conseguimos resolver.

7. Bibliografia

- https://stackoverflow.com/
- https://api.ipma.pt/
- https://docs.microsoft.com/pt-pt/

8. Conclusão

Este trabalho permitiu-nos expandir os nossos horizontes no que toca ao desenvolvimento de software, ensinando-nos a criar aplicações que não dependam de Bases de Dados locais, ou que todos os métodos estejam implementados localmente. Também serviu para ganharmos mais conhecimento na área do JSON, o que é bastante importante.