

The page features a decorative graphic consisting of three concentric blue circles of varying sizes, each with a lighter blue outer ring. These circles are positioned in the top right, middle right, and bottom right areas of the page. Thin blue lines extend from the top left and bottom right corners towards the circles, creating a sense of movement or connection.

Battlenet Profile Analyzer

Miguel Burdeos Tébar

Ciclo Desarrollo de Aplicaciones
Multiplataforma

Memoria del proyecto de DAM

IES Abastos. Curso 2015/2016. Grupo 7N. 18
de Enero de 2016

Tutor: Beatriz Pérez Oñate

Índice

1.	Introducción, justificación y objetivos del proyecto	2
2.	Herramientas y tecnologías software utilizadas.....	5
	Concepto de API.....	5
	La API pública de Blizzard	5
	Términos de uso de la API	7
	Lenguaje C Sharp (C#)	9
	Visual Studio 2012 como IDE para el desarrollo de la aplicación	10
	Adobe Photoshop CC 2015.....	11
	Deserializador Json .dll por Newtonsoft	11
	IExpress 2.0 como empaquetador para la instalación de la aplicación	13
3.	Planificación del proyecto, sus fases y metodología usada	13
	Proceso de planificación de tareas <i>SCRUM</i>	14
	Daily Scrum o Stand-up meeting.....	16
	Planificación del Sprint.....	16
	Backlog de la aplicación.....	16
	Coding Guidelines generales	17
	Coding guidelines usados para C#.....	19
4.	Licencia de la aplicación “Proprietary License Software”	20
5.	Desarrollo del proyecto	20
	Splash Screen.....	20
	GameChooser.....	22
	ProfileForm (Diablo 3)	23
	ProfileShower (Diablo 3)	24
	DiabloCharForm (Visualizador de personaje).....	26
	WowProfileForm (World of Warcraft).....	29
	WowProfileShower (Visualizador de personaje).....	30
	Desarrollo del multilenguaje de la aplicación.....	32
6.	Evaluación y conclusiones finales	34
7.	Referencias (bibliografía, revistas, webs, etc.)	35
8.	Agradecimientos	35

1. Introducción, justificación y objetivos del proyecto

El proyecto que se presenta a continuación no estará basado en las actividades realizadas en el centro de trabajo (FCT), en esta ocasión este proyecto ha sido realizado de forma independiente por el alumno, de esta manera lo que se intenta es mostrar un carácter más independiente evitando que las actividades del centro de trabajo sean el único guion necesario para la realización del mismo.

Principalmente los motivos que me han llevado a realizar un proyecto independiente (además de lo ya explicado anteriormente) han sido la motivación por desarrollar algo propio sin depender de nada o nadie, la materia que se ha desarrollado (basado en videojuegos) y el lenguaje utilizado a la hora del desarrollo, y es que hoy en día C# supone uno de los lenguajes de programación más importante, con más futuro y que permite trabajar en un entorno de desarrollo realmente potente y robusto como es Visual Studio, de la mano de Microsoft.



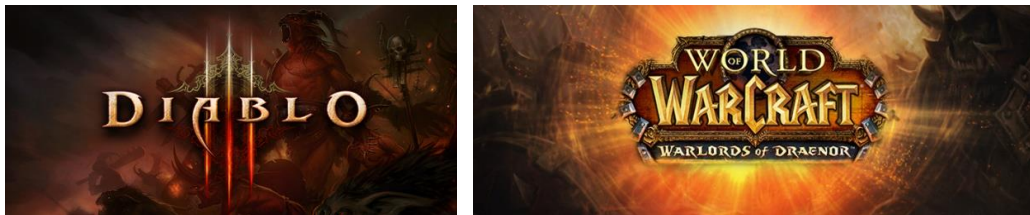
A día de hoy la realización de proyectos independientes permite al desarrollador nutrirse de conocimiento por cuenta propia y aumentar su portafolio de cara a un entorno laboral real, ambos imprescindibles para que el desarrollador pueda prosperar y mejorar. A continuación se resumirán seis de los métodos más importantes que puede utilizar un desarrollador independiente para poder prosperar por si mismo:

- **Atraer clientes:** En ocasiones, empresas o individuos pueden necesitar un programador para sus proyectos. Si creas un sitio web atractivo que aparezca en los buscadores de Internet, mediante el pago de publicidad o la optimización de los buscadores de Internet, los clientes podrán ponerse en contacto contigo para discutir sobre proyectos. También puedes conseguir este tipo de trabajo mediante la publicación de tu currículum en bolsas de trabajo, indicando específicamente que estás buscando empleo como trabajador independiente.
- **Licitación de proyectos:** Algunos sitios como Guru.com o ScriptLance.com actúan como intermediarios entre los programadores independientes y los clientes. El cliente deberá enviar una descripción del trabajo y los programadores calificados prepararán una oferta, incluyendo el tiempo que se tardarán en hacer el proyecto y cuánto costará. Después de comparar las ofertas y la calidad del trabajo, el cliente selecciona a un programador. Después de terminar el proyecto, el cliente te califica, lo que te sirve para la reputación de tu sitio web y así atraer a más clientes.
- **Contrato de trabajo:** Algunas empresas contratan a programadores independientes mediante una base de datos casi permanente de estos trabajadores. Puedes trabajar en el lugar con la empresa, ganando un sueldo fijo, pero el proyecto suele durar poco tiempo. Esto ofrece la seguridad de dinero constante durante el tiempo que dura el proyecto, pero también tienes la libertad de trabajar en otro lugar cuando lo finalices.
- **Venta de programas:** Crea tu propio programa de software y luego véndelo para obtener tu retribución. La ventaja de este método es que haces el trabajo una vez pero sigues ganando dinero a costa de la obra. Considera la posibilidad de programar software orientado a la práctica, como presupuestos y cálculos, software de organización, juegos o aplicaciones para el iPhone.
- **Programas gratuitos:** La venta de software no es la única manera de generar ganancias a través de tus programas. También puedes crear programas de software libre para que te conozcan. Para ganar dinero haciendo esto, debes ofrecer una versión "premium" del software que incluya más funciones. Alternativamente, puedes cobrar a la gente por manejar problemas de soporte técnico.
- **Trabajo subcontratado:** No tienes que hacer todo el trabajo de programación tú mismo. Si tienes una habilidad especial para obtener proyectos, puedes subcontratar el trabajo de otros programadores por menos dinero, revisar su trabajo, luego cobrar al cliente directamente. Este modelo de negocio permite que tomes varios clientes a la vez, potenciando las ganancias de dinero a medida que tu negocio va creciendo.

Debido a que el desarrollo del proyecto ha sido de forma independiente y durante la realización de las FCT esto ha supuesto que el tiempo empleado para el desarrollo de la aplicación ha tenido que ser fraccionado imponiendo fechas personales (cumplidas satisfactoriamente), por lo que me ha ayudado a mejorar mi capacidad de auto gestión del tiempo, cosa que es muy importante para un trabajador, sobre todo en este ámbito.

En este caso, la idea principal de esta aplicación se basa en la utilización de la API pública que la compañía Blizzard dispone para cualquier desarrollador que quiera utilizarla, en este caso la API permitirá al desarrollador acceder a secciones públicas de sus bases de datos, de esta manera existen múltiples usos para esta API, como la realización de una página web que permita manejar datos de los diferentes Videojuegos que Blizzard dispone, una App para Android/IOS o una aplicación con interfaz completa para escritorio como en el caso de este proyecto.

La aplicación Battlenet Profile Analyzer permitirá a cualquier usuario acceder a sus perfiles públicos de dos de los juegos con más auge que la compañía Blizzard dispone a día de hoy como son Diablo 3 y World of Warcraft.



La ventaja de Battlenet Profile Analyzer es que el usuario será capaz de acceder a toda la información de su cuenta que hay disponible públicamente de manera mucho más rápida ya que de la manera convencional el usuario es obligado a introducir sus credenciales en el sitio web determinado para cada juego (o la información de alguno de sus personajes).

En este caso el usuario tendrá en una sola aplicación el acceso a ambos juegos y con menos pasos intermedios para acceder a ellos, introduciendo su BattleTag (en el caso de diablo 3) o el nombre y servidor de uno de sus personajes (en el caso de World of Warcraft).

Esto permite, como ya se ha explicado antes un acceso mucho más rápido, además la interfaz de la aplicación es totalmente intuitiva y fácil de utilizar, con una usabilidad bien adaptada y un diseño acorde a cada juego, lo cual permite que el usuario este en un ambiente más cómodo sin necesidad de usar el navegador y con un tiempo de carga mucho menor.

Por supuesto la aplicación está traducida a los 4 idiomas principales del continente europeo (siendo su idioma por defecto el inglés), lo cual permite que más usuarios puedan acceder a ella y que se permita abarcar un rango de edades mayor.

Podríamos decir que lo dicho anteriormente se basa en ideas, ventajas y características a nivel de usuario, lo que la aplicación ofrece. Hablando sobre objetivos de la aplicación entorno al desarrollo podríamos dividirlo en los siguientes apartados:

- Desarrollo con API pública de Blizzard.
- Acceso a datos.
- Diseño de Interfaz con WinForms.
- Desarrollo de la aplicación en lenguaje C#.
- Deserialización de objetos JavaScript (json) devueltos por la API.
- Diseño de usabilidad óptima en una aplicación de escritorio.
- Localización de una aplicación (Multilanguage).

Sobre el futuro de la aplicación habría que tener en cuenta que ya que dependemos de los datos almacenados en las BBDD de Blizzard, el cambio de su estructura o actualización de la API supondría una actualización de la aplicación.

2. Herramientas y tecnologías software utilizadas

En este apartado se intentará recopilar toda la información sobre las herramientas y tecnologías utilizadas en este proyecto y que son responsables de la composición de la aplicación, en este caso puesto que la aplicación es totalmente dependiente de la API de Blizzard empezaremos desarrollando el concepto de API para posteriormente explicar como esta misma afecta a la aplicación.

Concepto de API

Una API representa la capacidad de comunicación entre componentes de software. Se trata del conjunto de llamadas a ciertas bibliotecas que ofrecen acceso a ciertos servicios desde los procesos y representa un método para conseguir abstracción en la programación, generalmente (aunque no necesariamente) entre los niveles o capas inferiores y los superiores del software. Uno de los principales propósitos de una API consiste en proporcionar un conjunto de funciones de uso general, por ejemplo, para dibujar ventanas o iconos en la pantalla.

De esta forma, los programadores se benefician de las ventajas de la API haciendo uso de su funcionalidad, evitándose el trabajo de programar todo desde el principio. Las API asimismo son abstractas: el software que proporciona una cierta API generalmente es llamado la implementación de esa API.

Sin embargo, las API de alto nivel generalmente pierden flexibilidad; por ejemplo, resulta mucho más difícil en un navegador web hacer girar texto alrededor de un punto con un contorno parpadeante que programarlo a bajo nivel. Al elegir usar una API se debe llegar a un cierto equilibrio entre su potencia, simplicidad y pérdida de flexibilidad.

La API pública de Blizzard

Antes de saber de qué trata la API pública que esta compañía comparte con el resto de desarrolladores es importante que sepamos qué tipo de compañía es Blizzard y que nos hagamos una idea de cómo está colocada en el mercado de videojuegos.



Blizzard Entertainment® es una de las principales compañías de desarrollo y publicación de software de entretenimiento. Tras fundar la marca Blizzard en 1994, la compañía se convirtió rápidamente en uno de los fabricantes de videojuegos más famosos y respetados. Al concentrar su energía en la creación de experiencias lúdicas excepcionales, Blizzard ha sabido mantener una reputación de calidad sin igual desde su fundación.

Blizzard ha acumulado los récords de las mejores ventas durante más de una década, y con grandes éxitos como los recientes *World of Warcraft®: Warlords of Draenor™*, *Hearthstone®: Heroes of Warcraft™*, *Diablo® III: Reaper of Souls™* y *StarCraft® II: Heart of the Swarm®*. La compañía ha conseguido de manera consecutiva varios premios al Juego del Año. El servicio gratuito de juego online de la compañía, *Battle.net®*, es uno de los mayores del mundo, con millones de usuarios en activo.

Los productos de alta calidad basados en los juegos de Blizzard también han tenido una muy buena acogida y han sido aclamados por la crítica. Entre estos productos se encuentran figuras, novelas, cómics manga, juegos de mesa, juegos de rol, prendas, el *juego de miniaturas de World of Warcraft* y el *juego de cartas coleccionables de World of Warcraft*, un éxito de ventas en su categoría. En la actualidad, Blizzard extiende aún más el alcance de sus universos de juego, y trabaja en la creación de una película sobre Warcraft® junto a Legendary Pictures, el estudio que ha producido películas como *Man of Steel*, *El Caballero Oscuro*, *Inception*, *Watchmen* y *300*.

Los productores convierten los juegos y los servicios de Blizzard en el mejor producto posible. Ayudan a creadores con trasfondos dispares a comunicarse en el mismo idioma y supervisan esfuerzos complejos repartidos en múltiples regiones de todo el mundo.

El objetivo principal de un productor es encargarse de las dificultades que pudieran suponer un problema para el trabajo de los miembros del equipo. Los productores de Blizzard no están "al mando", en vez de eso, hacen uso de las herramientas de gestión de proyectos, de una gran empatía y de la comunicación con todo el equipo para convertirse en un punto de apoyo en el huracán que es el desarrollo de un juego. Los productores se aseguran de comprobar hasta el último detalle y de que las fechas de entrega sean las adecuadas y se cumplan.

La producción de Blizzard se enorgullece de mantener a los ingenieros escribiendo código, a los artistas dibujando y a los diseñadores desarrollando sistemas de juego divertidos. La labor del productor es resolver cualquier cosa que interfiera con el trabajo de otras disciplinas. Los productores crean el entorno que posibilita el desarrollo de los productos y servicios de Blizzard y son los que transmiten la visión de Blizzard a sus equipos para que nunca se desvíen de su curso.

Además del éxito ya explicado anteriormente podemos decir abiertamente que Blizzard es una de las pocas compañías de videojuegos que muestra tanto interés por los nuevos desarrolladores, ofreciendo puestos de trabajo para desarrolladores tanto senior como nuevos y además ofreciendo el uso de sus datos públicos mediante su API para que cualquier desarrollador que quiera pueda usarlos en su aplicación sin necesidad de requerir beneficios por ventas o porcentajes de las ganancias, incluso sin inmiscuirse en los derechos de la aplicación, los cuales recaen en su totalidad en el desarrollador siempre y cuando este haga un buen uso de las términos de uso de la API.

Términos de uso de la API

En el siguiente link se detallan los términos de uso de la API:

<https://dev.battle.net/policy>

Las políticas de uso en resumen serían:

- La API tiene que ser accedida mediante la "API Key" (que obtendremos al registrar nuestro proyecto y nuestra cuenta de usuario) la cual habrá que incluir en cada request mediante la URI.
- Cada usuario obtendrá una sola licencia para la utilización de la API.
- No están permitidas las versiones "Premium" en aplicaciones que utilicen la API.
- La aplicación no puede contener material ofensivo.
- La aplicación no puede perjudicar a los productos ya existentes de Blizzard.
- La aplicación tiene que estar dirigida para edades superiores a 13 años.
- El desarrollador es el único responsable de la aplicación.
- No se puede usar la API por motivos de marketing de otras aplicaciones.
- No puedes vender tu licencia de uso sobre la API.
- No se puede integrar datos obtenidos de la API en un producto físico.
- Tu "API Key" debe ser confidencial.
- No se puede utilizar ingeniería inversa sobre la API.
- Debes proteger los datos obtenidos con la API.
- Tu aplicación debe tener políticas de privacidad.
- No se puede realizar datamining con la API.

En nuestro caso para poder utilizar la API deberemos registrarnos en la web:

<https://dev.battle.net>

Una vez hecho esto podremos registrar nuestro proyecto, en el cual utilizaremos la API, entonces obtendremos nuestra API Key:

Battlenet Profile Analyzer

Name: Battlenet Profile Analyzer
Registered: [3 months ago](#)

API	Key
Game APIs: Basic Plan	dzswy2ebvfn3bm839vevuqhe6vhzcfm

[EDIT](#) [DELETE APPLICATION](#)

Una vez hecho esto podremos incluso testear los request de la API desde la propia web, mediante el apartado API Docs:

CHARACTER PROFILE API

[LIST METHODS](#) | [EXPAND METHODS](#)**GET** CHARACTER PROFILE /WOW/CHARACTER/:REALM/:CHARACTERNAME

The Character Profile API is the primary way to access character information. This Character Profile API can be used to fetch a single character at a time through an HTTP GET request to a URL describing the character profile resource. By default, a basic dataset will be returned and with each request and zero or more additional fields can be retrieved. To access this API, craft a resource URL pointing to the character who's information is to be retrieved.

Parameter	Value	Type	Description
:realm	<input type="text" value="test-realm"/>	string	The character's realm. Can be provided as the proper realm name or the normalized realm name.
:characterName	<input type="text" value="Peratryn"/>	string	The name of the character you want to retrieve.
fields	<input type="text"/>	string	The dataset you wish to retrieve for the character. Each field value is explained in more detail in the following methods. If no fields are specified the API will only return basic data about the character.
locale	<input type="text" value="en_US"/>	string	What locale to use in the response
jsonp	<input type="text"/>	string	Request data to be returned as a JsonP callback

[Try it!](#)

En general la utilización de la API de Blizzard se basa en Request (peticiones) mediante URI's las cuales nos devolverán objetos JavaScript en formato JSON, los cuales a su vez deserializaremos en nuestra aplicación mediante código C# y formatearemos los resultados para usarlos como queramos.

La simplicidad de JSON ha dado lugar a la generalización de su uso, especialmente como alternativa a XML en AJAX. Una de las supuestas ventajas de JSON sobre XML como formato de intercambio de datos en este contexto es que es mucho más sencillo escribir un analizador sintáctico (parser) de JSON. En JavaScript, un texto JSON se puede analizar fácilmente usando la función `eval()`, lo cual ha sido fundamental para que JSON haya sido aceptado por parte de la comunidad de desarrolladores AJAX, debido a la ubicuidad de JavaScript en casi cualquier navegador web.

En la práctica, los argumentos a favor de la facilidad de desarrollo de analizadores o del rendimiento de los mismos son poco relevantes, debido a las cuestiones de seguridad que plantea el uso de `eval()` y el auge del procesamiento nativo de XML incorporado en los navegadores modernos. Por esa razón, JSON se emplea habitualmente en entornos donde el tamaño del flujo de datos entre cliente y servidor es de vital importancia (de aquí su uso por Yahoo, Google, etc, que atienden a millones de usuarios) cuando la fuente de datos es explícitamente de fiar y donde no es importante el no disponer de procesamiento XSLT para manipular los datos en el cliente. Si bien es frecuente ver JSON posicionado *contra* XML, también es frecuente el uso de JSON y XML en la misma aplicación.

Lenguaje C Sharp (C#)



En su totalidad, el lenguaje utilizado para el desarrollo de esta aplicación ha sido C#, no solo por su robustez, sino porque actualmente (puesto que fue el lenguaje más utilizado durante el curso) es donde mejor me desenvuelvo programando, además es un lenguaje que se integra perfectamente con las interfaces de escritorio debido a su relación con WinForms.

C# (pronunciado *si sharp* en inglés) es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET, que después fue aprobado como un estándar por la ECMA (ECMA-334) e ISO (ISO/IEC 23270). C# es uno de los lenguajes de programación diseñados para la infraestructura de lenguaje común.

Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma .NET, similar al de Java, aunque incluye mejoras derivadas de otros lenguajes.

El nombre C Sharp fue inspirado por la notación musical, donde '#' (sostenido, en inglés *sharp*) indica que la nota (C es la nota do en inglés) es un semitono más alta, sugiriendo que C# es superior a C/C++. Además, el signo '#' se compone de cuatro signos '+' pegados.²

Aunque C# forma parte de la plataforma .NET, ésta es una API, mientras que C# es un lenguaje de programación independiente diseñado para generar programas sobre dicha plataforma. Ya existe un compilador implementado que provee el marco Mono - DotGNU, el cual genera programas para distintas plataformas como Windows, Unix, Android, iOS, Windows Phone, Mac OS y GNU/Linux.

Durante el desarrollo de la plataforma .NET, las bibliotecas de clases fueron escritas originalmente usando un sistema de código gestionado llamado Simple Managed C (SMC). En enero de 1999, Anders Hejlsberg formó un equipo con la misión de desarrollar un nuevo lenguaje de programación llamado Cool (Lenguaje C orientado a objetos). Este nombre tuvo que ser cambiado debido a problemas de marca, pasando a llamarse C#.³ La biblioteca de clases de la plataforma .NET fue migrada entonces al nuevo lenguaje.

Hejlsberg lideró el proyecto de desarrollo de C#. Anteriormente, ya había participado en el desarrollo de otros lenguajes como Turbo Pascal, Delphi y J++.

Visual Studio 2012 como IDE para el desarrollo de la aplicación



Existen varios compiladores e IDE's para desarrollar en C#, sin embargo en la realización de este proyecto se ha usado en su totalidad Visual Studio 2012, no solo por su potencia y robustez, sino por sus herramientas de gran utilidad como **Microsoft IntelliSense**, la aplicación de autocompletar, mejor conocido por su utilización en Microsoft Visual Studio entorno de desarrollo integrado. Además de completar el símbolo de los nombres que el programador está escribiendo, IntelliSense sirve como documentación y desambiguación de los nombres de variables, funciones y métodos de utilización de metadatos basados en la reflexión.

El nombre proviene de una mala pronunciación de "intelligence" de un empleado de Microsoft llamado Juan Guardiola, que estaba tomando cerveza al ocurrírsele el nombre.

Microsoft Visual Studio es un entorno de desarrollo integrado (IDE, por sus siglas en inglés) para sistemas operativos Windows. Soporta múltiples lenguajes de programación tales como C++, C#, Visual Basic .NET, F#, Java, Python, Ruby, PHP; al igual que entornos de desarrollo web como ASP.NET MVC, Django, etc., a lo cual sumarle las nuevas capacidades online bajo Windows Azure en forma del editor Monaco.

Visual Studio permite a los desarrolladores crear sitios y aplicaciones web, así como servicios web en cualquier entorno que soporte la plataforma .NET (a partir de la versión .NET 2002). Así se pueden crear aplicaciones que se comuniquen entre estaciones de trabajo, páginas web, dispositivos móviles, dispositivos embebidos, consolas, etc.

En concreto la versión 2012 de Visual Studio permite trabajar con los siguientes Frameworks:

- .NET Framework 2.0
- .NET Framework 3.0
- .NET Framework 3.5
- .NET Framework 4.0
- .NET Framework 4.5 (utilizado en este proyecto)

¿Qué es .NET Framework?

.NET Framework es un entorno de ejecución administrado que proporciona diversos servicios a las aplicaciones en ejecución. Consta de dos componentes principales: Common Language Runtime (CLR), que es el motor de ejecución que controla las aplicaciones en ejecución, y la biblioteca de clases de .NET Framework, que proporciona una biblioteca de código probado y reutilizable al que pueden llamar los desarrolladores desde sus propias aplicaciones.

Adobe Photoshop CC 2015



En este caso el uso de esta herramienta de edición no ha sido muy grande, puesto que muchas de las adaptaciones de la interfaz podían realizarse en el mismo IDE Visual Studio 2012, sin embargo para adaptar imágenes a un tamaño en concreto de la ventana esta herramienta ha sido muy útil, ya que nos ha permitido cambiar el tamaño directamente en píxeles de diferentes imágenes usadas para fondos, el splash screen, etc... de manera que han podido ser adaptadas a una resolución en concreto, además de transparencias utilizadas, etc...

Adobe Photoshop es un editor de gráficos rasterizados desarrollado por Adobe Systems Incorporated. Usado principalmente para el retoque de fotografías y gráficos, su nombre en español significa literalmente "taller de fotos". Es líder mundial del mercado de las aplicaciones de edición de imágenes y domina este sector de tal manera que su nombre es ampliamente empleado como sinónimo para la edición de imágenes en general.

Actualmente forma parte de la familia Adobe Creative Suite y es desarrollado y comercializado por Adobe Systems Incorporated inicialmente para computadores Apple pero posteriormente también para plataformas PC con sistema operativo Windows. Su distribución viene en diferentes presentaciones, que van desde su forma individual hasta como parte de un paquete, siendo éstos: Adobe Creative Suite Design Premium y Versión Standard, Adobe Creative Suite Web Premium, Adobe Creative Suite Production Studio Premium y Adobe Creative Suite Master Collection.

Deserializador Json .dll por Newtonsoft

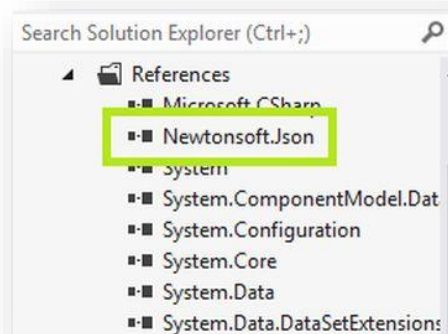


En el caso de este proyecto, y como ya se ha explicado en el apartado de la API, una vez que hagamos un request mediante una URI determinada, la API de Blizzard nos devolverá información en formato Json el cual es un formato ligero para el intercambio de datos.

JSON es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML, debido que a priori este formato no es compatible con C# es necesario parsear los datos, o como en nuestro caso, deserializarlos.

Lamentablemente C# no contiene ninguna clase en sus librerías nativas para tratar datos en formato Json, sin embargo Newtonsoft ofrece públicamente y de manera gratuita una DLL que añadiéndola a las dependencias de nuestro proyecto nos permitirá deserializar el contenido obtenido en formato Json a XML (entre otros) o serializar de nuevo a objeto JavaScript.

Aquí podemos ver un ejemplo de la librería añadida como dependencia a nuestro proyecto:



Para poder utilizar esta librería tenemos dos opciones, descargando las .dll necesarias en la web del desarrollador <http://www.newtonsoft.com/json> y añadirlas como dependencia a nuestro proyecto como se muestra en la anterior imagen o bien descargarla e instalarla automáticamente en nuestro proyecto mediante el gestor de extensiones y actualizaciones del que dispone Visual Studio:



IEExpress 2.0 como empaquetador para la instalación de la aplicación

IEExpress es una herramienta de Microsoft incluida en varias ediciones de los sistemas operativos Windows (32-bit y 64-bit): Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008, Windows 7, Windows 8, and Windows 10. Fue también incluida como parte de todo el Internet Explorer Administration Kit.

Esta herramienta es usada para crear un paquete de auto-extracción formado por un conjunto de ficheros a elección del usuario. Estos paquetes pueden ser usados para instalar aplicaciones, ejecutables, drivers, sistemas de componentes o setup bootstrappers, gracias a esto, estos paquetes pueden ser distribuidos a múltiples equipos, ya sean locales o remotos.



Todo esto mediante un asistente (IExpress Wizard) o mediante una directiva personalizada llamada SED file, la cual puede ser modificada con cualquier editor de texto o ASCII como notepad. IEXPRESS.EXE está ubicado en el directorio SYSTEM32 de nuestro sistema, tanto en versión 32 como 64 bits, o también puede ser iniciado mediante su ejecución escribiendo IExpress.exe en el RUN de nuestro menú de Windows.

3. Planificación del proyecto, sus fases y metodología usada

En el caso de este proyecto se ha usado una metodología de planificación estructurada, la cual me ha permitido fraccionar las tareas según su tiempo requerido, importancia, viabilidad, etc...

La realización del proyecto ha pasado por varias fases, desde la creación de ideas, requisitos funcionalidades, etc... hasta el desarrollo de la aplicación, pasando por diseño, pruebas de errores... etc. En concreto podríamos dividirlo en las siguientes etapas:

- **Fase de planificación:** Se trata de establecer cómo se deberá satisfacer las restricciones de prestaciones, planificación temporal, etc... Una planificación detallada da consistencia al proyecto y evita sorpresas que nunca son bien recibidas.
- **Nacimiento de la idea del proyecto:** Exposición de necesidades. Se crea un primer documento breve que recoge el anteproyecto y es aprobado por la dirección o el comité correspondiente (en este caso Bea).

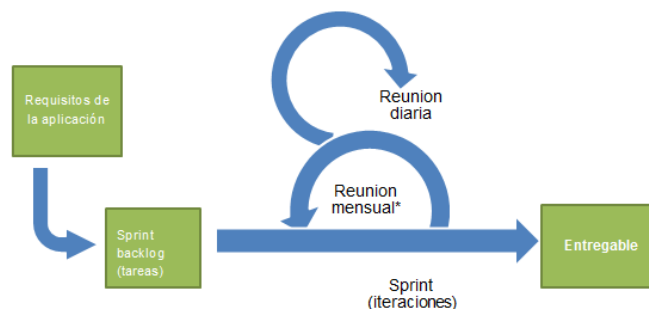
- **Estudio detallado:** Se define ya en detalle el contenido del proyecto, su análisis funcional, las cargas de trabajo previstas y la metodología a desarrollar, así como que herramientas software o tecnologías utilizaremos.
- **Definición del diseño de la interfaz:** Antes de empezar a implementar la interfaz gráfica se ha un boceto de la misma, sobre papel o mediante software determinado, de esta manera podemos tener una imagen preconcebida de cómo quedará y ahorrar tiempo en caso de estar pensando en el diseño durante el desarrollo.
- **Programación y pruebas:** Se realiza la programación de la aplicación y las pruebas para programación. De hecho, como el curso pasado nunca se ha programado deserializando objetos JavaScript, se realizó una IPU pertinente donde empleábamos las librerías descargadas mediante la web de Newtonsoft en un proyecto a parte.
- **Puesta en marcha (fase final):** La puesta en marcha de la aplicación es una fase delicada que requiere una estricta vigilancia hasta comprobar su correcto funcionamiento.

Inevitablemente, puesto que la aplicación está expuesta a sufrir cambios durante su desarrollo, ya sea de interfaz o de programación, la división de tareas fue variando a lo largo del desarrollo del proyecto entero, es por esto mismo, que la metodología utilizada fue la llamada *SCRUM*.

Proceso de planificación de tareas *SCRUM*

Scrum es el nombre con el que se denomina a los marcos de desarrollo ágiles caracterizados por:

- Adoptar una estrategia de desarrollo incremental, en lugar de la planificación y ejecución completa del producto.
- Basar la calidad del resultado más en el conocimiento tácito de las personas en equipos auto organizados, que en la calidad de los procesos empleados.
- Solapamiento de las diferentes fases del desarrollo, en lugar de realizar una tras otra en un ciclo secuencial o de cascada.



En el proceso de planificación *SCRUM* se aplican de manera regular un conjunto de buenas prácticas para trabajar colaborativamente, en equipo, y obtener el mejor resultado posible de un proyecto. Estas prácticas se apoyan unas a otras y su selección tiene origen en un estudio de la manera de trabajar de equipos altamente productivos.

En Scrum se realizan entregas parciales y regulares del producto final, priorizadas por el beneficio que aportan al receptor del proyecto. Por ello, Scrum está especialmente indicado para proyectos en entornos complejos, donde se necesita obtener resultados pronto, donde los requisitos son cambiantes o poco definidos, donde la innovación, la competitividad, la flexibilidad y la productividad son fundamentales.

Scrum también se utiliza para resolver situaciones en que no se está entregando al cliente lo que necesita, cuando las entregas se alargan demasiado, los costes se disparan o la calidad no es aceptable, cuando se necesita capacidad de reacción ante la competencia, cuando la moral de los equipos es baja y la rotación alta, cuando es necesario identificar y solucionar ineficiencias sistemáticamente o cuando se quiere trabajar utilizando un proceso especializado en el desarrollo de producto.

Un principio clave de Scrum es el reconocimiento de que durante un proyecto se puede cambiar de idea sobre lo que se quiere y se necesita (a menudo llamado *requirements churn*), y que los desafíos impredecibles no pueden ser fácilmente enfrentados de una forma predictiva y planificada. Por lo tanto, Scrum adopta una aproximación pragmática, aceptando que el problema no puede ser completamente entendido o definido, y centrándose en maximizar la capacidad del equipo de entregar rápidamente y responder a requisitos emergentes.

Las características más marcadas que se logran notar en Scrum serían:

- **Flexibilidad a cambios.** Gran capacidad de reacción ante los cambiantes requerimientos generados por las necesidades del cliente o la evolución del mercado. El marco de trabajo está diseñado para adecuarse a las nuevas exigencias que implican proyectos complejos.
- **Reducción del Time to Market.** El cliente puede empezar a utilizar las características más importantes del proyecto antes de que esté completamente terminado.
- **Mayor calidad del software.** El trabajo metódico y la necesidad de obtener una versión de trabajo funcional después de cada iteración, ayuda a la obtención de un software de alta calidad.
- **Mayor productividad.** Se logra, entre otras razones, debido a la eliminación de la burocracia y la motivación del equipo proporcionado por el hecho de que pueden estructurarse de manera autónoma.
- **Maximiza el retorno de la inversión (ROI).** Creación de software solamente con las prestaciones que contribuyen a un mayor valor de negocio gracias a la priorización por retorno de inversión.
- **Predicciones de tiempos.** A través de este marco de trabajo se conoce la velocidad media del equipo por sprint, con lo que es posible estimar de manera fácil cuando se podrá hacer uso de una determinada funcionalidad que todavía está en el Backlog.

- **Reducción de riesgos** El hecho de llevar a cabo las funcionalidades de mayor valor en primer lugar y de saber la velocidad a la que el equipo avanza en el proyecto, permite despejar riesgos efectivamente de manera anticipada.

Daily Scrum o Stand-up meeting

Cada día de un sprint, se realiza la reunión sobre el estado de un proyecto. Esto se llama *daily standup* o *Stand-up meeting*. El scrum tiene unas guías específicas:

- La reunión comienza puntualmente a su hora.
- Todos son bienvenidos, pero sólo los involucrados en el proyecto pueden hablar.
- La reunión tiene una duración fija de 15 minutos, no importa el tamaño del equipo.
- La reunión debe ocurrir en la misma ubicación y a la misma hora todos los días.

Durante la reunión, cada miembro del equipo contesta a tres preguntas:

- ¿Qué has hecho desde ayer?
- ¿Qué es lo que harás para mañana?
- ¿Has tenido algún problema que te haya impedido alcanzar tu objetivo? (Es el papel del ScrumMaster recordar estos impedimentos).

Planificación del Sprint

Al inicio de cada ciclo de Sprint (cada 15 o 30 días), se lleva a cabo una *reunión de planificación del Sprint*. Se pretende:

- Seleccionar qué trabajo se hará.
- Preparar, con el equipo completo, el Sprint Backlog que detalla el tiempo que llevará hacer el trabajo.
- Identificar y comunicar cuánto del trabajo es probable que se realice durante el actual Sprint.
- Realizarse esta planificación en ocho horas como tiempo límite.

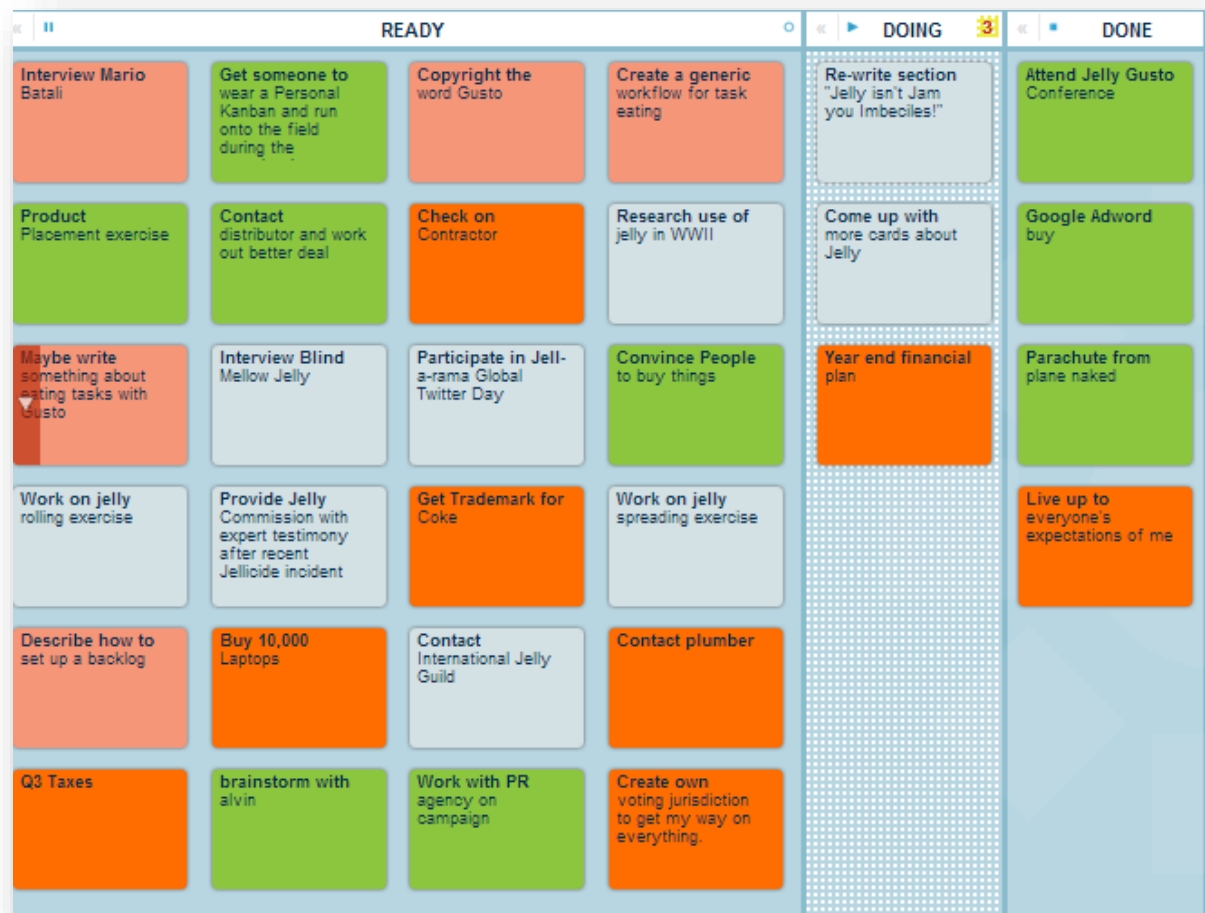
Backlog de la aplicación

El **backlog** de la aplicación se trata como un documento de alto nivel para todo el proyecto. Es el conjunto de todos los requisitos de proyecto, el cual contiene descripciones genéricas de funcionalidades deseables, priorizadas según su retorno sobre la inversión (ROI). Representa el *qué* va a ser construido en su totalidad.


Es abierto y solo puede ser modificado por el *product owner*. Contiene estimaciones realizadas a grandes rasgos, tanto del valor para el negocio, como del esfuerzo de desarrollo requerido.


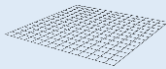
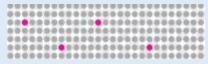











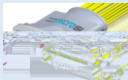
Esta estimación ayuda al *product owner* a ajustar la línea temporal (KEV) y, de manera limitada, la prioridad de las diferentes tareas. Por ejemplo, si dos características tienen el mismo valor de negocio la que requiera menor tiempo de desarrollo tendrá probablemente más prioridad, debido a que su ROI será más alto.





En la siguiente imagen se muestra un backlog modelo de simple diseño:



Coding Guidelines generales

1	ENGLISH. All comments & variables names & classes & docs, etc IN ENGLISH	
2	BEAUTY. Beautiful is better than ugly (indent, make it nice)	<pre>if (x == 3) { for (i = 1; i <= 10; i++)</pre>
3	EXPLICIT. Explicit is better than implicit. (variable names, classes, methods, comments..)	EXPLICIT CONTENT

4	SIMPLICITY. Simple is better than complex.	
5	FLAT. Flat is better than nested	
6	SPARSE. Sparse is better than dense (simplicity of functions, classes, etc)	
7	READABILITY. Readability counts	
8	YOU ARE NOT SPECIAL. Special cases aren't special enough to break the rules.	
9	PRACTICALITY. Practicality beats purity	
10	ERRORS. Errors should never pass silently, unless explicitly silenced.	
11	AMBIGUITY. In the face of ambiguity refuse temptation to guess	
12	QUICK & DIRTY Now is better than never, although never is often better than *right* now.	
13	SIMPLICITY If it is hard to explain, it is a bad idea	
14	SIMPLICITY If it is easy to explain, it may be a good idea.	
15	API. Use extensive documentation for the API for outsourcing including use examples.	
16	STRINGS. NEVER hardcode strings that will be shown to the user. Use resources instead	
17	COHERENCE. If something is the same, CALL IT the same everywhere	
18	DRY is better than WET. You can copy a code 3 times, more than that create a function / procedure to reuse it. DRY (Don't Repeat Yourself) is better than WET (Write Everything Twice, or We Enjoy Typing)	
19	INTERFACES. Interface names and variables SHOULD NOT BE RENAMED unless there is an important reason to do so. If you must do it, tell the group before.	

20	REUSE. Do not reinvent the wheel / Do not repeat yourself Think, comment, search and wait approval before start coding !	
21	YAGNI = You Aren't Gonna Need It. Do not add functionality until it is really necessary	
22	SCALABILITY. Think about scalability. Everything may grow, make it easy to maintain.	
23	KISS = Keep It Simple Stupid !	

Coding guidelines usados para C#

1	As a rule of thumb, write long variable names, classes & methods. (2 words min, 255 chars max). Do not abbreviate. Exception : for loops & matrix scanning it is accepted (i,j,k – m,n)	getFBPostShares
2	Classes. Pascal Naming (start with capital letter), in the middle do capitalize	ClassLikeThis
3	Methods. Camel naming (start with lower case), in the middle do capitalize	methodLikeThis
4	Constants are all capital letters. E.g. NUMBERPI=3.14159 Undeline is accepted to separate words in constants E.g. PI_NUMBER Always use a public static class called Constants (use namespace if needed)	CONSTANTS
5	Write a small description, 1-3 lines per class Write a single line per variable / method	//comment
6	1 class per file, even if it is small (if the name of files grow too much use directories). The file name is EXACTLY as the class name	1 class = 1 file
7	DO NOT USE GLOBAL VARIABLES, as a rule of thumb	GLOBAL VARS
8	Methods. As a rule of thumb, use verb+object of the verb. Ex. getList, writeFile, showDialog. Name should be self-descriptive	verbObject
9	Main. Program.cs should only run the main application, nothing else	Program.cs
10	Exception. Manage exceptions whenever possible. If you cannot manage it, throw it and add it at the method documentation	try { } catch

4. Licencia de la aplicación “Proprietary License Software”

Debido al estado actual de la aplicación, puesto que no se ha realizado ningún estudio sobre si la aplicación seguirá evolucionando o las plataformas que abarcará en un futuro se ha decidido que la licencia será de software propietario “Proprietary License Software”, en la cual el titular del software decidirá en todo momento si será de libre distribución según que plataforma, además el código fuente solo estará disponible para el propietario.

El término de software propietario ha sido creado para designar al antónimo del concepto de software libre, por lo cual en diversos sectores se le han asignado implicaciones políticas relativas al mismo. Para la Fundación para el Software Libre (FSF), este concepto se aplica a cualquier programa informático que no es libre o que sólo lo es parcialmente (semilibre), sea porque su uso, redistribución o modificación está prohibida, o sea porque requiere permiso expreso del titular del software.

La persona física o jurídica (compañía, corporación, fundación, etc.), al poseer los derechos de autor sobre un software, tiene la posibilidad de controlar y restringir los derechos del usuario sobre su programa, lo que en el software no libre implica por lo general que el usuario sólo tendrá derecho a ejecutar el software bajo ciertas condiciones, comúnmente fijadas por el proveedor, que signifique la restricción de una o varias de las cuatro libertades.

Según la opinión de algunos activistas del Movimiento de Software Libre, el término "software propietario" fue introducido por empresas desarrolladoras de software privativo como campaña publicitaria para desacreditar al software libre en cuanto a la propiedad del mismo haciéndola parecer como difusa y sin ninguna garantía de soporte legal para quien lo adquiría. La expresión software privativo comenzó al ser utilizada por Richard Stallman, desde el año 2003, en sus conferencias sobre software libre, pues sería más adecuada que "software propietario".

Por otra parte, en una sociedad de la información, el software se ha convertido en una herramienta importante de productividad, y una licencia de software propietario constituye un acuerdo o contrato entre dos sujetos jurídicos que voluntariamente acuerdan las condiciones de uso de un programa.

5. Desarrollo del proyecto

En este apartado se detallará como ha ido evolucionando la aplicación ordenadamente según como fue desarrollada, así mismo también se explicarán las funciones necesarias, lógica del código y el por qué de según qué clases o de cómo se han programado las diferentes funcionalidades de la aplicación.

Splash Screen

El Splash Screen supone la ventana de presentación de la aplicación, algo que aunque parezca de poca importancia le da al usuario una primera imagen, además de permitir al desarrollador o compañía presentarse.

En concreto un Splash Screen es un elemento de control gráfico que consiste en una ventana con una imagen, en resumen la presentación o logo del software, normalmente el Splash Screen aparecerá cuando la aplicación es lanzada (como es en este caso).

En el caso de los smartphones el Splash Screen de la aplicación cubrirá toda la pantalla, en cambio en las aplicaciones de escritorio se mostrará una imagen de un determinado tamaño en el centro de la pantalla, o en otro caso (como las aplicaciones web) cubriendo el contenido del navegador.

El principal propósito además de la presentación es el notificar que la aplicación está cargándose, acompañados de una barra de carga como feedback, o derivados, una vez la aplicación haya cargado en su totalidad este desaparecerá. Normalmente los Splash Screen tienen un diseño enlazado con el diseño principal de la aplicación o web donde son lanzados.

En el caso de este proyecto el método utilizado para la creación del Splash Screen ha sido mediante la creación de una ventana o formulario, la cual contiene la siguiente imagen cubriendo todo su contenido:



Una vez creado, añadiremos un Timer que llamaremos `timeLifeController` al cual modificaremos su propiedad `Interval` poniendo un valor de 3500 (milisegundos, o lo que es lo mismo, segundos) de esta manera el Timer realizará ticks de tiempo de 3,5 segundos, una vez hecho esto controlaremos el evento de tick en el cual pararemos el Timer, haremos que el `DialogResult` del form devuelva OK y lo cerraremos:

```
public partial class Splash : Form
{
    public Splash()
    {
        InitializeComponent();

        timeLifeController.Enabled = true;
        timeLifeController.Interval = 3500;
        this.pictureBox1.ImageLocation = "../CommonResources/splash.png";
    }

    private void timeLifeController_Tick(object sender, EventArgs e) //this event will be called at the first tick and then this form will die.
    {
        timeLifeController.Stop();
        this.DialogResult = DialogResult.OK;
        this.Close();
    }
}
```

Por último se ha realizado un añadido en el código por defecto que trae la clase `Program.cs` de nuestro proyecto, la cual se encarga de la inicialización de la aplicación, en ella instanciaremos la clase de nuestro Splash Screen y controlaremos mediante un `if` que cuando devuelva un `DialogResult OK` (al realizar el primer tick del timer) la aplicación iniciará con nuestra clase de inicio, llamada `GameChooser.cs`:


```
[STAThread]
static void Main()
{
    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);
    Splash sp = new Splash();
    if (sp.ShowDialog() == DialogResult.OK)
    {
        Application.Run(new GameChooser());
    }
}
```

GameChooser

Como hemos explicado en el apartado anterior la primera ventana que mostrará la aplicación al iniciar después del Splash Screen será el GameChooser, el cual tendrá un diseño bastante simple, debido a que su función será simplemente la de elegir sobre que juego visualizaremos datos de perfil.

Como hemos dicho, al tener una funcionalidad simple no debemos sobrecargar la ventana de controles innecesarios, de esta manera también damos a la ventana una imagen más moderna y atractiva para el usuario, además de amigable.

En el código de la ventana manejaremos los eventos de cada botón para abrir las ventanas correspondientes a cada juego, además del botón de salir de la aplicación el cual cambiará cuando el puntero del ratón pase por encima, ejemplo del código:

```
private void exitButton_MouseUp(object sender, MouseEventArgs e)
{
    DialogResult confirmation = MessageBox.Show("Are you sure you want to exit?", "Warning", MessageBoxButtons.YesNo);
    if (confirmation == DialogResult.Yes)
    {
        Application.Exit();
    }
    else
    {
        exitButton.ImageLocation = "../CommonResources/exitbutton.png";
    }
}

private void diabloButton_Click(object sender, EventArgs e)
{
    this.Hide();
    ProfileForm d3Form = new ProfileForm();
    if (d3Form.ShowDialog() == DialogResult.Cancel)
    {
        this.Show();
    }
}

private void wowButton_Click(object sender, EventArgs e)
{
    this.Hide();
    WowProfileForm wowForm = new WowProfileForm();
    if (wowForm.ShowDialog() == DialogResult.Cancel)
    {
        this.Show();
    }
}

private void exitButton_MouseDown(object sender, MouseEventArgs e)
{
    exitButton.ImageLocation = "../CommonResources/exitbutton2.png";
}
```

ProfileForm (Diablo 3)



Esta ventana será la que la aplicación mostrará si elegimos el juego Diablo 3, de nuevo, esta tendrá un diseño simple en la que tendremos que escribir nuestro BattleTag el cual está compuesto de nuestro Nick y una ID (el textBox controlará que solo se puedan introducir números), además contendrá un comboBox para cambiar el idioma de la aplicación, un botón para ir a la ventana anterior, el botón que limpiará todo el texto que hayamos escrito y el botón Profile el cual nos lanzará la primera ventana con datos de nuestro perfil.

En cuanto a funcionalidad de interfaz estas serían las más importantes, en cuanto a código la importancia de esta ventana recae en que será la que almacene variables importantes con las que formaremos nuestra URI para así comunicarnos con la API como ya hemos explicado en apartados anteriores.

La manera en la que pasaremos estas variables será pasando la clase actual como propiedad a la hora de instanciar la clase del siguiente form, de esta manera la clase DiabloCharForm podrá acceder a estas variables.

```
public String BattleTag { get; set; }
public String Language { get; set; }
private Image btnetBackground;
private Boolean profileFound;

public ProfileForm()
{
    InitializeComponent();
    LanguageCombo.SelectedIndex = 0;
    pictureBox1.ImageLocation = "../CommonResources/battlenet.png";
    pictureBox1.BackColor = System.Drawing.Color.Transparent;
    pictureBox2.ImageLocation = "../EuropeFlags/UnitedKingDom.png";
    pictureBox2.BackColor = System.Drawing.Color.Transparent;
    btnetBackground = new Bitmap("../CommonResources/background2.png");
    this.BackgroundImage = btnetBackground;
    label1.BackColor = System.Drawing.Color.Transparent;
    label1.ForeColor = System.Drawing.Color.LightBlue;
    label2.BackColor = System.Drawing.Color.Transparent;
    label2.ForeColor = System.Drawing.Color.LightBlue;
    label3.BackColor = System.Drawing.Color.Transparent;
    label3.ForeColor = System.Drawing.Color.LightBlue;
}

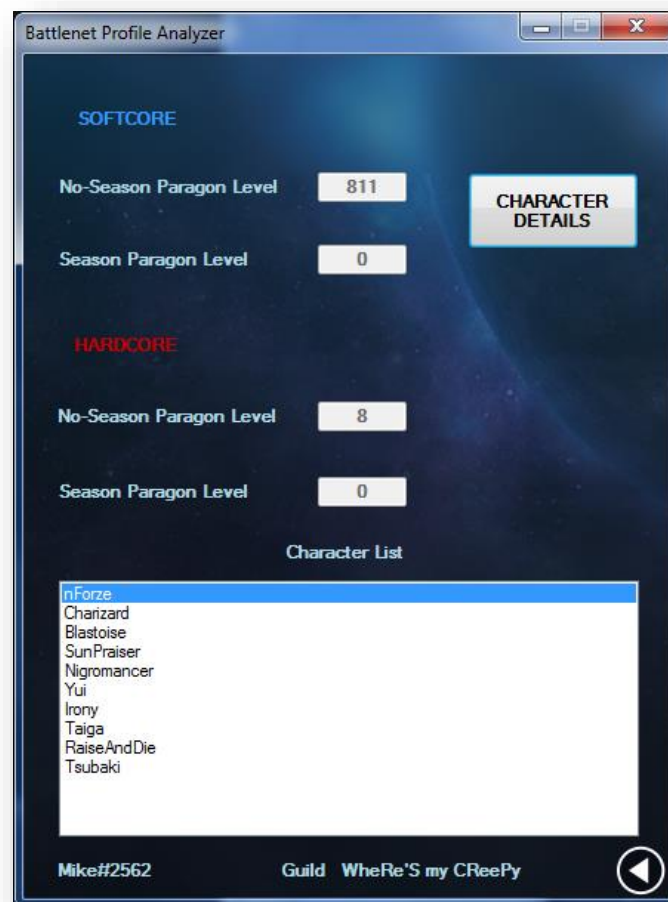
private void button2_Click(object sender, EventArgs e)
{
    NameTextBox.Text = "";
    IDTextBox.Text = "";
}

private void button1_Click(object sender, EventArgs e)
{
    BattleTag = NameTextBox.Text + "-" + IDTextBox.Text;
    switch (LanguageCombo.SelectedIndex)
    {
        case 1:
```

```
this.Hide();
ProfileShower profileShower = new ProfileShower(this);
if (profileShower.ShowDialog() == DialogResult.Cancel)
{
    profileFound = profileShower.profileFound;
    this.Show();

    if (!profileFound)
    {
        MessageBox.Show("Incorrect profile or doesn't exist", "Not Found", MessageBoxButtons.OK);
    }
}
```

ProfileShower (Diablo 3)



Esta será una de las ventanas principales de la aplicación para el juego Diablo 3, como se puede apreciar en la imagen es ahora cuando estaremos mostrando los primeros datos del perfil, en este caso ya nos comunicamos con la API de Blizzard (más adelante explicamos cómo).

Entre las funcionalidades de la interfaz se encuentra un visualizador de personajes, un indicador de los diferentes tipos de niveles que contendrá nuestro perfil, nuestro BattleTag, la hermandad en la que nuestros personajes se encuentran, el botón para volver a la ventana anterior y el botón que mostrará los detalles de cada personaje en una nueva ventana.

En la parte del código, esta clase se encargará de comunicarse con la API, almacenar el contenido obtenido (el cual recibiremos en formato Json) y deserializarlo a XML (en nuestro caso) para después formatear los datos obtenidos y utilizarlos como queramos.

Para ello, formaremos la URI con los datos recogidos de la clase ProfileForm, la cual almacenaremos en startForm desde nuestro constructor de clase. Una vez hecho esto, mediante la clase WebClient nativa de C# almacenaremos en un String todo el contenido que nos devuelva esa URI.

Después crearemos un método en el que deserializaremos todo el contenido del string en un objeto y como return devolveremos el objeto serializado debidamente para volverlo a deserializar como nodo XML, finalmente guardaremos el XML creado (llamado profile.xml) con los nodos creados a partir del objeto deserializado.

```
public ProfileShower(ProfileForm startForm)
{
    InitializeComponent();
    btnetBackground2 = new Bitmap("./CommonResources/background4.png");
    this.BackgroundImage = btnetBackground2;
    this.profileForm = startForm;
    this.battletag = profileForm.BattleTag;
    this.language = profileForm.Language;
    this.masterURL = "https://eu.api.battle.net/d3/profile/" + battletag + "?locale=" + language + "&apikey=dzswy2ebv";
    jsonString = client.DownloadString(masterURL);
    doc = JsonConvert.DeserializeXmlNode(format_json(jsonString), "profile"); //MUST INDICATES THE NAME OF THE ROOT NO
    doc.Save("./profile.xml");

    ds = new DataSet();
    ds.ReadXml(sourceXML);
}
```

Este será el método que crearemos para deserializar/serializar el objeto desde el string recibido:

```
private static string format_json(string json) //This will allow to format a json string properly
{
    dynamic parsedJson = JsonConvert.DeserializeObject(json);
    return JsonConvert.SerializeObject(parsedJson, Newtonsoft.Json.Formatting.Indented);
}
```

Mediante el evento Load de nuestro form lo primero que haremos será controlar si el perfil existe o no, haciendo que la ventana anterior devuelva un mensaje de error.

```
private void ProfileShower_Load(object sender, EventArgs e)
{
    if (jsonString.Contains("NOTFOUND"))
    {
        profileFound = false;
        this.DialogResult = DialogResult.Cancel;
    }
    else
    {
        profileFound = true;
    }
}
```

Para ello, tendremos un boolean a modo de flag el cual obtendrá false si encontramos "NOTFOUND" en el contenido de la URI, el DialogResult de nuestra clase actual pasará a ser Cancel y como hemos mostrado en una imagen anterior la clase ProfileForm dependerá de este flag el cual recogerá para mostrar la ventana de error.

Lo siguiente desarrollado ha sido la manera de enlazar el contenido guardado en formato xml (profile.xml), para el listado de personajes crearemos un DataSource y lo enlazaremos con la propiedad DataSource del listBox que contendrá el listado, después haremos que muestre el nodo "name".

Para el resto de controles, enlazaremos a la propiedad determinada del control (en el caso de un textbox será la propiedad text) con DataBindings indicando que nodo del xml mostrar:

```
//Filling Character List
listBox1.DataSource = ds.Tables["heroes"];
listBox1.DisplayMember = "name";
SNParagon.DataBindings.Add("Text", ds.Tables["profile"], "paragonLevel");
SSParagon.DataBindings.Add("Text", ds.Tables["profile"], "paragonLevelSeason");
HNParagon.DataBindings.Add("Text", ds.Tables["profile"], "paragonLevelHardcore");
HSParagon.DataBindings.Add("Text", ds.Tables["profile"], "paragonLevelSeasonHardcore");
battleTagLabel.DataBindings.Add("Text", ds.Tables["profile"], "battleTag");
guildLabel.DataBindings.Add("Text", ds.Tables["profile"], "guildName");

hideIdContainer.DataSource = ds.Tables["heroes"];
hideIdContainer.DisplayMember = "id";

//we ensure that selectedCharId gets a initial value
selectedCharId = hideIdContainer.GetItemText(hideIdContainer.SelectedItem);
```

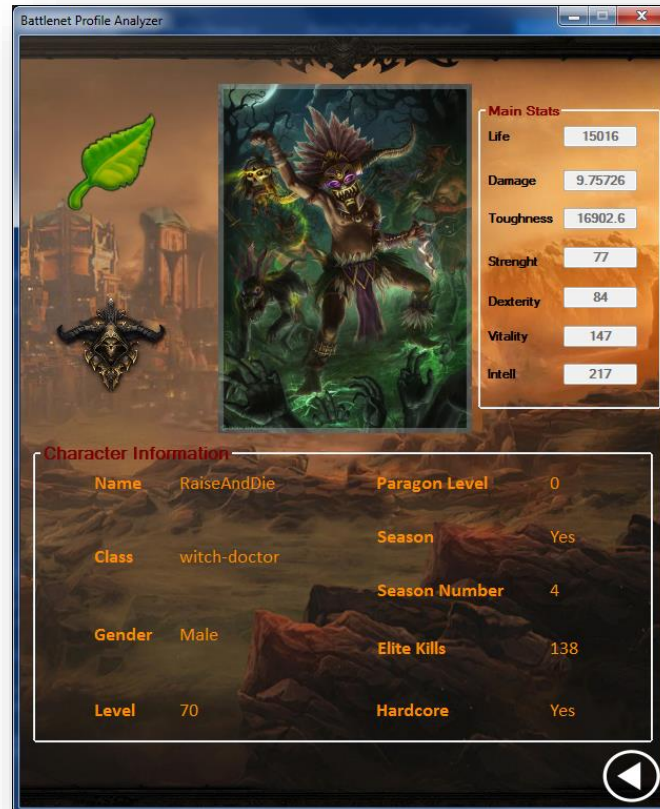
La variable selectedCharId contendrá una ID que usaremos para componer la URI que obtendrá detalles de cada personaje, esta variable obtendrá el valor de un ListBox oculto, que contendrá todas las ID's de cada personaje, el cual estará enlazado con la selección del personaje, por lo que cuando seleccionemos un personaje obtendremos su ID a la vez como vemos en la imagen:

```
private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    //we get the properly char id from getting the item's text from a hide listbox which is binded to the character list listbox (easy way)
    selectedCharId = hideIdContainer.GetItemText(hideIdContainer.SelectedItem);
}
```

DiabloCharForm (Visualizador de personaje)

Finalmente, esta será la ventana que muestre los detalles del personaje, en ella podemos encontrar una imagen principal que corresponderá a la clase del personaje seleccionado, los puntos de estadísticas principales de ese personaje en concreto, la información general del personaje (su nombre, clase, nivel, sexo, cuantos elites ha matado, etc...) y además dos PictureBox que indicarán si el personaje es de temporada o hardcore, por supuesto también encontraremos un botón para volver a la ventana anterior.

La usabilidad de cara al usuario se mantiene en todo momento ya sea en esta ventana o en cualquier otra de la aplicación, en concreto en esta, todos los colores están acordes al tipo de juego, así como los colores de las fuentes que permiten una buena lectura de las mismas, también por supuesto la posición de los controles del Form mantienen una coherencia con respecto a los datos mostrados.



En cuanto a código se refiere la clase `DiabloCharForm` contendrá funciones parecidas a la anterior, pero trataremos de diferente manera los datos, ya que, no podemos presentar en ventana algunos de los datos recogidos desde el XML, por ejemplo el sexo, el cual viene indicado en un dígito que equivaldrá a “Masculino” o “Femenino”, de esta manera es necesario realizar un format sobre el `DataBinding` determinado.

De la misma manera que en la clase anterior almacenaremos la URI en un string que usaremos con el `WebClient`, esta URI estará compuesta por el `BattleTag` almacenado, el `selectedCharId` y el lenguaje que habremos recogido desde la clase `ProfileForm`, una vez hecho esto pasaremos por el mismo proceso de deserialización del objeto `Json` mediante nuestra función y la clase `JsonConvert`:

```
public DiabloCharForm(ProfileShower profileShower)
{
    InitializeComponent();
    this._profileShower = profileShower;
    this.masterURL = "https://eu.api.battle.net/d3/profile/" + _profileShower.battletag + "/hero/" + _profil
    jsonString = client.DownloadString(masterURL);
    doc = JsonConvert.DeserializeXmlNode(formatJson(jsonString), "profile"); //MUST INDICATES THE NAME OF TH
    doc.Save(sourceXML);

    ds = new DataSet();
    ds.ReadXml(sourceXML);

    try
    {
        res_man = new ResourceManager("D3armoryTest.Language.Res", typeof(DiabloCharForm).Assembly);

        switch (_profileShower.language)
        {
            case "es_ES":
                cul = CultureInfo.CreateSpecificCulture("es");
                break;
        }
    }
}
```


El proceso de enlazar los controles del Form con los datos recogidos del Json y deserializados a XML será parecido, solo que, como hemos dicho anteriormente algunos de los datos hará falta formatearlos para que su salida hacia el control esté como nosotros queramos.

Mientras que los controles que contengan el valor directo del XML tendrán su propiedad DataBinding cambiada, para el resto de controles deberemos de crear un DataBinding aparte, el cual como primer parámetro pondremos la propiedad del control que editaremos, después la tabla en el XML donde sacaremos los datos y por último el nodo XML.

Una vez creado el DataBinding modificaremos su propiedad Format para crear un nuevo ConvertEventHandler y finalmente añadirlo al DataBinding del control:

```
private void DiabloCharForm_Load(object sender, EventArgs e)
{
    Binding gender = new Binding("Text", ds.Tables["profile"], "gender");
    gender.Format += new ConvertEventHandler(FormatGender);
    charGender.DataBindings.Add(gender);

    Binding hc = new Binding("Text", ds.Tables["profile"], "hardcore");
    hc.Format += new ConvertEventHandler(FormatHc);
    isCharHc.DataBindings.Add(hc);

    Binding season = new Binding("Text", ds.Tables["profile"], "seasonal");
    season.Format += new ConvertEventHandler(FormatSeason);
    isCharSeason.DataBindings.Add(season);

    switch (charClass.Text)
    {
        case "crusader":
            if (charGender.Text.Equals("Male"))
            {
                charPicture.ImageLocation = ("./DiabloImages/crusader.jpg");
            }
            else
            {
                charPicture.ImageLocation = ("./DiabloImages/crusaderfemale.jpg");
            }
            break;
    }
}
```

Aquí vemos como se ha realizado uno de las funciones del ConvertEventHandler:

```
private void FormatGender(object sender, ConvertEventArgs e)
{
    if (e.Value.ToString().Equals("0"))
    {
        e.Value = "Male";
    }
    else
    {
        e.Value = "Female";
    }
}
```


WowProfileForm (World of Warcraft)

Esta será la primera ventana que la aplicación lance cuando elijamos el juego World of Warcraft en la ventana del GameChooser, en la cual a diferencia de la de Diablo 3 tendremos que escribir el nombre del personaje que queremos analizar además del servidor (Realm) en el que reside, esta ventana también contendrá un ComboBox para la elección del idioma, el botón para limpiar los TextBox, el botón que nos permitirá ir a la ventana anterior y el botón Search que lanzará la visualización del personaje:



En cuanto al código de la clase WowProfileForm lo más importante será la manera en la que almacenamos los datos introducidos por el usuario, los cuales serán necesarios para montar la URI que se comunicará con la API para recoger datos del personaje.

Al igual que en la clase ProfileForm cuando hagamos la instanciación a la clase WowProfileShower (que mostrará el personaje) le pasaremos como parámetro toda la clase actual, de esta manera podremos acceder a los datos introducidos por el usuario (los cuales estarán almacenados en propiedades públicas) para así formar nuestra URI.

```
this.Hide();
WowProfileShower wPS = new WowProfileShower(this);
if (wPS.ShowDialog() == DialogResult.Cancel)
{
    charFound = wPS.charFound;
    this.Show();

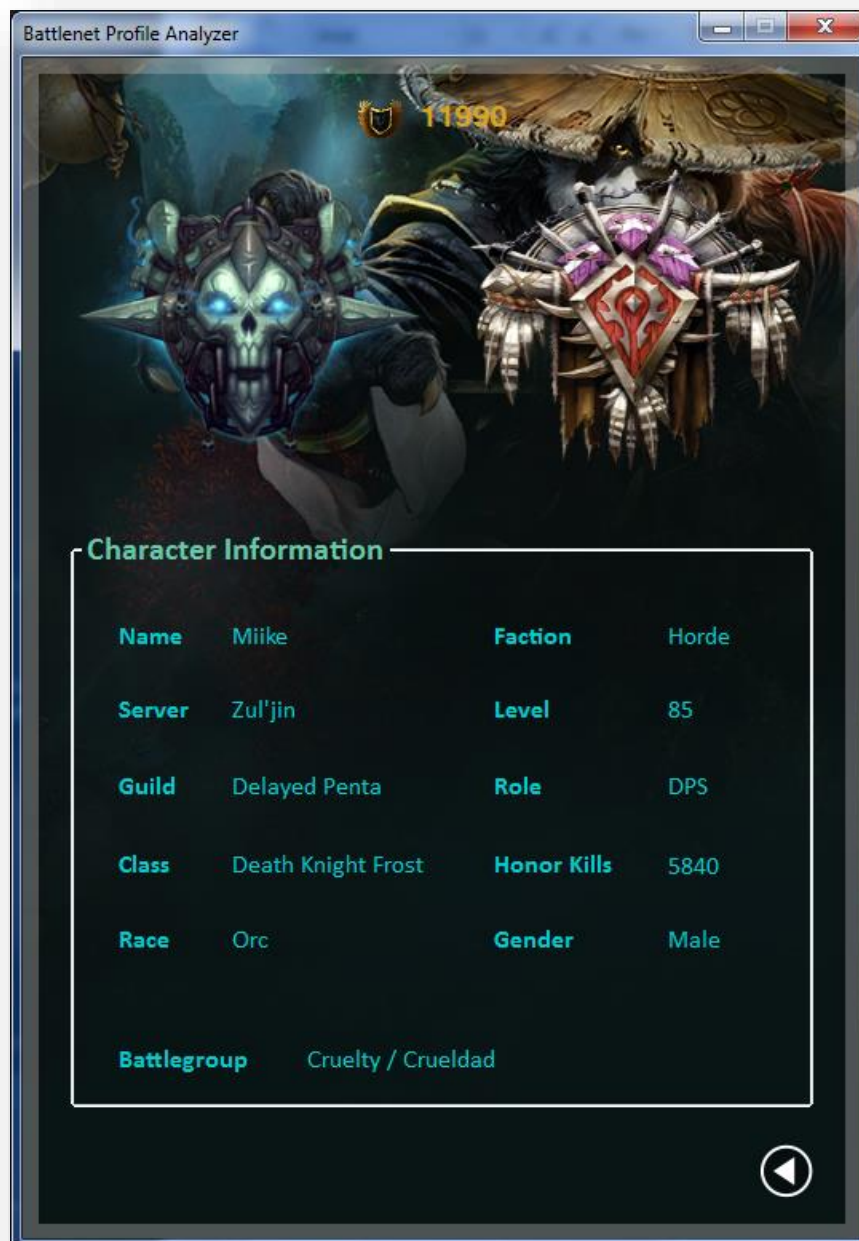
    if (!charFound)
    {
        MessageBox.Show("Character or realm doesn't exist", "Not Found", MessageBoxButtons.OK);
    }
}
}
```

Como se puede observar en la imagen también utilizaremos una variable booleana a modo de Flag para saber si el personaje no existe o es incorrecto, y de esta manera indicárselo al usuario mediante un MessageBox, para ello la variable charFound será true o false dependiendo de si a la hora de deserializar el objeto Json encontramos una excepción o no. Por último, la usabilidad de esta ventana será parecida a la de ProfileForm ya que en ambas se le pide al usuario datos sobre su perfil/personaje.

WowProfileShower (Visualizador de personaje)

Esta será la última ventana de la aplicación, en ella a diferencia del visualizador de personaje en Diablo 3, contendrá dos PictureBox que determinaran la clase y la raza del personaje (mediante datos tratados del XML), un contador de puntos de logros, el botón para volver a la ventana anterior y el apartado en el que estará reunida toda la información del personaje.

En este caso será necesario también tratar algunos de los datos recogidos una vez deserialicemos mediante un Format sobre el nuevo DataBinding que creemos:



La combinación de colores de las fuentes con el fondo permiten una correcta lectura de todo el texto, es importante mantener una buena usabilidad de cara al usuario para que la ventana parezca atractiva y amigable, además permitirá búsquedas rápidas de más personajes volviendo atrás e introduciendo nuevos datos.

En cuanto al código esta vez hemos empleado dos creado dos URI's para comunicarnos con la API ya que necesitaremos sacar datos de diferentes tablas de la BD de Blizzard, al igual que antes, crearemos las URI's con los parámetros recogidos de las diferentes clases desde el constructor de nuestra clase, almacenaremos todo el Json en formato String mediante la clase WebClient nativa de C# y su función DownloadString(), deserializaremos y crearemos el XML serializado:

```
public WowProfileShower(WowProfileForm wPForm)
{
    InitializeComponent();

    this._wPForm = wPForm;
    this.BackgroundImage = new Bitmap("./WowImages/charbackground.jpg");
    this.backButton.ImageLocation = (".CommonResources/backbutton.png");
    this.achievementPicture.ImageLocation = (".WowImages/achievementlogo.png");
    this.masterURL = "https://eu.api.battle.net/wow/character/" + _wPForm.Server + "/" + _wPForm.CharName + "?fields=";
    this.masterURL2 = "https://eu.api.battle.net/wow/character/" + _wPForm.Server + "/" + _wPForm.CharName +

    try
    {
        res_man = new ResourceManager("D3armoryTest.Language.Res", typeof(WowProfileShower).Assembly);

        jsonString = client.DownloadString(masterURL);
        jsonString2 = client.DownloadString(masterURL2);
        charFound = true;

        doc = JsonConvert.DeserializeXmlNode(format_json(jsonString), "profile"); //MUST INDICATES THE NAME OF
        doc.Save("./profile.xml");

        doc = JsonConvert.DeserializeXmlNode(format_json(jsonString2), "profile"); //MUST INDICATES THE NAME OF
        doc.Save("./chardata.xml");

        ds = new DataSet();
        ds.ReadXml(sourceXML);

        ds2 = new DataSet();
        ds2.ReadXml(sourceXML2);
    }
    catch (Exception)
    {
        charFound = false;
        this.DialogResult = DialogResult.Cancel;
    }
}
```

Como hemos dicho anteriormente se han realizado formats sobre DataBindings para dar como salida el texto formateado de una manera determianda. A la hora de acceder a diferentes nodos del XML ocurre que existen nodos con nombres duplicados que contienen datos diferentes, por lo tanto hemos tenido que almacenar todos los nodos a los que queríamos acceder y que tenían el mismo nombre en un XmlNodeList, lo cual convierte a un array de nodos XML, una vez esto hemos accedido al índice necesario y extraído sus datos mediante la propiedad InnerXML a un String para así escoger lo que queríamos mediante la función Split():

```
XmlNodeList allSpecs = doc.SelectNodes("//spec");
String allSpecsToString = allSpecs[0].InnerXml;
roleLabel.Text = allSpecsToString.Split('>', '<')[6];
spec = allSpecsToString.Split('>', '<')[2];
classLabel.Text += " " + spec;
```

Los Formats realizados sobre los DataBindings son parecidos a los hechos en la clase DiabloCharForm.

En este caso hemos tratado la manera de indicar al usuario si el personaje/servidor existe o el servidor es incorrecto de diferente manera, para ello aprovecharemos el TryCatch realizado en la función de deserialización Json para poner como false el flag de nuestra variable booleana charFound, además que hacer que el DialogResult del form sea Cancel, para así volver a la ventana anterior:

```
private string format_json(string json) //This will allow to format a json string properly
{
    try
    {
        dynamic parsedJson = JsonConvert.DeserializeObject(json);
        return JsonConvert.SerializeObject(parsedJson, Newtonsoft.Json.Formatting.Indented);
    }
    catch (Exception)
    {
        charFound = false;
        this.DialogResult = DialogResult.Cancel;
        return "not found";
    }
}
```

Desarrollo del multilinguaje de la aplicación

Existen varias maneras de que la aplicación sea multilinguaje, en nuestro caso hemos optado por utilizar la clase ResourceManager nativa de .NET. Mostraremos como se realizado poniendo de ejemplo la clase WowProfileShower, para ello empezaremos declarando un objeto ResourceManager y CultureInfo, necesitaremos añadir los using System.Globalization; y Sytem.Resources;

Después agregaremos un nuevo archivo de recursos a nuestro proyecto con click derecho > Agregar > Nuevo elemento > Archivo de recursos. En este nuevo fichero de recursos añadiremos todo el texto de cada control que queramos traducir y le asociaremos un nombre. Tendremos que repetir este paso por cada idioma que queramos añadir a nuestra aplicación:

Nombre	Valor	Comentario
BattleGroup	Grupo Batalla	
CharacterDetails	DETALLE DE PERSONAJES	
CharacterInfo	Información del Personaje	
CharacterList	Lista de Personajes	
Class	Clase	
Damage	Daño	
Dexterity	Destreza	
EliteKills	Asesinatos Élite	
Faction	Facción	
Gender	Sexo	
Guild	Herm.	
Hardcore	HARDCORE	
HardCore2	Hardcore	
HNSeasonParagonLevel	No-Temporada Paragon Nivel	
HonorKills	Honor Muertes	
HSeasonParagonLevel	Temporada Paragon Nivel	
Intelligence	Intell	
Level	Nivel	
Life	Vida	
MainStats	Estadísticas Principales	
Name	Nombre	
ParagonLevel	Nivel Paragon	
Race	Raza	
Role	Rol	
Season	Temporada	
SeasonNumber	Temporada Número	
Server	Servidor	

Una vez hecho esto, desde el constructor de nuestra clase instanciaremos el objeto creado ResourceManager pasándole como parámetro donde se encuentran nuestros ficheros de recursos y un typeof de la clase actual en la que nos encontremos:

```
res_man = new ResourceManager("D3armoryTest.Language.Res", typeof(WowProfileShower).Assembly);
```

Después haremos lo mismo con el objeto CultureInfo y su función CreateSpecificCulture() a la cual pasaremos como parámetro el nombre del fichero de cada idioma según corresponda, en nuestro caso lo haremos con un switch dependiendo de lo que recojamos en la propiedad Language de la clase WowProfileForm (recordemos que esta obtendrá el valor del combobox del idioma):

```
switch (_wPForm.Language)
{
    case "es_ES":
        cul = CultureInfo.CreateSpecificCulture("es");
        break;
    case "en_GB":
        cul = CultureInfo.CreateSpecificCulture("en");
        break;
    case "de_DE":
        cul = CultureInfo.CreateSpecificCulture("de");
        break;
    case "fr_FR":
        cul = CultureInfo.CreateSpecificCulture("fr");
        break;
}
```

Una vez que nuestros objetos ResourceManager y CultureInfo están creados tendremos que acceder a los campos en los archivos de recursos para obtener el texto y poder ponerlos en la propiedad Text de nuestros controles, para ello utilizaremos la función GetString de la clase ResourceManager a la cual le pasaremos el identificador del texto que habremos puesto en el archivo de recurso y el objeto CultureInfo para que sepa a qué archivo tiene que acceder según el idioma establecido:

```
this.softcore.Text = res_man.GetString("Softcore", cul);
this.SNSeasonParagonLevel.Text = res_man.GetString("SNSeasonParagonLevel", cul);
this.SSeasonParagonLevel.Text = res_man.GetString("SSeasonParagonLevel", cul);
this.hardcore.Text = res_man.GetString("Hardcore", cul);
this.HNSeasonParagonLevel.Text = res_man.GetString("HNSeasonParagonLevel", cul);
this.HSeasonParagonLevel.Text = res_man.GetString("HSeasonParagonLevel", cul);
this.characterlist.Text = res_man.GetString("CharacterList", cul);
this.guild.Text = res_man.GetString("Guild", cul);
this.characterdetails.Text = res_man.GetString("CharacterDetails", cul);
```

Así pues, podríamos decir que en resumen este ha sido toda la explicación sobre el desarrollo de nuestra aplicación.

6. Evaluación y conclusiones finales

Personalmente la realización de este proyecto ha sido bastante positiva para mí, debido a que nunca había programado comunicación con API's y esto me ha ayudado a aprender cómo usarlas de cara a un futuro y a entender la importancia de estas, puesto que cualquier software que necesite comunicarse con otro software de un diferente desarrollador necesitará usar su API o viceversa. También he necesitado documentarme sobre ellas y la manera de tratarlas.

La mentalidad que he empleado al realizar este proyecto ha sido como si de una aplicación comercial se tratase, por lo que esto me ha ayudado a mejorar a la hora de la estructuración del código, la usabilidad y el uso que el posible usuario puede darle a mi aplicación.

Debido a que ha sido una aplicación desarrollada a partir de 0 y que no ha tenido nada que ver con el centro de trabajo a veces los periodos de tiempo para desarrollarla han sido justos puesto que tenía que compaginarlo con el horario de trabajo, el cual era una jornada laboral normal y corriente.

Todo esto me ha obligado a imponerme fechas límites personales a lo largo de todo el desarrollo para cada parte de la aplicación y personalmente estoy contento de que he podido cumplir todas estas fechas sin problemas aun cuando al ser personales no tienes la presión de fechas límite reales.

Por supuesto ha habido conceptos que he aprendido durante el curso de DAM pero la mayoría he tenido que aprenderlos por mí mismo, desde la comunicación de las API's hasta la realización del multilinguaje, deserialización de Json, etc...

Puesto que el aspecto visual de una aplicación es importantísimo de cara al usuario final he intentado que toda la interfaz sea atractiva y amigable, para ello he tenido que utilizar edición de imagen en prácticamente cualquier diseño de la interfaz, usando Photoshop, lo cual me ha permitido aprender algo sobre diseño gráfico, algo que está muy ligado en el mundo de la programación y que permite no depender de terceros en el desarrollo futuro de otras aplicaciones.

Creo que todo esto es algo que debería tenerse en cuenta a la hora de evaluar este proyecto, puesto que no he requerido de ningún recurso del centro de trabajo ni me he basado en nada de lo desarrollado allí, algo que sin duda es de ayuda para aquellos que si lo hagan, al igual que eso permite que la aplicación pueda contener más complejidad, ya que se trataría de una aplicación desarrollada por una empresa y no por un desarrollador Junior.

Finalmente gracias al desarrollo de este proyecto la mentalidad de desarrollador que tenía ha cambiado, esto hace que tenga más ganas de desarrollar aplicaciones sobre algo en lo que tenga motivación, como ha sido este caso, lo cual creo que es importante, es por eso que a partir de ahora intentaré siempre estar con el desarrollo de algún proyecto, ya sea en conjunto o por mi cuenta, además de seguir mejorando como desarrollador creando nuevas aplicaciones y en diferentes plataformas, ya sea C#, Android o incluso Web.

7. Referencias (bibliografía, revistas, webs, etc.)

Concepto de API y documentación pública de Blizzard:

<http://www.ticbeat.com/tecnologias/que-es-una-api-para-que-sirve/>

https://es.wikipedia.org/wiki/Interfaz_de_programaci%C3%B3n_de_aplicaciones

<https://dev.battle.net/policy>

<http://eu.blizzard.com/es-es/company/careers/>

Deserializador Json por Newtonsoft:

<http://www.newtonsoft.com/json>

Documentación sobre C# y Visual Studio

https://es.wikibooks.org/wiki/C_sharp_NET

<http://si.ua.es/es/documentacion/c-sharp/>

https://es.wikipedia.org/wiki/C_Sharp

SCRUM y planificación

<http://proyectosagiles.org/que-es-scrum/>

[https://en.wikipedia.org/wiki/Scrum_\(software_development\)](https://en.wikipedia.org/wiki/Scrum_(software_development))

8. Agradecimientos

Me gustaría agradecer a mi tutora de proyecto Beatriz Pérez y a mi tutor de las FCT David Soriano, puesto que ambos se han interesado en todo momento por el proyecto, queriendo aconsejarme y preocupándose por las fechas del mismo, lo cual ha permitido en cierta medida que no me “durmiera en los laureles”.