

**Documentación Parte IV**

Univ. Tecnológica Nacional

Facultad Regional Rosario

Cátedra: Proyecto Final

Alumno: Fabricio Montes

Profesora: Daniela Díaz

Fecha: 01/08/2017

**Descripción**: Con objetivo de cumplir los requisitos de la cátedra en cuanto a la presentación de la documentación, se dividió en 4 partes la misma con el objetivo de facilitar la búsqueda de la información y su organización.

Cada una de las partes cuenta con:

- Descripción del documento, objetivos generales y alcance

- Índice

- Desarrollo de los temas cubiertos

**Objetivos y alcance**: El presente documento cubre la documentación completa de la programación del proyecto, como así también la estrategia de pruebas. Son dos documentos unificados, cada uno con su respectivo índice. A continuación se adjuntan los 2 documentos.



**Documentación de programación del proyecto**

Univ. Tecnológica Nacional

Facultad Regional Rosario

Cátedra: Proyecto Final

Alumno: Fabricio Montes

Profesora: Daniela Díaz

Fecha: 01/08/2017

**Tabla de contenidos**

1. Introducción

1.1 Propósito

1.2 Alcance

1.3 Definiciones, acrónimos y abreviaturas

1.4 Referencias

1.5 Información general

2. Organización del código y estilo

3. Comentarios

4. Nombramiento

5. Declaración

6. Expresiones y Declaraciones

7. Gestión de la memoria

8. Control de errores y excepciones

9. Portabilidad

10. Reutilización 11.Compilación 12. Anexo: Resumen de las Directrices

**1. Introducción**

* 1. **Propósito**

El presente documento tiene como propósito servir de guía para todos los involucrados en el desarrollo del aplicativo. Tanto para los desarrolladores que necesiten incorporar evolutivos o resolver incidentes productivos, como para aquellos encargados de realizar pruebas sobre el sitio y que necesiten asegurar la calidad del mismo. El documento busca explicar las características básicas del software realizado como así también los patrones y prácticas utilizados, con la intención de respetar estos estándares a medida que la aplicación evoluciona con el tiempo.

* 1. **Alcance**

Pretende ser una documentación interna para el equipo de desarrollo, y una base en la cual consultar en el futuro, tanto para desarrollo, como para pruebas y mantenimiento de la aplicación.

* 1. **Definiciones, acrónimos y abreviaturas**

|  |  |
| --- | --- |
| **Término** | **Definición** |
| IDE | Entorno de desarrollo integrado |
| VS | Visual Studio. IDE utilizado |
| C# | Lenguaje de programación utilizado |
| Razor | Motor de vistas utilizado para crear las páginas web del sitio |
| BD | Base de datos |
| UI | Interfaz de usuario |

* 1. **Referencias**

[1] Guía para documentar la Programación de un Proyecto de Software v1.1

* 1. **Información general**

El documento presenta en el punto 1 una introducción al documento, explicando objetivos y alcance del mismo. Desde el punto 2, se introducen la estructura del sitio, con vistas gráficas de la solución y de la relación entre los módulos, con trozos de código mostrando declaración de variables, comentarios, nomenclatura, manejo de errores. Al final del documento se hacen consideraciones sobre portabilidad y compilación.

**2. Organización del código y estilo**

La solución cuenta con 4 proyectos:

* API.Services
* ProyectoFinal.Tests
* ProyectoFinal.Utils
* ProyectoFinal

A continuación, se describe cada uno y las relaciones entre ellos.

**API.Services**: Este proyecto contiene clases que se encargan de la integración con los servicios de envío de email y de SMS, de la creación de templates, y del seteo de credenciales para las API utilizadas. El proyecto *ProyectoFinal* tiene una referencia a este proyecto.

Tipo proyecto: Librería de clases. Lenguaje: C#.

**ProyectoFinal.Tests**: Contiene todos los tests unitarios del aplicativo, separado por clases según los controladores que existen.

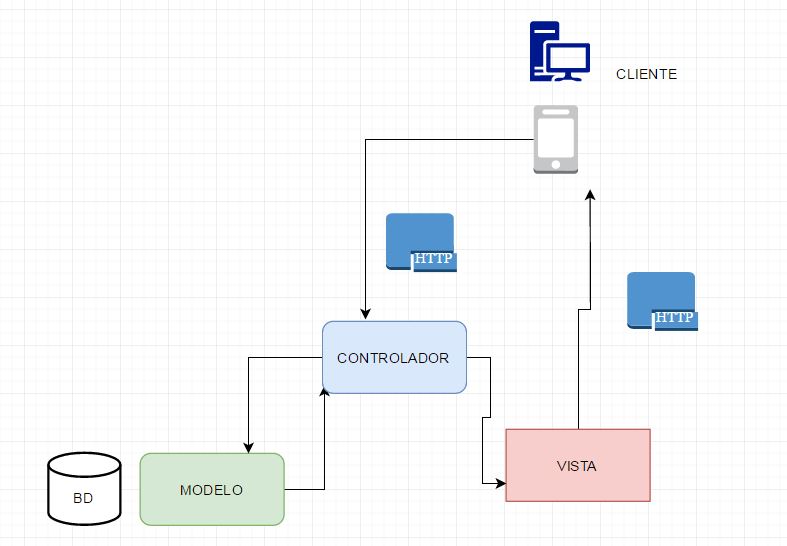
Tipo de proyecto: Prueba unitaria. Lenguaje: C#.

**ProyectoFinal.Utils**: Proyecto que contiene funcionalidades de uso común y que pueden ser fácilmente reutilizables, como por ej. el hasheo de una password y la comparación de una password ingresada con la correspondiente en base, métodos para recorrer listas, enumeraciones, entre otros. El proyecto *ProyectoFinal* tiene una referencia a este proyecto.

Tipo de proyecto: Librería de clases. Lenguaje: C#.

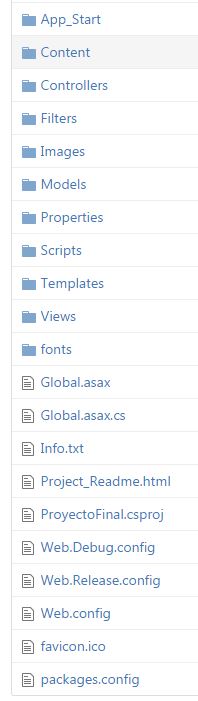
**ProyectoFinal**: Proyecto core de la aplicación. Es un proyecto web escrito en C# utilizando el patrón MVC. Contiene los controladores que aceptan los requests del browser, los modelos que se encargan de recuperar y persistir datos en BD, y las vistas para crear la UI.

Este proyecto contiene la parte principal de la aplicación, y el punto de entrada a la misma. A continuación se detalla un gráfico que muestra como funciona el patrón MVC.



El browser envía un request HTTP que es recibido por un controlador, que es una clase escrita en C# encargada de manejar las peticiones entrantes. Este controlador, consulta al modelo para consultar, editar, guardar, eliminar registros en base, y prepara y devuelve una vista, que es lo que luego se renderizará al cliente mediante una respuesta HTTP.

Estructura de folders en el proyecto *ProyectoFinal*:



Descripción de los contenidos de las carpetas más importantes:

App\_Start: Contiene clases C# encargadas de configuración de manejo de errores, ruteo, minificación de archivos.

Content: Contiene archives .css encargados de la presentación de las vistas.

Controllers: Contiene clases C# que manejan los requests HTTP y son el punto de entrada de la aplicación.

Filters: Contienen clases C# que representan lógica de pre y post procesamiento reutilizables para los controladores.

Images: Contiene recursos de imagen para uso del sitio.

Models: Contiene clases C# que representan las entidades con las que trabaja el sitio. Usa el patrón repositorio para comunicarse contra la BD.

Scripts: Contiene archives .js tales como jQuery.js y bootstrap.js.

Templates: Archivos .html utilizados para enviar emails

Views: Contiene las vistas de la aplicación, representan la interfaz de usuario. Son archivos .cshtml. Razor interpreta código C# en ellas y renderiza código HTML en el navegador.

Web.config: Arhivo XML utilizado para configuraciones como cadena de conexión a la BD, pares de clave/valor configurables, entre otros.

**3. Comentarios**

Comentarios utilizados en código C#:

- De una línea: //comentario

- De múltiples linas: /\*multiples líneas \*/

- De resumen: Buscan brindar información detallada de un método. Al instanciarse el método/clase correspondiente, se brinda esta información.

Ejemplo:

///<summary>

/// Se encarga de procesar un String con un formato definido. Devuelve una

/// instancia de Target con la informacion recuperada

///</summary>

///<param name="entries">String que contiene la data necesaria para armar un

/// Target

///</param>

**4. Nombramiento**

Se utiliza notación húngara para las variables, y notación húngara comenzando con mayúscula para nombres de métodos y clases. Además, se utilizan declarativas using para evitar escribir todo el espacio de nombres de un método / clase.

Ejemplo de variable:

private IClientRepository clientRepository;

Ejemplo de constructor:

publicClientsController()

{

this.clientRepository = new ClientRepository(new GymContext());

}

Ejemplo de método:

publicstaticboolCompare(stringattemptedPass, stringpass, stringsalt)

{

if (string.IsNullOrEmpty(attemptedPass) || string.IsNullOrEmpty(pass) || string.IsNullOrEmpty(salt))

returnfalse;

var attempt = PasswordUtilities.GenerateSHA256Hash(attemptedPass, salt);

return String.Equals(attempt, pass);

}

Ejemplo de declaración de clase:



Código fuente del aplicativo: <https://github.com/cpique/ProyectoFinal>

**5. Declaración**

Las declaraciones de variables respetan la siguiente estructura:

[Ambito] [Tipo Dato] [Nombre Variable] = [Dato asignado];

* Ambito

Visibilidad que tendrá esta variable en el proyecto. C# cuenta con 5 ámbitos distintos:

-Public: Variables visiblesen todas las clasesdel proyecto.

-Private: Variable solo podrá ser accedida desde la misma clase en la que la declaramos.

-Protected: Una variable protected solo sera accesible desde la clase en la que se declaro y desde sus clases heredadas.

-Internal: La variable es accesible para cualquier código dentro del mismo namespace

-Protected Internal: Variable visible en el namespace en el que reside y en las clases externas que sean derivadas.

Los más utilizados en el proyecto, son *private* y *public*.

* TipoDato

Ejemplos son String, int, DateTime, object, float, entre otros.

* Nombre Variable

Identificador de la variable. El proyecto sigue una notación húngara.

* Dato asignado

Valor asignado a la variable. Debe respetar el tipo de la variable.

**6. Expresiones y Declaraciones**

A continuación se brinda trozos de código para ejemplificar expresiones y declaraciones utilizadas:

-Declaración de variable:

var accountSid = Environment.GetEnvironmentVariable("ENVIRONMENT\_VARIABLE\_ACCOUNT\_SID");

-Constructor de una clase:

#region Constructor

publicPaymentsController() { …..}

#endregion

-Instanciación de una clase:

SMS smsService = new SMS();

-Creación de una propiedad en una clase:

publicintActivityID { get; set; }

-Uso de atributos para los controladores:

[AuthorizationPrivilege(Role = "Admin")]

[HandleError()]

publicclassPaymentsController : Controller { ……… }

-Código Razor en las vistas:

@{ViewBag.Title = "About"; }

**7. Gestión de la memoria**

La capa controladora de Windows Azure se encarga de escalar y de manejar la confiabilidad del sistema evitando así que los servicios se detengan si alguno de los servidores de datos de Microsoft tiene problemas y a su vez maneja la información de la aplicación web del usuario. Una de las muchas funciones es manejar los recursos de la memoria o el balanceo del uso de la aplicación.

**8. Control de errores y excepciones**

La aplicación cuenta con uso de sentencias *try catch* con el fin de ‘atrapar’ excepciones e informar los errores producidos. Además cuenta con páginas de error con el objetivo de brindar un feedback al usuario cuando una operación no se realiza de manera satisfactoria.

Se utilizan tanto excepciones propias del framework como excepciones tipadas.

**9. Portabilidad**

.NET Framework utiliza un lenguaje intermedio común a todos los sistemas operativos que luego es convertido a código de máquina cuando es necesario.

.NET Framework y Mono, la versión opensource de .NET Framework, trabajan de la siguiente forma: al compilar un programa escrito, no lo hacen en código máquina, sino en un lenguaje intermedio conocido como CIL (Common Intermediate Language) o MSIL (Microsoft Intermediate Language). Este lenguaje universal, al no ser cercano a la máquina, es independiente de ella y se puede utilizar en cualquier ordenador. Un programa en CIL podrá ser ejecutado en cualquiera de los siguientes sistemas operativos: Windows, GNU/Linux, Mac OS y para móviles con Windows Mobile o iPhone OS (este último gracias a la plataforma Mono), ya que soportan .NET Framework y Mono.

**10. Reutilización**

El proyecto [*ProyectoFinal.Utils*](https://github.com/cpique/ProyectoFinal/tree/master/ProyectoFinal.Utils)fue desarrollado con la intención de reutilizarlo a lo largo del aplicativo.

En los modelos del aplicativo, se utiliza el patrón Repository, con el fin de reutilizar los métodos a la hora de realizar operaciones sobre la BD, y, de requerirse cambios, cambiar solo una parte del código, programando contra una interfaz.

Además, las vistas son dinámicas, trabajan sobre un modelo que reciben, lo que permiten que muestren diferentes datos según el usuario logueado o la condición que se decida necesaria.

**11. Compilación**

*ProyectoFinal* tiene referencias a todos los demás proyectos a excepción del proyecto de tests. VS maneja las dependencias automáticamente y compila en orden los proyectos, compilando primero aquellos que se requieren como referencia en otros.

**12. Anexo: Resumen de las Directrices**

- Se debe utilizar el patrón Repository para creación y manipulación de modelos.

- Se utiliza Code First para cambios en la base de datos.

- Se utiliza GitHub para commitear los cambios en un repositorio y posteriormente Azure para deployar los cambios, una vez se finalice con cualquier módulo

- El desarrollo debe acompañarse continuamente con pruebas unitarias para fomentar la detección de defectos en etapas tempranas.

- Deben respetarse las convenciones del patrón MVC dentro del .NET Framework (convenciones de ubicaciones, de nombres, de rutas).

- No deben exponerse claves, keys o cualquier credencial en el repositorio de la aplicación. Deben manejarse como variables globables desde Azure.

- Debe utilizarse el layout por defecto para las páginas, para mostrar una interfaz consistente.

- El sitio debe ser responsive mediante el uso de Bootstrap.

**Revisiones**

|  |  |  |  |
| --- | --- | --- | --- |
| Fecha | Versión | Descripción | Autor |
| 06/11/2016 | 1.0 | Versión inicial | Cristian Piqué – Fabricio Montes |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |



**Estrategia de Pruebas**

Univ. Tecnológica Nacional

Facultad Regional Rosario

Cátedra: Proyecto Final

Alumno: Fabricio Montes

Profesora: Daniela Díaz

Fecha: 01/08/2017

**ÍNDICE**

**1. Introducción**

**a. Objetivo**

**b. Alcance**

**c. Audiencia**

**d. Terminología del Documento y Acrónimos**

**e. Referencias**

**f. Identificación y Justificación de Pruebas**

**2. Pruebas de casos de uso**

**3. Necesidades o Requerimientos de Entorno**

**a. Sistema Básico de Hardware**

**b. Elementos Básicos de Software en las Pruebas de Entorno**

**c. Configuraciones de Entornos de Prueba**

**d. Responsabilidades, Staff y Necesidades de Capacitación**

**e. Personas y Roles**

**f. Riesgos, Dependencias, Supuestos y Restricciones**

**g. Reporte de Problemas, Temas y Resolución**

**4. Realización de las pruebas de cada uno de los casos de prueba definidos**

**1 Introducción**

**1.a Objetivo**

El objetivo de este documento es recoger los casos de pruebas que verifican que el sistema satisface los requisitos especificados, buscando detectar defectos o problemas para una rápida solución de los mismos. Contiene la definición de los casos de prueba, la estrategia a seguir en la ejecución de las pruebas, los distintos niveles de prueba seleccionados como así también la configuración necesaria para la ejecución de las mismas. La estrategia de pruebas seleccionada permitirá asegurar la calidad del producto.

**1.b Alcance**

Los tipos de prueba que se realizarán y documentarán son:

- Pruebas unitarias: Utilizadas para probar unidades específicas de la aplicación. Son individuales e independientes. Se ejecutan automáticamente y fueron creados utilizando MS UT Framework. En conjunto sirven para probar la funcionalidad de un módulo del aplicativo, como por ej el ABML de Socios.

- Pruebas de integración: Utilizadas para probar la integración de dos o más módulos del aplicativo, y asegurar el correcto funcionamiento de los módulos combinados como grupo. Estas pruebas se realizan manualmente utilizando el aplicativo deployado en Azure.

- Pruebas de regresión: Se evalúa el correcto funcionamiento del aplicativo frente a cambios o evolutivos desarrollados. El propósito de éstas es asegurar que los casos de prueba que ya habían sido probados y fueron exitosos permanezcan así. Se recomienda que este tipo de pruebas sean automatizadas para reducir el tiempo y esfuerzo en su ejecución. Estas pruebas se realizan manualmente utilizando el aplicativo deployado en Azure.

- Prueba de carga: Se simula carga de usuario en el aplicativo sobre un período de tiempo específico, midiendo la respuesta de la aplicación. Muestra que tan rápido se responde a las peticiones de los usuarios (requests) y cuántos de ellos fallan, indicando problemas en la aplicación. Se realizan desde el Portal de Azure, configurando sitio web, número de usuarios concurrentes, y tiempo de duración de la prueba.

**1.c Audiencia**

El documento está dirigido a todos los involucrados en el proceso de pruebas del aplicativo, desde las pruebas unitarias en fase de desarrollo, hasta la prueba de sistema una vez que el aplicativo este deployado, pasando por las pruebas de integración de los módulos, las pruebas de regresión y las pruebas de carga.

**1.d. Terminología del Documento y Acrónimos**

a

|  |  |
| --- | --- |
| ACRONIMO | DEFINICION |
| VS | Visual Studio (IDE) |
| UT | Unit Test |
| AmosGym | Nombre del aplicativo |
| SendGrid | Plataforma para envío de comunicaciones vía email |
| Twilio | Plataforma para envío de comunicaciones vía SMS |
| C# | Lenguaje de programación utilizado para el desarrollo |
| MS UT Framework | Microsoft Unit Test Framework |
| ABML | Alta, baja, modificación, lista |
| MVC | Modelo, vista, controlador. Patrón de desarrollo de arquitectura software. |
| API | Application Programming Interface. Conjunto de funciones que pueden ser utilizadas por otro software |

**1.e. Referencias**

[1]. UTN\_Proyecto Final\_Estrategias de Prueba

**1.f. Identificación y Justificación de Pruebas**

- Pruebas unitarias: Se justifican ya que permiten una rápida prueba de una nueva funcionalidad desarrollada; son rápidas de escribir y ejecutar, y totalmente integradas con VS. Además, el patrón MVC utilizado para el desarrollo alienta y justifica el uso de test unitarios.

- Pruebas de integración: Son utilizados de 2 formas en el aplicativo. En primer lugar, sirven para probar módulos en conjunto y asegurar el correcto funcionamiento de las distintas partes del aplicativo funcionando como un todo. En segudo lugar, se ejecutan para probar la integración con APIs o servicios de terceros, como SendGrid y Twilio, utilizados para enviar comunicaciones masivas a los clientes.

- Pruebas de regresión: Debido al posible cambio de requerimientos, se justifica las pruebas de regresión, para asegurar la retrocompatibilidad de las funciones anteriores con las más nuevas, y asegurar el correcto funcionamiento general de la aplicación a medida que evoluciona.

- Prueba de carga: Permite medir la carga que soporta el aplicativo en el ambiente productivo. Se asigna una carga determinada de usuarios y un tiempo específico de prueba. Se justifica debido a su rápida configuración y la útil información brindada. Este tipo de pruebas están integradas en Azure.

**2. Pruebas de casos de uso**

**2.a Caso de Prueba: Gestionar socios. Tipo Prueba: Caja Negra**

En este caso de prueba podremos comprobar la funcionalidad del caso de uso gestionar socios y verificar que sus resultados son los esperados.

|  |  |
| --- | --- |
| **Objetivos** | Comprobar el funcionamiento de las altas, bajas y modificaciones de socios bajo un conjunto diferente de lotes de prueba, mediante una prueba de caja negra |
| **Técnica** | Seleccionar lotes de prueba y realizar la ejecución del caso de uso |
| **Criterios de éxito** | El criterio de aprobación para el caso de uso es que la solución muestre que se realiza correctamente el alta, modificación o eliminación de un socio. Los registros guardados en BD deben corresponderse con el formato establecido. |
| **Herramientas requeridas** | Sólo el sistema en fase beta. |
| **Consideraciones especiales** | Para poder registrar o modificar un socio correctamente, se deberá completar ciertos campos obligatorios del formulario correspondiente. |

**2.b Caso de Prueba: Prueba de carga. Tipo Prueba: Carga**

En este caso de prueba podremos comprobar la concurrencia de usuarios permitida por el aplicativo y los tiempos de respuesta brindados.

|  |  |
| --- | --- |
| **Objetivos** | Comprobar el tiempo de respuesta del sistema ante un número alto de usuarios concurrentes en el sistema durante un intervalo de tiempo |
| **Técnica** | Se ejecutará una prueba de carga desde Azure |
| **Criterios de éxito** | El criterio de aprobación para el caso de uso es que la aplicación muestre tiempos razonables de respuesta, sin demoras significativas, permitiendo concurrencia a un número de usuarios determinado |
| **Herramientas requeridas** | El sistema en producción |
| **Consideraciones especiales** | Para poder realizar la prueba se necesitan las credenciales para acceder al portal de Azure. |

**2.c Caso de Prueba: Gestionar comunicaciones. Tipo Prueba: Caja Negra**

En este caso de prueba podremos comprobar la integración con los servicios utilizados para enviar comunicaciones vía email y SMS.

|  |  |
| --- | --- |
| **Objetivos** | Comprobar el funcionamiento del envío de emails y de SMS desde la aplicación a los socios del gimnasio. |
| **Técnica** | Seleccionar lotes de prueba y realizar la ejecución |
| **Criterios de éxito** | El criterio de aprobación es que el socio seleccionado reciba un email y/o un SMS, con la información correspondiente a la función de comunicación seleccionada |
| **Herramientas requeridas** | El sistema en producción |
| **Consideraciones especiales** | Para poder realizar el envío de un email o SMS, se debe poseer el rol de administrador del sistema. |

**2.d Caso de Prueba: Gestionar informes. Tipo Prueba: Caja Negra**

En este caso de prueba podremos comprobar la funcionalidad del caso de uso gestionar informes y verificar que sus resultados son los esperados.

|  |  |
| --- | --- |
| **Objetivos** | Comprobar el funcionamiento de los informes del módulo de estadísticas del aplicativo |
| **Técnica** | Seleccionar informes y compararlos contra los registros en la BD. |
| **Criterios de éxito** | El criterio de aprobación para el caso de uso es que la solución muestre reportes con datos conformes a los registros que se encuentren en la BD, que sea consistente y dinámico al cambiar, agregarse o eliminarse datos de la BD. |
| **Herramientas requeridas** | Sólo el sistema en fase beta. |
| **Consideraciones especiales** | Para poder observar los informes se debe poseer el rol de administrador del sistema. |

**2.e Caso de Prueba: Pruebas de integración.**

En este caso de prueba podremos comprobar la funcionalidad del aplicativo general al realizar funcionalidades asociadas a más de un módulo o caso de uso.

|  |  |
| --- | --- |
| **Objetivos** | Comprobar el funcionamiento general de la aplicación mediante la ejecución de varias funcionalidades asociadas a distintos módulos |
| **Técnica** | Seleccionar un rol en particular y realizar un camino a lo largo del aplicativo |
| **Criterios de éxito** | El criterio de aprobación para el caso de uso es que la solución muestre una correcta integración entre los diferentes módulos del sistema. |
| **Herramientas requeridas** | Sólo el sistema en fase beta. |
| **Consideraciones especiales** | La prueba de integración se realiza manualmente, creando registros, navegando el sitio, realizando operaciones, eliminando registros, verificando la consistencia e integridad del aplicativo. |

**3. Necesidades o requerimientos de entorno**

**3.a. Sistema Básico de Hardware**

Pruebas unitarias: Se necesita VS 2015 y el código fuente descargado del proyecto, para poder ejecutar las pruebas unitarias. El mismo se encuentra en un repositorio público de GitHub.

Pruebas de integración: Se requiere acceso al sitio web y credenciales para acceder a las páginas que requieren autenticación.

Pruebas de regresión: Se requieren los accesos de pruebas unitarias y de integración mencionados anteriormente

Pruebas de carga: Se requieren credenciales para acceder al portal de Azure, donde está alojado el aplicativo.

**3.b. Elementos Básicos de Software en las Pruebas de Entorno**

Más allá de las herramientas mencionadas en el punto anterior, se necesita tener instalado SQL Server y SQL Server Managment Studio 2005 o superior.

**3.c. Configuraciones de Entornos de Prueba**

El entorno de prueba será el entorno local para las pruebas unitarias y de regresión, y el entorno productivo para las pruebas de carga y de integración. Se brindarán las credenciales, código fuente o IDE en el caso que corresponda.

**3.d. Responsabilidades, Staff y Necesidades de Capacitación**

Los encargados de realizar las pruebas serán miembros del equipo de desarrollo a designar. Deben contar con conocimientos tanto de la aplicación como de las distintas pruebas seleccionadas y los resultados esperados de cada una.

**3.e. Personas y Roles**

Mencioado en punto 3.d

**3.f. Riesgos, Dependencias, Supuestos y Restricciones**

-

**3.g. Reporte de Problemas, Temas y Resolución**

Los problemas o defectos encontrados se anotarán en una tabla, indicando descripción del defecto, prueba realizada, caso de uso que fallo, y posible solución. Además, se informará a los desarrolladores con el objetivo de atacar el problema encontrado.

**4. Realización de las pruebas de cada uno de los casos de prueba definidos**

* **Caso de Prueba: 2.a Gestionar socios. Tipo Prueba: Caja Negra**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Caso de prueba** | **Acción realizada** | **Criterio de éxito** | **Salida resultante** | **Acciones correctivas** |
| 2.a Gestionar socios | Registrar socio mediante formulario de alta | Nuevo registro en BD | Form completado con éxito y socio registrado | Se corrigieron validaciones sobre el formulario |
| 2.a Gestionar socios | Editar socio | Registro modificado en BD | Cambios de datos en el formulario de edición de socio | Se corrigió formato de fecha que se visualiza en el form |
| 2.a Gestionar socios | Búsqueda de socios en grilla | Se filtran socios por la palabra buscada | Socios filtrados | - |
| 2.a Gestionar socios | Ordenamiento de socios en grilla | Se ordenan los socios según criterio de ordenamiento | Los socios quedan ordenados en forma desc/asc | - |
| 2.a Gestionar socios | Eliminar socio | Registro eliminado de BD | Form completado y socio eliminado de grilla | - |
| 2.a Gestionar socios | Paginado | Se accede a las distintas páginas de la grilla | Se visualiza la página correspondiente con sus registros | - |
|  |  |  |  |  |

* **Caso de prueba: 2.b Prueba de carga**

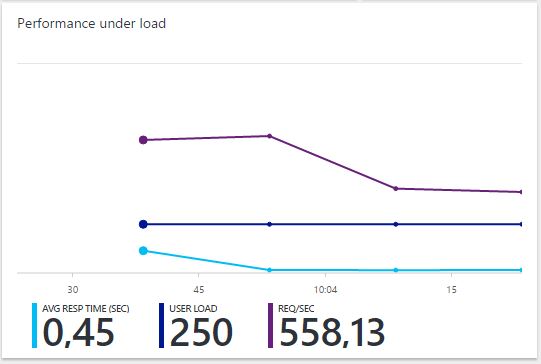
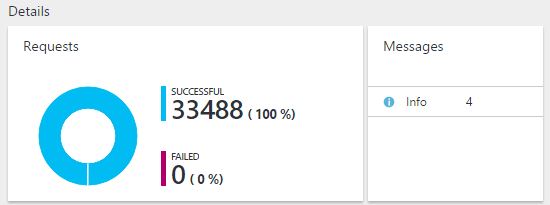
Se realizó desde el portal Azure, con la siguiente configuración:

Usuarios concurrentes: 40

Tiempo de ejecución: 1 min

(Mantiene peticiones de 40 usuarios concurrentes durante 1 minuto)

Como se ve en los gráficos, los requests fueron todos exitosos y el tiempo de respuesta del aplicativo aceptable.



* **Caso de Prueba: 2.c Gestionar comunicaciones. Tipo Prueba: Caja Negra**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Caso de prueba** | **Acción realizada** | **Criterio de éxito** | **Salida resultante** | **Acciones correctivas** |
| 2.a Gestionar comunicaciones | Registrar socio mediante formulario de alta y verificar la recepción de email de bienvenida | Socio recibe email de bienvenida en casilla de correo | Form completado con éxito. Socio recibe email de bienvenida en casilla de correo |  |
| 2.a Gestionar comunicaciones | Envío de email masivo a socios de una actividad | Socios de una actividad específica reciben email con información | Pantalla de éxito | Se corrigieron bugs cuando se hace un envío masivo (a más de un destinatario) |
| 2.a Gestionar comunicaciones | Se verifican los links contenidos en el email | Los links deben apuntar correctamente a páginas del sitio | Los links apuntan correctamente a páginas del sitio | - |
|  |  |  |  |  |

* **Caso de Prueba: 2.d Gestionar informes. Tipo Prueba: Caja Negra**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Caso de prueba** | **Acción realizada** | **Criterio de éxito** | **Salida resultante** | **Acciones correctivas** |
| 2.d Gestionar informes | Evaluación de gráfico Bar Chart | Se evalúa que los datos reflejan los registros de BD en númeroy porcentaje | Los datos son consistentes con los registros en BD |  |
| 2.d Gestionar informes | Evaluación de gráfico Line Chart | Se evalúa que los datos reflejan los registros de BD en númeroy porcentaje | Los datos son consistentes con los registros en BD |  |
| 2.d Gestionar informes | Evaluación de gráfico Pie Chart | Se evalúa que los datos reflejan los registros de BD en númeroy porcentaje | Los datos son consistentes con los registros en BD |  |
|  |  |  |  |  |

* **Caso de Prueba: 2.e Pruebas de integración.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Caso de prueba** | **Acción realizada** | **Criterio de éxito** | **Salida resultante** | **Acciones correctivas** |
| 2.e Pruebas de integración | Logueo de usuario. Edición de datos personales. Consulta de catálogos de actividades y abonos | Correcta navegación sobre las páginas. El usuario se loguea, edita datos y ve el listado de catálogos correctamente | Usuario pudo navegar y realizar las acciones mencionadas correctamente |  |
| 2.e Pruebas de integración | Admin inicia sesión, registra socios, registra actividades, registra abonos, consulta módulo de estadística | Admin realiza las operaciones satisfactoriamente sin pantallas o mensajes de error. BD debe contener los nuevos registros. | Admin realiza las operaciones satisfactoriamente sin pantallas o mensajes de error. BD contiene los nuevos registros creados. |  |
| 2.e Pruebas de integración | Usuario ingresa número de documento al arribar al gimnasio | Se informa los abonos disponibles y el estado de los mismos | Sistema informa sólo el abono más reciente y su estado | Se incorporó lógica para manejar múltiples abonos |
|  |  |  |  |  |

**Revisiones**

|  |  |  |  |
| --- | --- | --- | --- |
| Fecha | Versión | Descripción | Autor |
| 04/11/2016 | 1.0 | Versión inicial | Cristian Piqué-Fabricio Montes |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |