Triage, Sistema de gestión para sala de Guardia Hospitalaria

Néstor Muñoz
Marcia Tejeda

Director: Nicolás Paez
Co-Director: Pablo E. Martínez López

6 de noviembre de 2014

Resumen

El presente Trabajo de Inserción Profesional (TIP) fue realizado en el contexto del desarrollo de un sistema para resolver necesidades de la guardia del Hospital Oñativia en Rafael Calzada.
El problema presentado en este TIP es automatizar la recepción en la guardia del Hospital utilizando el método de emergencias conocido como Triage.

La solución propuesta es el desarrollo de una aplicación web que sea accesible desde todos los puntos de atención de las guardias y que permita evaluar los casos que se presentan de manera eficiente y con los mismos parámetros.

Índice

1. Introducción

Actualmente la guardia del H.Z.G.A "Dr. Arturo Oñativia" de la localidad de Rafael Calzada, a cargo del Doctor Luis Reggiani, utiliza el método Triage [1, 2] para la clasificación de pacientes según los síntomas que presenten. Triage es un método de la medicina de emergencias y desastres para la selección y clasificación de los pacientes basándose en las prioridades de atención, privilegiando la posibilidad de supervivencia, de acuerdo a las necesidades terapéuticas y los recursos disponibles. Trata de evitar el agravamiento del diagnóstico del paciente a causa de demora en la atención. Un nivel que implique que el paciente puede ser demorado no quiere decir que el diagnóstico final no pueda ser una enfermedad grave, ya que un cáncer, por ejemplo, puede tener funciones vitales estables que no lleve a ser visto con premura. El triage prioriza el compromiso vital inmediato y las posibles complicaciones.

Hasta el momento todo el proceso de Triage en el hospital se hace en forma manual, lo que implica algunos contratiempos:

- Depender de una persona (o varias) con todo el conocimiento.
- Emplear demasiado tiempo para guardar datos y recolectarlos.
- Obtener diferentes resultados (algunas veces incorrectos), pues diferentes personas usan en ocasiones criterios diferentes para la toma de decisiones.

Según el Dr. Reggiani informatizar el proceso de Triage implicaría una mejora notable en el desempeño de la guardia. Se lograría una estandarización en la clasificación de síntomas, se agilizaría el ingreso y la obtención de datos de pacientes, se mejoraría la atención en general y se distinguirían de una manera más eficaz aquellos pacientes que necesiten una atención inmediata.

En este trabajo proponemos desarrollar y poner en funcionamiento un sistema informático que dé soporte al proceso de Triage en la guardia del H.Z.G.A "Dr. Arturo Oñativia" de la localidad de Rafael Calzada. Dado que el sistema podría incluir muchísimas funcionalidades y al mismo tiempo existe una especificación detallada de los requerimientos, planteamos el proyecto con alcance variable con el compromiso de entrega de un software que resuelva la parte central del proceso de Triage. La idea es que el sistema desarrollado en el contexto de este trabajo sea puesto en marcha y utilizado por la institución promotora.

Dado el contexto en el cual debemos realizar el proyecto, consideramos que lo más apropiado es el uso de una metodología ágil [3]. En este sentido trabajamos con iteraciones de tiempo fijo de una semana de duración y cada incremento del sistema es validado por el Dr. Reggiani quien ocupa simultáneamente los roles de responsable de producto y especialista de negocio.

Concretamente el sistema debe cubrir las siguientes funcionalidades mínimas:

- Recepción de pacientes mediante búsqueda de aquellos que ya fueron atendidos en el hospital e ingreso de los que se atienden por primera vez.
- Toma de impresión visual inicial del paciente.
- Toma de los signos vitales que presenta el paciente: presión arterial (sístole y diástole), frecuencia cardíaca, saturación de O2, frecuencia respiratoria, temperatura y glucosa.

- Ingreso de los síntomas que presenta el paciente.
- División de los síntomas por categorías (discriminantes) y asociación de prioridades a los mismos.
- Lógica variada para los síntomas, según se trate de un paciente adulto o pediátrico, tanto para los valores de los signos vitales como para las prioridades de los síntomas.
- Emisión de alerta al momento de detectarse un síntoma de prioridad uno, para que se ingrese al paciente de inmediato al shock room.
- Posibilidad de extraer reportes de cantidad de consultas realizadas según prioridad y promedio de tiempo de espera de atención según prioridad.
- Puesta en funcionamiento en cada sala de recepción de pacientes de guardia.

Este informe cuenta nuestro trabajo en el desarrollo, implementación y puesta en funcionamiento de una aplicación web que cubre todas las funcionalidades mínimas detalladas anteriormente y además realiza las siguientes:

- Generación de reportes por paciente a modo de historial de atenciones en guardia con detalle de fecha, síntomas presentados, signos vitales, prioridad asignada y tipo de atención recibida.
- Diferenciación entre usuarios administradores del sistema y usuarios comunes.
- Posibilidad de detallar el tipo de atención recibida por el paciente luego de pasar por el proceso de Triage
- Alta, baja y modificación de pacientes, síntomas, discriminantes de síntomas y usuarios del sistema.

1.1. Sumario

En la sección 2 de este documento hablaremos del planteo del problema: cómo es la dinámica de trabajo en el hospital, cuáles son los requerimientos del cliente y cómo proponemos que sea la aplicación resultante de este trabajo. Luego, en la sección 3 describiremos todas las herramientas tecnológicas utilizadas para el desarrollo así como las metodologías de trabajo implementadas y ampliaremos los detalles de diseño e implementación. En la sección 4, detallaremos las pruebas realizadas y explicaremos cómo procedimos a la instalación del sistema. Para finalizar, en la sección 5 presentaremos las conclusiones.

2. Planteo

Comenzamos el presente trabajo con el relevamiento de información mediante reuniones con el Dr. Reggiani, con quien tuvimos contacto constante durante todo el desarrollo. Las primeras reuniones fueron para describir el problema y las necesidades reales. Luego, durante la etapa de desarrollo, cada pantalla y

funcionalidad fue validada por el usuario, con el propósito de llegar a un producto que fuera útil y consistente. A medida que avanzamos con el producto, se fue negociando el alcance agregando o quitando funcionalidades dependiendo del tiempo disponible y consultando con el Dr. la prioridad para cada tarea.

En esta sección iremos presentando los resultados de estos encuentros.

2.1. Dinámica de trabajo en el hospital

La guardia del H.Z.G.A "Dr. Arturo Oñativia" opera recibiendo a los pacientes en dos sectores: Pediatría y Adultos. Cada sector tiene definidos sus parámetros de evaluación de pacientes, pudiendo un síntoma tener una prioridad para los adultos y otra para los pacientes pediátricos. Hay una división entre los pacientes pediátricos también dependiendo de la edad, diferenciando bebés de meses y niños más grandes.

Al llegar a la guardia los pacientes son recibidos por el enfermero de guardia, quién utilizando el sistema desarrollado toma una impresión visual del paciente. Luego se pasa a la sala de toma de signos vitales donde otro enfermero controla la presión, glucosa en sangre, entre otros, y graba en el sistema los síntomas que presenta el paciente. En caso de encontrar algún síntoma de prioridad UNO (peligro de muerte o daño permanente) en alguna de las tres instancias mencionadas antes (Impresión Visual, Signos Vitales o Síntomas), el sistema deriva al paciente de inmediato a la sala de Shock.

Una vez conocida la prioridad del paciente ingresado, hay tres caminos:

- Atención Inmediata.
- Atención dentro de los próximos 30 minutos.
- Atención en Consultorios Externos.

Una vez atendido el paciente, se termina el ciclo dentro del sistema; esto es, el sistema no guarda información post-triage.

2.2. Requerimientos del cliente

La característica más mencionada por el Dr. fue, en las primeras entrevistas, registrar adecuadamente a los pacientes que ingresan. Para ello, decidimos guardar todos los datos de los pacientes (tal como Nombre, Apellido, Teléfono, DNI, Dirección, entre otros) para poder contar con una base de datos de todas las personas atendidas en caso de necesitarla.

Otra de las prioridades que detectamos fue la necesidad de completar el Triage de forma eficiente y con una respuesta rápida ante casos de urgencias. El cliente solicitó de manera excluyente que el sistema debía cortar cualquier interacción en el momento de detectar un caso de Prioridad UNO, para poder actuar con el apremio necesario.

Detallamos en secciones futuras los reportes que el cliente requerió.

2.3. Pantallas y dinámica de uso

Las pantallas, como ya mecionamos, se fueron diseñando y validando con el cliente en las reuniones periódicas.

La pantalla inicial (y principal) del sistema permite buscar a los pacientes por nombre, apellido, DNI o fecha de nacimiento. En el caso de que ya hayan sido atendidos en algún momento en la guardia, serán encontrados por el buscador y se podrá proceder a completar los datos del Triage. En caso de no encontrarlos, la misma pantalla permite ingresarlos al sistema en el momento generando un nuevo registro de una persona.

La pantalla de Triage está divida en tres: Impresión Visual, Síntomas y Signos Vitales. Tiene una navegación definida por pestañas que permite navegar de forma fluida entre los tres formularios. Una vez que se cargan los datos deseados, el paciente pasa a una "Lista de espera", otra pantalla que permite ver qué pacientes están esperando atención. Permite también continuar el Triage; esto es, ingresar nuevamente a la pantalla de Triage y poder modificar o cargar nuevos síntomas. Esta característica es necesaria cuando la persona que toma la Impresión Visual está en un lugar físico distinto al del enfermero que toma los signos vitales, por ejemplo.

En el caso de que el paciente haya sido atendido, o derivado a consultorio externo, y se retire del hospital, el enfermero, o administrativo, ubicado en el puesto de salida debe buscar al paciente en la lista de espera y marcar la finalización de la atención con alguna de las opciones mencionadas anteriormente: Atención Inmediata, Atención dentro de los próximos 30 minutos o Atención en consultorios externos.

Entre las pantallas administrativas, o de configuración, se encuentran las de Alta y Modificación de:

- Síntomas
- Discriminantes
- Usuarios

permitiendo crear nuevos registros o modificar los existentes. En el caso de los usuarios, es posible también dar la baja.

2.4. Informes

El cliente pidió pantallas con los informes detallados a continuación.

Tiempo de espera para cada prioridad
: Reporte que muestra el tiempo de espera medio para cada prioridad en un rango de tiempo dado por dos fechas.

Cantidad de atenciones para cada prioridad
: Reporte que muestra la cantidad de atenciones para cada prioridad en un rango de tiempo dado por dos fechas.

Reporte de Personas
: Lista con todas las personas que se atendieron. Permite ver individualmente los datos de cada persona y una lista que muestra todas las veces que fue atendida, los síntomas presentados y el tipo de atención recibida.

3. Metodología de trabajo e implementación

En esta sección en primer lugar hablamos sobre la metodología de trabajo que utilizamos durante todo el proyecto. En segundo lugar describimos las tecnologías utilizadas para el desarrollo de cada una de las partes de la aplicación: el *front-end* (interfaz de usuario), el *back-end* (lógica del negocio e interacción con la base de datos) y la base de datos. Finalmente describimos las arquitecturas que utilizamos.

3.1. Metodología de trabajo

Decidimos darle al desarrollo un enfoque ágil[3]. Así dar visibilidad constante a todos los interesados fue uno de los principios transversales a todo el proyecto. La comunicación fue muy fluida, tanto por mail, como a través de reuniones presenciales o virtuales (en forma remota). Otro de los pilares del enfoque ágil fue trabajar en forma iterativa e incremental. Es decir que trabajamos con iteraciones de tiempo fijo de una semana de duración. Al final de cada iteración los avances eran validados por el Dr. Reggiani.

3.1.1. Resumen del itinerario del proyecto

Lo primero que hicimos fue varias reuniones entre todos los interesados en el proyecto: los desarrolladores, los directores y el Dr. Reggiani. De esas reuniones y de una visita al hospital obtuvimos los requerimientos.

El segundo paso fue la elección de las tecnologías. Entre el basto abanico de posibilidades NodeJS[1] y Grails[2] aparecían como las predilectas ya que son tecnologías modernas y teníamos buenas referencias de ambas. Para decidirnos por una de las dos desarrollamos un conversor web muy simple de Farenheit a Celcius. A partir de esa primer experiencia finalmente optamos por usar Grails ya que se asemeja más que NodeJS a las tecnologías que veníamos utilizando en las distintas materias a lo largo de la carrera.

Si bien Grails es *full stack*[3], decidimos utilizar una tecnología que resuelva las vistas del lado del cliente, se comunique con el *back-end* mediante una interfaz REST[4] y sea *responsive*[5]. Así surgió la idea de utilizar AngularJS[6] que cubre ampliamente todos esos requisitos.

En tercer lugar hicimos una estimación relativa a grandes rasgos. Allí calculamos cuanto tiempo nos iba a demandar cada funcionalidad requerida y la fecha de cierre del proyecto. Luego hicimos una planificación en donde ordenamos los requerimientos dentro de las iteraciones según las prioridades del Dr. Reggiani.

A partir de ahí comenzamos con el desarrollo recorriendo las iteraciones planificadas. Al promediar el proyecto hicimos una instalación de prueba en el hospital. Y al finalizar el mismo hicimos la instalación definitiva del producto terminado en una máquina de dicha institución.

[1] http://nodejs.org/

[2] https://grails.org/

[3] Se dice que una tecnología es *full stack* cuando posee todas las herramientas necesarias para desarrollar una aplicación

[4] http://es.wikipedia.org/wiki/Representational_State_Transfer

[5] http://es.wikipedia.org/wiki/Diseño_web_adaptable

[6] https://angularjs.org/

Sprint 1		Hs Estimadas	Hs Insumidas
	Configurar el servidor de integración continua	5	4
	Configurar Heroku	5	10
	Configurar el controlador de versiones	5	6
	Crear los prototipos de pantallas.	4	4
	Como enfermero (o administrativo) quiero poder crear un nuevo paciente.	14	18
	Testing		
	Total hs:	33	42
Sprint 2			
	Como enfermo (o administrativo) quiero poder buscar un paciente por sus datos personales, y, en el caso de encontrarlo, seleccionarlo para completar el proceso. En el caso de no encontrarlo, tener un camino para crear un nuevo paciente.	20	26
	Configurar Postgres	5	8
	Total hs:	25	34
Sprint 4			
	Como enfermero (o administrativo) quiero poder ingresar los síntomas del paciente a través de la impresión visual (pre-triage).	19	22
	Como enfermero (o administrativo) quiero que el sistema frene el ciclo de ingreso de datos cuando se detecta un síntoma de prioridad 1.	4	4
	Total hs:	23	26
Sprint 5			
	Como enfermero quiero poder ingresar los síntomas que presenta el paciente al momento del triage.	22	17
	Total hs:	22	17

Figura 1: Planificación

3.1.2. Flujo de trabajo en una iteración

Al inicio de cada iteración estimabamos cuanto tiempo nos iba a llevar cada tarea y enviabamos un email con los detalles sobre lo que ibamos a hacer durante esa semana. También haciamos prototipos de las pantallas a realizar que eran validados por el Dr. Reggiani. Además dejamos sentado en una hoja de cálculo los detalles de cada tarea: el tiempo de realización estimado, la fecha de realización y el tiempo real insumido. Por otro lado cada día que avanzabamos con alguna tarea enviabamos un reporte por email informando lo que habíamos hecho y si había surgido algún contratiempo. Por último, al finalizar la iteración, enviabamos otro email con los detalles de lo realizado.

3.1.3. Herramientas que utilizamos

Para la comunicación via email creamos un grupo en Google Groups[7]. Para toda la documentación compartida utilizamos Google Drive[8]. Para hacer reuniones remotas utilizamos Skype[9] y Google Hangouts[10]. Utilizamos Balsamiq[11] para hacer los prototipos de pantallas. Utilizamos Git[12] y Github[13] para versionar el código. Y utilizamos Travis[14] como servidor de integración continua[15].

[7] https://groups.google.com
[8] https://drive.google.com/
[9] http://www.skype.com.ar/es/
[10] https://plus.google.com/hangouts
[11] https://balsamiq.com/
[12] http://git-scm.com/
[13] https://github.com/
[14] https://travis-ci.org/
[15] http://es.wikipedia.org/wiki/Integración_continua

Tarjeta	Tarea	La realizó	fecha	Hs insumidas	Hs estimadas
Como usuario del sistema, quiero poder loguearme y desloguearme.	Crear la vista del login	Nestor	28/08/2014	1	3
	Crear el controlador del log	Nestor	28/08/2014	5	5
	Crear pantalla de cambio de contraseña	Nestor	12/11/2014	2	2
	Crear tests funcionales	Nestor	17/09/2014	3	2
	Escribir casos de prueba	Nestor	28/08/2014	1	1
Como enfermero (o administrativo) quiero poder crear un nuevo paciente.	Crear el controlador de paciente	Nestor	13/02/2014	3	3
	Crear la clase de dominio paciente	Nestor	7/2/2014	1	2
	Crear la vista del alta de paciente	Marcia	7/2/2014	7	3
	Crear la clase de dominio persona	Nestor	7/2/2014	1	2
	Crear el controlador de persona	Nestor	13/02/2014	3	3
	Crear el controlador Angular de Persona	Marcia	8/2/2014	7	4
	Crear tests unitarios	Nestor	18/02/2014	4	1
	Crear tests funcionales	Marcia	16/2/2014	6	2
	Crear casos de prueba del alta de paciente	Marcia	13/2/2014	2	1
	Crear validaciones para el alta de paciente	Marcia	25/2/2014	12	2
Como enfermo (o administrativo) quiero poder buscar un paciente por sus datos personales, y, en el caso de encontrarlo, seleccionarlo para completar el proceso. En el caso de no encontrarlo, tener un camino para crear un nuevo paciente.	Crear la vista de la búsqueda	Nestor	20/02/2014	3	3
	Crear API REST de la búsqueda	Marcia	19/02/2014	4	6
	Crear controlador de la búsqueda en Grails	Marcia	20/02/2014	3	2
	Crear controlador de la búsqueda en Grails	Nestor	28/02/2014	2	
	Crear controlador de la búsqueda en Angular	Nestor	28/02/2014	6	4
	Crear test funcionales	Nestor	25/03/2014	6	3
	Crear test unitarios	Marcia	20/02/2014	2	2
	Crear test unitarios	Nestor	1/3/2014	1	
	Escribir casos de prueba	Nestor	19/02/2014	1	1

Figura 2: *Tracking* de tareas

3.2. Tecnologías

Todas las tecnologías que utilizamos en el desarrollo de la aplicación son de código abierto. Para desarrollar el *front-end* elegimos AngularJS[16]. Para el *back-end* elegimos Grails[17].

Cabe aclarar que ambos frameworks cubrieron ampliamente todo lo que nosotros necesitábamos de ellos para hacer este trabajo, por ello sólo utilizamos una pequeña parte de los mismos. Por ejemplo, de Grails casi no utilizamos la parte de las vistas ya que las mismas las desarrollamos del lado del cliente en AngularJS.

Además de estas dos tecnologías utilizamos Bootstrap[18] para obtener un diseño amigable para el usuario. Este framework también nos permitió realizar una aplicación *responsive* que se adapta a cualquier tamaño de pantalla, incluso de teléfonos celulares.

3.2.1. Sobre el *front-end*

AngularJS es un framework de aplicaciones web de código abierto escrito en JavaScript. Es desarrollado y mantenido por Google. La primer versión fue lanzada en el año 2010 y desde ese momento viene ganando espacio en la industria.

AngularJS es un conjunto de herramientas para la creación de aplicaciones web de una sola página (*single-page applications*). Este framework maneja contenido dinámico y permite extender el vocabulario HTML[19] obteniendo un entorno más expresivo, legible y práctico para el programador. La filosofía de AngularJS es que la programación declarativa es la que debe utilizarse para generar interfaces de usuario.

[16] https://angularjs.org/
[17] https://grails.org/
[18] http://getbootstrap.com/
[19] http://es.wikipedia.org/wiki/HTML

3.2.2. Sobre el *back-end*

Grails es un framework de aplicaciones web de código abierto para la máquina virtual de Java (JVM). Es un framework *full stack*, es decir que integra todas las herramientas necesarias para desarrollar una aplicación web.

Está escrito en Groovy, un lenguaje de programación que a su vez está desarrollado en Java. Uno de sus principios es la convención sobre la configuración que busca decrementar el número de decisiones que un desarrollador necesita tomar, ganando así en simplicidad pero no perdiendo flexibilidad por ello. La primer versión fue lanzada en el año 2006.

Como está desarrollado en Java, Grails es multiplataforma. Además toma de su lenguaje padre tecnologías ampliamente utilizadas en la industria como Hibernate[20], para la persistencia de datos, y Spring[21], para la seguridad, la autenticación, las pruebas, la gestión de transacciones, etc.

3.2.3. Sobre la base de datos

Para desarrollar una aplicación como la requerida se hizo indispensable utilizar una base de datos para almacenar toda la información ingresada. Los pacientes, los síntomas, las prioridades, los tiempos de espera, etc. son almacenados en una base de datos.

Durante todo el desarrollo de la aplicación utilizamos una base de datos H2[22] que viene embebida en Grails especialmente para utilizar en la etapa de desarrollo. También utilizamos esa tecnología en la instalación de prueba en el hospital. Pero para la instalación definitiva no utilizamos H2 ya que consideramos más apropiado y seguro tener una base de datos independiente al resto del sistema. Por eso utilizamos PostgreSQL[23]. PostgreSQL es un sistema de gestión de bases de datos relacional orientado a objetos. Lo elegimos porque varios artículos lo recomiendan como la mejor opción para usar junto a Grails[24].

3.3. Arquitectura

Utilizamos una arquitectura cliente-servidor[25] con AngularJS del lado del cliente y Grails del lado del servidor.

3.3.1. Comunicación entre el cliente y el servidor

La comunicación se da mediante peticiones HTTP[26] desde el cliente (AngularJS) al servidor (Grails). La información viaja en formato JSON[27]. Ejemplo de petición HTTP desde AngularJS:

```
$http.post('usuario/login',{
  usuario: $scope.nombre,
  password: $scope.password
```

[20] http://es.wikipedia.org/wiki/Hibernate
[21] http://es.wikipedia.org/wiki/Spring_Framework
[22] http://www.h2database.com
[23] http://www.postgresql.org.es/
[24] https://devcenter.heroku.com/articles/getting-started-with-grails
[25] http://es.wikipedia.org/wiki/Cliente-servidor
[26] http://es.wikipedia.org/wiki/Hypertext_Transfer_Protocol
[27] http://es.wikipedia.org/wiki/JSON

```
}).success(function(usuario){
  //hago algo cuando la peticion fue exitosa
}).error(function(){
  //hago algo cuando la peticion falló
});
```

En este ejemplo, se hace una petición HTTP de tipo POST con un JSON como parámetro. Del lado de Grails quien recibe la petición es el controlador de la clase de dominio Usuario que ejecutara el método login.

3.3.2. Arquitectura del *front-end*

En AngularJS cada pantalla esta definida dentro de un archivo HTML. A cada HTML le corresponde un controlador. Todos los controladores están definidos en un módulo de AngularJS dentro de un archivo JavaScript.

Al ser una aplicación de una sóla página, hay un único HTML principal que va cambiando parte de su contenido de manera dinámica a medida que el usuario navega. De esta manera también hay partes de la aplicación que están presentes en todas las pantallas, como el menú principal y el pie de página.

3.3.3. Arquitectura del *back-end*

Grails está dividido en vistas, controladores y clases del dominio. Como se trata de una aplicación de una sola página entonces tenemos una única vista. Hay un controlador por cada clase de dominio. Los controladores son los que procesan y responden las peticiones HTTP. Las clases del dominio son las que interactúan con los controladores y la base de datos.

4. Pruebas e instalación

En esta sección describimos todos los tipos de pruebas que realizamos junto con el desarrollo de la aplicación: pruebas unitarias, de integración y funcionales. En segundo lugar mencionamos algunos problemas con los que nos encontramos al incorporar las pruebas al desarrollo. En último lugar describimos como realizamos la instalación del producto final.

4.1. Pruebas

Desarrollamos la aplicación realizando pruebas unitarias y de integración de cada clase del dominio así como también pruebas funcionales de cada pantalla.

Cada funcionalidad desarrollada tiene su conjunto de pruebas correspondiente. Algunas de las ventajas de usar pruebas automatizadas son las siguientes:

- Se robustece la aplicación.
- Se genera confianza en el programador al hacer modificaciones.
- Se ahorra tiempo.
- Se disminuye el margen de error en el código.

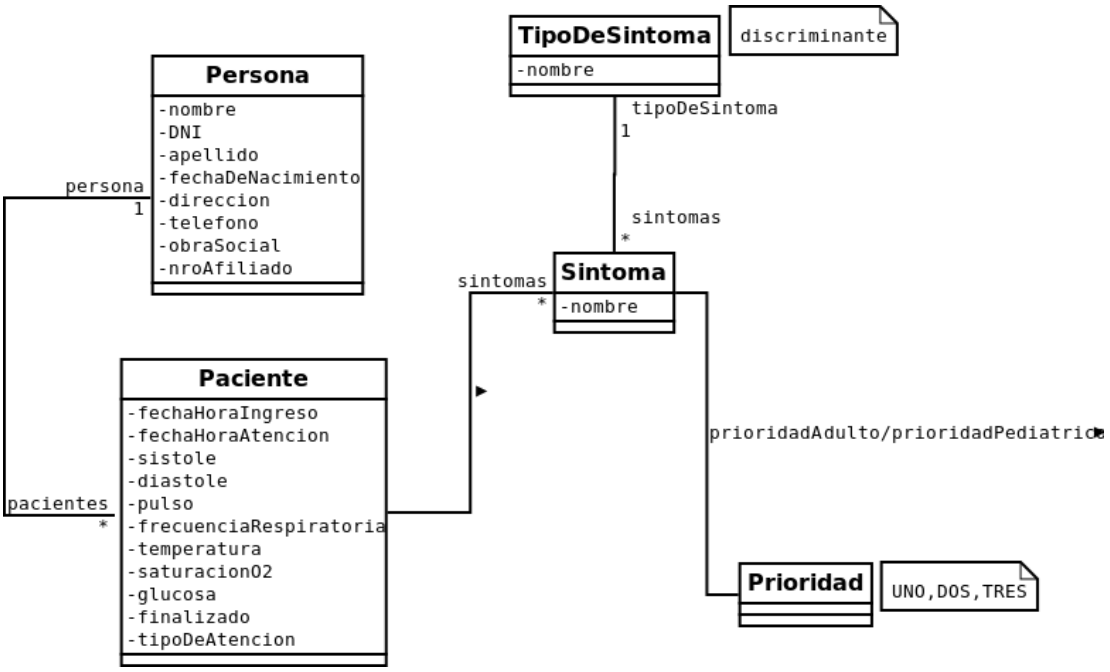

Figura 3: Diagrama de clases del dominio

4.1.1. Pruebas unitarias

En programación, una prueba unitaria es una forma de comprobar el correcto funcionamiento de un módulo de código. Esto sirve para asegurar que cada uno de los módulos funcione correctamente por separado. Luego, con las Pruebas de Integración, se podrá asegurar el correcto funcionamiento del sistema o subsistema en cuestión. Cabe mencionar que las pruebas unitarias no tienen repercución en la base de datos. Para realizarlas utilizamos la herramienta nativa de Grails[28] con la librería de Java JUnit[29]. Así cubrimos el comportamiento de las clases del dominio definidas en el *back-end*

4.1.2. Pruebas de integración

Las pruebas de integración son aquellas que se realizan en el ámbito del desarrollo de software una vez que se han aprobado las pruebas unitarias. Se refieren a las pruebas de todos los elementos unitarios que componen un proceso, hechas en conjunto, de una sola vez. Consiste en realizar pruebas para verificar que un gran conjunto de partes de software funciona bien. Las pruebas de integración preceden a las pruebas funcionales del sistema. Cabe mencionar que este tipo de pruebas tiene repercución en la base de datos. Para realizarlas utilizamos la herramienta nativa de Grails[30]. Con ello cubrimos el comportamiento de los controladores definidos en el *back-end*.

[28] http://grails.org/doc/latest/guide/testing.html#unitTesting
[29] http://junit.org/
[30] http://grails.org/doc/latest/guide/testing.html#integrationTesting

4.1.3. Pruebas funcionales

Las pruebas funcionales se basan en la ejecución, revisión y retroalimentación de las funcionalidades previamente diseñadas para el software. Se hacen mediante el diseño de modelos de prueba que buscan evaluar cada una de las opciones con las que cuenta el paquete informático. Dicho de otro modo son pruebas específicas, concretas y exhaustivas para probar y validar que el software hace lo que debe y sobre todo, lo que se ha especificado. Para realizarlas utilizamos CasperJS[31] y Protractor[32] haciendo una simulación del usuario final utilizando la aplicación.

En un primer momento utilizamos CasperJS pero tuvimos muchos problemas para hacerlo funcionar correctamente con AngularJS. Por eso dejamos de usarlo y lo reemplazamos por Protractor.

Para utilizar Protractor necesitamos usar Selenium[33], un entorno de pruebas de software para aplicaciones basadas en la web, con un *driver* para el navegador Chrome[34].

4.1.4. Problemas que tuvimos con el desarrollo de las pruebas

- Hay algunas funciones de Grails que no son soportadas por su ambiente de pruebas. Por eso para que las pruebas pasen nos vimos obligados a usar sólo aquellas funciones que no tenían dicho problema.

- Tuvimos problemas con Protractor al probar pantallas con ventanas modales. Las ventanas modales son elementos que al aparecer bloquean la ventana principal de la aplicación. Por eso para poder probar estas pantallas nos vimos obligados a dormir la ejecución de la prueba durante un segundo. Así el modal tenía tiempo para desaparecer y la ventana principal de desbloquearse. Si no hacíamos esto la prueba fallaba ya que la ejecución de la misma, luego de cerrar el modal, intentaba interactuar con elementos de la ventana principal que aún permanecían bloqueados.

4.2. Instalación

Acordamos con el Dr. Reggiani hacer la instalación en una máquina del hospital. Como las computadoras están en una misma red entonces la aplicación se encuentra accesible desde cualquier punto del lugar.

La primer instalación (de prueba) la hicimos al promediar el proyecto. El objetivo de la misma fue que los usuarios finales se familiaricen con el producto, nos den *feedback* y propongan modificaciones de creerlo necesario. La segunda instalación (definitiva) la hicimos al finalizar el proyecto.

Para la instalación de prueba usamos un servidor Tomcat[35] al cual le insertamos el WAR[36] de nuestra aplicación que contenía a su vez una base de datos H2[37] embebida.

[31] http://casperjs.org/
[32] https://github.com/angular/protractor
[33] http://www.seleniumhq.org/
[34] http://www.google.com/intl/es-419/chrome/
[35] http://tomcat.apache.org/
[36] http://es.wikipedia.org/wiki/WAR_(archivo)
[37] http://www.h2database.com

La instalación final fue similar a la de prueba con la salvedad que no utilizamos la base de datos H2 embebida en Grails. En su lugar instalamos una base de datos PostgreSQL[38] independiente del resto del sistema y, por lo tanto, más segura y confiable.

5. Conclusiones

El presente trabajo abordó el planteo, diseño, implementación, desarrollo y puesta en producción de "Triage, Sistema de gestión para sala de Guardia Hospitalaria"

La primer conclusión importante que obtuvimos es la facilidad con la que se pudo desarrollar el trabajo gracias a la interacción continua con el cliente y los directores. Esta modo de trabajo permitió llevar el enfoque general del desarrollo hacia el producto final que necesita el cliente.

La segunda conclusión que obtuvimos al terminar el desarrollo de este trabajo es que el contenido en general de la carrera nos dió las herramientas y el conocimiento para poder encarar un proyecto de una magnitud mayor a lo aprendido en cualquiera de las materias cursadas. Permitiendo así la familiarización rápida con tecnologías nunca utilizadas.

Finalmente queda por mencionar que, como todo diseño, si bien el trabajo efectuado es de calidad y funcionalidad, queda abierto a mejoras y agregación de nuevas funcionalidades en futuros TIPs.

[38] http://www.postgresql.org.es/

Referencias

[1] Derlet R, Kinser D, Lou R, et al. Prospective identification and triage of nonemergency patients out of an Emergency Department: a 5 years study. Ann Emerg Med 1996; 25:215-223.

[2] Manual de procedimiento. Recepción, Acogida y Clasificación. MSPBS. Paraguay 2011.

[3] Shore J, Warden S, The Art of Agile Development, O'Reilly Media, 2007.

Figura 4: Pantalla inicial

A. Manual del Usuario

Las siguientes secciones tienen como objetivo describir y explicar las funcionalidades del Sistema de gestión para sala de Guardia Hospitalaria, que utiliza el método Triage para la recepción de los pacientes. En las mismas, se detalla la funcionalidad de cada pantalla con screenshot y ejemplos básicos y funcionales al manual.

B. Pantalla Inicial

En esta sección del manual explicaremos cómo ingresar nuevas personas al sistema o cómo buscarlas en el caso de que ya hayan sido atendidas.

B.1. Ingreso de un nuevo paciente

En la pantalla de Inicio del sistema (figura 4)

C. Alta, baja y modificación de datos (ABMs)

Para poder acceder al menú de configuración de datos el usuario actual debe tener el rol de administrador. En caso contrario dicho menú permanecerá oculto.

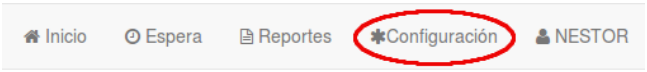

Figura 5: Menú de configuración visible

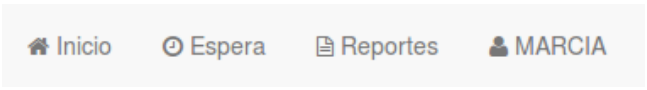

Figura 6: Menú de configuración oculto

C.1. ABM de síntomas

Para acceder a la pantalla de administración de síntomas dirigirse hacia "Configuración" y luego a "Síntomas".

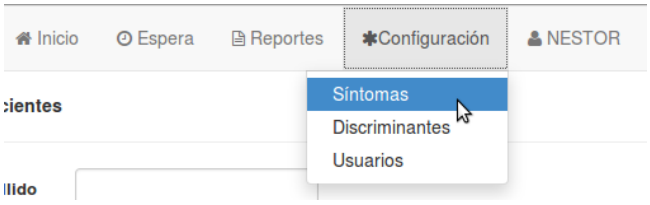

Figura 7: Menú de síntomas

Cuando hacemos click en "Síntomas" se nos muestra el listado de todos los síntomas cargados en el sistema.

Figura 8: Listado de síntomas

C.2. Filtrado del listado

Para facilitar la búsqueda de un síntoma podemos filtrar el listado llenando los campos de "Síntoma" y/o "Discriminante"[39]. No hace falta ingresar la palabra entera. Es suficiente con ingresar las primeras letras. No se distinguen mayúsculas y minúsculas. Luego de ingresar el valor deseado hay que hacer click en el botón "Buscar"

[39]El filtrado de los listados de síntomas, discriminantes y usuarios funciona de la misma manera. Por lo tanto en este manual sólo se explicará el filtrado del listado de síntomas

Figura 9: Listado de síntomas filtrado