Triage, Sistema de gestión para sala de Guardia hospitalaria

Néstor Muñoz
Marcia Tejeda

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

25 de octubre de 2014

Resumen

El presente Trabajo de Inserción Profesional (TIP) se desarrolló 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 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 a veces criterios diferentes para la toma de decisiones.

Según Luis 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. Luis 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.

2. Planteo

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 que es quién utiliza el sistema desarrollado. Éste toma una impresión visual del paciente. Luego se pasa a la sala de toma de signos vitales donde un 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 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. Reuniones y contacto con el usuario

Durante el trabajo hubo contacto constante con Luis Reggiani.
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 cosas dependiendo del tiempo disponible y consultando con Luis la prioridad para cada tarea.

2.3. Requerimientos del cliente

Lo más importante para Luis fue desde un primer momento registrar adecuadamente a los pacientes que ingresan. Para ello, se decidió guardar todos los datos de las personas: Nombre, Apellido, Teléfono, DNI, Dirección -entre otros- para poder contar con una base de datos de todas las personas atendidas en caso de necesitarla.
Entre las cosas más prioritarias, se tomó la idea de completar el Triage de forma eficiente y con una respuesta rápida ante casos de urgencias. Se pidió que el

sistema corte cualquier interacción ante un caso de Prioridad UNO, para poder actuar con el apremio necesario.
Se detallan en secciones futuras los reportes que fueron requeridos.

2.4. Pantallas y dinámica de uso

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". Esta otra pantalla 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. Esto es útil cuando la persona que toma la Impresión Visual está en un lugar físico distinto del de el 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:

- Síntomas
- Discriminante
- Usuarios

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

2.5. Informes

El cliente pidió pantallas con los informes detallados a continuación.

2.5.1. 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.

2.5.2. 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.

2.5.3. 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. Diseño e implementación

3.1. Tecnologías

Todas las tecnologías que utilizamos en el desarrollo de la aplicación son de código abierto. Para desarrollar el front-end (interfaz de usuario) elegimos AngularJS[1]. Para el back-end (lógica del negocio e interacción con la base de datos) elegimos Grails[2].
Cabe aclarar que ambos frameworks cubrieron ampliamente todo lo que nosotros necesitabamos de ellos para hacer este trabajo, por ello sólo utilizamos una pequeña parte de los mismos. Por ejemplo, de Grails casi no utilizamos las vistas ya que las mismas las desarrollamos del lado del cliente en AngularJS.
Otra tecnología que utilizamos fue Bootstrap[3]. Con esta herramienta pudimos atenuar nuestras carencias en diseño web y además pudimos hacer una aplicación responsive que se adapta a cualquier tamaño de pantalla, incluso de teléfonos celulares.

3.1.1. AngularJS

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.
HTML es suficiente para hacer páginas web estáticas. Pero no alcanza para desarrollar vistas dinámicas en aplicaciones web. 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 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.
Si bien nosotros no conociamos AngularJS, habiamos escuchado buenos comentarios sobre este framework moderno y novedoso, entonces decidimos aprenderlo utilizandolo en este TIP.

3.1.2. Grails

Grails es un framework de aplicaciones web de código abierto, full stack (contiene todo lo necesario para desarrollar una aplicación web), para la máquina virtual de Java (JVM).
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 (convention over configuration) 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 y Spring.
Al igual que con AngularJS, nosotros no conociamos Grails de antemano. Tuvimos que aprender a utilizarlo en el desarrollo de este TIP.

[1] https://angularjs.org/
[2] https://grails.org/
[3] http://getbootstrap.com/

3.1.3. Bases de datos

Durante todo el desarrollo de la aplicación utilizamos una base de datos H2 que viene embebida en Grails. También utilizamos H2 en la instalación de prueba en el hospital. Pero para la instalación final utilizamos Postgres[4].
Elegimos Postgres porque es lo que recomienda Heroku[5] para utilizar con Grails[6]. Heroku es una plataforma de la nube que soporta varios lenguajes de programación y ofrece un servicio de hosting básico gratuito. Nuestra idea original era instalar la aplicación en dicha plataforma, pero el límite de memoria que ofrece el servicio gratuito nos obligó a buscar otra alternativa. Finalmente decidimos hacer la instalación en una máquina del hospital.

3.2. Metodología de trabajo

Desde el primer momento el director del TIP nos indujo a 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 "cliente".

3.2.1. 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 "cliente". 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[7] y Grails aparecían como las predilectas aunque nunca habiamos trabajado con ninguna de las dos. Por ello, para obtener una impresión general de cada una y así resolver la elección, desarrollamos un conversor de Farenheit a Celcius muy simple. 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, el director del TIP quería que la aplicación sea más moderna. Entonces nos recomendó utilizar una tecnología que resuelva las vistas del lado del cliente, se comunique con el back-end mediante una API(Application Programming Interface) REST[8] y sea responsive[9]. Así surgio la idea de utilizar AngularJS 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 "cliente".

[4] http://www.postgresql.org.es/
[5] https://www.heroku.com/
[6] https://devcenter.heroku.com/articles/getting-started-with-grails
[7] http://nodejs.org/
[8] http://es.wikipedia.org/wiki/Representational_State_Transfer
[9] http://es.wikipedia.org/wiki/Diseño_web_adaptable

A partir de ahí comenzamos con el desarrollo recorriendo las iteraciones planificadas. Al promediar el proyecto hicimos una instalación de prueba en el hospital. La idea era que los usuarios se familiaricen con el producto, nos den un feedback y propongan modificaciones de creerlo necesario.
Por último hicimos la instalación definitiva del producto terminado en una máquina del hospital.

3.2.2. 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 "cliente". 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. Al finalizar la iteración enviabamos otro email con los detalles de lo realizado.

3.2.3. Herramientas que utilizamos

Para la comunicación via email creamos un grupo en Google Groups[10]. Para toda la documentación compartida utilizamos Google Drive[11]. Para hacer reuniones remotas utilizamos Skype[12] y Google Hangouts[13]. Utilizamos Balsamiq[14] para hacer los prototipos de pantallas. Utilizamos Git[15] y Github[16] para versionar el código. Y utilizamos Travis[17] como servidor de integración continua[18].

3.3. Arquitectura

Utilizamos una arquitectura cliente-servidor[19] 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[20] desde el cliente (AngularJS) al servidor (Grails). La información viaja en formato JSON[21]. Ejemplo de petición HTTP desde AngularJS:

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

[10] https://groups.google.com
[11] https://drive.google.com/
[12] http://www.skype.com.ar/es/
[13] https://plus.google.com/hangouts
[14] https://balsamiq.com/
[15] http://git-scm.com/
[16] https://github.com/
[17] https://travis-ci.org/
[18] http://es.wikipedia.org/wiki/Integración_continua
[19] http://es.wikipedia.org/wiki/Cliente-servidor
[20] http://es.wikipedia.org/wiki/Hypertext_Transfer_Protocol
[21] 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 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 en Grails

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.

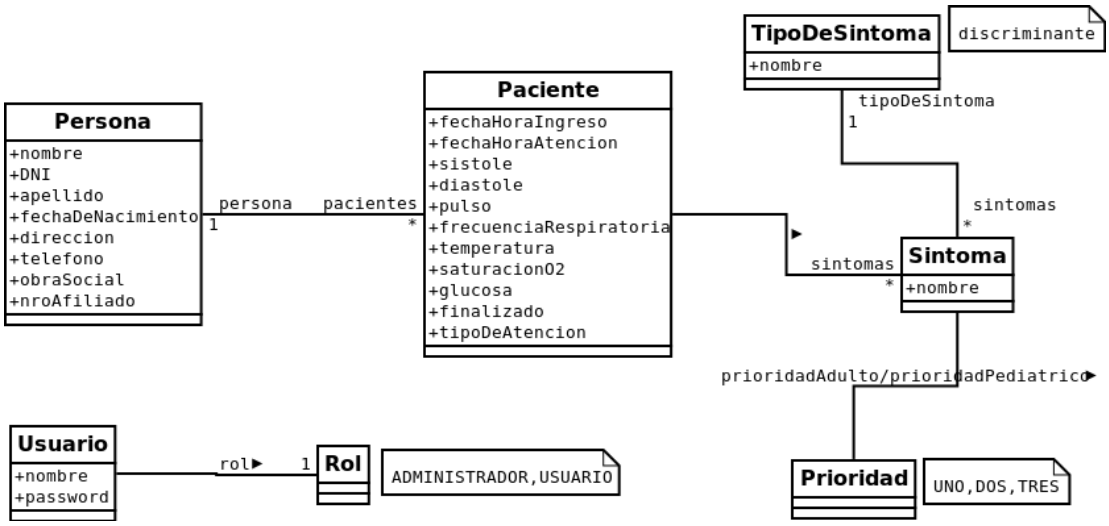

Figura 1: Diagrama de clases del dominio

3.4. Testing

Desarrollamos la aplicación realizando testing unitario y de integración de cada clase de dominio así como también testing funcional de cada pantalla.
Cada funcionalidad desarrollada tiene su testing correspondiente. Algunas de las ventajas de usar testing automatizado 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.

3.4.1. Testing unitario

Para los tests unitarios utilizamos la herramienta nativa de Grails[22] con la librería de Java JUnit. Con ello cubrimos el comportamiento de las clases de dominio definidas en Grails que no tiene repercución en la base de datos.

3.4.2. Testing de integración

Realizamos los tests de integración con la herramienta nativa de Grails[23]. Con ello cubrimos el comportamiento de los controladores definidos en Grails que tiene repercución en la base de datos.

3.4.3. Testing funcional

Para testear el correcto funcionamiento de cada pantalla emulamos al usuario final usando dos herramientas de testing funcional: CasperJS[24] y Protractor[25]. 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[26], un entorno de pruebas de software para aplicaciones basadas en la web, con un driver para el navegador Chrome[27].

3.4.4. Problemas que tuvimos con el testing

- Hay algunas funciones de Grails que no son soportadas por el ambiente de test del mismo framework. Entonces para que los test pasen nos vimos obligados a usar sólo aquellas funciones que no tenían dicho problema.
- Tuvimos problemas con Protractor al testear pantallas con modales. Como los modales bloquean el resto de la pantalla nos vimos obligados, siempre que aparece un modal en un caso de prueba, a dormir la ejecución del test para darle tiempo al modal a desaparecer y desbloquear el resto de la pantalla. Si no haciamos esto el test no pasaba.

[22] http://grails.org/doc/latest/guide/testing.html#unitTesting
[23] http://grails.org/doc/latest/guide/testing.html#integrationTesting
[24] http://casperjs.org/
[25] https://github.com/angular/protractor
[26] http://www.seleniumhq.org/
[27] http://www.google.com/intl/es-419/chrome/

3.5. Deploy e Instalación

La idea original fue tener la aplicación en línea corriendo en Heroku[28]. Pero tuvimos que descartar esa posibilidad ya que la cantidad de memoria que la aplicación necesita para funcionar es mayor a la que ofrece el servicio gratuito. Entonces acordamos 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.
Para la instalación de prueba usamos un servidor Tomcat[29] al cual le insertamos el WAR[30] de nuestra aplicación que contiene a su vez una base de datos H2[31] embebida.
La instalación final es igual a la de prueba pero no utilizamos la base de datos H2 y en su lugar instalamos una base de datos Postgres.

3.5.1. Manual de usuario

Se confeccionó un manual de usuario...

3.5.2. Aprendizaje del usuario

Cuánto tiempo llevó explicar el sistema, si fue fácil de entender..

[28] https://www.heroku.com/
[29] http://tomcat.apache.org/
[30] http://es.wikipedia.org/wiki/WAR_(archivo)
[31] http://www.h2database.com

4. Conclusiones

Cómo atacamos el problema
Contamos como siempre estuvimos en contacto con el cliente, validando cada pantalla y cada funcionalidad
Sobre lo que sabíamos y lo que no
Sobre la necesidad del codirector o 'nexo' con el cliente

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.