Trabajo de Inserción Profesional

Néstor Muñoz
Marcia Tejeda

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

19 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

1.1. Contexto general

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. 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.

1.2. Sobre el TRIAGE

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 que se retrase la atención del paciente que empeoraría su pronóstico por la demora en su 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.

1.3. Propuesta de solución

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.

1.4. Objetivo General

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.

1.5. Resultado Final

Desarrollamos 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.

1.6. Síntesis de trabajo

TODO

2. Planteo

2.1. Problema en detalle

Cómo se trabaja en el hospital, cuál es la necesidad real

2.2. Reuniones y contacto con el usuario

Contacto constante (Esto aparece abajo también, no sé dónde iría o cómo dividirlo)

2.3. Requerimientos del cliente

?

2.4. Pantallas

Contamos en general cuál sería la dinámica de uso y las pantallas principales

2.5. Informes

Qué informes nos pidió el cliente y para qué

2.5.1. Tiempo de espera para cada prioridad

Consultar a Luis para qué necesita este reporte

2.5.2. Cantidad de atenciones para cada prioridad

Consultar a Luis para qué necesita este reporte

2.5.3. Reporte de Personas

Lista con todas las personas que se atendieron y el detalle de cada atención. Preguntar a Luis por qué es útil

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

Los primero que hicimos fueron 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 aparecian 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 veniamos 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. Arquitetura 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 interactuan con los controladores y la base de datos.

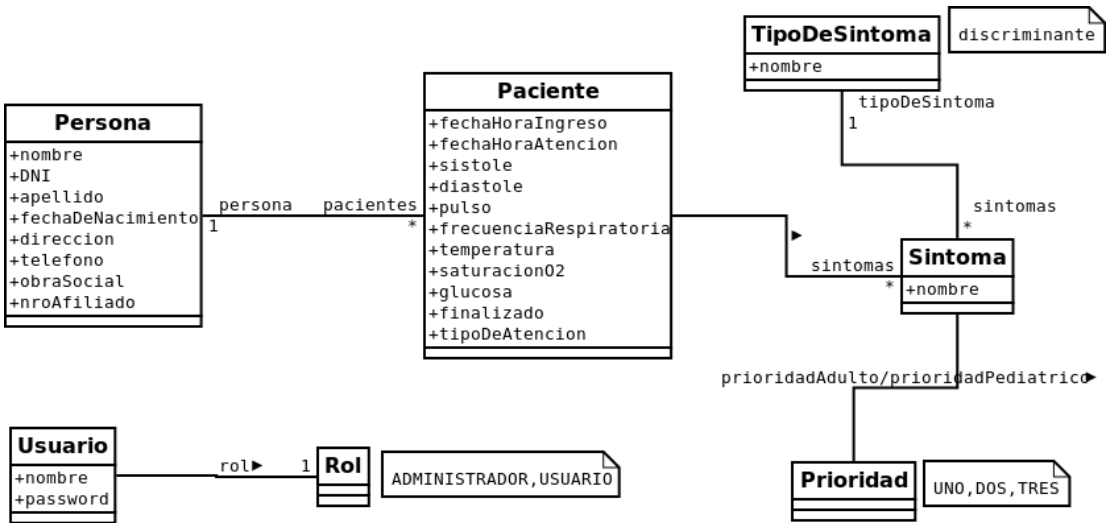

Figura 1: Diagrama de clases del dominio

3.4. Detalle técnico

?

3.5. Problemas que tuvimos

?

3.6. Detalles interesantes del código

En el caso de que los haya

3.7. Testing

Herramientas que usamos para testear, qué tipo de testing hicimos (funcional, unitario, de integración...)

3.7.1. Funcional

Selenium.. / Protractor / Casper

3.7.2. Unitario

Con Grails

3.7.3. Integración

Con Grails

3.8. Deploy e Instalación

.war Tomcat H2 Postgres

Cómo realizamos la instalación en el entorno de trabajo donde se usará el producto

3.8.1. Manual de usuario

Se confeccionó un manual de usuario...

3.8.2. Aprendizaje del usuario

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

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.