FACULTAD DE INGENIERÍA Y CIENCIAS AGROPECUARIAS

SISTEMA DE GESTIÓN DE TURNOS ONLINE PARA LA CLÍNICA ODONTOLÓGICA DE LA UNIVERSIDAD DE LAS AMÉRICAS. (SOLUCIÓN CLOUD)

Trabajo de Titulación presentado en conformidad a los requisitos establecidos para optar por el título de Ingeniero en Sistemas de Computación e Informática

Profesor Guía

Ing. Juan José León Guerrero Msc.

Autor

Pablo Alexander Jácome Arrobo

Año

2016

DECLARACIÓN DEL PROFESOR GUÍA

“Declaro haber dirigido este trabajo a través de reuniones periódicas con el estudiante, orientando sus conocimientos y competencias para un eficiente desarrollo del tema escogido y dando cumplimiento a todas las disposiciones vigentes que regulan los Trabajos de Titulación”

_____

Juan José León Guerrero
Master Especializado en Ingeniería en Medios Numéricos
Ingeniero de Sistemas y Computación
1707506760

DECLARACIÓN DE AUTORÍA DEL ESTUDIANTE

“Declaro que este trabajo es original, de mi autoría, que se han citado las fuentes correspondientes y que en su ejecución se respetaron las disposiciones legales que protegen los derechos de autor vigentes”

_____

Pablo Alexander Jácome Arrobo

C.I: 1718805748

AGRADECIMIENTOS

A mis padres y mi hermana, quienes con mucha paciencia, ayuda y perseverancia no dejaron que mi objetivo se trunque.

A mis compañeros y amigos, con los cuales he compartido inolvidables momentos.

DEDICATORIA

Quisiera dedicar el presente trabajo a todas las personas que quieran construir algo desde 0. No sólo sistemas y software, sino también cualquier cosa que sea beneficiosa para la humanidad.

Espero que este trabajo sirva de inspiración para crear más soluciones cloud en el Ecuador y que estas tengan mejor demanda.

RESUMEN

Las soluciones o sistemas Cloud en el Ecuador son poco comunes, a pesar de las ventajas que estas presentan sobre los sistemas tradicionales. De la misma manera, en el Ecuador existen pocos hospitales o clínicas que posean un sistema de gestión de turnos y horas laboradas. El presente proyecto presenta el levantamiento de requerimientos, diseño e implementación de una solución Cloud, utilizando metodología Scrum, para la clínica odontológica de la Universidad de las Américas, en donde no existe una administración y contabilidad eficiente de las horas de práctica de los estudiantes. Razón por la cual, se hizo el levantamiento de requerimientos necesario y se desarrollaron las historias de usuario que permitieron el desarrollo de la solución "SISCOUDLA". El desarrollo de aplicación fue divido en 4 Sprints, los mismo que fueron validados por el Scrum Master. En el desarrollo de la solución se utilizaron varias herramientas de programación libres, entre ellas: Java, JavaScript, Jquery, JSon, Ajax, Html5, CSS3, Bootstrap, etc., que finalmente fueron desplegadas en el Google App Engine. Esta solución utiliza arquitectura SOA, y aprovecha todas las posibilidades de las tecnologías de Google para su desarrollo e implementación.

ABSTRACT

Cloud systems or solutions in Ecuador are not common, in spite of the advantages of these systems over the traditional ones. The same way in Ecuador there are few hospitals and clinics that manage the hours of attention and work using a system. This project it shows the requirements lifting, analysis and development of a cloud solution, using the Scrum methodology, for the dental clinic of the Universidad de las Américas, where there are not a system to manage the practicing hours of the students. For this reason, the requirements were lifted and the user's cases were developed, for the purpose of finish the product: SISCOUDLA. In order to develop this solution, it was necessary 4 sprints that were validated for the Scrum master at the end of each sprint. For the development of this solution some open source tools were used, for example: Java, JavaScript, Jquery, JSon, Ajax, Html5, CSS3, Bootstrap, etc., that were deployed in the Google App Engine. This solution uses SOA architecture and takes advantage of all the capabilities of Google's technologies in order to develop and deploy a final product.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Introducción

Antecedentes

Una de las tendencias poco explotadas en el Ecuador, es la de soluciones empresariales en la nube, tal vez por su desconocimiento o falta de popularidad. Sin embargo, es una de las propuestas más factibles que pueden existir en el mercado para la implementación de sistemas para empresas. Las soluciones en la nube permiten a un departamento de TI centrarse en aportar valor, en lugar de verse obligado a dedicar la mayor parte de su tiempo al mantenimiento y actualización de su infraestructura tecnológica, con las mismas garantías de seguridad y rendimiento que si la tuviera alojada en su propia empresa. Es sabido que las empresas que no han adoptado Cloud, dedican un 70% de su presupuesto de TI al mantenimiento y sólo un 30% a nuevos proyectos e innovación. (Softeng, s.f.)

La atención odontológica en el ciudad de Quito puede ser muy limitada o de altos costes. Razón por la cual la Facultad de Odontología de la Universidad de las Américas ha puesto a disposición del público en general su clínica odontológica, con una atención de calidad y calidez bajo la responsabilidad de profesionales altamente capacitados. La clínica se encuentra ubicada en el Campus Colón. (Universidad de las Américas, s.f.)

En la actualidad, la facultad de odontología de las Universidad de las Américas está estructurada en Área de Simulación, Área de Laboratorios, Área de Clínica y Área de Cirugía. En el Área de Clínica, se atiende al público y los estudiantes hacen sus horas de prácticas para las materias Clínica I y II. Para poder tomar dichas materias, los estudiantes deben aprobar las simulaciones y cursar el quinto semestre de la carrera. (UDLA, Quito, s.f.)

El proceso de atención y registro de las horas de práctica, es completamente manual; estas son registradas en papel, para luego ser contabilizadas y calificadas. Al ser un proceso manual, se corre el riesgo de que la información

sea alterada o perdida y no se pueda recuperar, afectando así a la contabilidad de las horas de práctica de los estudiantes y por ende su calificación. Además, que no se tiene almacenados los datos personales de los pacientes y existe desorganización en el manejo de los cubículos disponibles y reservados.

Dados estos antecedentes, el presente proyecto pretende crear una solución empresarial en la nube llamada SISCOUDLA para la clínica odontológica de la Universidad de las Américas que permita gestionar las horas de práctica de los estudiantes y la organización de cubículos.

Alcance

El alcance de este trabajo de titulación es proporcionar a la facultad de odontología un sistema con el cual se pueda gestionar las horas de práctica de los estudiantes. De la misma manera, el sistema servirá para organizar el uso de los cubículos y los turnos reservados.

El sistema contará con una plataforma web, en donde los estudiantes reservarán un cubículo: de acuerdo al tratamiento que van a realizar y en su horario correspondiente.

El sistema permitirá a los coordinadores y administradores, la generación de reportes acerca del uso de cubículos y tratamientos realizados en la clínica odontológica.

El sistema guardará la información de los pacientes atendidos en la clínica odontológica, y permitirá crear reportes acerca de los mismos.

El sistema contará con una plataforma web, en la cual las personas encargadas de la administración, registrarán los estudiantes, tratamientos, horarios, etc. que se involucren en la atención de la clínica odontológica.

El sistema no modificará o intervendrá de alguna manera en el proceso de calificación de los estudiantes, sino que contabilizará las horas de atención y mostrará reportes de los tratamientos realizados.

El sistema, no tendrá alguna plataforma móvil nativa; sin embargo tendrá la capacidad de adaptarse a cualquier tipo pantalla.

El sistema, será implementado en la nube con una arquitectura orientada a servicios (S.O.A) en la plataforma de servicios Cloud de Google.

El sistema, utilizará metodología SCRUM para su desarrollo.

Justificación

En la actualidad cada vez son más frecuentes los sistemas online de reserva de turnos, para la atención del público, en clínicas y hospitales. Los cuales facilitan las labores de atención y la administración de las horas trabajadas por los médicos, enfermeras, etc.

La clínica odontológica de la UDLA es una institución médica que brinda atención al público en general y no posee un sistema de automatización y gestión de turnos, de manera que esta información es tratada manualmente con el riesgo de ser manipulada o perdida. La constante necesidad de un sistema de gestión y la falta de éste en la clínica odontológica de la Universidad de las Américas ha inspirado este proyecto.

Adicionalmente, y a diferencia del resto de sistemas que existen en la actualidad, el sistema de la clínica odontológica de la UDLA será implementado con arquitectura SOA y en Cloud, lo cual es un campo poco experimentado en el país y que fomentaría el desarrollo de más soluciones de este estilo.

Objetivo General

Implementar una plataforma virtual online y en la nube, utilizando metodología Scrum, para la gestión de turnos y horas de práctica en la clínica odontológica de la Universidad de las Américas. “SISCOUDLA”.

Objetivos específicos

- Levantar y analizar los requerimientos del usuario final con el fin de desarrollar una plataforma virtual online “a medida” que satisfaga las necesidades actuales.
- Desarrollar e implementar los requerimientos del usuario en una plataforma online.
- Utilizar la metodología Scrum en el proceso de desarrollo del sistema.
- Implementar el sistema en la nube, utilizando la plataforma Cloud de Google.
- Realizar las pruebas de aceptación y calidad respectivas por parte del Scrum Master.

1. Capítulo I. Estado de la Cuestión

1.1. Cloud Computing

El termino Cloud Computing, en resumen, significa almacenar y acceder a información y programas a través del internet en vez de utilizar recursos locales, como discos duros o redes internas.

Para el presente proyecto se utiliza el Cloud Computing con todas sus prestaciones y utilidades para el desarrollo e implementación de un sistema completo de nivel empresarial en el cual se presentan los siguientes elementos: (Developer, s.f.)

- **Infraestructura:** Generalmente las aplicaciones empresariales necesitan de servidores dedicados para hospedar las aplicaciones propias de la institución. En el caso del Cloud Computing se utilizan servidores de los proveedores de servicios cloud, en este caso Google.
- **Plataforma:** Generalmente las aplicaciones empresariales necesitan de una plataforma instalada en los servidores de aplicaciones para así poder ser ejecutadas. En el caso del Cloud Computing, las plataformas están instaladas en los proveedores de cloud, en este caso Google App Engine.
- **Software:** Generalmente las aplicaciones empresariales necesitan de software de terceros para su funcionalidad. Muchas de ellas se encuentran hospedadas en diversos servidores. En el caso de Cloud Computing no importa en donde se encuentren hospedadas las aplicaciones sino simplemente consumir sus servicios.

Estos elementos respetan la arquitectura recomendada para el cloud computing como es la SOA (Services Oriented Architecture), la cual permite que todos los elementos de una solución sean independientes entre sí pero

que actúen como un solo sistema. Se utiliza SOA para el desarrollo e implementación de este proyecto.

1.2. Ventajas de Cloud Computing

Las principales ventajas del Cloud Computing son: (Developer, s.f.)

- **Costes bajo demanda**: Una de las principales ventajas del Cloud Computing tiene que ver con los costes que representa tener una solución empresarial en Cloud vs una solución in-house. Las soluciones Cloud tienen la característica de que se paga únicamente lo que se consume. Es decir, los costes de infraestructura y servicios se reducen a únicamente lo que se haya utilizado durante un tiempo determinado.
- **Elasticidad:** El Cloud Computing permite a las aplicaciones crecer o reducir sus requisitos de recursos a petición del cliente, sin la necesidad de preocuparse por cuestiones de infraestructura como nuevos nodos de servicios, discos duros, etc.
- **Seguridad:** Las principales preocupaciones en la implementación de sistemas web, es la seguridad. El Cloud Computing utiliza la seguridad de sus proveedores y garantiza que toda la información almacenada sea fidedigna y segura.
- **Múltiples data centers:** Generalmente por cuestión de costes y complejidad, las empresas cuentan con un único data center en donde almacenan su información y aplicaciones. El Cloud Computing permite utilizar todos los data centers de sus proveedores sin importar la localización de la petición del servicio.
- **Tiempo de respuesta:** Las aplicaciones web necesitan una gran infraestructura para que los tiempos de respuesta sean pequeños y no afecten la experiencia de usuario. Sin embargo, el acortar estos tiempos de respuesta requiere de muchos recursos de infraestructura y configuraciones que muchas veces no pueden ser fácilmente implementadas. El Cloud Computing libera al cliente de esta

preocupación ya que todas las configuraciones e infraestructura es manejada por el personal propio de los proveedores.

1.3. Comparativa de proveedores Cloud en el mercado

En la actualidad en el mercado existen diversos proveedores de Cloud Computing entre los cuales se destacan: (CBT Nuggets, s.f.)

- **Amazon Web Services:** AWS fue uno de los pioneros en prestar este tipo de servicios, tiene mayor antigüedad y experiencia en la prestación de servicios cloud, sin embargo suele ser susceptible en su tiempo de respuesta. Debido a que para implementar una solución en AWS, es necesario configurar y conocer las zonas con las que trabajan los servicios de AWS. Es más difícil de implementar soluciones.
- **Microsoft Azure:** Azure es la propuesta de Microsoft en la prestación de servicios cloud, tiene la ventaja de ser seguros y facilita la integración de las aplicaciones cloud con software propio de Microsoft y sistemas operativos. Sin embargo los costes de implementar soluciones cloud en Azure suelen ser mayores debido a que se debe cancelar licencias adicionales para utilizar los servicios necesarios para nuestra aplicación.
- **Google Cloud Services:** Google ha sido reconocido durante años en tener la mejor infraestructura del mercado. Gracias a esto, en la actualidad muchas empresas están migrando su información y sistemas a los servicios cloud de Google. Además, Google propone el Google App Engine, el cual permite crear soluciones cloud de manera fácil y ágil.

Para el presente proyecto se utilizó Google Cloud Services, debido a la facilidad de desarrollo de soluciones cloud y la integración con sus aplicaciones o APIs. Además, en la Universidad de las Américas, los estudiantes utilizan correos electrónicos basados en tecnología Google lo

cual me permitió integrar fácilmente estos con la aplicación desarrollada: SISCOUDLA.

1.4. Services Oriented Architecture (SOA)

Al hablar de Cloud Computing, se debe hablar estrictamente de arquitectura SOA, la cual se orienta a servicios. Servicios tales como infraestructura, plataforma, software, etc. Quienes funcionan en conjunto para prestar a los clientes, soluciones prácticas e inmediatas, pero que sin embargo pueden ser utilizadas por separado y ser independientes unas de otras. Se puede decir que esta infraestructura se enfoca en el negocio. (MSDN Microsoft, s.f.)

La arquitectura SOA permite el desenvolvimiento de una solución Cloud, pues deja a la libertad del desarrollador, seleccionar los componentes que se van a utilizar, además de que estos puedan ser de fabricantes diferentes, o diferentes tecnologías.

1.5. SCRUM

SCRUM en los últimos años ha sido una de las metodologías más exitosas en el desarrollo de software a medida. Razón por la cual se decidió utilizar esta metodología en el desarrollo de este proyecto, pues permite la rápida adaptabilidad a los cambios y la entrega de productos cada cierto tiempo, demostrando el trabajo realizado por el equipo de desarrollo. (Scrum Methology, s.f.)

Scrum es una metodología de desarrollo muy simple que requiere de mucho trabajo, dado que su gestión no se basa en el seguimiento de un plan, sino en la adaptación continua sobre la marcha. (Palacio, 2007)

En la figura 1, se puede observar el ciclo de vida que posee un proyecto realizado con metodología Scrum, en este se puede observar cada uno de los artefactos que serán utilizados en este proyecto:

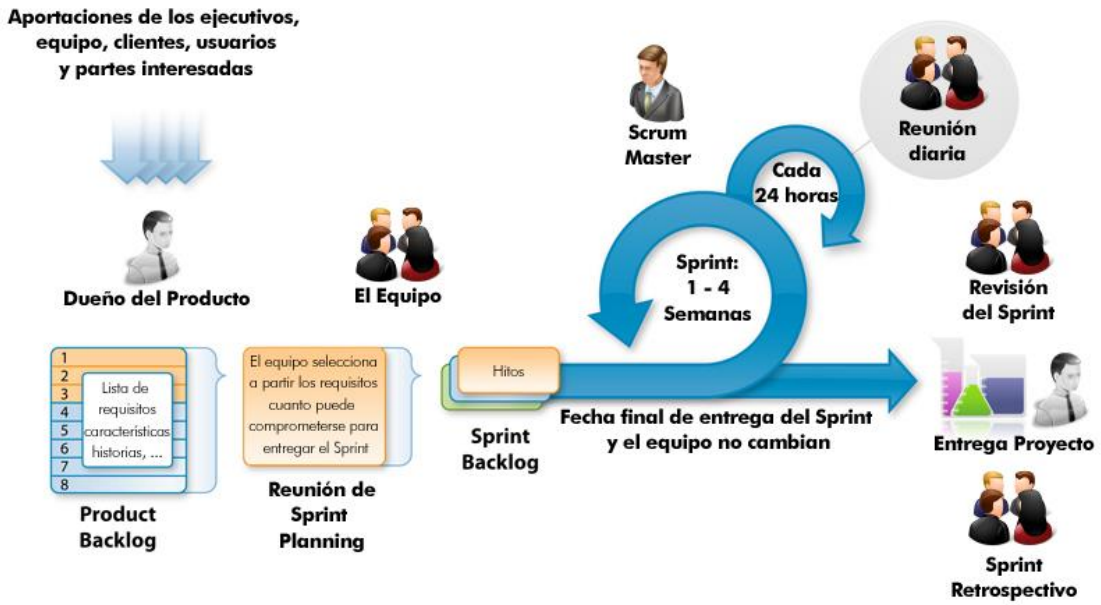

Figura 1. Ciclo de Vida SCRUM
Tomado de (IslaVisual, s.f.)

La ventaja que presenta Scrum en el desarrollo de soluciones Cloud, se demuestra al fraccionar el trabajo en fases o Sprints con lo que se puede trabajar en cada uno de los servicios de la arquitectura SOA por separado.

2. Capítulo II. Desarrollo

El presente proyecto se realizó utilizando metodología Scrum, para lo cual a continuación se presentan los artefactos utilizados además de todo el proceso que propone la metodología.

2.1 Recolección de Historias de Usuario

Las historias de usuario fueron recolectadas en base a una reunión con las coordinadoras de la clínica. En la tabla 1, se presenta información general que contiene: el nombre del proceso, el usuario involucrado, la funcionalidad que se quiere implementar, el número de la historia de usuario que va a describir esta funcionalidad y la prioridad asignada.

Tabla 1. Recolección de Historias de Usuario

Proceso	Usuario	Requerimiento	Historia de Usuario	Prioridad
Registro	Estudiante	Registrar datos personales.	HU01	Alta
Registro	Estudiante	Registrar datos del paciente.	HU02	Alta
Registro	Estudiante	Registro de turno.	HU03	Alta
Confirmación	Coordinador	Revisión de turnos, cubículos e impresión de lista.	HU04	Alta
Reportes	Administrador	Generación de reportes.	HU05	Alta
Administración	Administrador Sistema	Creación de Coordinadores y Administradores.	HU06	Media
Administración	Administrador Sistema	Administración de tablas del sistema.	HU07	Baja

Tomado de *(Palacio, 2016)*

Para describir las historias de usuario se usará el formato indicado en la tabla 2, donde:

- **Número:** El numero secuencial de la historia de usuario.
- **Nombre de historia:** Texto descriptivo de la historia de usuario.
- **Usuario:** Se refiere a la persona que realiza la funcionalidad.
- **Descripción:** Texto que detalla la historia de usuario.
- **Condiciones de satisfacción:** Describe el resultado esperado.
- **Prerrequisitos:** Son los requisitos previos para realizar la historia de usuario.

Tabla 2. Formato de Historias de Usuario

Historia de Usuario **Número**
Nombre Historia: **Usuario:** **Descripción:** **Condiciones de Satisfacción:** **Prerrequisitos:**

Tomado de (Palacio, 2016)

Desde la tabla 3 hasta la tabla 9 se describen las historias de usuario identificadas en la tabla 2.

2.1.1 HU01

Tabla 3. Historia de Usuario 1

Historia de Usuario **HU01**
Nombre de historia: Registrar datos personales. **Usuario:** Estudiante **Descripción:**

1. Comprobar si el estudiante está registrado en el sistema realizando una consulta mediante su número de matrícula. 2. Si el estudiante está registrado, desplegar los datos y permitir actualizar los mismos. 3. Si el estudiante no está registrado, se debe ingresar sus nombres, apellidos, clínica a la que pertenece, correo electrónico, número de identificación, número de matrícula y teléfono. 4. Validar los datos ingresados 5. Guardar los datos en las tablas correspondientes. **Condiciones de Satisfacción:** La información del estudiante queda registrada. **Prerrequisitos:** Ninguno

2.1.2 HU02

Tabla 4. Historia de Usuario 2

Historia de Usuario **HU02**
Nombre de historia: Registrar datos del paciente. **Usuario:** Estudiante **Descripción:** 1. Comprobar si el estudiante ha ingresado al sistema. 2. Comprobar si el paciente existe en el sistema haciendo una consulta usando su número de identificación. 3. Si el paciente no se encuentra registrado en el sistema, se debe ingresar sus nombres, apellidos, número de historia si la tiene, correo electrónico, número de identificación, género y teléfono. 4. Validar los datos registrados. 5. Guardar los datos en las tablas correspondientes. **Condiciones de Satisfacción:** La información del paciente queda registrada. **Prerrequisitos:** HU01

2.1.3 HU03

Tabla 5. Historia de Usuario 3

Historia de Usuario **HU03**
Nombre de historia: Registro de turno. **Usuario:** Estudiante **Descripción:** 1. Se debe comprobar que el estudiante ha ingresado al sistema y tenga un paciente registrado. 2. Se debe validar el horario que seleccionó y que este le corresponda. 3. El estudiante debe seleccionar el tratamiento que va a realizar y según su elección, el sistema le concederá un cubículo libre. 4. Validar toda la información registrada. 5. Guardar la información en las tablas correspondientes. **Condiciones de Satisfacción:** La información del turno queda registrada. **Prerrequisitos:** HU01**,** HU02

2.1.4 HU04

Tabla 6. Historia de Usuario 4

Historia de Usuario **HU04**
Nombre de historia: Revisión de turnos, cubículos e impresión de lista.
Usuario: Coordinador
Descripción: 1. El administrador del sistema debe crear un usuario coordinador con el rol respectivo. 2. El usuario coordinador debe ingresar al sistema. 3. Se debe comprobar que el usuario esté registrado en el sistema y tenga asignado el rol correspondiente.

4. El sistema generará la información de los turnos registrados. 5. Existirá una opción de impresión, con la lista de estudiantes, pacientes y tratamiento por turno.
Condiciones de Satisfacción: La información registrada es la correcta y se puede imprimir la lista.
Prerrequisitos: HU03

2.1.5 HU05

Tabla 7. Historia de Usuario 5

Historia de Usuario **HU05**
Nombre de historia: Generación de reportes.
Usuario: Administrador
Descripción: 1. El administrador del sistema debe crear un usuario administrador con el rol respectivo. 2. El usuario administrador debe ingresar al sistema. 3. Se debe comprobar que el usuario esté registrado en el sistema y tenga asignado el rol correspondiente. 4. El sistema generará la información de los turnos registrados. 4. Existirá una opción de impresión, con la lista de estudiantes, pacientes y tratamientos.
Condiciones de Satisfacción: La información registrada es la correcta y se puede generar reportes.
Prerrequisitos: HU04

2.1.6 HU06

Tabla 8. Historia de Usuario 6

Historia de Usuario **HU06**
Nombre de historia: Creación de Coordinadores y Administradores.
Usuario: Administrador del Sistema
Descripción: 1. El Súper usuario (Desarrollador) debe crear un usuario administrador del sistema con el rol respectivo. 2. El usuario administrador del sistema debe ingresar al sistema. 3. Se debe comprobar que el usuario esté registrado en el sistema y tenga asignado el rol correspondiente. 4. El usuario podrá crear usuarios coordinadores y administradores. 5. Se validará la información registrada. 6. Guardar la información registrada en las tablas correspondientes.
Condiciones de Satisfacción: La información queda registrada.
Prerrequisitos: Ninguno

2.1.7 HU07

Tabla 9. Historia de Usuario 7

Historia de Usuario **HU07**
Nombre de historia: Administración de tablas del sistema.
Usuario: Administrador del Sistema
Descripción: 1. El usuario administrador del sistema debe ingresar al sistema. 2. Se debe comprobar que el usuario esté registrado en el sistema y tenga asignado el rol correspondiente. 3. El usuario podrá modificar las tablas del sistema según corresponda. 4. Se validará la información registrada. 5. Guardar la información registrada en las tablas correspondientes.
Condiciones de Satisfacción: La información queda registrada.
Prerrequisitos: Ninguno

2.2 Diseño e Implementación del Product Backlog

2.2.1 Lista inicial del producto

Para realizar el Product Backlog inicial se tomaron las historias de usuario organizadas de acuerdo a la prioridad: alta, media, baja. Las historias de usuario de prioridad alta son: HU01, HU02, HU03, HU04, HU05. Las historias que tienen prioridad media son: HU06. Las historias que tienen prioridad baja son: HU07. El Product Backlog inicial se muestra en la tabla 10 que posee las siguientes columnas:

- **Orden:** Prioridad del requerimiento.
- **Requerimiento:** texto descriptivo de la historia de usuario.
- **Historia de usuario:** es el número de historia.

Tabla 10. Product Backlog Previo

Orden	Requerimiento	Historia de Usuario
1	Creación de arquitectura y ambientes de desarrollo.	Definido por el autor
2	Diseño del Front End	Definido por el autor
3	Login Google y Tradicional	Definido por el autor
4	Registrar datos personales.	HU01
5	Registrar datos del paciente.	HU02
6	Registro de turno.	HU03
7	Revisión de turnos, cubículos e impresión de lista.	HU04
8	Generación de reportes.	HU05
9	Creación de Coordinadores y Administradores.	HU06
10	Administración de tablas del sistema.	HU07
11	Manejo de sesiones y roles	Definido por el autor

Adaptado de (Palacio, 2016)

2.2.2 Refinamiento del Product Backlog

En el refinamiento del Product Backlog se añaden tareas que deben realizarse para cumplir con cada requerimiento. En la tabla 11 se muestra el refinamiento del Product Backlog.

Tabla 11. Product Backlog Refinado

Orden	Requerimiento	Tarea
1	Creación de arquitectura y ambientes de desarrollo.	1.1 Definir la arquitectura de la aplicación
		1.2 Crear el modelo conceptual de la base de datos
		1.3 Crear el modelo físico de la base de datos
		1.4 Generar el código SQL
2	Diseño del Front End	2.1 Diseño e implementación de Landing Page
		2.2 Diseño e implementación de dashboards por rol
		2.3 Diseño e implementación de módulos por rol
3	Login Google y Tradicional	3.1 Login usando el API de Google
		3.2 Login tradicional
4	Registrar datos personales. (HU01)	4.1 Crear un método que permita buscar un estudiante de acuerdo a su número de matrícula
		4.2 Crear un método que permita actualizar los datos del estudiante
		4.3 Validar los datos ingresados
		4.4 Crear un método que permita ingresar al estudiante, en caso de que la búsqueda no devuelva ningún resultado

		4.5 Mostrar un mensaje de confirmación al guardar la información del estudiante
5	Registrar datos del paciente. (HU02)	5.1 Crear un método que permita buscar un paciente de acuerdo a su número de historia clínica
		5.2 Crear un método que permita actualizar los datos del paciente.
		5.3 Validar los datos ingresados
		5.4 Crear un método que permita ingresar el paciente, en caso de que la búsqueda no devuelva ningún resultado.
		5.5 Mostrar un mensaje de confirmación al guardar/actualizar la información personal del paciente.
6	Registro de turno. (HU03)	6.1 Ingresar datos de paciente y validarlos
		6.2 Presentar Horarios del Estudiante Ingresado
		6.3 Presentar Especialidades y Tratamientos
		6.4 Presentar cubículo y registro de turno
7	Revisión de turnos, cubículos e impresión de lista. (HU04)	7.1 Crear un método de consulta de turnos
		7.2 Presentar turnos reservados, ocupados y cancelados al Estudiante
		7.3 Presentar turnos reservados al Coordinador (Toma de Lista)
		7.4 Presentar turnos reservados, ocupados y cancelados al Administrador
8	Generación de reportes. (HU05)	8.1 Reporte de Turnos
		8.2 Reporte de Estudiantes
		8.3 Reporte varios
9	Creación de	9.1 Definir forma de creación de

	Coordinadores y Administradores. (HU06)	coordinadores y administradores
		9.2 Funcionamiento y administración de roles
10	Administración de tablas del sistema. (HU07)	10.1 Crear módulos de administración para Administrador
		10.2 Administración de tablas catálogos
		10.3 Administración de tablas core
11	Manejo de sesiones y roles	11.1 Crear módulo de autenticación
		11.2 Crear usuarios y roles
		11.3 Configurar usuarios y roles
		11.4 Realizar pruebas

Adaptado de (Palacio, 2016)

2.2.3 Definición de los Sprints

Luego de definir el Product Backlog se procede a especificar los Sprints, en la reunión Sprint Planning Meeting realizada con el Scrum Master. Se define lo siguiente:

- Para el presente proyecto se utilizarán números de la serie Fibonacci como medida de estimación de esfuerzo (0,1, 2, 3, 5, 8, 13).
- El Scrum Master asignará la media de las estimaciones.
- En caso de existir discrepancia en la medida de la estimación, el Scrum Master asignará el valor correspondiente.

2.2.3.1 SPRINT I

El Sprint I se inició el 14/04/2016 y terminó el 29/04/2016 con una duración total de 12 días. Para este Sprint se diseñaron e implementaron los siguientes requerimientos: Creación de arquitectura y ambientes de desarrollo y Diseño del Front End considerados de alta prioridad.

El objetivo del primer Sprint es tener el esquema y arquitectura necesarios para la elaboración del sistema y así poder desarrollar las historias de usuario. En la tabla 12, se detalla las tareas de este sprint.

Tabla 12. Primer Sprint

Orden	Requerimiento	Tarea	Estimación
1	Creación de arquitectura y ambientes de desarrollo.	1.1 Definir la arquitectura de la aplicación	13
		1.2 Crear el modelo conceptual de la base de datos	8
		1.3 Crear el modelo físico de la base de datos	5
		1.4 Generar el código SQL	5
2	Diseño del Front End	2.1 Diseño e implementación de Landing Page	8
		2.2 Diseño e implementación de dashboards por rol	8
		2.3 Diseño e implementación de módulos por rol	8
		TOTAL	55

Adaptado de (Palacio, 2016)

Diseño/Implementación: Las tareas que requieren diseño en este Sprint son:

- 1.1 Definir la arquitectura de la aplicación
- 1.2 Crear el modelo conceptual de la base de datos
- 2.1 Diseño de Landing Page
- 2.2 Diseño de dashboards por rol
- 2.3 Diseño de módulos por rol

Arquitectura de la aplicación

- **Arquitectura**

La aplicación utiliza la arquitectura SOA propia de un sistema Cloud. En esta aplicación podemos encontrar los siguientes elementos, representados en la figura 2:

- **IaaS:** La aplicación utiliza la infraestructura Google, debido a que reside en los servidores Cloud de Google App Engine y no necesitará alguna configuración adicional o mantenimiento.
- **PaaS:** Las plataformas que utiliza la aplicación son:
    a) Google App Engine, como motor principal de desarrollo.
    b) Google Cloud SQL, como motor de base de datos.
- **SaaS:** El software que utiliza la aplicación es:
    a) Cloud Debugger API, para depurar la aplicación en el entorno cloud.
    b) Google Calendar API, para el manejo de fechas.
    c) Google Cloud Logging API, para el manejo de registro de logs.
    d) Identity Toolkit API, para el manejo de sesiones y login.

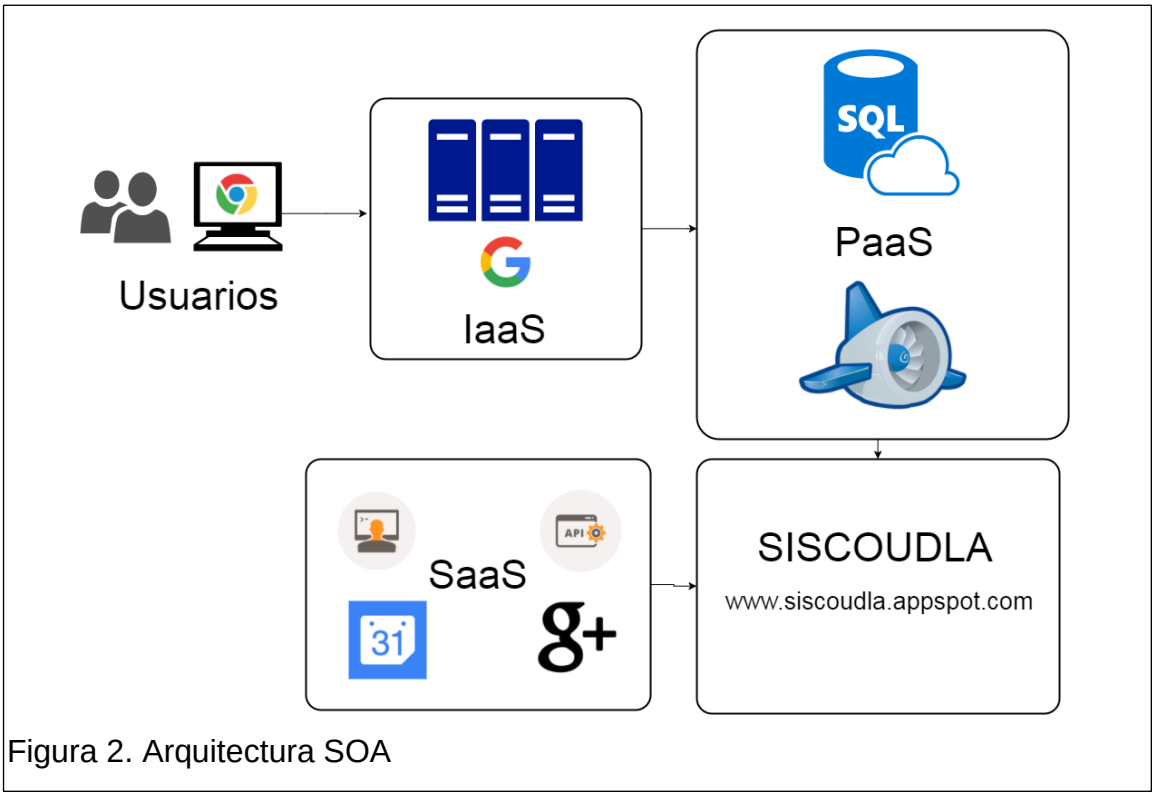

Figura 2. Arquitectura SOA

A nivel de código, la aplicación posee la siguiente estructura:

- **Front End:** La aplicación en el Front End utiliza HTML5, CSS3 y el framework Bootstrap dentro de archivos JSP, y a su vez estos utilizan JavaScript, JQuery, Json y Ajax para la comunicación e interacción con el Back End.

  El Front End sigue el patrón modelo-vista-controlador, podemos ver la representación de este patrón en la figura 3 donde, el controlador (a través de archivos JS) es el encargado de la lógica del Front End y a su vez de interactuar con los objetos Json (modelo) para luego ser presentados en la vista. Este patrón, tiene la ventaja de que todo el trabajo se realiza en el cliente y reduce la interacción con el servidor a simples llamadas Ajax, lo cual permite que la aplicación responda más rápidamente a las peticiones y acciones del usuario.

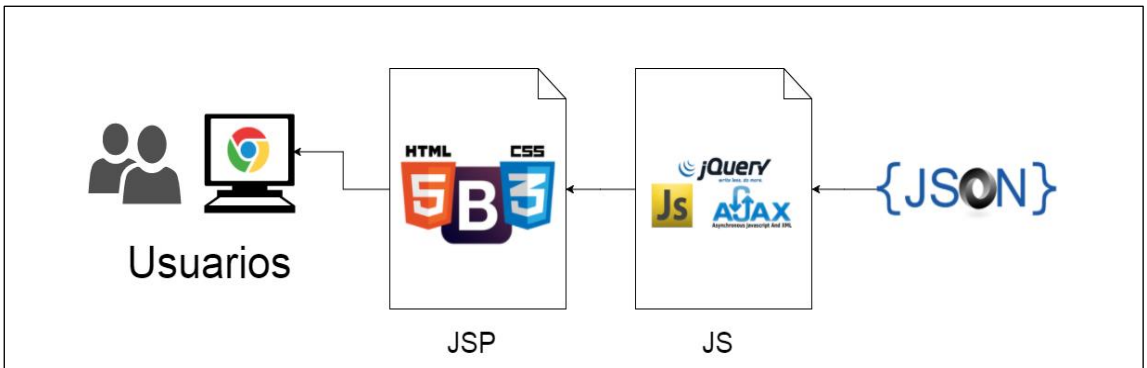

Figura 3. Arquitectura Front End

- **Back End:** La aplicación en el Back End utiliza el lenguaje de programación Java, de la misma manera utiliza el patrón modelo-vista-controlador, siendo los controladores (archivos java) los encargados de la lógica en el Back End así mismo interactúan con los objetos POJO, a través del motor JPA y Eclipse Link para la persistencia de datos en la base. Para la vista se utiliza archivos JSP, explicados anteriormente. En la figura 4 se describe el patrón MVC, esta vez para la parte del Back End, el desglose de la vista (archivos JSP) lo podemos revisar en la figura 3.

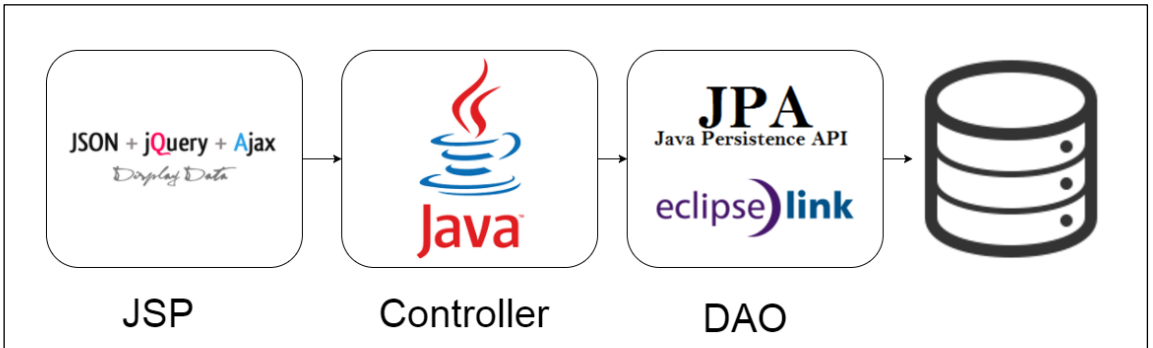

Figura 4. Arquitectura Back End

Diagrama de Despliegue

El diagrama de despliegue presenta los componentes de un sistema además de cómo estos se comunican entre sí.

El diagrama de despliegue para esta solución lo podemos ver en la figura 5, el cual consiste de 2 componentes principales: un computador personal en donde

el usuario utilizará un Web Browser y se comunicará con el Google App Engine mediante HTTP, el otro componente trata del Google App Engine en donde existen los subcomponentes: Google SQL el cual es el motor de la base de datos, Google APIs las cuales son las aplicaciones de Google que utilizará la solución y el servidor de despliegue de la aplicación. Todos estos componentes se comunican mediante HTTP:

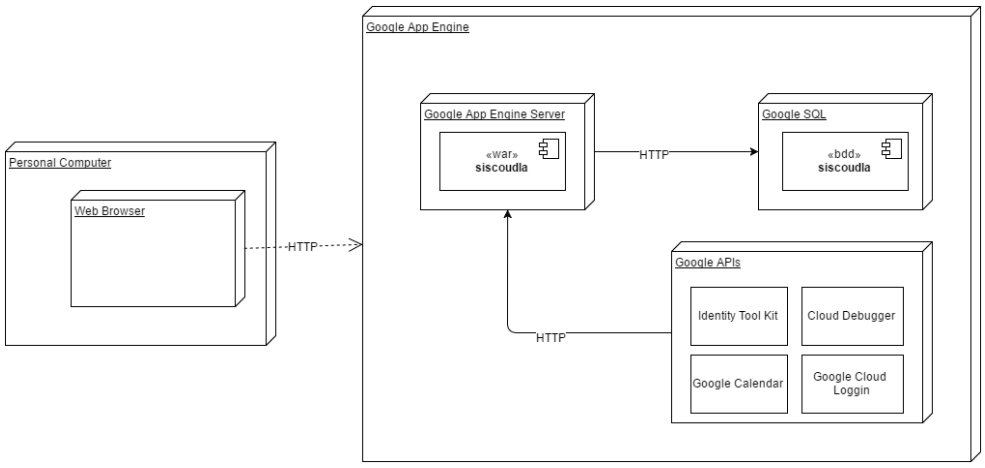

Figura 5. Diagrama de Despliegue

Diagrama de Clases

El diagrama de clases presenta un esquema de como las entidades u objetos serán utilizadas para el desarrollo de una aplicación.

El diagrama de clases de la aplicación se representa en la figura 6, en donde se tiene como clases principales a Persona (de la cual heredan Estudiante, Paciente y Usuario). Las clases Turno, Horario y Cubículo permiten el registro de futuras atenciones a pacientes. Las clases Especialidad y Tratamiento sirven para identificar que cubículos serán utilizados en la asignación de turnos.

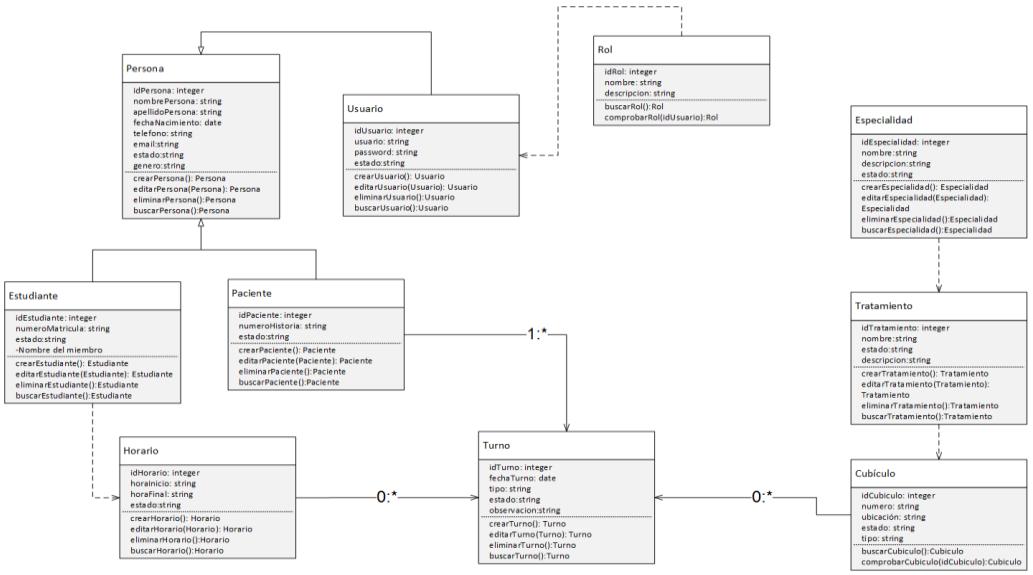

Figura 6. Diagrama de Clases

- **Diagrama de Casos de Uso**

El diagrama de casos de uso muestra las tareas o actividades que un Usuario o Actor realizará en el sistema, además de las sub tareas que estas puedan tener.

El diagrama de casos de uso para los actores Estudiante y Coordinador se representa en la figura 7, en donde se muestra las actividades que pueden realizar. El estudiante podrá administrar su información y reservar turnos, mientras el coordinador administrará su información, los cubículos y tomará lista de acuerdo a los turnos reservados por los estudiantes:

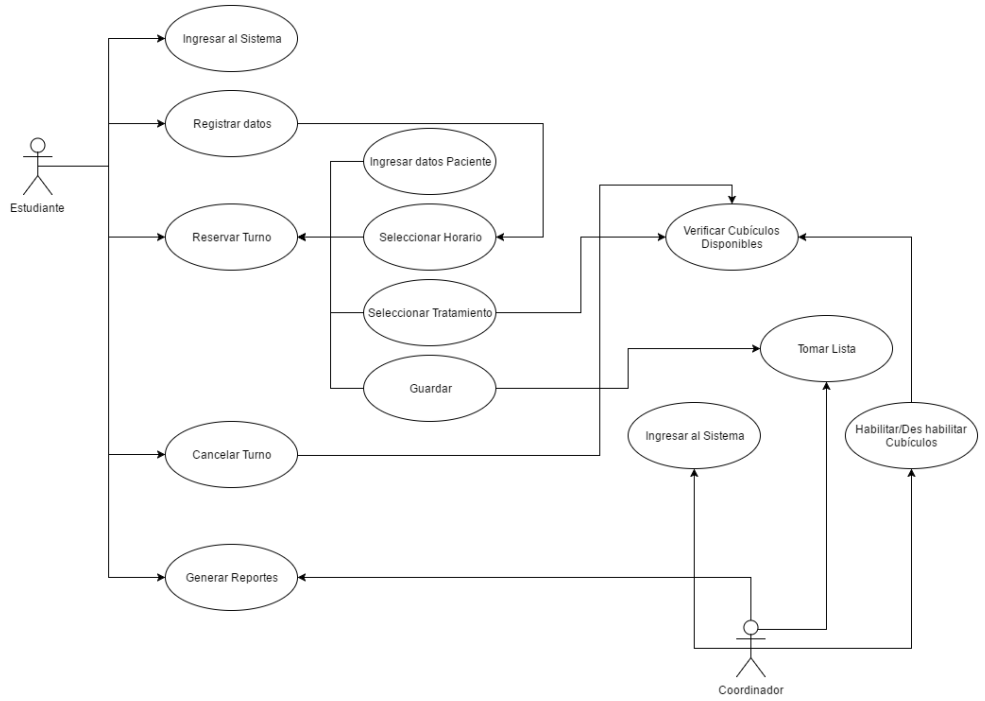

Figura 7. Diagrama Casos Sistema

El diagrama de casos de uso para el actor Administrador se representa en la figura 8, donde las actividades del Administrador son administrar su información y de la clínica, crear coordinadores y generar reportes:

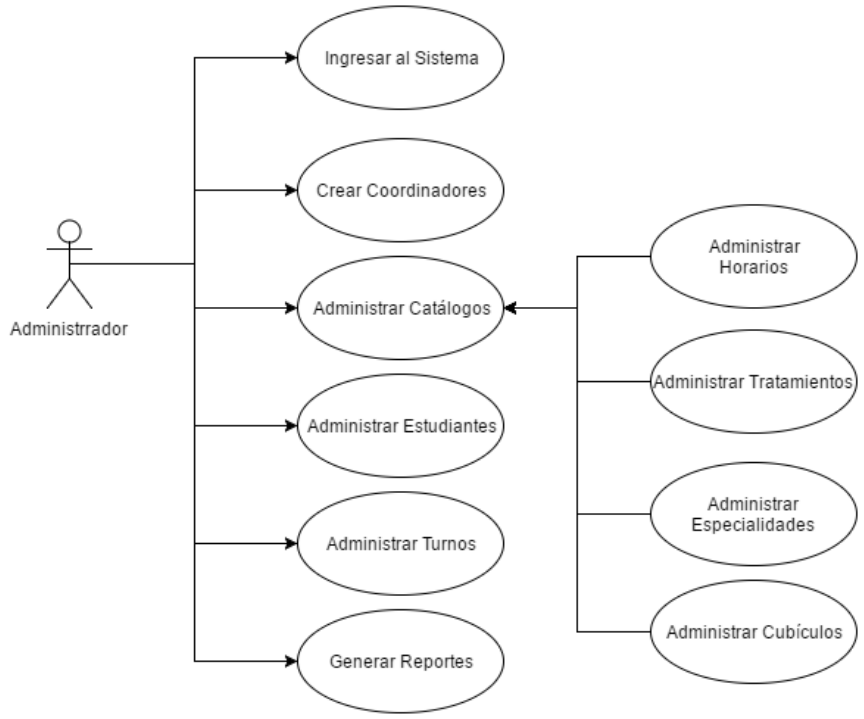

Figura 8. Diagrama de Casos del Administrador

- **Diagramas de Actividad**

El diagrama de actividad presenta los flujos o secuencia de actividades que se realizan para un determinado proceso.

En la figura 9 se muestra el diagrama de actividad del flujo Registrar Turno, en donde el actor Estudiante debe registrarse en el sistema, registrar su información estudiantil y proceder a registrar turnos. Para el registro de turnos se presenta una secuencia de pasos:

- **Ingresar datos del Paciente:** El Estudiante ingresará el número de historia clínica y el Sistema validará si el paciente ya existe o no.
- **Seleccionar Fecha y Horario:** Cuando el Estudiante haya registrado sus horarios para el semestre podrá seleccionar fechas de reserva de turnos.
- **Seleccionar Tratamiento:** El Sistema mostrará las especialidades y tratamientos disponibles para la reserva de turnos. Una vez seleccionado el tratamiento, el Sistema verificará si existen cubículos

disponibles para ese horario e informará del cubículo disponible o falta de este.

- **Guardar Turno:** Una vez que el estudiante guarda el turno, el turno es reservado y el cubículo deja de estar disponible.

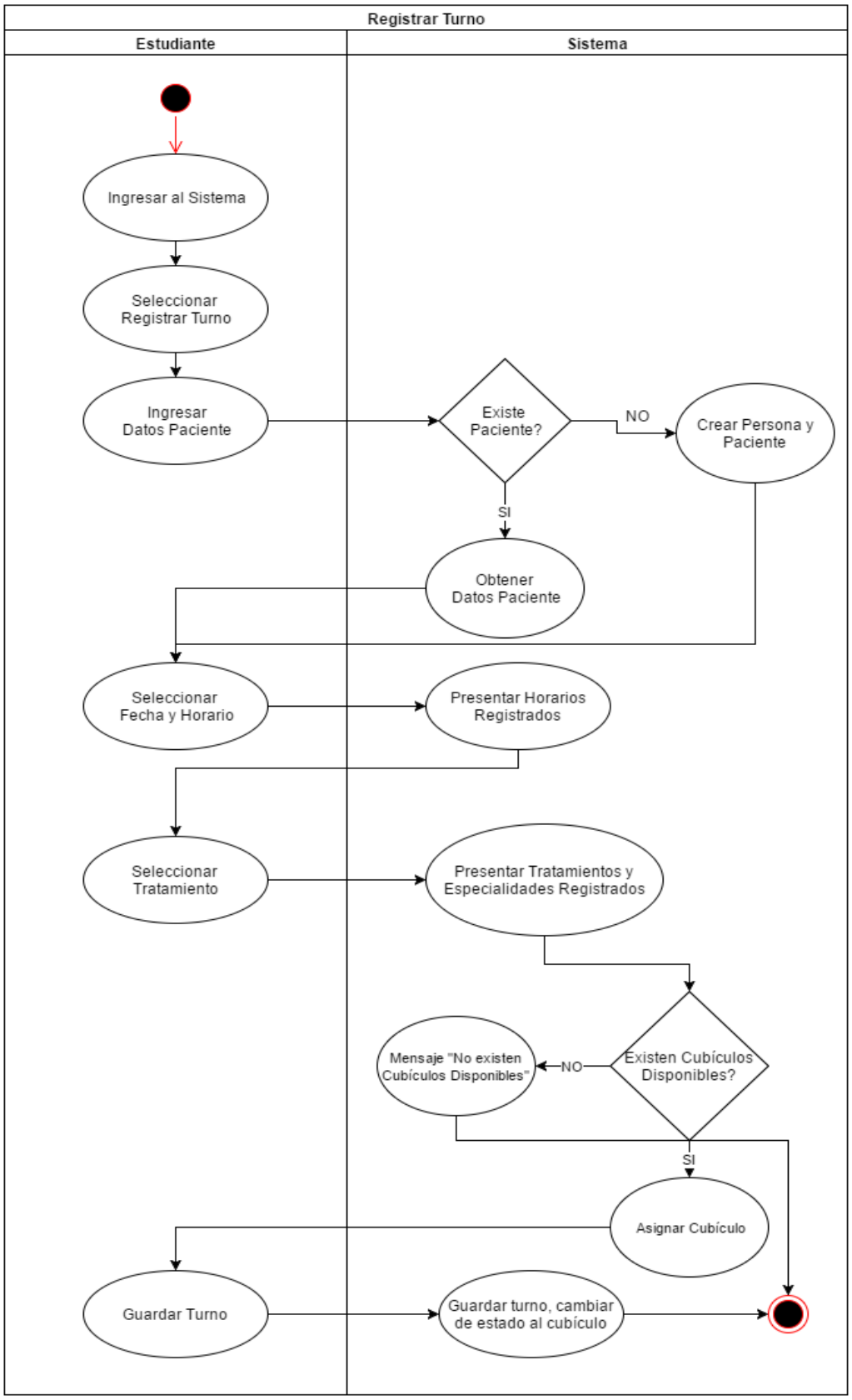

Figura 9. Diagrama Actividad Registrar Turno

El diagrama de actividad Tomar Lista se representa en la figura 10, en donde el Coordinador debe ingresar al sistema y seleccionar la opción de Tomar Lista, se le presentarán los datos registrados previamente por los Estudiantes y procederá a confirmar su turno. Al confirmar el turno, el coordinador cambia el estado del cubículo y del turno del Estudiante:

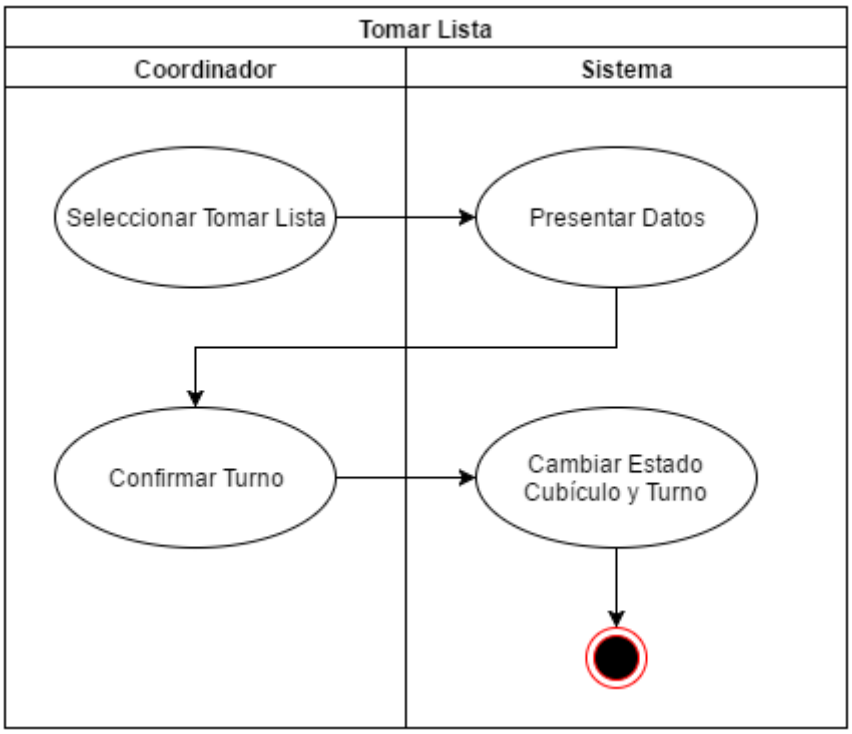

Figura 10. Diagrama de Actividad Tomar Lista

Diagrama de Secuencia

El diagrama de secuencia presenta el grupo de pasos a seguir para realizar un proceso de una manera más detallada.

El diagrama de secuencia de la actividad Registrar Turno se representa en la figura 11 y 12. En la figura 11, se presenta la primera parte del flujo, en donde se desglosa cada una de las actividades y métodos que se ejecutan en el sistema:

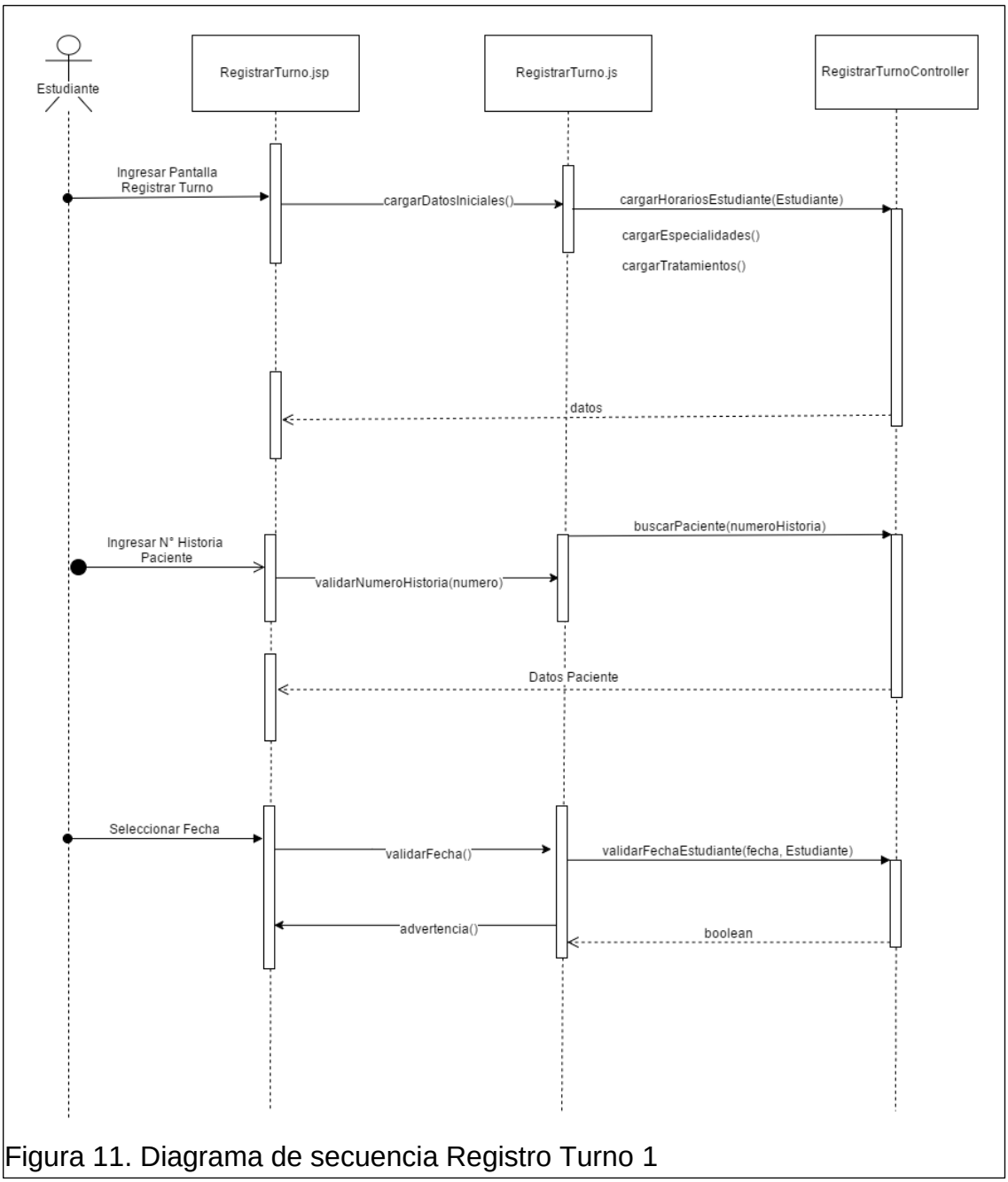

Figura 11. Diagrama de secuencia Registro Turno 1

En la figura 12, se puede apreciar la continuación del flujo en donde ya se han ejecutado los primeros métodos descritos en la figura 11 y que llevan a la culminación del flujo Registrar Turno por parte del Estudiante:

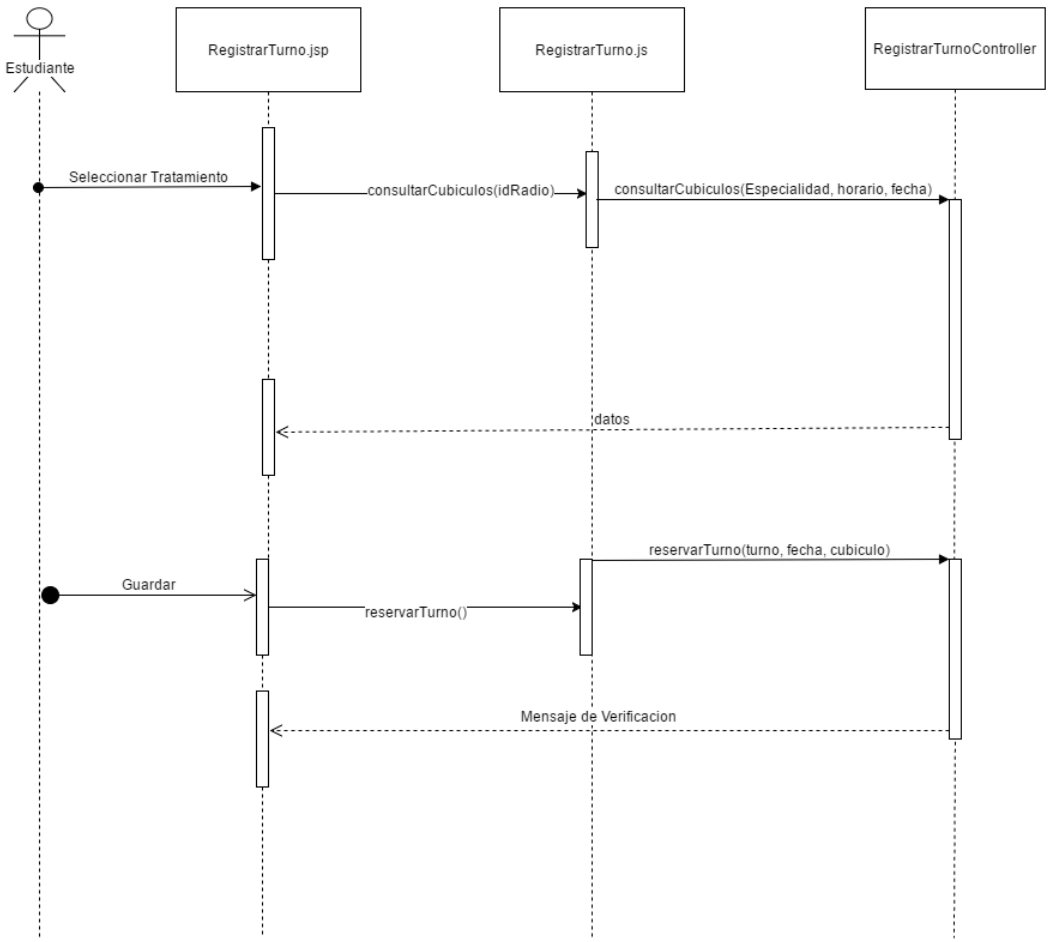

Figura 12. Diagrama de secuencia Registro Turno 2

Crear el modelo conceptual de la base de datos

El modelo conceptual de la base de datos se creó una vez levantados y analizados los requerimientos. A partir del diagrama de clases se pudo implementar el modelo entidad-relación de la base de datos para su posterior implementación.

En la figura 13, podemos observar el diagrama entidad-relación que, en primeras instancias y en el ambiente de desarrollo, se implementó en MySQL.

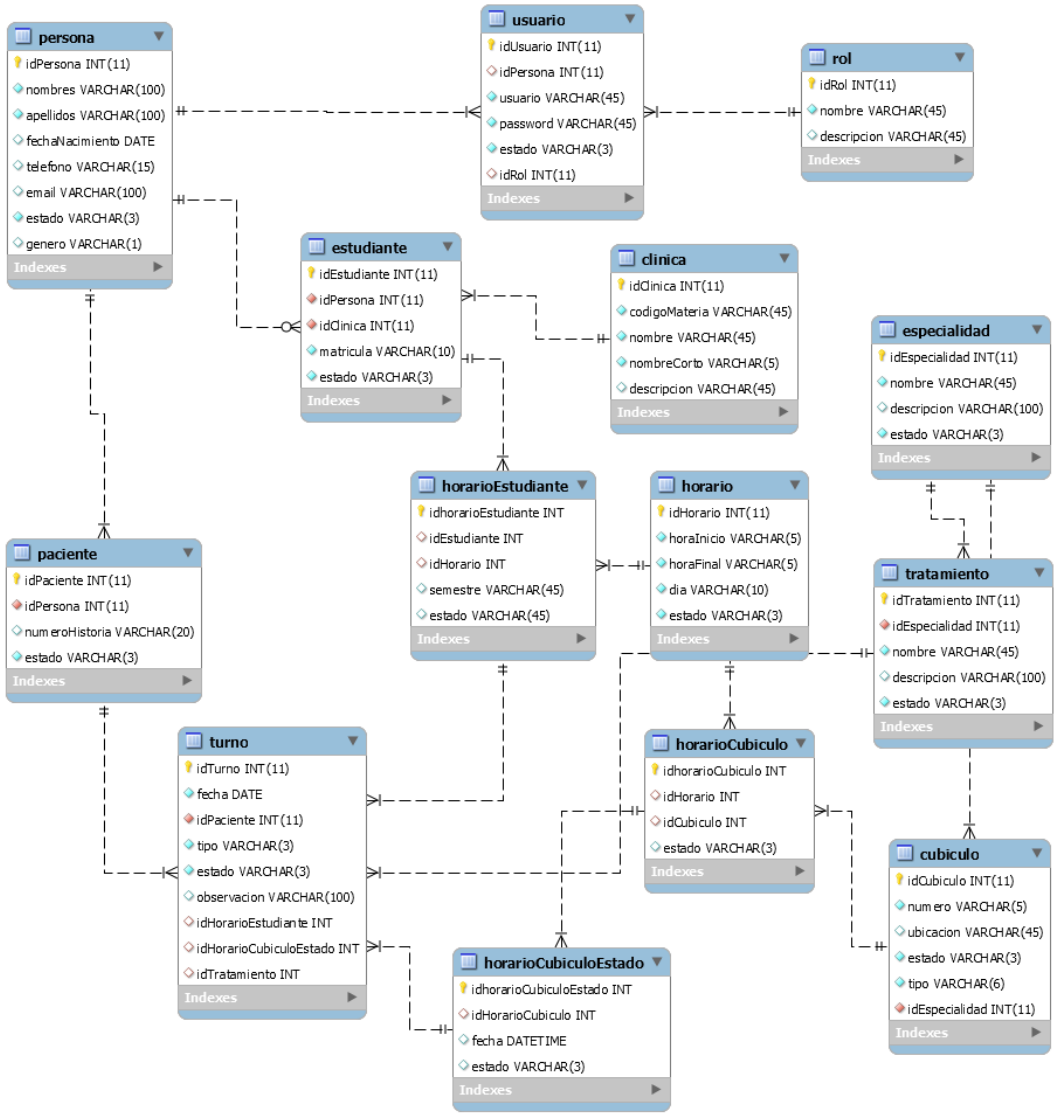

Figura 13. Modelo Entidad-Relación

Diseño de Landing Page

Todo el Front End de la aplicación fue desarrollado utilizando HTML5, CSS3 y el framework Bootstrap, con el motivo de que este se adapte a todas las pantallas posibles. Se seleccionó un tema libre que utiliza estas tecnologías y se adaptó a las necesidades del sistema, dando como resultado las figuras 14 y 15.

En la figura 14, se muestra la primera pantalla que los usuarios verán al ingresar al sistema, en donde se presenta la opción de ingresar al sistema ya sea por login tradicional o usando una cuenta de Google:

Figura 14. Landing Page Login

En la figura 15, como parte del Landing Page se puede crear una cuenta de la manera tradicional al no utilizar una cuenta de Google y en donde se debe ingresar los datos del usuario como son: Nombres, Apellidos, E-mail y su la contraseña para el sistema:

Figura 15. Landing Page Crear Cuenta

Diseño de dashboards por rol

Para cada rol es necesario un dashboard específico, debido a que cada uno de los usuarios tendrá acciones distintas. Los diseños de los dashboards se presentan en las figuras 16 y 17.

En la figura 16, se muestra el dashboard del Estudiante en donde se presentará la información registrada así como los turnos reservados, ocupados y cancelados. A la izquierda se detallan las demás actividades que el Estudiante puede realizar en el sistema.

Figura 16. Dashboard Estudiante

En la figura 17, de la misma manera que el Estudiante, se mostrará la información del Administrador y los turnos reservados, ocupados y cancelados para toda la clínica. A la izquierda se muestran las actividades que puede realizar el Administrador:

Figura 17. Dashboard Administrador

Diseño de módulos por rol

Debido a que cada usuario tiene una acción específica, es necesarios módulos distintos, en las figuras 18, 19, 20, 21 y 22 se muestra el diseño de cada uno de ellos.

En la figura 18 se muestra el formato de las pantallas de mantenimiento de la solución por parte del Administrador en donde a primera vista se presenta la lista de registros de la tabla, a la derecha existen los botones de Actualizar y

Eliminar que modifican los registros y en la parte superior derecha existe el botón de Nuevo que despliega una pantalla para ingresar nuevos registros. Además existe una barra de filtrado de registros, en la cual se puede buscar algún registro en específico.

Figura 18. Módulo de Mantenimiento

En la figura 19, se muestra el primer paso para el registro de turnos por parte del Estudiante, en donde se debe ingresar los datos del paciente. En el caso de que el paciente ya haya asistido a la clínica previamente se debe ingresar solamente el número de historia clínica.

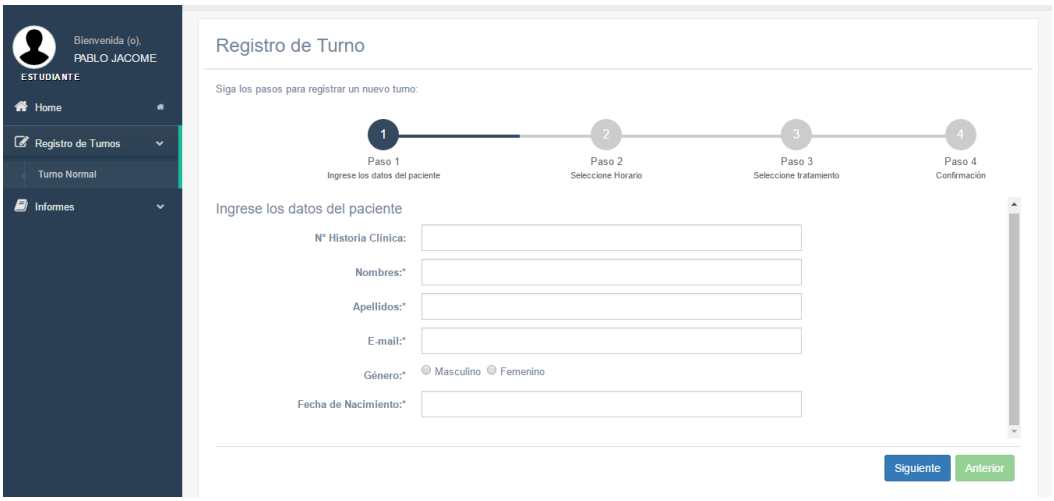

Figura 19. Registro Turnos 1

En la figura 20, se presenta el segundo paso de la reserva de turnos en donde se presenta los horarios ingresados para el Estudiante previamente y se debe seleccionar la fecha en la que se reservará el turno.

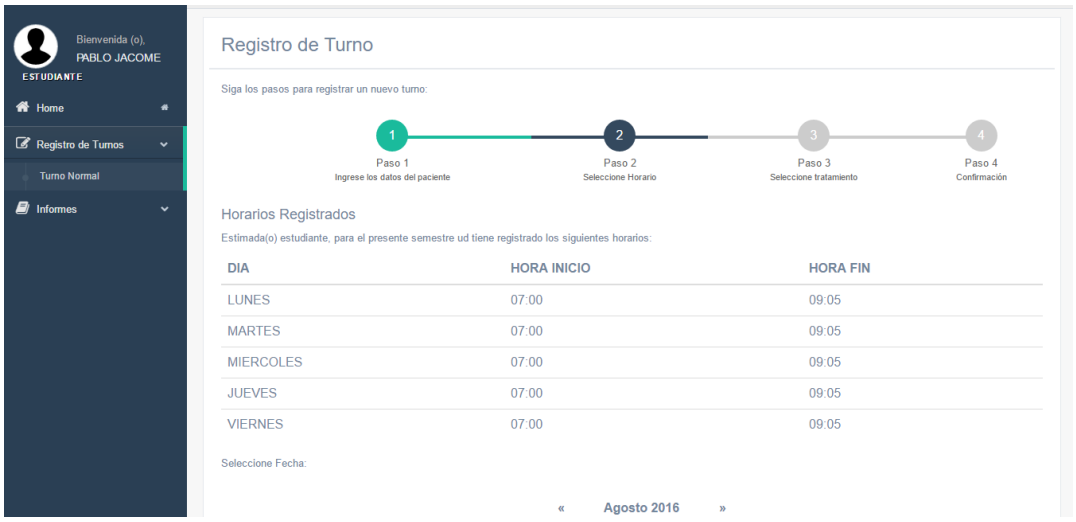

Figura 20. Registro de Turnos 2

En la figura 21, se presenta el tercer paso del registro de turnos en donde se presenta las especialidades y tratamientos que están registrados en la clínica. Se debe seleccionar un tratamiento para continuar con los siguientes pasos.

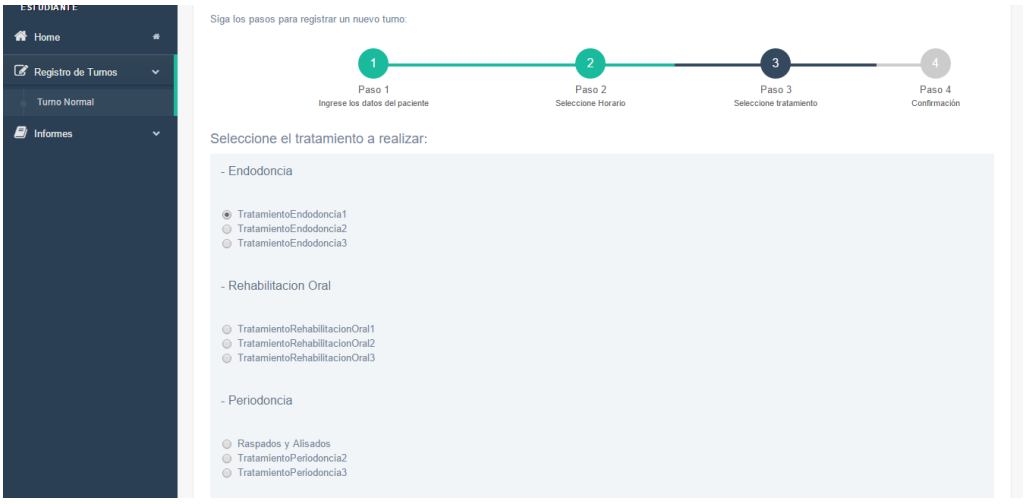

Figura 21. Registro de Turnos 3

En la figura 22, se presenta el cuarto y último paso del registro de turnos en donde se presenta toda la información ingresada previamente y si existen cubículos disponibles se procede a guardar el turno, caso contrario se presenta un mensaje de falta de cubículos disponibles.

Figura 22. Registro de Turnos 4

2.2.3.2. SPRINT II

El Sprint II se inició el 02/05/2016 y terminó el 17/05/2016 con una duración total de 12 días. Para este Sprint se diseñaron e implementaron los siguientes requerimientos: Login Google y Tradicional, Registrar Datos Personales (HU01) y Registrar Datos Paciente (HU02) considerados de alta prioridad.

El objetivo de este Sprint es dar funcionamiento a las primeras pantallas del sistema, además de incorporar la integración del Identity Tool Kit de Google en la aplicación. En la tabla 13 encontramos la descripción de este Sprint.

Tabla 13. Segundo Sprint

Orden	Requerimiento	Tarea	Estimación
1	Login Google y Tradicional	1.1 Login usando el API de Google	13
		1.2 Login tradicional	5
2	Registrar Datos Personales (HU01)	2.1 Crear un método que permita buscar un estudiante de acuerdo a su número de matrícula	3
		2.2 Crear un método que permita actualizar los datos del estudiante	3
		2.3 Validar los datos ingresados	5
		2.4 Crear un método que permita ingresar datos, en caso de que la búsqueda no devuelva ningún resultado	3
		Tarea	Estimación
		2.5 Mostrar un mensaje de confirmación al guardar/actualizar la información personal del estudiante	3
3	Registrar Datos Paciente (HU02)	3.1 Crear un método que permita buscar un paciente de acuerdo a su número de historia clínica	3
		3.2 Crear un método que permita actualizar los datos del paciente	3
		3.3 Validar los datos ingresados	5
		3.4 Crear un método que permita ingresar el paciente, en caso de que la búsqueda no devuelva ningún resultado.	3
		Tarea	Estimación
		3.5 Mostrar un mensaje de confirmación al guardar/actualizar la información personal del paciente.	3
TOTAL			52

Adaptado de (Palacio, 2016)

Para la implementación de este Sprint se necesitó seguir el tutorial de la documentación del Google Identity Kit. (Google Developers, s.f.) En la figura 23 se muestra el método encargado de la integración:

```
//Login Google
function onSignIn(googleUser) {
      var profile = googleUser.getBasicProfile();
        tipoConsulta = "loginGoogle";
        nombres=profile.getGivenName()
        apellidos=profile.getFamilyName();
        email=profile.getEmail();
        contrasena=profile.getId();
        $.ajax({
            url : '../IndexController',
            data : {
                "nombres" : nombres,
                "apellidos" : apellidos,
                "email" : email,
                "contrasena" : contrasena,
                "tipoConsulta" : tipoConsulta
            },
            type : 'POST',
            datatype : 'json',
            success : function (data) {
                window.location = "dashboard.jsp";
            }
        });
    }//Fin LoginGoogle
```

Figura 23. Método Login Google JS

El resto de métodos y funcionalidad utilizada son programación java tradicional y que serán adjuntados en el CD de anexos, además que se encuentra todo el código del proyecto en la siguiente dirección:¿ https://github.com/AecKz/siscoudla

Resultado:

En la figura 24, se muestra la pantalla de login Google, donde se usa la API de autenticación OAuth que es compatible con el Google App Engine y permite obtener la información del usuario para registrarla en la base de datos del sistema, como se muestra en la figura 25.

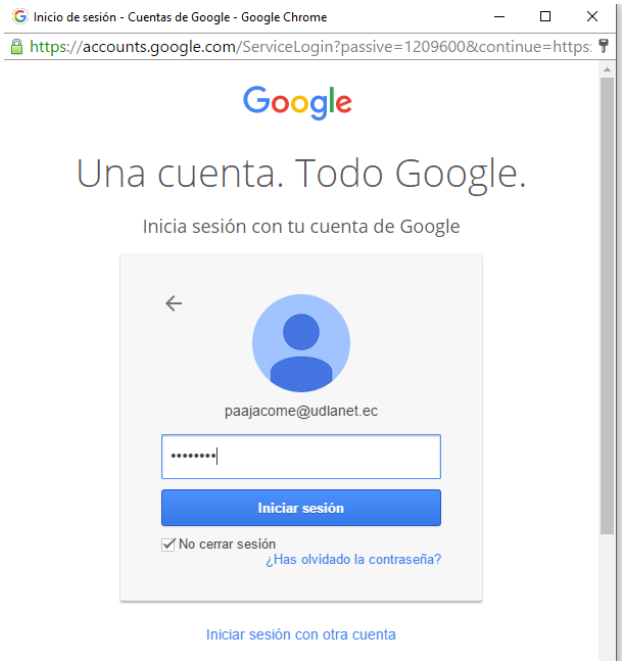

Figura 24. Autenticación Google

idPersona	nombres	apellidos	fechaNacimiento	telefono	email	estado	genero
1	PABLO	JACOME	1990-09-06	0984024446	paajacome@udlanet.ec	ACT	M

Figura 25. Registro en la Base de Datos

2.2.3.3. SPRINT III

El Sprint III se inició el 18/05/2016 y terminó el 31/05/2016 con una duración total de 12 días. Para este Sprint se diseñaron e implementaron los siguientes requerimientos: Registrar Turno (HU03), Revisión de turnos, cubículos e impresión de lista. (HU04) Generación de Reportes (HU05) considerados de alta prioridad. Al completar estas historias se obtiene el entregable funcional: Proceso Core del sistema.

El objetivo de este Sprint es completar el funcionamiento del sistema Core. En la tabla 14 se describe este Sprint.

Tabla 14. Tercer Sprint

Orden	Requerimiento	Tarea	Estimación
1	Registrar Turno (HU03)	1.1 Ingresar datos de paciente y validarlos	5
		1.2 Presentar Horarios del Estudiante	8
		1.3 Presentar Especialidades y Tratamientos	8
		1.4 Presentar cubículo y registro de turno	8
2	Revisión de turnos, cubículos e impresión de lista. (HU04)	2.1 Crear un método de consulta de turnos	5
		2.2 Presentar turnos reservados, ocupados y cancelados al Estudiante	5
		2.3 Presentar turnos reservados al Coordinador (Toma de Lista)	3
		Tarea	Estimación
		2.4 Presentar turnos reservados, ocupados y cancelados al Administrador	3
3	Generación de Reportes (HU05)	3.1 Reporte de Turnos	3
		3.2 Reporte de Estudiantes	3
		3.3 Reporte varios	3
TOTAL			54

Adaptado de (Palacio, 2016)

En este Sprint se desarrollaron varios métodos para el proceso de reservar el turno, el método principal de este Sprint lo podemos observar en la figura 26:

```
function reservarTurno(){
        //Ocultamos el boton
        $('#btnAceptar').hide();
        //Enviamos datos del paciente
        var historiaClinica = $('txtHistoriaClinica').val();
        var nombresPaciente = $('#txtNombresPaciente').val();
        var apellidosPaciente = $('#txtApellidosPaciente').val();
        var emailPaciente = $('#txtEmailPaciente').val();
        var generoPaciente = $('#genero input:radio:checked').val();
        var fechaNacimientoPaciente =  $('#fechaNacimientoPaciente').datepicker('getDate');
        var fechaSeleccionada = $('#fechaSeleccionada').datepicker('getDate');
        var cubiculoAsignadoTurno = $('#lblCubiculoAsignado').text();
        var tratamientoSeleccionado = $('#lblnTratamiento').text();
        $.ajax({
            url : '../RegistroTurnosController',
            data : {
                "tipoConsulta" : "reservarTurno",
                "historiaClinica":historiaClinica,
                "nombresPaciente":nombresPaciente,
                "apellidosPaciente":apellidosPaciente,
                "emailPaciente":emailPaciente,
                "generoPaciente":generoPaciente,
                "fechaNacimientoPaciente":fechaNacimientoPaciente,
                "fechaSeleccionada":fechaSeleccionada,
                "fechaNacimientoPaciente":fechaNacimientoPaciente,
                "tratamientoSeleccionado":tratamientoSeleccionado,
                "cubiculoAsignadoTurno" : cubiculoAsignadoTurno
            },
            type : 'POST',
            datatype : 'json',
            success : function(data) {
                var resultado = data.resultado;
                switch (resultado)
                {
                    case "ok":
                        alert('Turno Reservado!');
                        break;
                    case "error":
```

Figura 26. Método Reservar Turno JS

El resto de métodos y funcionalidad utilizada son programación java tradicional y que serán adjuntados en el CD de anexos, además que se encuentra todo el código del proyecto en la siguiente dirección: https://github.com/AecKz/siscoudla

Resultado:

En las siguientes figuras se muestran los resultados del flujo de registrar turno. Al registrar los datos, seleccionar los tratamientos y horarios correspondientes se pueden presentar 2 pantallas: la figura 27 en la cual el sistema le ha asignado un cubículo o la figura 28 en la cual el sistema le muestra un mensaje de advertencia. Para el caso de existir cubículos disponibles, al presionar el botón de “Aceptar” se mostrará la figura 29. En la figura 30, se muestra la pantalla de informes de turnos.

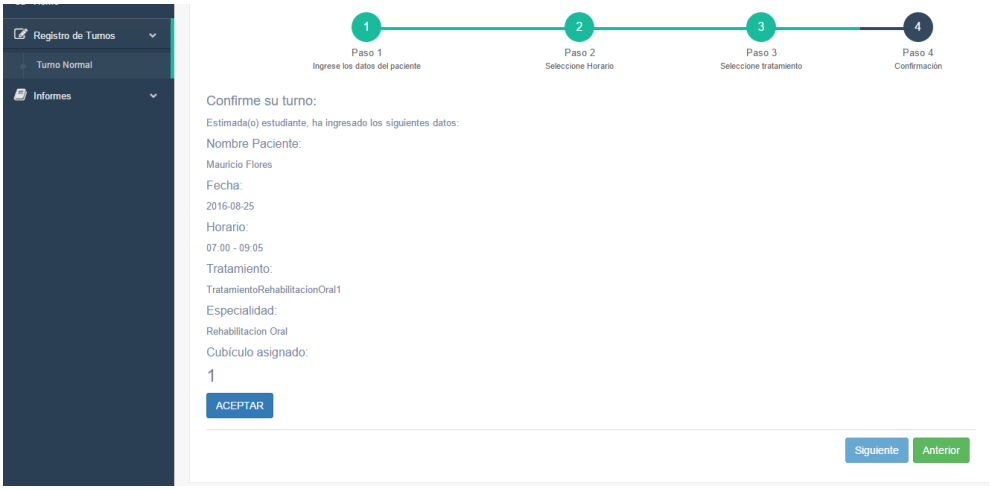

Figura 27. Cubículo Asignado

Figura 28. Advertencia Cubículos No Disponibles

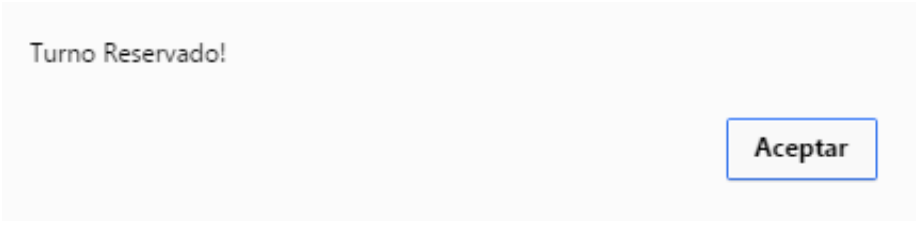

Figura 29. Mensaje de Reserva

Figura 30. Pantalla de Informes

2.2.3.4. SPRINT IV

El Sprint IV se inició el 01/06/2016 y terminó el 16/06/2016 con una duración total de 12 días. Para este Sprint se diseñaron e implementaron los siguientes requerimientos: Creación de Coordinadores y Administradores (HU06), Administración de tablas del sistema (HU07) y Manejo de Sesiones y Roles considerados de media y baja prioridad.

El objetivo de este Sprint es terminar el desarrollo del sistema y comenzar las pruebas de validación con el cliente final. En la tabla 15, se muestra la descripción de este Sprint.

Tabla 15. Cuarto Sprint

Orden	Requerimiento	Tarea	Estimación
1	Creación de Coordinadores y Administradores (HU06)	1.1 Definir forma de creación de coordinadores y administradores	5
		1.2 Funcionamiento y administración de roles	8

2	Administración de tablas del sistema (HU07)	2.1 Administración de tablas catálogos	8
		2.2 Administración de tablas core	8
3	Manejo de Sesiones y Roles	3.1 Crear módulo de autenticación	8
		3.2 Crear usuarios y roles	3
		3.3 Configurar usuarios y roles	3
		3.4 Realizar pruebas	8
	TOTAL		51

Adaptado de (Palacio, 2016)

En este Sprint se reutilizaron los métodos de consulta y presentación implementados anteriormente, además que se utiliza la tecnología JPA para facilitar el desarrollo de las pantallas de mantenimiento. En la figura 31, se muestra uno de los métodos encargados del mantenimiento de tablas.

```
            /* Inicio Controles Actualizar Registro*/
            $(".actualizar-btn").bind({click: function() {
                    $("#addButton").trigger("click");
                    $("#codigo").val($(this).parent().children().first().val());
                    var elem = $(this).parent();
                    var bandera = 1;
                    do {
                        elem = elem.prev();
                        if (elem.is("td")){
                            var elemCode = elem.attr("relation");
                            elementType(elem.text(), elemCode, $("#"+elemCode).attr("type"));
                        }else {
                            bandera = 0;
                        }
                    } while (bandera == 1);
                }
            });
            /* Fin Controles Actualizar Registro*/

            /* Inicio Controles Eliminar Registro */
            $(".eliminar-btn").bind({click: function() {
                    var r = confirm("Seguro que desea eliminar el Horario " + $(this).parent().parent().children().first().text());
                    if (r == true){
                        codigo = $(this).parent().children().first().val();
                        horaInicio = ""; horaFinal = ""; dia="";
                        tipoConsulta = "eliminar";
                        enviarDatos(codigo, horaInicio, horaFinal, dia, tipoConsulta);
                        $(this).parent().parent().remove();
                    }
                }
            });
            /* Fin Controles Elminar Registro */
        }else{
            $("#dataTableContent").append("<tr><td colspan='2'>No existen Registros</td></tr>");
        }
    }
});
```

Figura 31. Método de Mantenimiento JS

El resto de métodos y funcionalidad utilizada son programación java tradicional y que serán adjuntados en el CD de anexos, además que se encuentra todo el código del proyecto en la siguiente dirección: https://github.com/AecKz/siscoudla

Resultado:

En la figura 32, se muestra una de las pantallas de mantenimiento de las tablas del sistema, en donde a primera vista se presenta la lista de registros de la tabla, a la derecha existen los botones de Actualizar y Eliminar que modifican los registros y en la parte superior derecha existe el botón de Nuevo que despliega una pantalla para ingresar nuevos registros (figura 33). Además existe una barra de filtrado de registros, en la cual se puede buscar algún registro en específico.

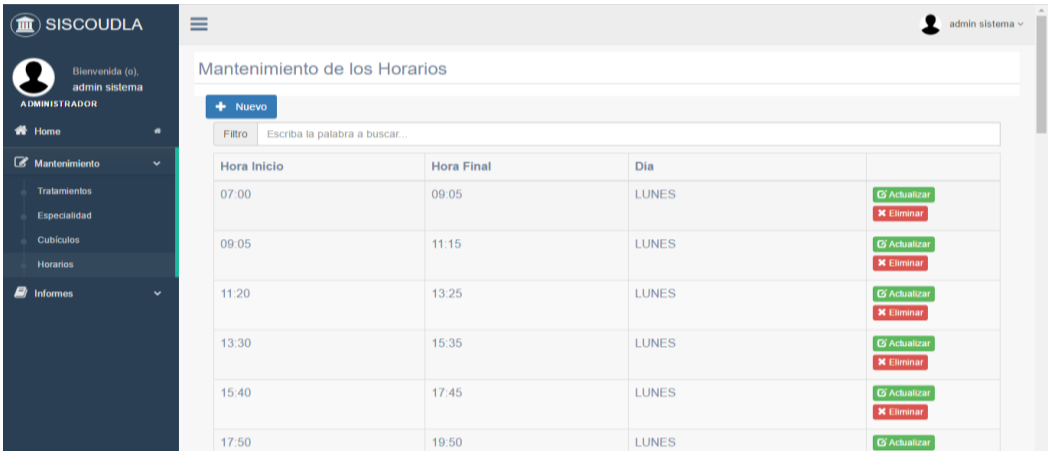

Figura 32. Mantenimiento de Horarios

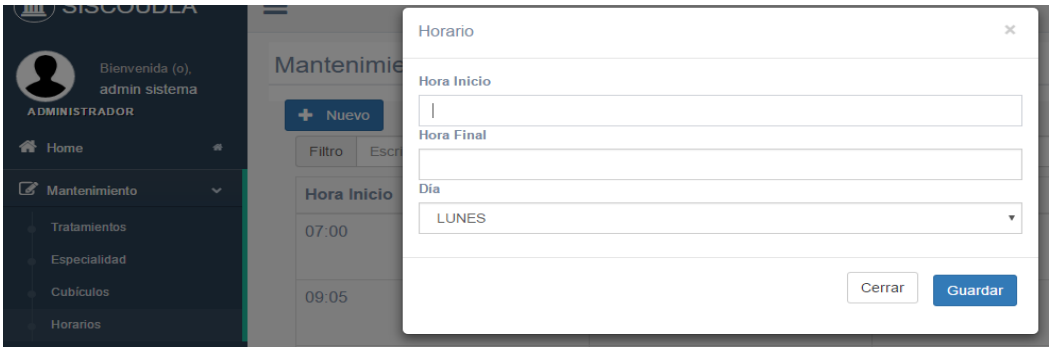

Figura 33. Pantalla de Nuevo Registro

2.3. Sprint Review

El propósito del Sprint Review es ir verificando el desempeño de cada uno de los Sprints y validar el cumplimiento de los requerimientos detallados en las historias de usuario. Se utilizará el formato descrito en la tabla 16, en donde:

- **Número:** Secuencial de la prueba de aceptación.
- **Historia de Usuario:** El número de la historia de usuario correspondiente.
- **Prerrequisitos:** Requisitos previos para realizar la prueba.
- **Resultado:** El resultado de la prueba.
- **Evaluación:** Calificación de la prueba.

Tabla 16. Formato Aceptación

Prueba de Aceptación **Número**
Historia de Usuario:
Prerrequisitos:
Pasos de Ejecución:
Resultado:
Evaluación:

Tomado de (Palacio, 2016)

A continuación se presenta las pruebas de aceptación de cada una de las historias, tablas 17, 18, 19, 20, 21,22 y 23, de usuario utilizando el formato descrito en la tabla 16.

Tabla 17. Aceptación de Registrar datos personales

Prueba de Aceptación **PA01**
Historia de Usuario: Registrar datos personales. (HU01)
Prerrequisitos: Arquitectura del Sistema, Diseño del Front End
Pasos de Ejecución: 1. Comprobar si el estudiante está registrado en el sistema realizando una consulta mediante su número de matrícula. 2. Si el estudiante está registrado, desplegar los datos y actualizar los mismos. 3. Si el estudiante no está registrado, se debe ingresar sus nombres, apellidos, clínica a la que pertenece, correo electrónico, número de identificación, número de matrícula y teléfono. 4. Validar los datos ingresados 5. Guardar los datos en las tablas correspondientes.
Resultado: La información del estudiante queda registrada.
Evaluación: Prueba Exitosa

Tabla 18. Aceptación de Registrar datos paciente

Prueba de Aceptación **PA02**
Historia de Usuario: Registrar Datos Paciente (HU02)
Prerrequisitos: Arquitectura del Sistema, Diseño del Front End
Pasos de Ejecución: 1. Comprobar si el estudiante ha ingresado al sistema. 2. Comprobar si el paciente existe en el sistema haciendo una consulta usando su número de identificación. 3. Si el paciente no se encuentra registrado en el sistema, se debe ingresar sus nombres, apellidos, número de historia si la tiene, correo electrónico, número de identificación, género y teléfono. 4. Validar los datos registrados. 5. Guardar los datos en las tablas correspondientes.
Resultado: La información del paciente queda registrada.
Evaluación: Prueba Exitosa

Tabla 19. Aceptación de Registro de turno

Prueba de Aceptación **PA03**
Historia de Usuario: Registro de turno. (HU03)
Prerrequisitos: HU01, HU02
Pasos de Ejecución: 1. Se debe comprobar que el estudiante ha ingresado al sistema. 2. Se debe validar el horario que seleccionó y que este le corresponda. 3. El estudiante debe seleccionar el tratamiento que va a realizar y según su elección, el sistema le concederá un cubículo libre. 4. Validar toda la información registrada. 5. Guardar la información en las tablas correspondientes.
Resultado: La información del turno queda registrada.
Evaluación: Prueba Exitosa

Tabla 20. Aceptación de Revisión de turnos

Prueba de Aceptación **PA04**
Historia de Usuario: Revisión de turnos, cubículos e impresión de lista. (HU04)
Prerrequisitos: HU03
Pasos de Ejecución: 1. El administrador del sistema debe crear un usuario coordinador con el rol respectivo. 2. El usuario coordinador debe ingresar al sistema. 3. Se debe comprobar que el usuario esté registrado en el sistema y tenga asignado el rol correspondiente. 4. El sistema generará la información de los turnos registrados. 5. Existirá una opción de impresión, con la lista de estudiantes, pacientes y tratamiento por turno.
Resultado: La información registrada es la correcta y se puede imprimir la lista.
Evaluación: Prueba Exitosa

Tabla 21. Aceptación de Generación de Reportes

Prueba de Aceptación **PA05**
Historia de Usuario: Generación de reportes. (HU05)
Prerrequisitos: HU04
Pasos de Ejecución: 1. El administrador del sistema debe crear un usuario con el rol respectivo. 2. El usuario administrador debe ingresar al sistema. 3. Se debe comprobar que el usuario esté registrado en el sistema y tenga asignado el rol correspondiente. 4. El sistema generará la información de los turnos registrados. 4. Existirá una opción de impresión, con la lista de estudiantes, pacientes y tratamientos.
Resultado: La información registrada es la correcta y se puede generar reportes.
Evaluación: Prueba Exitosa

Tabla 22. Aceptación de Coordinadores y Administradores

Prueba de Aceptación **PA06**
Historia de Usuario: Creación de Coordinadores y Administradores. (HU06)
Prerrequisitos: Ninguno
Pasos de Ejecución: 1. El Súper usuario (Desarrollador) debe crear un usuario administrador del sistema con el rol respectivo. 2. El usuario administrador del sistema debe ingresar al sistema. 3. Se debe comprobar que el usuario esté registrado en el sistema y tenga asignado el rol correspondiente. 4. El usuario podrá crear usuarios coordinadores y administradores. 5. Se validará la información registrada. 6. Guardar la información registrada en las tablas correspondientes.
Resultado: La información queda registrada.
Evaluación: Prueba Exitosa

Tabla 23. Aceptación de Administración de tablas del sistema

Prueba de Aceptación **PA07**
Historia de Usuario: Administración de tablas del sistema. (HU07)
Prerrequisitos: Ninguno
Pasos de Ejecución: 1. El usuario administrador del sistema debe ingresar al sistema. 2. Se debe comprobar que el usuario esté registrado en el sistema y tenga asignado el rol correspondiente. 3. El usuario podrá modificar las tablas del sistema según corresponda. 4. Se validará la información registrada. 5. Guardar la información registrada en las tablas correspondientes.
Resultado: La información queda registrada.
Evaluación: Prueba Satisfactoria

2.4. Análisis del Proyecto

Al finalizar el desarrollo e implementación de la solución SISCOUDLA se puede realizar el siguiente análisis:

Al iniciar el proyecto se planificaron 39 tareas las cuales fueron repartidas y cumplidas en cada Sprint.

2.4.1. Burndown Chart

El burndown chart nos permite observar el avance del proyecto. Para cada uno de los Sprints se tiene un burndown chart final:

2.4.1.1. Sprint I

En la figura 34 se muestra el Burndown chart de este Sprint:

Figura 34. Burndown Chart Sprint I

Como se puede observar en la figura 34, se cumplieron con los objetivos del Sprint en el tiempo adecuado. En este Sprint se completaron 55 Story Points en 12 días. Observamos en la figura que la línea de progreso es casi igual a la

ideal con un cambio a partir del día 9, pues a partir de ese día se trabajó menos puntos de historia que los previstos debido a que requerían menos esfuerzo que el planificado.

2.4.1.2. Sprint II

En la figura 35 se muestra el Burndown chart de este Sprint:

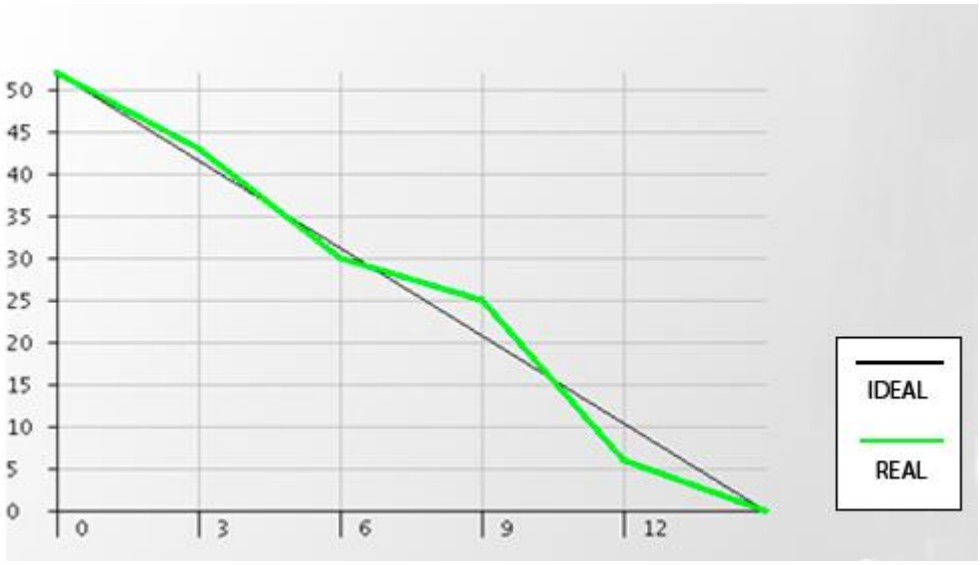

Figura 35. Burndown Chart Sprint II

Como se puede observar en la figura 35, se cumplieron con los objetivos del Sprint en el tiempo adecuado. En este Sprint se completaron 52 Story Points en 12 días. Observamos en la figura que existe una gran diferencia entre la línea de progreso y la ideal a partir del día 6 debido a que se realizó mayor esfuerzo creando métodos y pruebas unitarias hasta el día 10. Para los días 11 y 12, el esfuerzo fue menor y por lo tanto la línea decrece.

2.4.1.3. Sprint III

En la figura 36 se muestra el Burndown chart de este Sprint:

Figura 36. Burndown Chart Sprint III

Como se puede observar en la figura 36, se cumplieron con los objetivos del Sprint en el tiempo adecuado. En este Sprint se completaron 54 Story Points en 12 días. Observamos en la figura que la línea de progreso está por debajo de la línea ideal hasta el día 9 debido a que el desarrollo de las historias de usuario requirió menos esfuerzo al planificado ya que se tomó como base los métodos y pruebas unitarias realizadas en el Sprint II.

2.4.1.4. Sprint IV

En la figura 37 se muestra el Burndown chart de este Sprint:

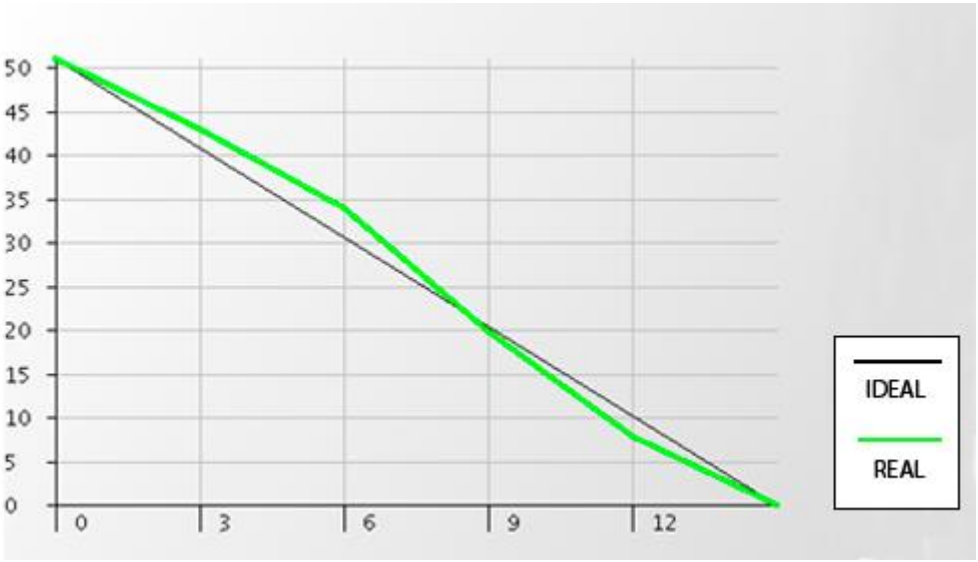

Figura 37. Burndown Chart Sprint IV

Como se puede observar en la figura 37, se cumplieron con los objetivos del Sprint en el tiempo adecuado. En este Sprint se completaron 51 Story Points en 12 días. Observamos en la figura que la línea de progreso está por encima de la línea ideal debido a que se requirió mayor esfuerzo al implementar las historias de usuario ya que se crearon métodos nuevos con lógica diferente a los Sprints anteriores, sin embargo para el día 7 la línea decrece pues los métodos ya fueron implementados y se requirió menos esfuerzo.

2.5. Pruebas Unitarias

El sistema SISCOUDLA, fue desarrollado utilizando metodología Scrum y lenguaje de programación Java. Para lo cual, siguiendo con las normativas y parámetros de desarrollo se utiliza la herramienta JUnit, con la finalidad de realizar tests para cada uno de los métodos que utiliza el sistema. Estos tests se adjuntan en el Anexo 1 y Anexo 2. El Anexo 1, presenta el Test del flujo principal de reserva de turnos para un determinado estudiante y en un horario

específico. El Anexo 2, presenta cada uno de los métodos necesarios para el funcionamiento de cada módulo o pantalla del sistema. En ambos anexos se adjunta la captura de pantalla de los resultados de las pruebas.

3. Capítulo III. Validación de Objetivos

3.1 Cumplimiento de Objetivos

- Se realiza el levantamiento de casos, análisis, diseño e implementación de una solución Cloud para la clínica odontológica de la Universidad de las Américas, utilizando metodología Scrum y las tecnologías Cloud de Google:

El sistema “SISCOUDLA” se encuentra desplegado en un ambiente de Producción bajo la siguiente dirección: http://siscoudla.appspot.com/ , en donde se encuentra listo para ser usado por la Clínica Odontológica de la Universidad de las Américas y sus estudiantes.

- Se comprueba la funcionalidad y versatilidad que presentan las tecnologías Cloud de Google para el desarrollo de sistemas complejos, utilizando software y herramientas libres:

El sistema “SISCOUDLA” fue desarrollado utilizando el lenguaje de programación JAVA, utilizando frameworks y tecnologías libres. De la misma manera se utilizaron herramientas como el IDE de desarrollo “Eclipse” en donde se le instaló el plugin de desarrollo del Google App Engine además del SDK del App Engine, los cuales permitieron programar y emular el sistema en un ambiente Cloud local. En la figura 38 se muestra el ambiente de programación Cloud local y en la figura 39 la configuración del SDK para el sistema.

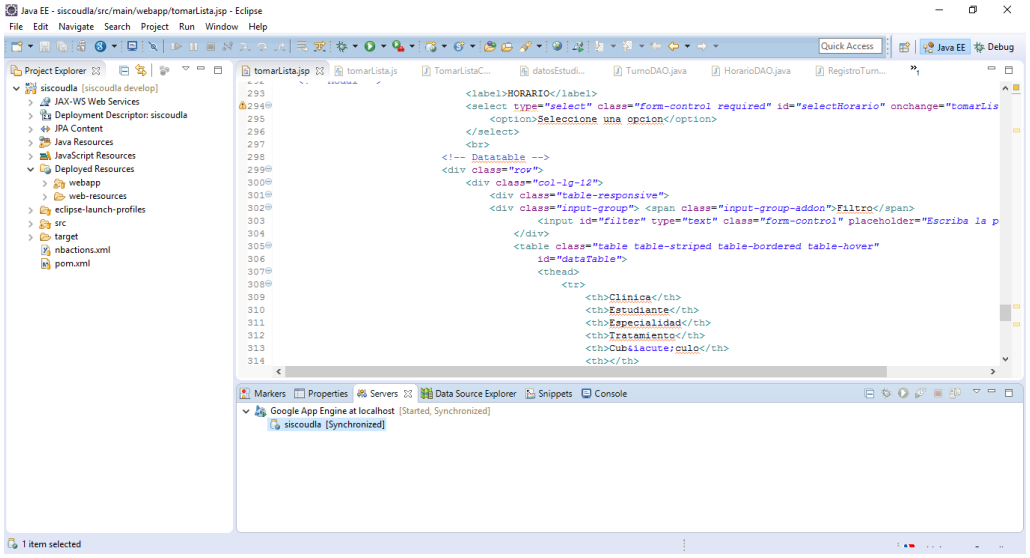

Figura 38. Ambiente de Desarrollo

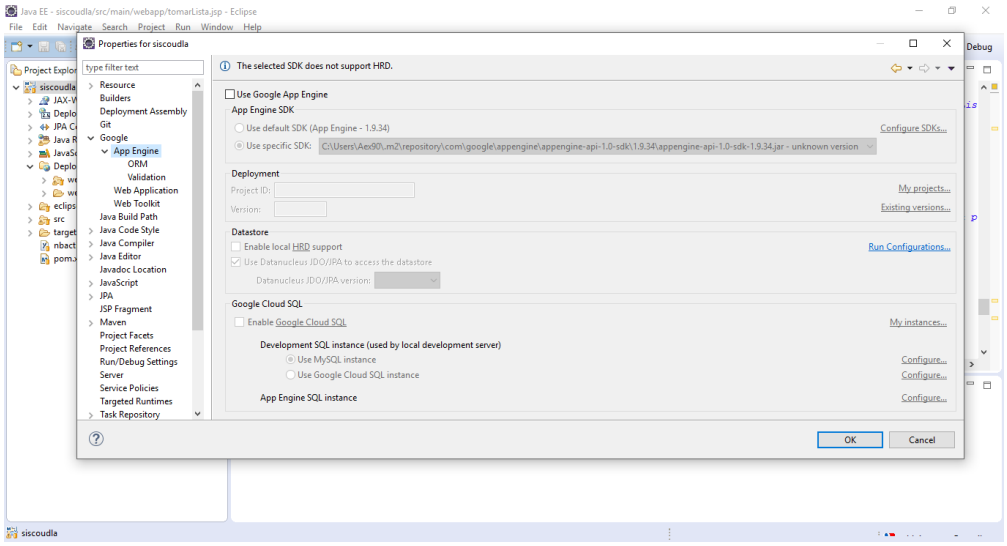

Figura 39. IDE instalado el SDK de Google

- Se comprueba la compatibilidad y beneficios de utilizar una metodología ágil, como es Scrum, para el desarrollo de soluciones Cloud:

Al iniciar el desarrollo del sistema “SISCOUDLA” se realizó una investigación de metodologías que sean compatibles con la programación de sistemas Cloud y se encontró que las metodologías ágiles respondían mejor a la satisfacción

laboral de los equipos de programación como lo observamos en la figura 40. (Palacio, Navegapolis, 2016)

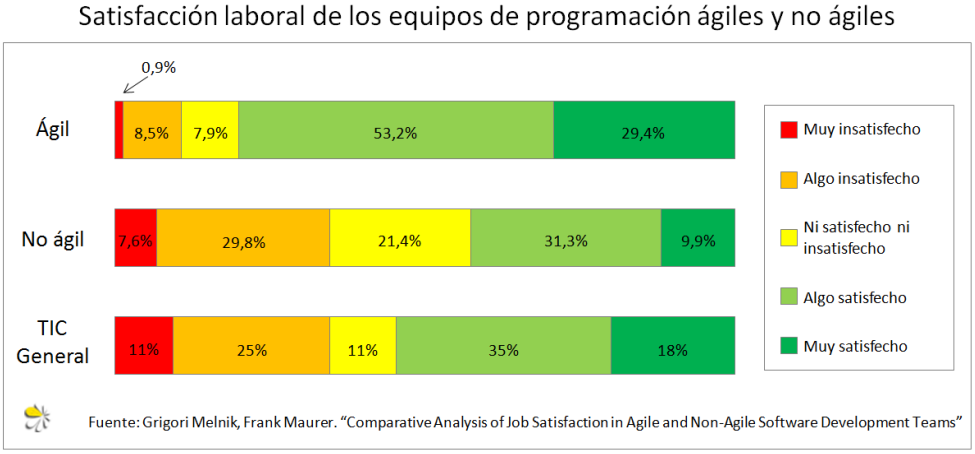

Figura 40.Satisfacción laboral de equipos de programación
Tomado de (Palacio, 2016)

Al finalizar el desarrollo del sistema, se comprobó las estadísticas de la figura 40, pues el sistema fue terminado con satisfacción en 39 tareas repartidas en 4 Sprints de 55, 52, 54 y 51 puntos de historia. Cada Sprint con una duración de 12 días dando un total de 48 días de desarrollo, que para el propósito del proyecto, cumplió con las expectativas.

Cabe recalcar que la implementación de un sistema de esta complejidad aproximadamente dura 6 meses (ERPToday, s.f.), y que el sistema "SISCOUDLA" tomó aproximadamente 2 meses de los cuales solo 2 días fueron dedicados a configuraciones y levantamiento del sistema en Producción, lo cual permite comprobar que el desarrollo de sistemas Cloud es más ágil y de rápida implementación que los sistemas tradicionales. Además de su compatibilidad con la metodología Scrum.

3.2 Estimación de Costos

Una de las ventajas de los sistemas Cloud es que para su funcionamiento se debe pagar únicamente lo que se consume. Para el sistema "SISCOUDLA" se

realizó una estimación de cuánto costaría mensualmente[1]. Dando un resultado de $388,17 dólares americanos, usando las siguientes especificaciones:

En la figura 41, se muestra la estimación de uso por mes del Compute Engine, que es la infraestructura necesaria para el alojamiento del sistema. Se estima el uso de una máquina virtual, ubicada en Estados Unidos, instalado un disco SSD de 375 GB con un uso total de 730 horas por mes.

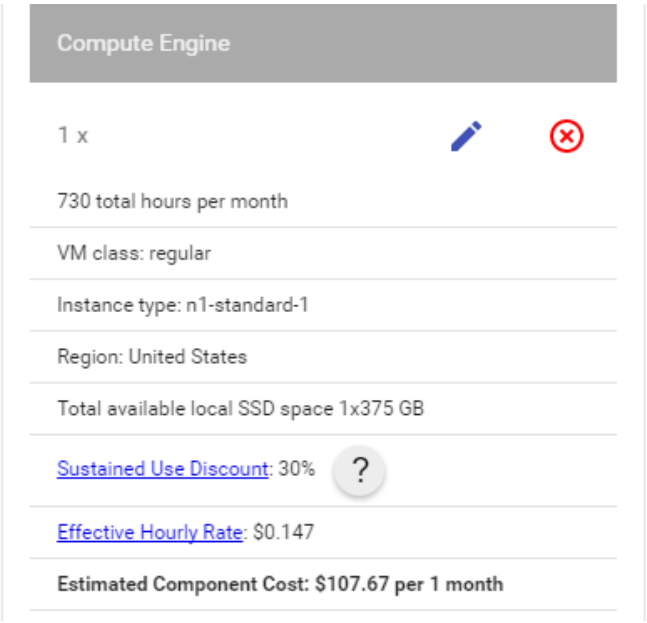

Figura 41. Estimación de Costos Compute Engine

En la figura 42, se muestra la estimación de costos de uso del Google App Engine, que es la plataforma que utilizará el sistema “SISCOUDLA” para su funcionamiento. Se estima el uso de 730 instancias por mes con un tráfico de 100 Gb y un almacenamiento de 5 Gb.

[1] Costos estimados para Agosto del 2016, utilizando la calculadora proporcionada por la plataforma, ubicada bajo la dirección: https://cloud.google.com/products/calculator

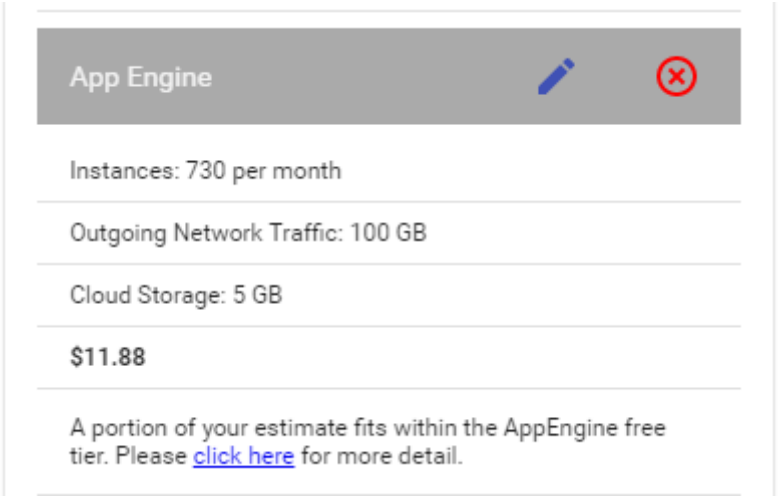

Figura 42. Estimación de Costos del App Engine

En la figura 43, se muestra la estimación de costos del Cloud SQL que es la base de datos que utilizará el sistema “SISCOUDLA” para su funcionamiento. Se estima el uso de 730 horas por mes con un almacenamiento de 500 Gb y utilizando una instancia.

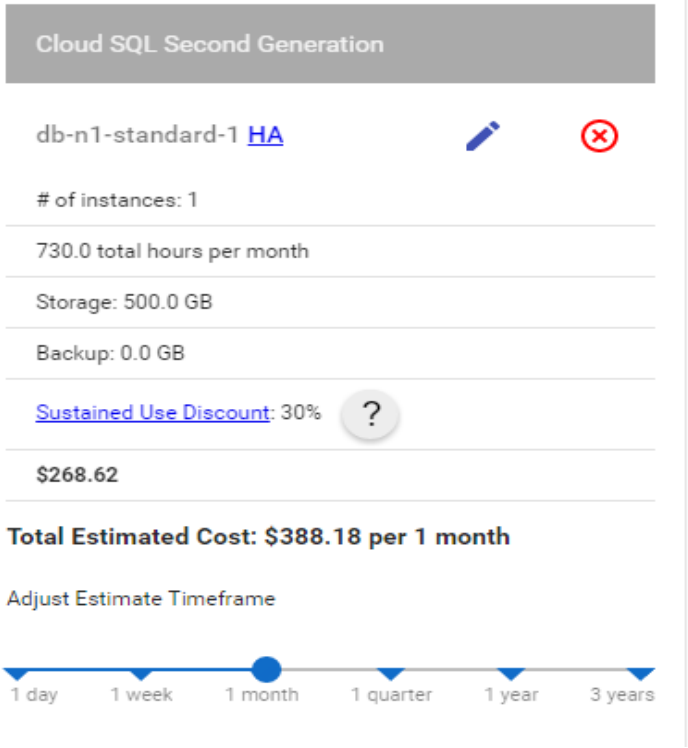

Figura 43. Estimación de Costos de SQL

Con esto se comprueba que implementar un sistema Cloud es más barato que un sistema tradicional pues un sistema tradicional requiere de más elementos y servicios para su funcionamiento y con ello los costos se incrementan.

Cabe recalcar que los precios estimados pueden variar con el uso del sistema y que se realizó esta estimación de uso para los 450 estudiantes que existen actualmente en la clínica. (Gudiño, 2016)

4. Capítulo IV. Conclusiones y Recomendaciones

4.1 Conclusiones

Mediante el uso de una metodología ágil (Scrum), se facilitó la entrega de productos finales de una manera rápida y continua, permitiendo a los interesados percibir el avance del proyecto.

Desarrollar este sistema con un Back End robusto como es Java, facilitó el diseño e implementación con una arquitectura flexible como es SOA, además de lograr una sencilla integración con otras tecnologías que forman parte de cloud.

Utilizar herramientas de software libre (Bootstrap en el front end y JPA en el back end) facilitaron el desarrollo de sistemas cloud al contar con toda la información necesaria provista por los autores.

Para el presente proyecto fue necesario optimizar cada método, transacción y llamadas a la base de datos, debido a que todas estas representaran un coste monetario al usuario final. En consecuencia, desarrollar soluciones cloud permite al programador enfocarse en escribir y optimizar el código, ya que no debe realizar configuraciones o solucionar problemas de infraestructura.

Google App Engine permite el desarrollo de aplicaciones altamente escalables, además de proporcionar al desarrollador una gran variedad de APIs propias de Google que facilitan y optimizan el desarrollo de sistemas.

4.2 Recomendaciones

El sistema utiliza herramientas de software libre bajo el motor Maven, el cual permite la actualización de librerías y dependencias de una manera centralizada. Es recomendable utilizar un motor similar, además de las versiones más estables de las librerías.

El sistema utiliza herramientas de pruebas unitarias como es el JUnit con el cual se pudo agilizar el desarrollo de métodos y procesos completos en el Back End antes de ser implementados en el Front End. Es recomendable utilizar este tipo de test en la etapa de desarrollo pues permite ahorrar tiempo y evitar errores.

Es recomendable utilizar tecnologías compatibles entre sí y respetar las convenciones y patrones de desarrollo que posee una aplicación Java EE.
Para la persistencia de datos en la nube, Google ofrece la posibilidad de usar una de sus plataformas ya sea Google SQL o el Google Data Storage. Es recomendable analizar el impacto y flujo de datos que tendrá el sistema y según eso utilizar la tecnología más adecuada.

Google App Engine permite el desarrollo de aplicaciones y pagar únicamente lo que se consume, pero es necesario optimizar el uso de las transacciones realizadas pues todas ellas tienen un costo. Por lo tanto es recomendable optimizar todas las transacciones que se realicen en Front End y minimizar las llamadas a la base de datos.

Google App Engine permite la integración con varias de las APIs de Google, sin embargo es necesario investigar y analizar cada una de ellas para así seleccionar las más adecuadas para el proyecto. Existen APIs que están instaladas por defecto en el proyecto y deben ser quitadas para que no generen conflictos a futuro.

REFERENCIAS

CBT Nuggets. (s.f.). *Cloud Wars: Amazon (AWS) vs. Google (GCP) vs. Microsoft (Azure).* Recuperado el 16 de enero de 2016, de https://www.youtube.com/watch?v=342KEaxFVjM

Developer. (s.f.). *Java Cloud Development: What Developers Need to Know*. Recuperado el 10 de enero de 2016, de http://www.developer.com/java/java-cloud-development-what-developers-need-to-know.html

ERPToday. (s.f.). *ERPToday*. Recuperado el 08 de enero de 2016, de http://www.erptoday.info/duracion-implementacion-erp/

Google Developers. (s.f.). *Developers*. Recuperado el 02 de enero de 2016, de https://developers.google.com/identity/sign-in/web/sign-in

Gudiño, C. (2016). Levantamiento de Requerimientos del sistema SISCOUDLA. (P. Jacome, Entrevistador)

IslaVisual. (s.f.). *IslaVisual.com*. Recuperado el 07 de enero de 2016, de http://www.islavisual.com/articulos/desarrollo_web/diferencias-entre-scrum-y-xp

MSDN Microsoft. (s.f.). *Chapter 1: Service Oriented Architecture (SOA)*. Recuperado el 12 de mayo de 2016, de https://msdn.microsoft.com/en-us/library/bb833022.aspx

Palacio, J. (2007). *Flexibilidad con Scrum.*

Palacio, J. (2016). *Navegapolis*. Recuperado el 18 de mayo de 2016, de http://navegapolis.com/

Scrum Methology. (s.f.). *An Empirical Framework For Learning (Not a Methodology)*. Recuperado el 05 de marxo de 2016, de http://scrummethodology.com/

Softeng. (s.f.). *¿Qué es la nube y cuáles son los beneficios que ofrece a las empresas?* Recuperado el 12 de mayo de 2016, de https://www.softeng.es/es-es/noticias/newsletter/marzo-abril-2011/que-es-la-nube-cuales-son-los-beneficios-que-ofrece-las-empresas.html

UDLA, Quito. (s.f.). *Facultad de Odontología UDLA*. Recuperado el 15 de mayo de 2016, de https://www.youtube.com/watch?v=o3-SVIMCzsw

Universidad de las Américas. (s.f.). *Vida universitaria servicios clinica odontologica*. Recuperado el 15 de mayo de 2016, de http://www.udla.edu.ec/vida-universitaria/servicios/clinica-odontologica/

ANEXOS

Anexo 1.- JUnit Test

Código:

```
package siscoudla;

import static org.junit.Assert.*;

import java.util.Date;
import java.util.List;
import com.udla.siscoudla.dao.HorarioCubiculoEstadoDAO;
import com.udla.siscoudla.dao.HorarioDAO;
import com.udla.siscoudla.dao.RolDAO;
import com.udla.siscoudla.dao.TratamientoDAO;
import com.udla.siscoudla.modelo.Estudiante;
import com.udla.siscoudla.modelo.Horario;
import com.udla.siscoudla.modelo.Rol;
import com.udla.siscoudla.modelo.Tratamiento;
import com.udla.siscoudla.util.Utilitarios;

public class Test {

	@org.junit.Test
	public void test() {
		//testFlujoTurnoNormal();
	}

	public void testFlujoTurnoNormal() {
		//Test del Flujo de reservar turnos normales
		System.out.println("*******Inicia el Flujo********");
		//1.- Se comprueba si el usuario ingresado es Estudiante
		RolDAO rolDAO = new RolDAO();
		String usuario = "paajacome@udlanet.ec";
```

```
                List<Rol> roles = rolDAO.buscarRolPorUsuario(usuario);
                if(roles.get(0).getNombre().equals("Estudiante")){
                        System.out.println("El usuario: " + usuario +" es
Estudiante");
                        System.out.println("Verificar o crear paciente");
                        //2.- Se crea un paciente si este no existe
                                //2.1-Verificar si existe el paciente a traves de
numero de historia
                                //2.2- Si no existe, crear objeto Persona y Paciente
                        //3.- Se presenta los horarios para el presente semestre del
estudiante
                        HorarioDAO horarioDAO = new HorarioDAO();
                        Estudiante estudiante = new Estudiante ();
                        estudiante.setIdEstudiante(Integer.parseInt("1"));
                        List<Horario> horarios =
horarioDAO.buscarHorariosEstudiante(estudiante);
                        if (!horarios.isEmpty()) {
                                System.out.println("El estudiante registra los
siguientes horarios: ");
                                for (Horario he : horarios) {

                                        System.out.println("Horario: " +he.getDia() +"-
"+ he.getHoraInicio() +"-"+ he.getHoraFinal());
                                }
                        } else {
                                System.out.println("No existen horarios para el
estudiante");
                        }
                        //4.- El estudiante selecciona el tratamiento a realizar
                        //Se presentan los tratamientos categorizados por
especialidad en la pantalla
```

```
                    TratamientoDAO tratamientoDAO = new
TratamientoDAO();
                    int idEspecialidad = 2;
                    List<Tratamiento> tratamientos =
tratamientoDAO.buscarTratamientosEspecialidad(idEspecialidad);
                    if (!tratamientos.isEmpty()) {
                            System.out.println("Existen los siguientes
tratamientos: ");
                            for (Tratamiento he : tratamientos) {
                                    System.out.println("Tratamiento: "
+he.getNombre() +"-"+ he.getEspecialidad().getNombre());
                            }
                    } else {
                            System.out.println("No existen tratamientos");

                    }
                    //5.- Se comprueba si existen cubiculos disponibles para el
turno
                    int idHorario = 1;
                    Date fecha = Utilitarios.stringToDate("2016-05-09");
                    String tipoCubiculo = "NORMAL";
                    HorarioCubiculoEstadoDAO hceDAO = new
HorarioCubiculoEstadoDAO();
                    List<String> cubiculos =
hceDAO.buscarCubiculosLibres(fecha, idEspecialidad, idHorario, tipoCubiculo);

                    if (!cubiculos.isEmpty()) {
                            System.out.println("Existen disponibles los
siguientes cubiculos: ");
                            for (String he : cubiculos) {
                                    System.out.println("Cubiculo: " +he);
                            }
```

```
                } else {
                        System.out.println("No trae resultados");

                }
                //6.- Se asigna un cubiculo al estudiante al azar
                //String cubiculo = cubiculos.get(new
Random().nextInt(cubiculos.size()));
                String cubiculo = cubiculos.get(0);
                System.out.println("Se ha asignado el cubiculo: "+
cubiculo);
                //7.- Si el estudiante acepta, se comprueba una vez mas si
el cubiculo asignado esta Libre
                //Ojo: Es posible que por concurrencia se asigne un
cubiculo, y luego otro usuario lo registre.
                Boolean flag =
hceDAO.verificarEstado(Integer.parseInt(cubiculo), "LIB");
                if(flag){
                        System.out.println("El cubiculo esta libre");
                        //Si el cubiculo está libre se crea el objeto Turno en
estado RESERVADO y se cambia el estado de hce a OCU
                        System.out.println("Se crea el Turno");
                }else{
                        System.out.println("El cubiculo NO esta libre");
                }
                System.out.println("*******Fin del Flujo********");
        }else{
                fail("No es estudiante");
        }
    }

}
```

Resultado:

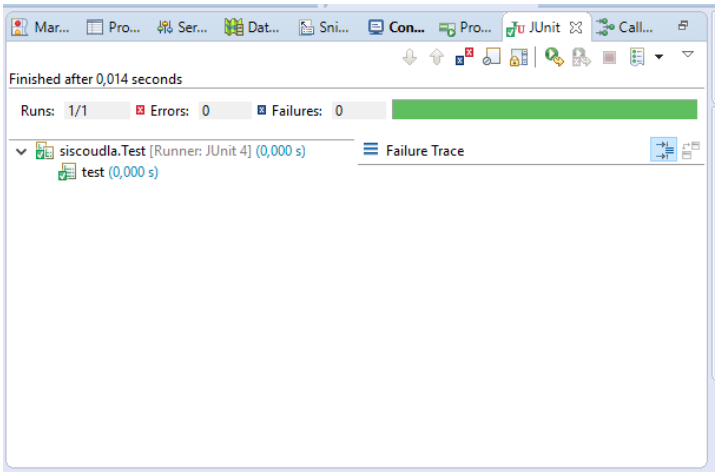

Figura 44. JUnit Test

Anexo 2.- JUnit TestDAO

Código:

```
 package siscoudla;

import static org.junit.Assert.*;

import java.util.Date;
import java.util.List;

import org.junit.Test;

import com.udla.siscoudla.dao.EspecialidadDAO;
import com.udla.siscoudla.dao.EstudianteDAO;
import com.udla.siscoudla.dao.HorarioCubiculoEstadoDAO;
import com.udla.siscoudla.dao.HorarioDAO;
import com.udla.siscoudla.dao.HorarioEstudianteDAO;
import com.udla.siscoudla.dao.PersonaDAO;
import com.udla.siscoudla.dao.RolDAO;
import com.udla.siscoudla.dao.TratamientoDAO;
import com.udla.siscoudla.dao.TurnoDAO;
import com.udla.siscoudla.dao.UsuarioDAO;
import com.udla.siscoudla.modelo.Especialidad;
import com.udla.siscoudla.modelo.Estudiante;
import com.udla.siscoudla.modelo.Horario;
import com.udla.siscoudla.modelo.Horariocubiculoestado;
import com.udla.siscoudla.modelo.Persona;
import com.udla.siscoudla.modelo.Rol;
import com.udla.siscoudla.modelo.Tratamiento;
import com.udla.siscoudla.modelo.Turno;
import com.udla.siscoudla.modelo.Usuario;
```

```
import com.udla.siscoudla.util.Utilitarios;

public class TestDAO {

	@Test
	public void test() {
		System.out.println("Iniciando Test DAO");
//		testBuscarHorariosEstudiante();
//		testBuscarRolUsuario();
//		testBuscarTratamientosEspecialidad();
		//testBuscarEspecialidadTratamientos();
		//testBuscarCubiculosLibres();
		//testVerificarEstado();
		//testBuscarPorUsuario();
		//testBuscarPorPersona();
		//testBuscarEspecialidadTratamientos();
		//testBuscarPorDiaEstudiante();
		//testBuscarPorNumero();
		//testBuscarTurnosEstudiante();
		testBuscarPorIdPersona();
		System.out.println("Fin Test DAO");
	}

	public void testBuscarRolUsuario() {
		try{
		RolDAO rolDAO = new RolDAO();
		List<Rol> roles =
rolDAO.buscarRolPorUsuario("paajacome@udlanet.ec");
		if (!roles.isEmpty()) {
			for (Rol rol : roles) {
				System.out.println("Resultado: " +rol.getNombre());
			}
```

```
            } else {
                System.out.println("No trae resultados");
            }
            }catch(Exception e){
                System.out.println("Error:" + e);
                fail("Fail: " + e);
            }
        }

        public void testBuscarHorariosEstudiante() {
            try{
            HorarioDAO horarioDAO = new HorarioDAO();
            Estudiante estudiante = new Estudiante ();
            estudiante.setIdEstudiante(Integer.parseInt("1"));
            List<Horario> horarios =
horarioDAO.buscarHorariosEstudiante(estudiante);
            if (!horarios.isEmpty()) {
                for (Horario he : horarios) {
                    System.out.println("Resultado: " +he.getDia() +"-"+
he.getHoraInicio() +"-"+ he.getHoraFinal());
                }
            } else {
                System.out.println("No trae resultados");
            }
            }catch(Exception e){
                System.out.println("Error:" + e);
                fail("Fail: " + e);
            }
        }

        public void testBuscarTratamientosEspecialidad() {
            try{
```

```
		TratamientoDAO tratamientoDAO = new TratamientoDAO();
		int idEspecialidad = 4;
		List<Tratamiento> tratamientos =
tratamientoDAO.buscarTratamientosEspecialidad(idEspecialidad);
		if (!tratamientos.isEmpty()) {
			for (Tratamiento he : tratamientos) {
				System.out.println("Resultado: " +he.getNombre()
+"-"+ he.getEspecialidad().getNombre());
			}
		} else {
			System.out.println("No trae resultados");
		}
		}catch(Exception e){
			System.out.println("Error:" + e);
			fail("Fail: " + e);
		}
	}

	public void testBuscarEspecialidadTratamientos() {
		try{
		EspecialidadDAO especialidadDAO = new EspecialidadDAO();
		int idTratamiento = 5;
		Especialidad especialidades =
especialidadDAO.buscarEspecialidadTratamiento(idTratamiento);
		if (especialidades!=null) {
				System.out.println("Resultado: "
+especialidades.getNombre());

		} else {
			System.out.println("No trae resultados");
		}
		}catch(Exception e){
```

```
                System.out.println("Error:" + e);
                fail("Fail: " + e);
            }
        }

        public void testBuscarCubiculosLibres() {
            try{
            int idHorario = 1;
            int idEspecialidad  = 2;
            Date fecha = Utilitarios.stringToDate("2016-05-09");
            String tipoCubiculo = "NORMAL";
            HorarioCubiculoEstadoDAO hceDAO = new
HorarioCubiculoEstadoDAO();
            List<String> cubiculos = hceDAO.buscarCubiculosLibres(fecha,
idEspecialidad, idHorario, tipoCubiculo);
            if (!cubiculos.isEmpty()) {
                for (String he : cubiculos) {
                    System.out.println("Resultado: " +he);
                }
            } else {
                System.out.println("No trae resultados");
            }
            }catch(Exception e){
                System.out.println("Error:" + e);
                fail("Fail: " + e);
            }
        }

        public void testVerificarEstado(){
            try{
                HorarioCubiculoEstadoDAO hceDAO = new
HorarioCubiculoEstadoDAO();
```

```
                Boolean flag = hceDAO.verificarEstado(1, "LIB");
                if(flag){
                        System.out.println("El cubiculo esta libre");
                }else{
                        System.out.println("El cubiculo NO esta libre");
                }
        }catch(Exception e){
                System.out.println("Error:" + e);
                fail("Fail: " + e);
        }
}

public void testBuscarPorUsuario(){
        try{
                PersonaDAO personaDAO = new PersonaDAO();
                Persona persona = new Persona();
                String usuario = "paajacome@udlanet.ec";
                persona = personaDAO.buscarPorUsuario(usuario);
                if(persona!=null){
                        System.out.println("La persona es:" +
persona.getNombres() +" "+persona.getApellidos());
                }else{
                        System.out.println("No existen datos");
                }
        }catch(Exception e){
                System.out.println("Error:" + e);
                fail("Fail: " + e);
        }
}

public void testBuscarPorPersona(){
        try{
```

```
                EstudianteDAO estudianteDAO = new EstudianteDAO();
                Estudiante estudiante = new Estudiante();
                Persona persona = new Persona();
                persona.setIdPersona(Integer.parseInt("1"));
                estudiante = estudianteDAO.buscarPorPersona(persona);
                if(estudiante!=null){
                        System.out.println("El estudiante es:" +
estudiante.getMatricula());
                }else{
                        System.out.println("No existen datos");
                }
        }catch(Exception e){
                System.out.println("Error:" + e);
                fail("Fail: " + e);
        }
    }

    public void testBuscarPorDiaEstudiante(){
        try{
                HorarioEstudianteDAO horarioEstudianteDAO = new
HorarioEstudianteDAO();
                String diaNombre = "LUNES";
                int idEstudiante = 1;
                int idHorarioEstudiante =
horarioEstudianteDAO.buscarPorDiaEstudiante(diaNombre, idEstudiante);
                if(idHorarioEstudiante!=0){
                        System.out.println("El id es:" + idHorarioEstudiante);
                }else{
                        System.out.println("No existen datos");
                }
        }catch(Exception e){
                System.out.println("Error:" + e);
```

```
                fail("Fail: " + e);
            }
        }

        public void testBuscarPorNumero(){
            try{
                HorarioCubiculoEstadoDAO hceDAO = new
HorarioCubiculoEstadoDAO();
                Horariocubiculoestado hce = new Horariocubiculoestado();
                String numeroCubiculo = "5";
                hce = hceDAO.buscarPorNumero(numeroCubiculo);

                if(hce!=null){
                    System.out.println("El id es:" +
hce.getIdhorarioCubiculoEstado());
                }else{
                    System.out.println("No existen datos");
                }
            }catch(Exception e){
                System.out.println("Error:" + e);
                fail("Fail: " + e);
            }
        }

        public void testBuscarTurnosEstudiante(){
            try{
                TurnoDAO turnoDAO = new TurnoDAO();
                String estado = "RES";
                int idEstudiante = 1;
                List<Turno> turnos =
turnoDAO.buscarReservadosPorEstudiante(idEstudiante, estado);
                if (!turnos.isEmpty()) {
```

```
                        for (Turno turno : turnos) {
                                System.out.println("Turno: "
+turno.getFecha()+ " - Horario:"
                                                +
turno.getHorarioestudiante().getHorario().getHoraInicio()
                                                + " - "
                                                +
turno.getHorarioestudiante().getHorario().getHoraFinal()
                                                + " -Tratamiento: "
                                                +
turno.getTratamiento().getNombre()
                                                + " -Paciente: "
                                                +
turno.getPaciente().getPersona().getNombres()
                                                + " - "
                                                +
turno.getPaciente().getPersona().getApellidos()
                                                + " - Cubiculo: "
                                                +
turno.getHorariocubiculoestado().getHorariocubiculo().getCubiculo().getNumero
()
                                                );
                                }
                        } else {
                                System.out.println("No trae resultados");

                        }
                }catch(Exception e){
                        System.out.println("Error:" + e);
                        fail("Fail: " + e);
                }
        }
```

```
public void testBuscarPorIdPersona(){
        try{
                UsuarioDAO usuarioDAO = new UsuarioDAO();
                Usuario usuario = new Usuario();
                int idPersona = 1;
                usuario = usuarioDAO.buscarPorIdPersona(idPersona);
                if (usuario.getIdUsuario()>0) {
                        System.out.println("Resultado: "+
usuario.getUsuario() +"-"+usuario.getRol().getNombre());
                } else {
                        System.out.println("No trae resultados");

                }
        }catch(Exception e){
                System.out.println("Error:" + e);
                fail("Fail: " + e);
```

Resultado:

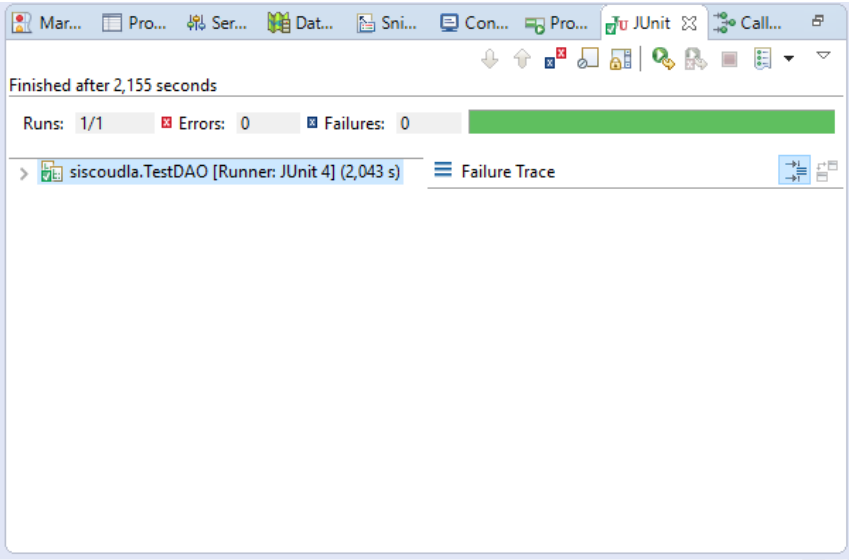

Figura 45. JUnit TestDAO

Anexo 3. Términos y definiciones

Google App Engine: Plataforma de servicios Cloud de Google.

IaaS: Siglas de infraestructura como servicio, la parte de hardware de una arquitectura SOA.

SaaS: Siglas de software como servicio, la parte de software de una arquitectura SOA.

PaaS: Siglas de plataforma como servicio, la parte de software e infraestructura de una arquitectura SOA.

SOA: Siglas de arquitectura orientada a servicios, arquitectura básica de una solución Cloud.

IDE: Siglas de Integrated Development Environment, ambiente de desarrollo integrado. Herramienta de desarrollo.

SDK: Siglas de Software Development Kit, kit de desarrollo de software.

SSD: Siglas de Solid State Disk, discos duros de estado sólido.

Landing Page: Sitio de “aterrizaje” a una página web.

Dashboard: Página de control por usuario.

JEE: Siglas de Java Platform Enterprise Edition, plataforma de desarrollo empresarial con lenguaje Java.

JUnit: Tecnología propia del desarrollo Java EE, para realizar pruebas unitarias de manera automatizada.

Java: Lenguaje de programación de propósito general, basado en el paradigma de orientación de objetos

JSP: Siglas de JavaServer Pages, tecnología utilizada para el desarrollo de aplicaciones web dinámicas basadas en html y xml.

HTML: Siglas de HyperText Markup Language, lenguaje marcado para elaboración de páginas web.

JavaScript: Lenguaje de programación interpretado utilizado principalmente en el lado del cliente.

Framework: Conjunto estandarizado de conceptos, prácticas y criterios para el desarrollo de software con un lenguaje especifico.

JQuery: Framework de desarrollo de JavaScript.

Ajax: Siglas de Asynchronous JavaScript And XML, técnica de desarrollo web interactivo.

JSon: Siglas JavaScript Object Notation, formato de texto ligero para el intercambio de datos.

Maven: Herramienta de software para la gestión y construcción de proyectos Java

A continuación se listan los archivos que se encuentran en el CD:

- Manual de usuario
- Código fuente
- Modelo de la BDD