ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA DE SISTEMAS

DESARROLLO DE UN SISTEMA WEB PARA LA ADMINISTRACIÓN DE LA INFRAESTRUCTURA FÍSICA DE LA ESCUELA POLITÉCNICA NACIONAL.

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS INFORMÁTICOS Y DE COMPUTACIÓN

CASTRO LIMA WILSON RAMIRO
orochi_cris@yahoo.com

GUACHAMÍN CORELLA LEONARDO DANIEL
mrdanleo_g@hotmail.com

Director: Ing. Sandra Sánchez MSc.
sandra.sanchez@epn.edu.ec

Quito, Agosto de 2014

DECLARACIÓN

Nosotros, Wilson Ramiro Castro Lima y Leonardo Daniel Guachamín Corella, declaramos bajo juramento que el trabajo aquí descrito es de nuestra autoría; que no ha sido previamente presentada para ningún grado o calificación profesional; y, que hemos consultado las referencias bibliográficas que se incluyen en este documento.

A través de la siguiente declaración cedemos nuestros derechos de propiedad intelectual correspondientes a este trabajo, La Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

Wilson Ramiro Castro Lima

Leonardo Daniel Guachamín Corella

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Wilson Ramiro Castro Lima y Leonardo Daniel Guachamín Corella, bajo mi supervisión.

Msc. Ing. Sandra Sánchez

DIRECTOR DE PROYECTO

AGRADECIMIENTOS

Agradezco a mis compañeros y profesores que dieron todo de ellos para que cumpla mi meta, la de ser mejor persona, aprender y ser un buen profesional.

A mis hermanos que dieron todo de ellos, para que durante mis estudios no me rinda ante las adversidades.

A mi madre quien me apoyó en cada momento de estos años de estudios y quien me brindo la posibilidad de estudiar y de ser mejor.

A la Msc. Ing. Sandra Sánchez quien me apoyó en cada momento como estudiante y tesista.

A la Tlga. Geovana Saltos

Wilson Castro L.

DEDICATORIA

La presente Tesis se la dedico a mi familia quienes me apoyaron durante estos largos años de arduo esfuerzo y trabajo.

A mi madre quien dio todo de ella para que me convirtiera en una mejor persona y pudiera cumplir todos mis estudios, siempre apoyándome en cada instante de mi vida.

A mi novia Mariana Cuadrado quien es un pilar fundamental para que me convierta en un mejor hombre, persona y profesional.

Wilson Castro L.

AGRADECIMIENTOS

Quiero agradecer a Dios por darme la sabiduría necesaria para culminar con éxito este trabajo.

A mis queridos padres por el esfuerzo y sacrificio que realizaron cada día para procurar que no me falte nada, por darme los consejos y regaños necesarios que me han servido durante el trascurso de mi vida, por darme todo su amor y cariño día a día.

A mis hermanos por brindarme su ayuda y consejos siempre que fue necesario, por compartir sus experiencias y amistad, sobre todo por ser un ejemplo a seguir.

A mis profesores ya que supieron transmitir sus conocimientos desde el inicio y hasta el final de mi carrera.

A la Ing. Sandra Sánchez ya que más que docente y tutora, ha demostrado ser una amiga en la que se puede confiar.

A mi amigo Wilson Castro quien compartió la idea del sistema desarrollado y conjuntamente nos embarcamos en la ejecución de esta meta y a la Tlga. Geovana Saltos que me permitió formar parte del desarrollo del sistema.

Leonardo Guachamín C.

DEDICATORIA

Dedico este trabajo a mis padres Jorge y Gloria, a mis hermanos Pablo y Christian ya que han sido mi inspiración todos los días, gracias a ellos puedo crecer, y volverme una persona mejor que trabaja duro para cumplir sus metas.

A todos los buenos amigos que he ido ganando lo largo de mi carrera y en mi vida profesional.

Leonardo Guachamín C.

CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

El presente proyecto de titulación tiene por objetivo realizar un **Sistema Web Para La Administración De La Infraestructura Física De La Escuela Politécnica Nacional**. Para el desarrollo del mismo se han estructurado 4 capítulos en los cuales se tratan tales temas.

En el capítulo uno, se realiza un análisis de la situación actual de la Escuela Politécnica Nacional (EPN) en cuanto a la Infraestructura Física de sus instalaciones, se abarca antecedentes históricos hasta los actuales pasos de la Institución en la acreditación de la misma, ubicándola en la Categoría A.

Se exponen las principales características de la EPN como son su Misión, Visión, Acción Afirmativa y la Ubicación, mencionando además las dependencias externas de la EPN como la Metalmecánica San Bartolo y el Observatorio Astronómico. Seguidamente se pone a conocimiento el principal problema que conlleva la administración de la Infraestructura Física de la EPN, describiendo a los edificios que conforman el Campus Politécnico José Rubén Orellana R. Al terminar el capítulo uno se detalla tanto la metodología y las herramientas que se usarán para el desarrollo del Sistema, estableciendo como metodologías a usar XCRUM y XP, y como principales herramientas: JBoss Server, JBoss Developer y PostgreSql.

En el capítulo dos, se especifican los requisitos, las historias de usuario y la pila del producto inicial, lo cual conlleva a la planificación de los Sprints, estableciendo fechas de inicio y de fin de cada uno de ellos. Luego se establece la ejecución de los Sprints con las fechas establecidas en la planificación añadiendo la arquitectura de la solución. La definición de los roles o perfiles de usuario también esta abarcado en este capítulo, por lo cual todos los puntos mencionados abarca en su totalidad el desarrollo del Sistema Web.

En el capítulo tres, se muestra la instalación y puesta en marcha del Sistema Web, estableciendo características necesarias para el correcto inicio del sistema. La recopilación de datos de inicio es una parte fundamental cuando se usará por primera vez el sistema, aquí se muestra cómo y qué datos se deben ingresar por medio del sistema, cabe mencionar que el inicio de datos debe llevarse a cabo

por un administrador, ya que él tiene todos los permisos de uso del sistema. En cuanto a la clientelización, es la presentación del sistema ante los clientes y personas que tengan interés en el sistema para tomar una retroalimentación y mejorar la funcionalidad del mismo.

Finalmente en el capítulo cuatro, se exponen las conclusiones y recomendaciones obtenidas con la realización del presente proyecto.

1 CAPÍTULO 1: DESCRIPCIÓN DEL PROBLEMA

1.1 SITUACIÓN ACTUAL DE LA EPN EN EL ÁMBITO DE INFRAESTRUCTURA FÍSICA

1.1.1 ANTECEDENTES

La infraestructura física con la que cuentan los centros de educación superior debe contar con todos los requerimientos apropiados como: aulas en perfecto estado (pintura, inmuebles), laboratorios tecnológicos, espacios verdes para deportes, salidas de emergencia en los edificios, cafeterías y bibliotecas los cuales son necesarios para garantizar el desarrollo de sus funciones y actividades de tal forma que pueda brindar un servicio de calidad a sus usuarios, en este caso estudiantes profesores, y demás personal administrativo que conforman la institución.

La Escuela Politécnica Nacional, es uno de los centros de educación superior más reconocidos en el país no solo por su infraestructura sino también por su calidad de enseñanza y su participación en proyectos en el campo investigativo, también cabe destacar que en el mes de Julio del 2014 el Dr. Holger Capa presidente de la Comisión Permanente de Evaluación y Acreditación de las Carreras del CEAACES entrega el certificado con fecha Quito, 26 de Noviembre de 2013, al Rector de la EPN acreditando a la Universidad como categoría “A” por 5 años más. [1]

Historia

La Escuela Politécnica Nacional fue fundada en 1869 por el presidente Gabriel García Moreno, siendo así el primer centro de investigación, docencia y formación de profesionales en ingeniería y ciencias de alto nivel, logrando consolidarse como uno de los centros de estudios superiores más prestigiosos del país, siguiendo estándares de excelencia, y siendo un referente en ciencia, tecnología e innovación.

En 1876 la Escuela Politécnica Nacional fue cerrada por el presidente Borrero debido a razones políticas. Los estudios como Matemáticas, Cosmografía, Física,

Química Aplicada, Electrotecnia, Ingeniería Minera y Geología fueron en los que se enfocó la Escuela Politécnica Nacional en el año de 1935 tras su reapertura por mandato del Presidente Velasco Ibarra. El 4 de junio de 1946 la Escuela Politécnica Nacional cambia su nombre actual de Instituto Superior Politécnico por el de Escuela Politécnica Nacional con el que se ha mantenido hasta nuestros actuales días. [1]

Misión

"La Escuela Politécnica Nacional, tiene como misión formar académicos y profesionales en ingeniería y ciencias, con conciencia ética, solidarios, críticos, capaces de contribuir al bienestar de la comunidad; así como generar, difundir y transmitir el conocimiento científico y tecnológico, con responsabilidad social, como resultado de una dinámica interacción con los actores de la sociedad ecuatoriana y la comunidad internacional."[1]

Visión

"La Escuela Politécnica Nacional es una universidad pública con estándares internacionales de excelencia, siendo un referente en ciencia, tecnología e innovación. Sus capacidades y esfuerzos están orientados al servicio de la comunidad, contribuyendo al desarrollo cultural, dentro de un marco de principios y valores transcendentales del ser humano."[1]

Acción Afirmativa

"La Escuela Politécnica Nacional es una institución laica y democrática, que garantizar la libertad de pensamiento, expresión y culto de todos sus integrantes, sin discriminación alguna. Garantiza y promueve el reconocimiento y respeto de la autonomía universitaria, a través de la vigencia efectiva de la libertad de cátedra y de investigación y del régimen de cogobierno."[1]

Ubicación

[1] Tomado de http://www.epn.edu.ec/

El Campus Politécnico se encuentra ubicado en la parroquia La Floresta, en la calle Ladrón de Guevara E11 – 253 frente al Coliseo Rumiñahui (Sector centro-oriental de Quito).

En la Figura 1.1 se muestra una vista satelital de la ubicación de la Escuela Politécnica Nacional.

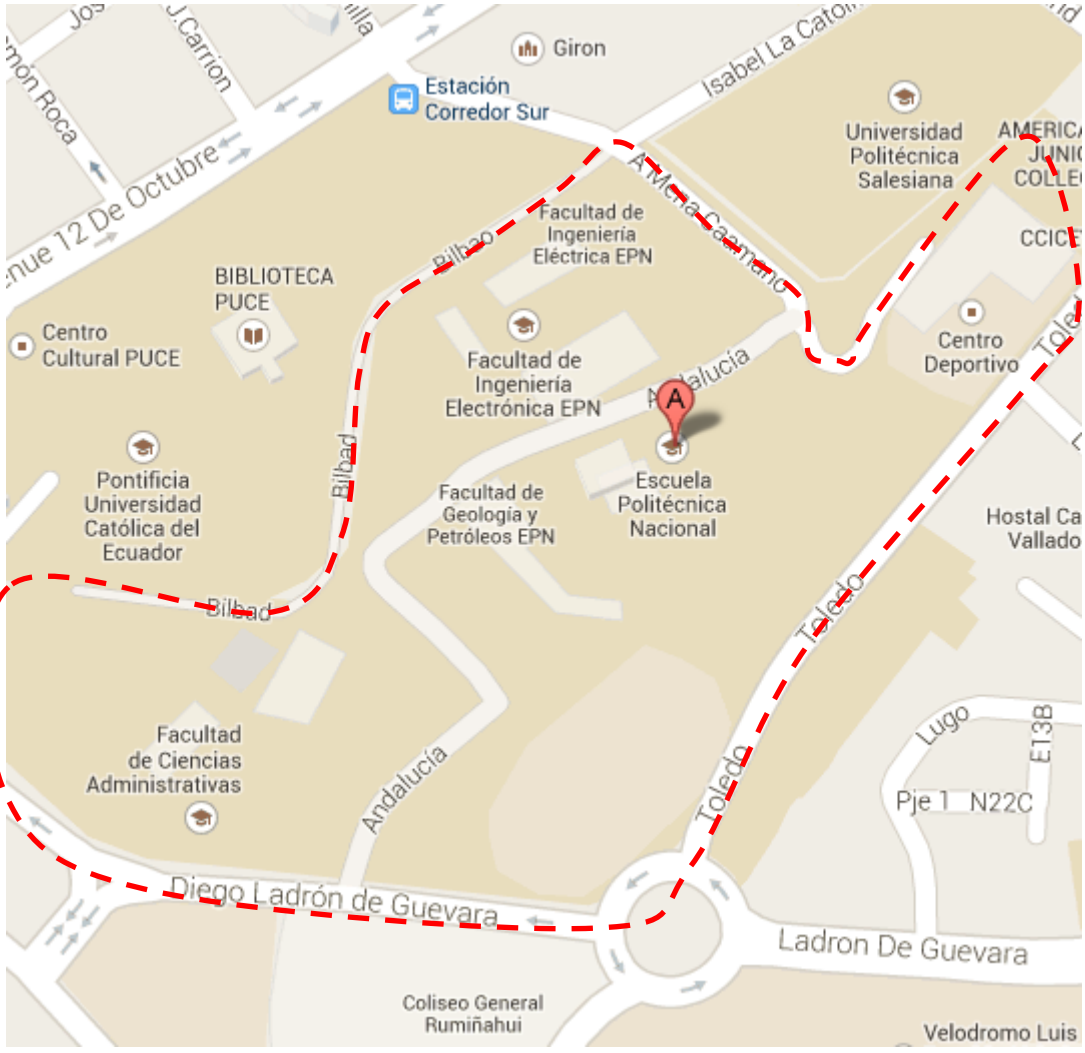


Fig. 1.1: Vista satelital de la ubicación de la Escuela Politécnica Nacional. Fuente: Google Maps.

El campus tiene un área aproximada de 152 mil metros cuadrados con una superficie de construcción de 67489 metros cuadrados que corresponde a laboratorios, centros de investigación, aulas, bibliotecas, oficinas administrativas,

talleres, teatro, canchas deportivas entre otros, además cuenta con cuatro entradas ubicadas en:

- Calle Isabel la Católica y Veintimilla (Entrada Facultad de Ingeniería Eléctrica)
- Calle Alfredo Mena Caamaño y Andalucía (Entrada Escuela de Tecnólogos)
- Calle Andalucía (Entrada de Facultad de Ingeniería Química y Agroindustria)
- Calle Ladrón de Guevara (Entrada Facultad de Ingeniería Civil) [1]

Se debe mencionar que el Rector de la Escuela Politécnica Nacional, Ing. Jaime Calderón Segovia, luego de varias gestiones realizadas con IMOBILIAR consiguió que se entregue el terreno que antiguamente pertenecía a ¨La Universal¨ a la Escuela Politécnica Nacional, lo cual fue ratificado por el Eco. Rafael Correa Delgado, Presidente de la República.

La nueva adquisición tiene alrededor 5 mil metros cuadrados y está ubicada en la intersección de las calles Isabel La Católica y Veintimilla (La Universal) junto a la Facultad de Ingeniería Eléctrica. [2]

En la Figura 1.2 se muestra una vista satelital del nuevo espacio adquirido por la Escuela Politécnica Nacional.

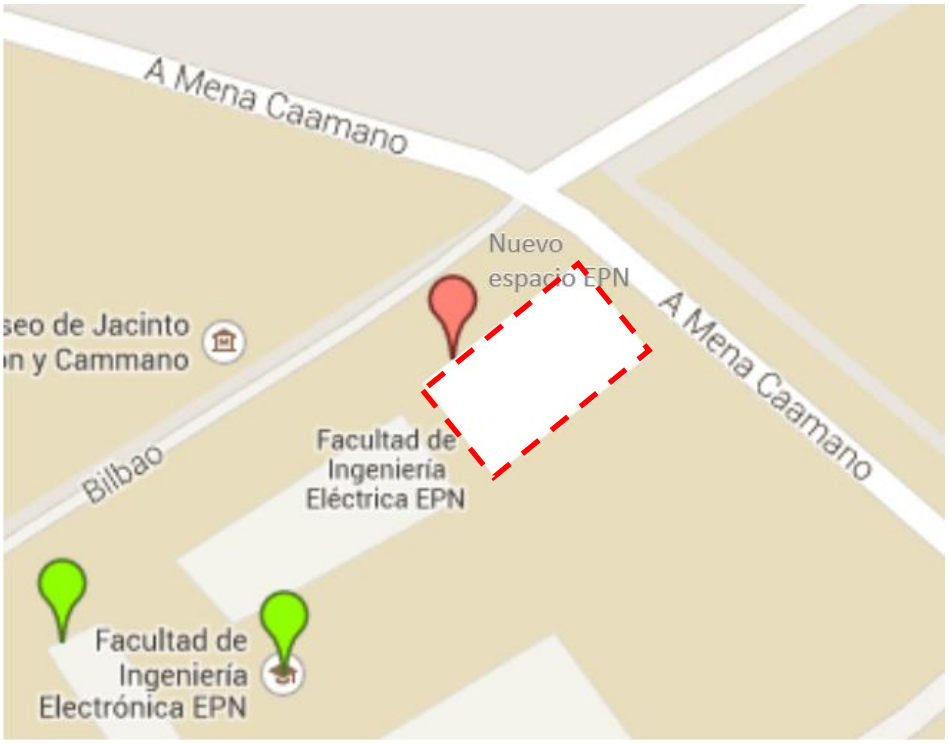


Fig. 1.2: Vista satelital de la ubicación del nuevo espacio adquirido por la EPN.

Fuente: GoogleMaps

La Facultad de Ingeniería Civil y Ambiental elabora estudios topográficos del terreno que permitirán realizar la planificación de obras de ampliación para campus con la creación de edificios inteligentes y sismo resistentes.

Existen dos dependencias que forman parte de la EPN y están fuera del campus politécnico y son:

Observatorio Astronómico Quito

Institución científica y cultural ecuatoriana ubicada en la ciudad de Quito. Su principal actividad está relacionada con la Astronomía en las áreas de investigación, educación y su difusión.

Se encuentra ubicada en la Av. Gran Colombia S/N y Av. 10 de agosto, Interior del parque "La Alameda".

En la Figura 1.3 se muestra una vista satelital de la ubicación del Observatorio Astronómico de Quito.

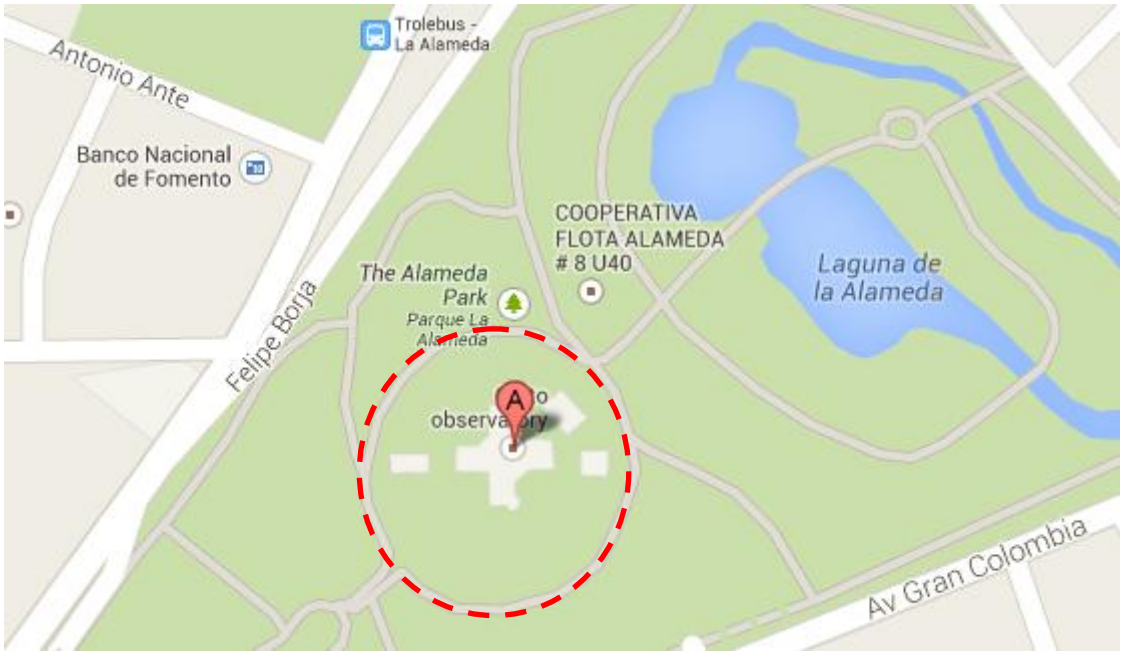


Fig. 1.3: Vista satelital de la ubicación del Observatorio Astronómico Quito.
Fuente: GoogleMaps

Metalmecánica San Bartolo de la Escuela Politécnica Nacional

Taller de propiedad de la Escuela Politécnica Nacional en donde se lleva acabo el mantenimiento, reparación y reconstrucción de equipo pesado y equipo industrial de grandes dimensiones, la Metalmecánica San Bartolo está ubicada la Avenida Pedro Vicente Maldonado No S15-283 y Balzar.

En la Figura 1.4 se muestra una vista satelital de la ubicación de la Metalmecánica de San Bartolo.

Fig. 1.4: Vista satelital de la ubicación de la Metalmecánica San Bartolo de la EPN.
Fuente: GoogleMaps

Consta de un terreno de extensión 26269.070 m2 y un área de construcción de 7 240.07 m2.

Instalaciones Académicas y Administrativas

El Campus “Rubén Orellana” cubre 15.2 hectáreas en Quito. Posee un total de 62.800 m2 de área construida, distribuida el 14% en aulas y el 24,1% en laboratorios. Se cuenta además, con el Edificio para Aulas y Relación con el Medio Externo con un área de 33.800 m2.

En el campus se distribuyen alrededor de veinte edificaciones, en las cuales se encuentran las respectivas facultades, departamentos, laboratorios, centros de investigación, aulas y oficinas.

El edificio de Administración Central, ubicado en el sureste del campus, alberga el Rectorado, Vicerrectorado, la Comisión de Evaluación Interna, las Direcciones: Jurídica, Administrativa, Planificación, Relaciones Institucionales, Recursos Humanos, Financiera, Comisiones de Investigación, Docencia y Extensión, Unidad de Desarrollo Curricular, la Secretaría General y otras dependencias administrativas. [1]

1.1.2 PROBLEMÁTICA

Actualmente la EPN no cuenta con información actualizada y suficiente, para la gestión de su propia infraestructura, provocando que la distribución de aulas, laboratorios, oficinas y demás espacios físicos dentro del campus sea un problema tanto para profesores, estudiantes, empleados y visitantes. La actual administración de los espacios físicos de la EPN se lleva a cabo por medio de un control establecido en hojas electrónicas y muchas veces de manera informal, por lo tanto, cuando se realizan nuevas construcciones o cambios en el Campus, el manejo de las hojas se vuelve un gran inconveniente para las personas encargadas de dicha administración por lo que el sistema web facilitará y resolverá tales inconvenientes.

Por otro lado la falta de planos actualizados de las edificaciones de la EPN hace difícil la visualización de los futuros cambios y los lugares en donde se van a llevar a cabo dichos cambios, por tanto, un Sistema Informático mantendría actualizado el número de edificaciones, las aulas, laboratorios y los espacios físicos que maneja la Institución para así llevar un control progresivo de la Infraestructura de la EPN.

Los visitantes, nuevo personal, estudiantes, personal docente y administrativo que se integran a la Politécnica, en muchos casos no conocen el lugar de ciertas instalaciones dentro del campus, lo cual dificulta el desplazamiento dentro del mismo a una hora exacta y en el lugar correcto, además se evitaría el inconveniente de preguntar la ubicación de ciertas oficinas y otros espacios a personas que en muchos casos tampoco saben la ubicación exacta. Además la programación de eventos especiales por parte de la comunidad Politécnica, es un inconveniente muy grande, ya que el cruce de horarios, lugares y fechas dificulta una buena realización de dichos eventos ocasionando en muchos de ellos la cancelación y por ende la pérdida de dinero por parte de ellos debido a la falta de programación y su seguimiento.

Es importante contar con un sistema, el cual beneficie a los usuarios administrativos de la infraestructura de la EPN, de manera interactiva, en la cual el usuario puede dejar de lado el manejo de hojas electrónicas, papel y horas

innecesarias al momento de distribuir los espacios físicos y la reprogramación de las mismas, es una tarea que un Sistema Informático conjuntamente con el Administrador del mismo pueden desarrollar de una manera efectiva, brindando así la solución a un sin número de problemas que se han dado a lo largo de la administración de la EPN en el ámbito de infraestructura física.

1.1.3 INFRAESTRUCTURA

En el Campus Rubén Orellana de la Escuela Politécnica Nacional, funciona el 80% de sus edificios, los cuales acogen: facultades, departamentos, laboratorios, oficinas administrativas, aulas de clases, espacios verdes además del funcionamiento del Teatro Politécnico y el Museo de Historia Natural.

En la Figura 1.5 se muestra de manera global a la Infraestructura Física de la EPN, rescatando sus edificios y presentándolos por colores.

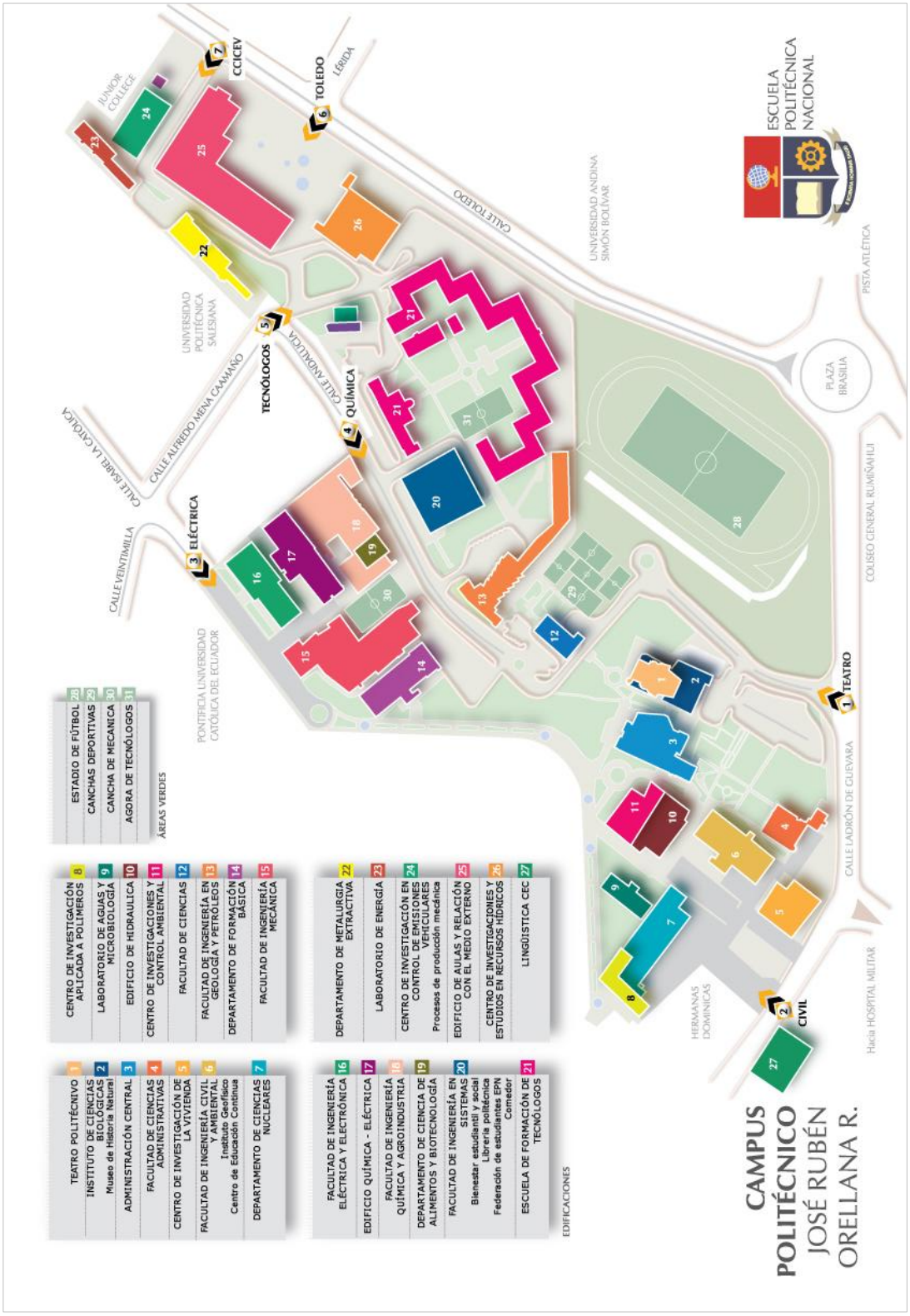


Fig. 1.5: Escuela Politécnica Nacional Campus José Rubén Orellana R. Fuente: http://goo.gl/d6ivTx, Modificado por los Autores.

1.1.4 EDIFICIOS

Los edificios del Campus Politécnico funcionan de manera diversa, alojando: facultades, bibliotecas, salas de lectura, oficinas, laboratorios, bodegas, asociaciones y espacios varios como al cafetería en el edificio de Sistemas.

En la Tabla 1.1 se muestra al Edificio de Sistemas con una breve descripción de su infraestructura.

Características	***Edificio: Facultad de Sistemas***
	Facultad: Sistemas
	Datos
No. De plantas	5
No. De subsuelos	1
No. De aulas	14
No. De laboratorios	7
No. De oficinas	29
No. De bodegas	3
Auditorios	0
Otros espacios	6/28
Observaciones:	*En el período 2014 A se inició una construcción de lado posterior del edificio, además de una adecuación de la cafetería.* *En una planta se encuentra Bienestar Estudiantil, instalaciones de atención médica y la FEPON (Federación de Estudiantes Politécnicos) con 28 espacios.* *Existen 6 espacios pertinentes a la facultad de Sistemas.* *En la planta baja se encuentra la cafetería de al EPN,*

Facultad de Sistemas

Tabla 1.1: Ficha de datos del Edificio de Sistemas.
Fuente: http://cei.epn.edu.ec/CEI-Descargas.html. Elaborado por los autores.

Las demás especificaciones y características de cada edificio se encuentran en el Anexo 1-Datos de Edificios.

1.2 JUSTIFICACIÓN DE LAS METODOLOGÍAS A UTILIZAR

Se conoce muy bien que de las diferentes metodologías ágiles existentes para el desarrollo de software, las más renombradas y utilizadas son Scrum y XP, teniendo a XP con una traza de errores menor por medio de la programación organizada en parejas y el desarrollo orientado a pruebas, teniendo así la satisfacción del programador, manteniendo por otro lado a Scrum alineando las diferentes necesidades del cliente con el equipo de trabajo, cumpliendo las expectativas del cliente mediante resultados anticipados, además de poseer flexibilidad y adaptación a cambios mitigando errores y controlando la calidad del producto en sus diferentes entregas hasta obtener el producto final.

Por los diferentes aspectos mencionados anteriormente hemos adoptado dichas metodologías para el desarrollo de nuestro sistema, rescatando lo mejor da cada una de ellas, además de reconocer que, para la gestión del proyecto y la mayoría

1.2.1 METODOLOGIA SCRUM

1.2.1.1 Historia

El origen de Scrum data de estudios realizados por Hirotaka Takeuchi e Ikujijo Nonaka en los años 80 principalmente en el año 86 en el campo de las prácticas de productos tecnológicos, en el año de 1993 fue cuando Jeff Sutherland aplicó dicho modelo en la Empresa Easel Corporation al desarrollo de software para luego aplicarlo en Informix y Ascential Software Corporation. En el año de 1996 presentó, junto con Ken Schwaber, las prácticas que empleaban como un proceso formal, para gestión del desarrollo de software en OOPSLA 96 (Schwaber & Sutherland, 1996). [25]

1.2.1.2 Fases

Las fases de Scrum son muy variadas a la hora de desarrollar sistemas software, depende del equipo de desarrollo y de las personas involucradas hay muchas formas de variar las fases pero las más utilizadas son las siguientes: planificación del sprint, refinamiento de la pila del producto lo cual conlleva a un seguimiento

del sprint y por último la revisión del sprint y retrospectiva del mismo, tal como se puede apreciar en la Figura 1.6

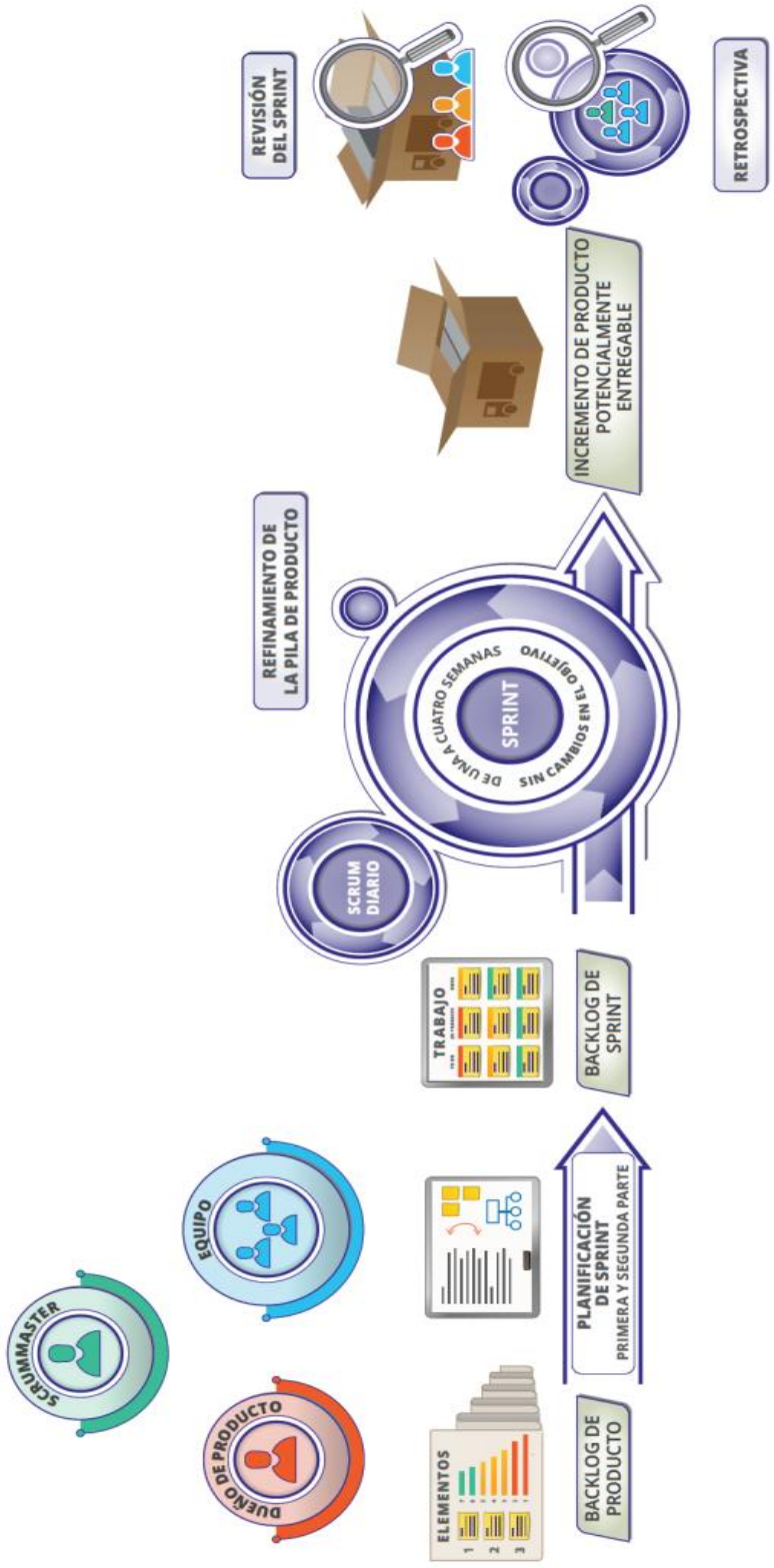
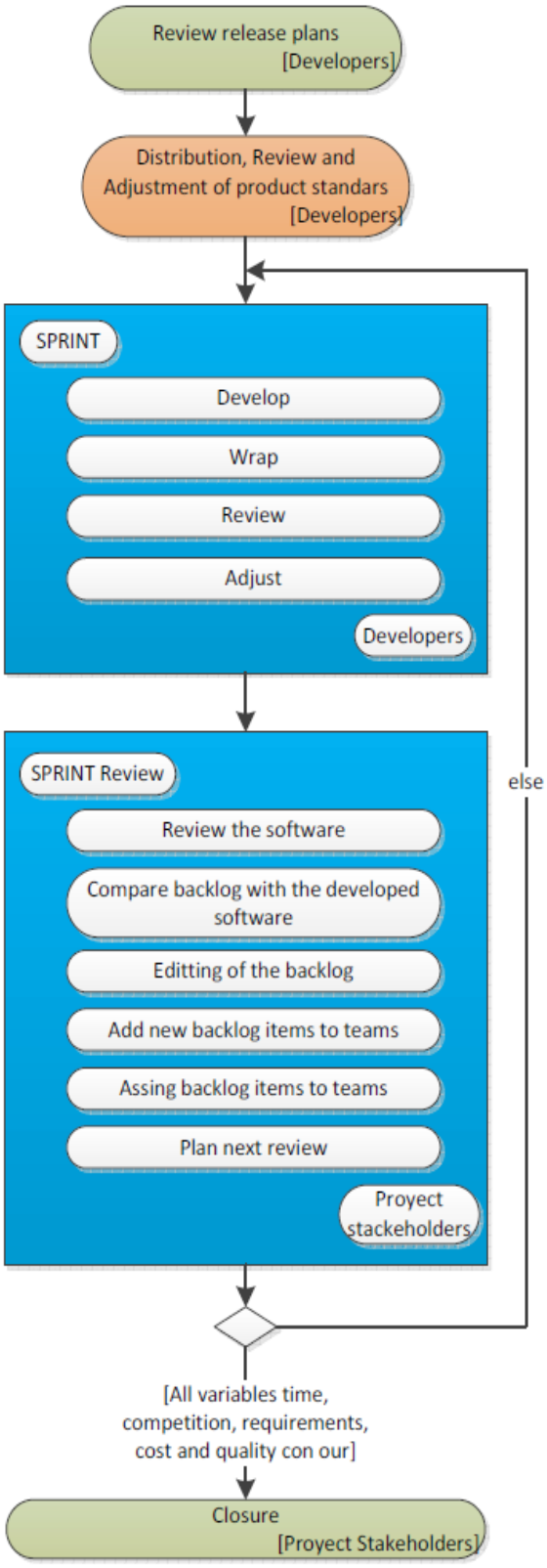


Fig. 1.6: Proceso Scrum

Fuente: Deemer P., Benefield G., Larman C., Vodde B., Scrum Primer, 2012, pág, 3

En la Figura 1.7 se describe las características de las fases de Scrum.



Revisiones de planes y distribución , revisiones y ajuste del producto y estandarización: Fase en la cual se realiza una revisión de las diferentes actividades, así como la forma de la distribución de cada una de ellas a los diferentes involucrados, se revisa estándares y tecnologías a usar.
Sprint: En esta fase se verifican, determinan y solucionan los diferentes inconvenientes de forma iterativa en el desarrollo, en esta fase se pueden añadir nuevos ítems, para lograr un ajuste y mejora tanto en código como en documentación.
Sprint Review: Fase determinada para la revisión de un Sprint conjuntamente con el SCRUM Master, esta actividad se la realiza en la última tarea del sprint con la posibilidad de añadir nuevos items.
Closure: Es la finalización de las actividades dentro de un proyecto, en esta fase se obtiene un producto o versión entregable por medio de las pruebas, la promoción y el marketing.

Fig. 1.7: Fases de Scrum.
Fuente: Spada D., Usabilidad en el proceso de desarrollo de SCRUM, pág 7. Modificado por los Autores.

1.2.1.3 Roles y Responsabilidades

En la Tabla 1.2 se muestra los roles y características de los responsables.

Actor	Características

Product Owner (Propietario del Producto).
 	
Encargado y responsable del Product BackLog, toma las ideas del usuario a fin de determinarlas como decisiones, esta tarea es realizada por una sola persona, la cual es la responsable de la financiación necesaria para el proyecto, prácticamente es la persona que lleva las decisiones que afectan al resultado del producto final, implantando fechas de lanzamiento, presentación a los clientes, publicidad vinculando toda actividad a un retorno de inversión.
	• Conocer el entorno del negocio, las necesidades y el objetivo que persigue el cliente con el sistema que se está desarrollando.
	• Conocer las metodologías ágiles como Scrum para realizar todas las actividades con la calidad y profesionalismo que el compete:
	  ○ Desarrollar y administrar la pila del producto.
	  ○ Presentar y participar en las reuniones de planificación de todos los sprints.
	• Retro-información del negocio, analizando cada aspecto y tareas durante el desarrollo del producto.
	• Conocer y/o haber realizado actividades similares previas con el mismo personal de trabajo o equipo (recomendable).

Scrum Team (Equipo de Desarrollo). El equipo de desarrollo es un grupo multidisciplinar con la cualidad de auto organización, uso común de buenas prácticas y tecnologías ágiles para realizar cada Sprint. Es recomendable el trabajo en equipos de entre 4 y 8 personas. Si se trabaja con un equipo	• Conocer y comprender la visión del cliente. • Aportar y colaborar con el cliente en la elaboración de la pila del producto. • Compartir conjuntamente el objetivo de cada sprint y cada responsabilidad de sus logros. • Participar en las decisiones. • Respetar las opiniones ajenas y aportaciones de cada miembro del equipo

muy grande como más de 10 personas, resulta difícil mantener la agilidad, produciendo rigidez en la dinámica de las personas y el trabajo en equipo.	y de los involucrados en el trabajo. • Conocer las metodologías agiles como Scrum para mayor fluidez en el trabajo.
Scrum Master (Líder del Proyecto). El Scrum Master ***no*** es el jefe de los miembros del Equipo, como tampoco es un jefe de proyecto sino que es el encargado de ayudar al área de productos en la aplicación de Scrum para definir y lograr un valor de negocio, realizando todo lo que esté a su alcance para ayudar a las personas vinculadas al proyecto así como adoptar prácticas de desarrollo modernas.	• Asesorar al cliente o dueños del producto. • Asesorar al equipo de trabajo. • Revisar y validar la pila del producto. • Mantener la compostura de las reuniones a lo largo del desarrollo del producto. • Solucionar los diferentes inconvenientes que se pueden suscitar en cada sprint y que pueden afecta en la realización y culminación de las tareas. • Configurar y mejorar continuamente las prácticas de Scrum en la organización.
Otros interesados	• Son las personas a quienes les interesa el producto final o han aportado en el desarrollo del proyecto.

Tabla 1.2: Roles y Características en SCRUM.
Fuente: Elaborado por los Autores.

1.2.1.4 Prácticas

En la Tabla 1.3 se muestran herramientas y prácticas para la gestión de las fases en SCRUM.

Herramientas	Descripción
Gráfico Burn-Up	Muestra una visión general del desarrollo del producto y de la evolución del mismo partiendo de temas previos asociados en el product backlog en un tiempo real.
Gráfico Burn-Down	Ayuda el seguimiento de cada Sprint para comprobar que cada tarea avanza de manera correcta y fluida verificando si el tiempo es el previsto. Ayuda a verificar cuantitativamente esfuerzo faltante o que no

	se ha realizado.
Juegos y Protocolos de Decisión	Juego o estimación de póker. Estimación a los chinos.
Métricas	**Descripción**
Tiempos	Trabajo en tiempo real que se suele medir en horas. El tiempo utilizado para realizar una tarea, básicamente es un tiempo hipotético o teórico manejado en condiciones ideales en caso de que no existan interrupciones.
Puntos de función o funcionalidad	Es la cantidad de trabajo por medio de medidas relativas, que se usa para la construcción de las historias de usuario del product backlog así como la funcionalidad.
Estimaciones	Es el esfuerzo necesario para realizar una tarea, una actividad o desarrollar una funcionalidad. Dichas estimaciones se basan en las cualidades, habilidades y conocimiento de cada miembro del equipo.
Velocidades absoluta	Es la cantidad de desarrollo o avance en un sprint. Medido en estimaciones (puntos de función, horas o días reales o teóricos). Unidad de tiempo de trabajo.

Tabla 1.3: Herramientas de Scrum.
Fuentes: Palacio J., Scrum Manager Manual Gestión de Proyectos, Septiembre 2008; Flexibilidad con Scrum, Octubre 2007, modificado por los autores.

1.2.2 METODOLOGIA XP

1.2.2.1 Historia

XP aplicada al desarrollo de software nació de la mano de Kent Beck en 1.996, cuando la compañía Chrysler Corporation, le encomendó trabajar en el desarrollo de una aplicación de nóminas, y que posteriormente la perfeccionó junto a Ward Cunningham y Ron Jeffries.

1.2.2.2 Fases

En la Tabla 1.4 se especifica las fases de XP.

Fases	Descripción
Exploración	Historias de usuario realizadas por clientes, se recomienda la

	construcción de un prototipo para comprobar y reconocer la tecnología y arquitectura.
Planificación	Los desarrolladores estiman esfuerzos, determinando el contenido y cronograma de la primera entrega, la cual no debería ser en más de 3 meses.
Iteración	Se intenta establecer la arquitectura y una producto entregable, seguida de pruebas ejecutadas al final de cada iteración realizadas por los clientes.
Mantenimiento	A veces se requiere nuevas personas dentro del equipo de trabajo debido a cambios en su estructura, donde la rapidez en el desarrollo puede variar por la puesta en producción.
Cierre del Proyecto	Aquí las historias de usuario han sido acopladas y desarrolladas en el proyecto. La falta de presupuesto o beneficios bajos se vinculan al cierre del sistema.

Tabla 1.4: Fases de XP.
Fuente: Letelier P, Penadés C; Metodologías Ágiles para el desarrollo de software: Xtreme Programming (XP), Abril 2006, pags. 10-12, modificado por los autores.

En la Figura 1.8 se muestra el ciclo de vida de Extreme Programming de una forma gráfica.

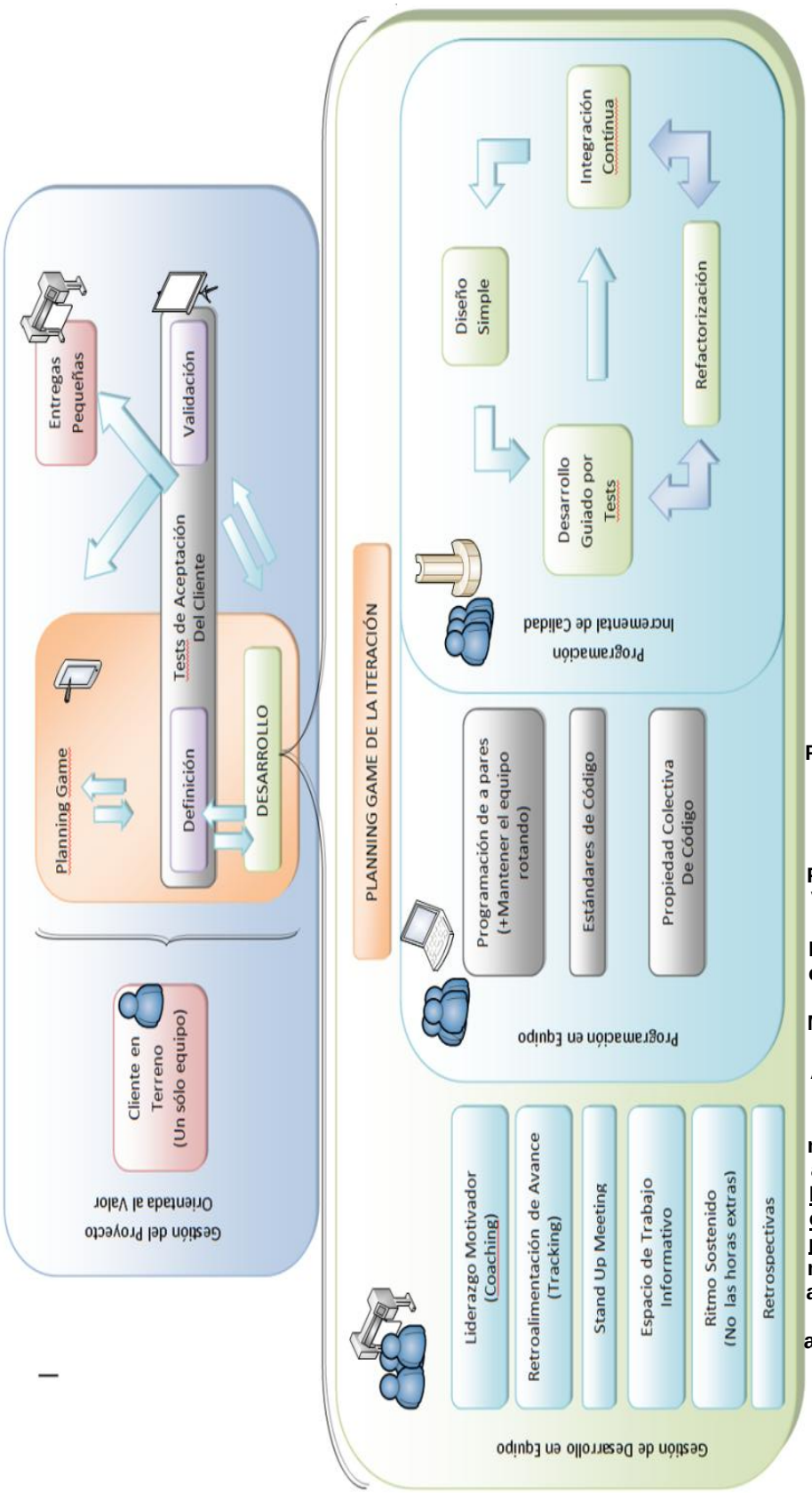


Fig. 1.8: Fases de XP. Fases de XP Fuente: Villena A, Introducción a los Métodos Ágiles, enlace corte modificado de http://goo.gl/NpL6M7, modificado por los autores.

1.2.2.3 Roles y Responsabilidades

En la Tabla 1.5 se lista y describe a los responsables y características de los principales involucrados en un proyecto con XP.

Rol	Descripción
Programador	Es el corazón de XP, con amplia gama de conocimientos de programación, debe informar el funcionamiento total del sistema y su evolución.
Cliente	Escribe y verifica la culminación de ciertas historias, los cambios en las condiciones del negocio, la tecnología, la composición y la capacidad del equipo.
Tester	Trabaja en conjunto con el Cliente para elegir y realizar pruebas funcionales, publicándolas en lugares accesibles para todos los interesados del proyecto.
Rastreador	Encargado de realizar principalmente un “FEEDBACK”, seguimiento en las iteraciones, donde mayor habilidad del rastreador es la de recolectar información.

Tutor o Coach	Encargado de hacer cumplir el proceso con XP supervisando y ayudando al equipo de trabajo, con su experiencia con proyectos en XP.
Consultor	Persona ajena al equipo, que ayuda al equipo a la solución de problemas que no logren resolver de una manera rápida y completa.
Gestor o Manager	Soluciona y eliminar las decisiones críticas referente al proyecto, brindando todo su conocimiento al equipo, cuando estos lo necesiten.

Tabla 1.5: Roles y Responsabilidades en XP.
Fuente: Kent Beck, Extreme Programming Explained, Septiembre 1999, Págs, 105-111, modificado por los autores.

1.2.2.4 Prácticas

La metodología XP se basa principalmente en la disminución de costo del cambio a lo largo del proyecto, y gracias a ello XP se ha convertido en el pionero de las metodologías ágiles alrededor del mundo.

El Juego de la Planificación.

Llevado a cabo por los programadores y los clientes al inicio de cada iteración para cada historia de usuario, realizando estimaciones de esfuerzo en las diferentes áreas del desarrollo del proyecto y así poder decidir los tiempos y riesgos de entrega.

Entregas Pequeñas.

Las entregas pequeñas son prototipos con características operativas y funcionales muy imitadas del sistema, pero que entreguen resultados de valor al cliente por lo cual no deberían tardar más de 3 meses.

Metáfora.

Una metáfora, no es más que una historia compartida, la cual describe cómo debería funcionar el sistema donde XP visualiza posibles inconvenientes que se pueden suscitar al no tener presente una arquitectura evolutiva al comienzo del proyecto.

Diseño Simple.

Un diseño es aquel que: supera las pruebas, no existe lógica duplicada, tienen el menor número de clases y métodos posibles lo cual refleja la intención de la implementación de los programadores.

Pruebas.

En esta práctica, predominan las pruebas unitarias en la producción del código, las cuales con escritas antes del mismo y con ejecutadas constantemente en cada modificación del sistema.

Refactorización.

Mejora la estructura interna del código sin alterar su comportamiento externo, eliminando información duplicada, mejorando la comunicación, simplificando y añadiendo flexibilidad para mejorar los posteriores cambios evolutivos y de funcionalidad que surjan en el transcurso del tiempo en el sistema.

Programación en Parejas.

Es la técnica usada para el desarrollo del sistema (codificación), dando beneficios como: inspecciones de códigos continuos, detección de errores en el código, las líneas de código son pocas, solución de problemas de programación, se comparten conocimientos de programación, varias personas entienden el funcionamiento que va a tener el sistema y así los programadores disfrutan más de su trabajo.

Propiedad colectiva del Código.

Esta práctica ayuda a que cualquier programador pueda manipular el código, realizando modificaciones o añadiendo nuevas líneas al mismo, aportar en el código dejando de lado la dependencia de código y programador.

Integración Continua.

Se da cuando el código es sometido a pruebas y logra exitosamente cumplirlas a cabalidad, puede ser construido e integrado varias veces en un solo día por los desarrolladores. Con esta práctica se reduce la falta de comunicación en cuanto a la reutilización y compartición, además de beneficiar al equipo de desarrollo en cuento a la preparación ante cualquier inconveniente por medio de detección y corrección de errores de integración.

40 horas por semana.

Si exceden este límite o realizan horas extras se desmotiva al equipo y posiblemente esté ocurriendo un inconveniente o problema el cual debe corregirse lo más pronto posible.

Disponibilidad del cliente.

Debe estar presente y disponible todo el tiempo. El cliente es la persona que guía hacia las historias de mayor valor para el negocio; juntamente con los programadores puede resolver de manera inmediata cualquier duda asociada.

Estándares de Programación.

Esta práctica ayuda a los programadores a estar constantemente comunicados y así poder establecer estándares de codificación, ya sea que estén utilizados dentro de la organización o utilizados mayormente por el equipo.

Comentarios Respecto a las Prácticas.

El mérito de XP está en agrupar de manera eficiente varias prácticas novedosas de ingeniería de software que han sido propuestas y en muchos casos demostradas con prácticas.

En la Figura 1.9 se muestra a aplicación y trabajo en conjunto de las prácticas.

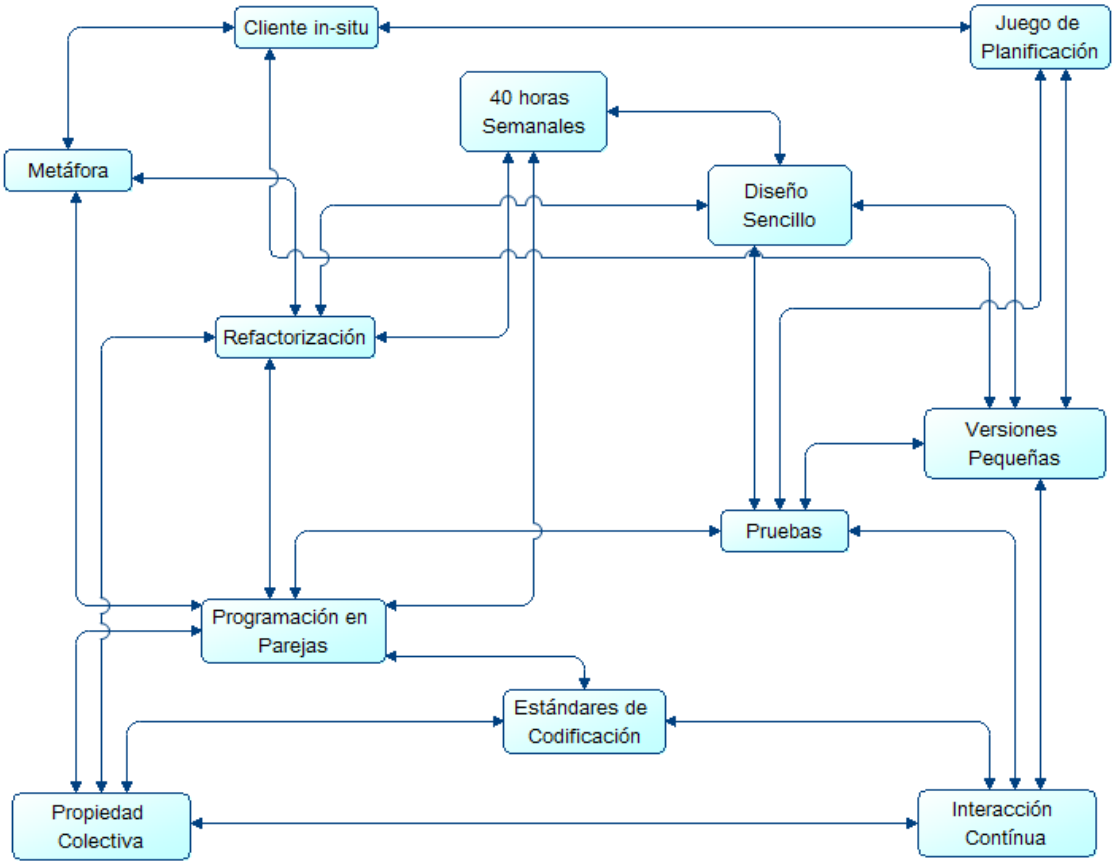


Fig. 1.9: Prácticas trabajando en conjunto.
Fuente: Letelier P, Penadés C; Metodologías Ágiles para el desarrollo de software: sXtreme Programming (XP), abril 2006, modificado por los autores.

1.2.3 COMBINACIÓN DE LAS METODOLOGÍAS SCRUM Y XP

La combinación de las metodologías ágiles SCRUM y XP se ha tomado en cuenta debido a que las dos se complementan, y de esa manera forma una metodología, con la cual se rija en base a los procesos que tendrán el sistema y el desarrollo que conlleva.

Scrum vincula las actividades que se van realizando a lo largo del desarrollo desde el punto de vista de organización y gestión, en cuanto a XP se enfoca en las diferentes y diversas prácticas de programación. Esa es la razón de que funcionen tan bien juntas: se vinculan por medio de diferentes áreas y se complementan entre ellas, logrando una administración de los procesos vinculadas al desarrollo de software. [24]

En la tabla 1.6 se muestran las características que envuelve le uso de Scrum y XP.

Características
Las dos son un manifiesto ágil.
Existe una interacción de usuario a usuario.
Realizan los Proyectos en corto tiempo.
Trabajan en Equipo.
Reuniones exprés, de pie, entre el equipo.
Entregas rápidas y continuas en corto tiempo.

Tabla 1.6: SCRUM y XP.
Fuente: Elaborado por los Autores.

1.2.3.1 Prácticas a utilizarse de cada Metodología.

Según Henril Kniberg en su libro SCRUM y XP desde las Trincheras, da a conocer las prácticas más relevantes y que han dado éxito al momento de utilizar al combinación de ambas metodologías, por lo cual hemos adoptado dichas prácticas como válidas para nuestro sistema.

Prácticas de SCRUM.

Las prácticas y técnicas que se han tomado en cuenta de la metodología SCRUM son las siguientes.

- **La pila del producto o product backlog:** Se define como la recolección de los requisitos del cliente plasmados en un documento necesario e indispensable para el desarrollo del producto.
- **La pila de tareas o sprint backlog:** Definida como el conjunto de todas las historias de usuario pertenecientes a la pila del producto, las cuales se usan en cada sprint.

- **Estimación del esfuerzo:** Corresponde a todos los días ideales de una persona en los cuales puede trabajar en una historia de usuario definiendo puntos de historia.
- **Gráfico Burn-down:** Este gráfico ayuda a medir la productividad y detectar riesgos de mala distribución del trabajo y desviaciones temporales.
- **Reuniones para cada sprint:** Reuniones finales en cada sprint donde se realiza una retroalimentación del funcionamiento de todas las atareas del sprint.
- **Reuniones de seguimiento diarias:** Reuniones expres donde se determina cambios o la incorporación de nuevos ítems en el sprint.
- **Planning Poker:** Técnica usada para realizar la estimación de las historias. Simula el juego de Poker para obtener una estimación colectiva y cooperativa de cada historia.

Prácticas de XP.

Las prácticas y técnicas que se han tomado en cuenta de la metodología XP son las siguientes.

- **Test Driven Development:** Diseñar los casos de pruebas antes que implementar las funcionalidades.
- **Refactorizar:** Rescribir métodos y clases, para mejorar la legibilidad del código. Realizada a partir de las pruebas de usabilidad y funcionamiento cuando se genera software de manera incremental y a partir de las pruebas.
- **Diseño incremental:** Mantener un diseño simple desde el principio e ir mejorándolo continuamente.
- **Integración continúa:** Técnica que permite compilar el código o cambios de código cada vez que se sube al repositorio de código, las pruebas de ejecución se las realiza en un ambiente de prueba ates de desplegarlo en un entorno de producción.

  A demás, se puede realizar dicho proceso de forma automática, permitiendo un cambio cada cierto tiempo, con lo cual se puede observar los diferentes resultados de las pruebas en diferentes condiciones y horas,

no obstante cada descripción de la integración continua se la debe realizar en un ambiente de test antes de lanzar los cambios a producción.[24]

Roles

De acuerdo a Kniberg Henrik en su libro Scrum y Xp desde las Trincheras, propone que un buen equipo de trabajo se basa en las diferentes opiniones, de acuerdo a vivencias y experiencias a la hora del trabajo en equipo y a las buenas prácticas.

En la Tabla 1.7 Se muestran los roles que se usaran en el proyecto.

SCRUM	XP
Product Owner: Geovana Saltos Manager: Sandra Sánchez Team: • Leonardo Guachamín • Wilson Castro Stackholder: Andrea Plaza	User: Geovana Saltos Tracker: Wilson Castro Coach: Sandra Sánchez Consulter: Andrea Plaza Tester: • Leonardo Guachamín • Wilson Castro Big Boss: Sandra Sánchez

Tabla 1.7: Roles XP y SCRUM
Fuente: Elaborado por los Autores

1.3 JUSTIFICACIÓN DE LAS HERRAMIENTAS A UTILIZAR

La Escuela Politécnica Nacional como entidad pública, apoya al Decreto Presidencial 1014, que adopta el Software Libre (SL) como política de estado: “Establecer como política pública para las Entidades de la Administración Pública Central la utilización de Software Libre en sus sistemas y equipamientos

informáticos.” [2] Por ende, la Institución ha establecido el uso las siguientes herramientas de software libre para el desarrollo de aplicaciones institucionales:

a. Java Enterprise Edition como modelo de programación. Este modelo provee un estándar simple y unificado basado en componentes, aportando así; flexibilidad, escalabilidad y portabilidad a los sistemas.
b. JavaServer Faces como marco de trabajo para la construcción de las interfaces de usuario debido a que posee una amplia librería de etiquetas para componentes UI, esta tecnología utiliza el modelo vista controlador permitiendo separar las vistas de la lógica del negocio lo que ayuda a simplificar el desarrollo de las aplicaciones.
c. JBOSS AS como servidor de aplicaciones, debido a su robustez, alto rendimiento y fácil configuración, brindando así, confiabilidad y rapidez a los sistemas.
d. PostgresSQL como motor de base de datos ya que permite administrar y almacenar la información aportando integridad y fiabilidad de la misma.

Debido a que nuestro sistema favorece a la automatización de procesos institucionales en este caso la administración de espacios físicos, se acoge a la utilización de las mismas herramientas que se han venido utilizando en la institución.

A continuación se describe cada herramienta y sus principales ventajas.

1.3.1 J2EE

Java 2EE consiste en un esquema de desarrollo de software empresarial basado en una arquitectura multicapas distribuida, con componentes reusables, un modelo de seguridad unificada, control de transacciones, y soporte para servicios web.

1.3.1.1 Características

- Es un conjunto de estándares y especificaciones para el desarrollo aplicaciones basadas en tecnología Java.

[2] Tomado del Anexo6 - Decreto 1014 Software Libre en Ecuador, artículo 1.

- Define un modelo de aplicaciones distribuido y multicapa con n-niveles.
- Posee una arquitectura simplificada que se basa en componentes, servicios y clientes estándar.
- Permite simplificar el modelo de desarrollo.

1.3.1.2 Arquitectura

La plataforma J2EE utiliza un modelo de programación distribuido en distintas capas, asegurando escalabilidad, accesibilidad y facilidad de gestión en un ámbito empresarial. Su arquitectura está compuesta por una capa cliente, una capa intermedia que a su vez esta formada por una o más subcapas, y una capa de datos, conocida como la capa de información empresarial (IE). [3]

- **Capa Cliente**

La capa cliente consiste de aplicaciones clientes que acceden al servidor Java EE que normalmente se encuentra en una máquina diferente a la del servidor. Es la responsable de la Interacción con el usuario. [4] La capa cliente soporta dos tipos básicos de aplicaciones clientes:

- Aplicaciones Web

  Estos utilizan los navegadores web como el software de cliente, como por ejemplo: Mozilla FireFox, Internet Explorer, Chrome, Opera, entre otros.

- Aplicaciones cliente Stand-Alone

  Son aplicaciones cliente que por lo general no están escritas en JAVA. Las aplicaciones cliente Stand-Alone se ejecuta en los equipos cliente y proporcionan funcionalidades de interfaz de usuario, como barras de herramientas, barras de menús, soporte para arrastrar y soltar, ayuda sensible al contexto, y funciones deshacer / rehacer.

- **Capa de Presentación**

Corresponde al punto de acceso que tiene el cliente con la aplicación para solicitar servicios, y la forma en que la aplicación envía las respuestas al cliente. [4]

En la Tabla 1.8 Se puede apreciar las tecnologías J2EE utilizadas en la capa de presentación.

Tecnología	Objetivo
Servlets	Lenguaje de programación Java, que procesa peticiones y construyen respuestas dinámicamente, por lo general para páginas HTML.
Tecnología Java Server Faces	Un framework de componentes de interfaces de usuario para aplicaciones web que permite incluir componentes como campos y botones en una página.
Tecnología Java Server Faces Facelets	Tipo de aplicaciones JavaServer Faces que utilizan páginas XHTML en lugar de páginas JSP.
Expression Language	Conjunto de etiquetas estándar utilizado en las páginas JSP y Facelets.
JavaServer Pages (JSP)	Documentos basados en texto que se compilan en servlets y definen cómo se puede agregar contenido dinámico a las páginas estáticas como páginas HTML.
JavaServer Pages Standard Tag Library	Librería de etiquetas que encapsula la funcionalidad básica común a las páginas JSP.
Componentes JavaBeans	Los objetos que actúan como almacenes de datos temporales para las páginas de una aplicación.

Tabla 1.8: Tecnologías JEE en la capa de presentación.
Fuente: Overview of Enterprise Applications, http://goo.gl/vML0u3, modificado por los autores.

- **Capa Lógica de Negocio**

Esta capa contiene los requerimientos funcionales solicitados para el desarrollo de la aplicación, utiliza Enterprise Java Beans (EJB) para encapsular la funcionalidad del sistema, que a su vez están desplegados dentro de un contenedor EJB el cual permite manejar transacciones, seguridad y conexiones remotas, permitiendo a los desarrolladores enfocarse en resolver la lógica del negocio. [5] [4]

En la Tabla 1.9 Se puede apreciar las Tecnologías J2EE utilizadas en la capa de negocio.

Tecnología	Descripción
Componentes Enterprise JavaBeans (enterprise bean)	Componentes gestionados que encapsulan la funcionalidad principal de la aplicación.
Web services JAX-RS RESTful	API para la creación de servicios web que responden a los métodos HTTP. Estos servicios se desarrollan de acuerdo a los principios de REST, o transferencia de estado representacional.
Web service endpoints JAX-WS	API para crear y consumir servicios web SOAP.
Entidades Java Persistence API	API para tener acceso a datos en bases de datos subyacentes y mapear esos datos a objetos del lenguaje de programación Java.
J2EE managed beans	Componentes gestionados que pueden proporcionar la lógica de negocio de una aplicación, pero no necesitan las características transaccionales o de seguridad de los enterprise beans.

Tabla 1.9: Tecnologías JEE en la capa de negocio
Fuente: Overview of Enterprise Applications, http://goo.gl/vML0u3, modificado por los autores.

- **Capa de Integración y Recursos**

La capa de Integración y Recursos maneja la persistencia de la aplicación utilizando sistemas controladores de bases de datos, y sus respectivos dispositivos de conexión. [4]

En la tabla 1.10 se pueda apreciar las Tecnologías J2EE utilizadas en la capa Integración y recursos.

Tecnología	Descripción

Java Database Connectivity API (JDBC)	API de bajo nivel para el acceso y recuperación de datos de los almacenes de datos subyacentes.
Java Persistence API	API para tener acceso a datos en almacenes de datos subyacentes y mapear esos datos a objetos del lenguaje de programación de Java.
J2EE Connector Architecture	API para conectarse a otros recursos de la empresa, como planificación de recursos empresariales o software del sistema de gestión de clientes.
Java Transaction API (JTA)	API para definir y gestionar las transacciones, incluyendo las transacciones distribuidas o transacciones que cruzan múltiples fuentes de datos subyacentes.

Tabla 1.10: Tecnologías JEE en la capa de integración y recursos
Fuente: Overview of Enterprise Applications, http://goo.gl/vML0u3, modificado por los autores.

En la Figura 1.10 podemos apreciar las diferentes capas que componen la tecnología J2EE.

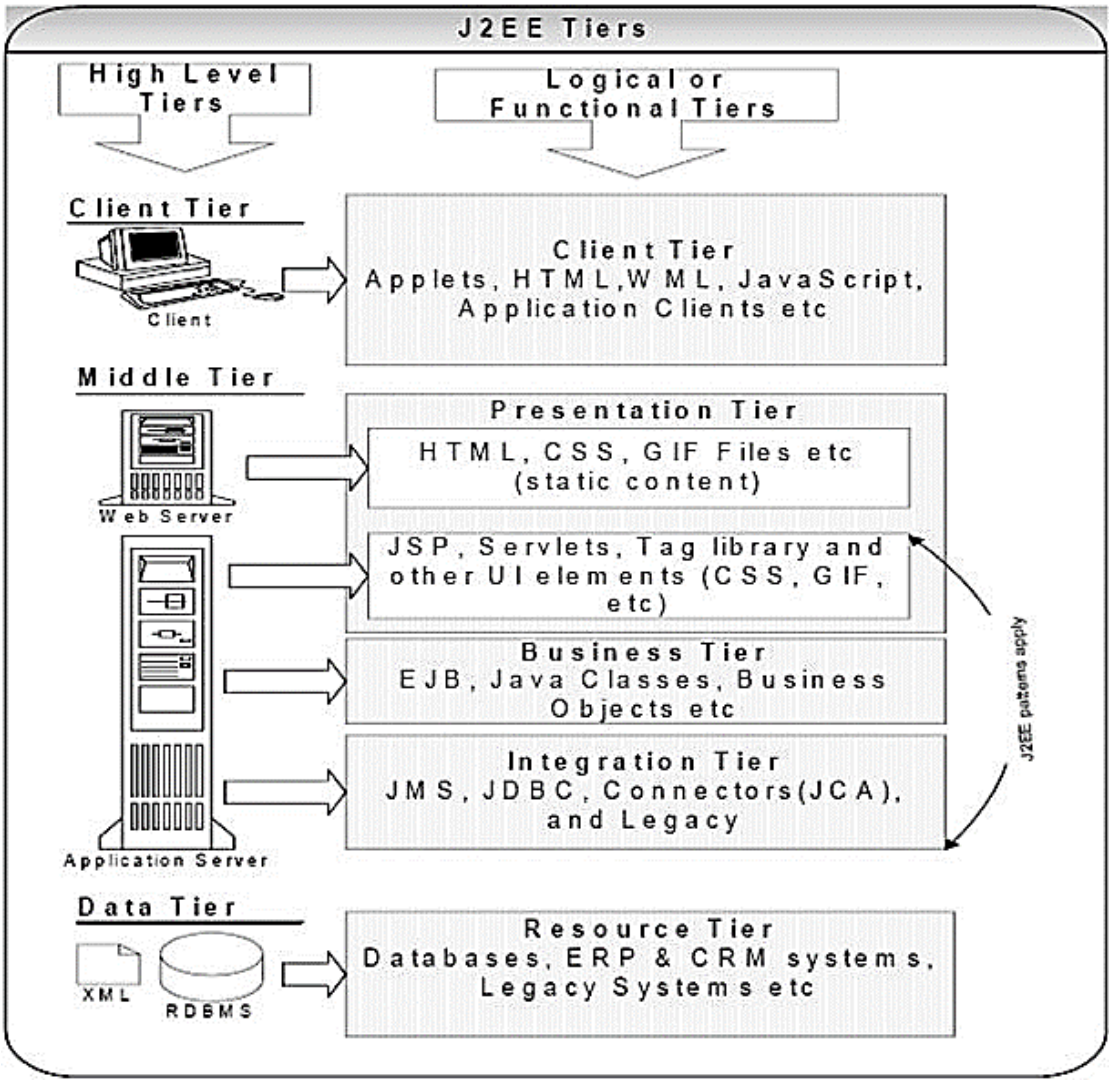


Fig. 1.10 Capas J2EE
Fuente: Java Forums, http://www.java-forums.org/blogs/J2EE/687-J2EE-tiers.html

1.3.1.3 Elementos de la Plataforma J2EE

Componentes J2EE

Es una unidad de software funcional auto contenida que se ensambla como parte de una aplicación J2EE y que puede interactuar con otros componentes. [6]

En la Tabla 1.11 se aprecia los tipos de componentes J2EE

Componentes cliente	• Aplicaciones Java SE (AWT/Swing, Applets) • Navegadores web: Flrefox, Chrome, lExplorer
Componentes web	• Java Servlets • JavaServer Pages (JSP) • JavaServer Faces (JSF)
Componentes de negocio	• Enterprise JavaBeans (EJB)

Tabla 1.11: Tipos de componentes J2EE.
Fuente: Elaborado por los Autores.

Contenedores J2EE

Los contenedores J2EE ofrecen una serie de servicios tales como seguridad, gestión de transacciones, gestión multiusuario, compartición de recursos entre otros, lo que permitiendo que el desarrollador se centrarse en el desarrollo de la lógica de negocio de la aplicación. [3]

Tipos de contenedores J2EE:

- Contenedor cliente (Application Client Container o Applet Container).
- Contenedor web (Web Container).
- Contenedor de negocio o de EJBs (EJB Container).

Servicios

Las especificaciones J2EE, definen una serie de tareas que los distintos tipos de contenedores deben implementar y ofrecer a los componentes, para que puedan ser utilizados por los desarrolladores. [7]

En la Tabla 1.12 se destacan ciertos servicios:

Servicio	Función
De directorio	Indexación y búsqueda de componentes y recursos.
De despliegue	Descripción y personalización de componentes a la hora de su instalación.
De transaccionalidad	Ejecutar distintas acciones en una misma unidad transaccional.
De seguridad	Autenticar y autorizar a los usuarios de una aplicación.
De acceso a datos	Facilitar el acceso a las Bases de Datos.
De conectividad	Facilitar el acceso a los distintos Sistemas de Información Empresarial.
De mensajería	Comunicarse con otros componentes, aplicaciones o Sistemas de Información Ejecutiva (EISs).

Tabla 1.12: Servicios en la plataforma J2EE. [7]
Fuente: Elaborado por los Autores.

1.3.1.4 Ventajas

- Productividad: es una plataforma multi-compatible con los servicios web e idiomas resultando productivo para una variedad de aplicaciones tales como e-learning, comercio electrónico, encuestas en línea y otros tipos de desarrollos a medida.
- Escalabilidad: incluyen la agrupación de recursos (agrupación de conexiones de base de datos, agrupación de instancias de beans de sesión y agrupación de hebras), paso de mensajes asíncronos y una gestión de ciclo de vida de componentes más eficiente.
- Portabilidad: Las aplicaciones y componentes J2EE son portables en relación con servidores J2EE compatibles, sin que sea necesario modificar el código.
- Reutilización de componente: Los componentes J2EE se pueden comprar como paquetes disponibles comercialmente y adjuntarlos a su aplicación J2EE según sea necesario.
- Costo: Es una plataforma de código abierto.
- Soporte y documentación: En su página oficial se puede obtener excelente documentación y soporte para la plataforma.
- Seguridad: Proporciona potentes y simples características de seguridad.

1.3.2 SERVIDOR DE APLICACIONES JBOSS

JBoss es un servidor de aplicaciones J2EE de código abierto implementado en Java puro. Al estar escrito íntegramente en Java, puede ser utilizado en cualquier sistema operativo que disponga de una Máquina Virtual de Java (JVM).

En 1999 Marc Fleury empezó el proyecto JBoss con una visión comunitaria pensando en un servidor de aplicaciones de código libre creado por y para desarrolladores. Para abril de 2006 Red Hat adquiere JBoss Inc., donde incorpora un portfolio Middleware, desde su adquisición está disponible en dos versiones, el comunitario JBoss-AS y el empresarial JBoss-EAP. [8] [9]

1.3.2.1 Características

JBoss en particular ha sido desde sus inicios un servidor de aplicaciones extensible y modular.

- Producto de licencia de código abierto sin coste (JBoss-AS).
- Cumple los estándares: JSR-168, JSR-170, JSR-252, JMX 1.2.
- Confiable a nivel de empresa.
- Incrustarle, orientado a arquitectura de servicios.
- Flexibilidad consistente.
- Servicios del middleware para cualquier objeto de Java.
- Soporte empresarial 24x7 (JBoss-EAP).
- Soporte completo para JMX. [10]

1.3.2.2 Arquitectura

El servidor de aplicaciones JBoss es una gran colección de componentes independientes e interdependientes, cada uno de los cuales se centra en un área específica de la funcionalidad J2EE.

JBoss fue construido alrededor del kernel JMX (Java Management Extension). Este kernel provee un conjunto básico de funcionalidades y todos los servicios suministrados por el servidor de aplicaciones se escriben como Beans gestionados (MBeans) conectados al kernel JMX. La arquitectura JMX está compuesto por tres niveles los cuales son:

- Nivel de Instrumentación: es donde se localizan los componentes que facilitan información necesaria para la gestión de una aplicación.
- Nivel de Agente: proporciona una interfaz de comunicación que permite la administración de los componentes del nivel de instrumentación.
- Nivel de Servicios distribuidos: mecanismo por el cual la administración aplicaciones interactúan con agentes y sus objetos administrados. [11]

En la figura 1.11 se muestra el funcionamiento arquitectónico de JBOSS.

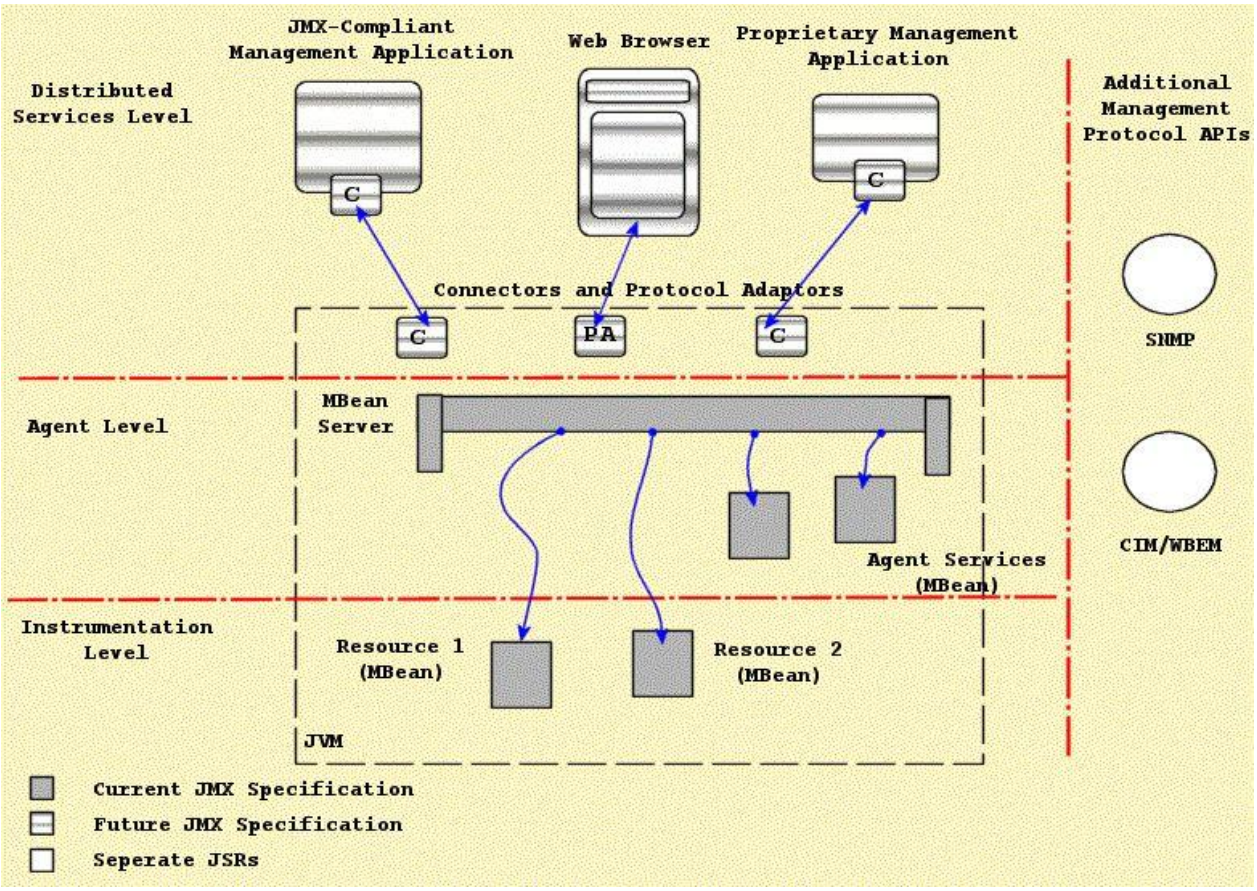


Fig. 1.11: Arquitectura JBOSS
Fuente: The JBoss 4 Application Server Guide, http://goo.gl/LdfLdb

1.3.2.3 Ventajas

JBoss Application Server aprovecha ventajas de versiones anteriores y ofrece un mejor rendimiento, una menor utilización de memoria, una gestión distribuida y certificación J2EE6 Web Profile.

Actualmente existen 7 razones para usar JBOSS AS y son las siguientes:

- Rapidez en el arranque: Los servicios se arrancan en paralelo para eliminar esperas innecesarias y aprovechar los procesadores multinúcleo.
- Ligero: Han mejorado la gestión de la memoria permitiendo ejecutar JBoss incluso en pequeños dispositivos.
- Núcleo modular: JBoss carga solamente las clases de aplicación necesarias. El cargador de clases es concurrente, para mejorar el desempeño.
- Despliegue incremental: tiene la posibilidad de editar recursos estáticos sin necesidad de desplegar otra vez.
- Administración elegante: posee una consola de administración web atractiva y amigable.
- Gestión de dominios: puede arrancar de dos maneras: modo standalone, donde se tiene una instancia, o modo Dominio, para crear una topología multi-servidor.
- Componentes de primera clase: se compone de varios proyectos Open-Source Software como, Hibernate, Mojarra, Apache CXF, Infinispan, RESTEasy, Weld, HornetQ, Iron Jacamar, JGroups. [12]

1.3.3 JAVASERVER FACES

JavaServer Faces fue creado a través del Java Community Process (JCP), por un grupo de líderes de tecnología como Sun Microsystems, Oracle, Borland, BEA e IBM, conjuntamente con un equipo de expertos en Web y Java.

Se lo puede catalogar como un framework de componentes para construir la interfaz de usuario de aplicaciones web de una forma más sencilla, eficaz y eficiente basándose en una arquitectura bien diseñada y fácil de mantener.

1.3.3.1 Características

- Representación de componentes de interfaz de usuario (UI-User Interface) y manejo de su estado.
- Manejo de eventos, conversión de datos y validación del lado del servidor.
- Definición de la navegación entre páginas.
- Soporte internacional y accesibilidad.

- Extensibilidad y accesibilidad.
- Bibliotecas de etiqueta para añadir componentes a páginas Web y para unir componentes con objetos del lado del servidor.

1.3.3.2 Modelo de navegación JSF

JSF sigue el paradigma de diseño Modelo-Vista-Controlador. El cual se divide en tres componentes:

- El modelo, que contiene la lógica de negocio o código de no-UI.
- La Vista, que es todo el código necesario para presentar una interfaz de usuario para el usuario.
- El controlador, que es un agente front-end que maneja directamente las peticiones del usuario y envía la vista apropiada.

En la Figura 1.12 se visualiza el modelo MVC.

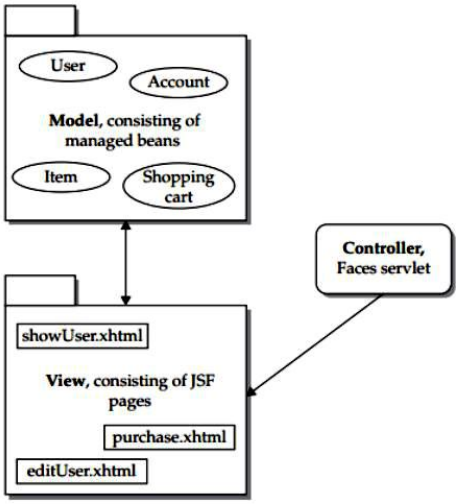


Fig. 1.12: Modelo Vista Controlador
Fuente: Ed Burns; Chris Schalk with Neil Griffin, JavaServer Faces 2.0: The Complete Reference, 2010 pág. 13

JavaServer Faces desde su principio fue creado precisamente para adherirse a la metodología de diseño MVC. Ofreciendo así un método limpio para separar el código de la presentación (Vista) del código back-end de la lógica del negocio (Modelo). También, proporciona un front-end servlet (Controlador) que controla

todas las solicitudes Faces y las envía junto con los datos necesarios de la aplicación, a la vista de componente apropiado (Página). [13]

1.3.3.3 Arquitectura

Una aplicación JSF es similar a cualquier otra aplicación web basado en tecnología Java la cual se ejecuta en un contenedor de servlets de Java.

En la Tabla 1.13 se describen los elementos que conforman esta tecnología.

Componente UI	Un objeto con estado, mantenido en el servidor, que proporciona funcionalidad específica para interactuar con un usuario final. Son javabeans con propiedades, métodos y eventos. Organizados en una vista (árbol de componentes visualizado como una página).
Renderer:	Responsable de mostrar un componente UI y traducir una entrada de usuario al valor del componente. Los renderers pueden ser diseñados para trabajar con uno o más componentes UI, y un componente UI se puede asociar con muchos renderers diferentes.
Validador:	Responsable de asegurar que el valor introducido por un usuario es aceptable. Uno o más validadores pueden ser asociados con un sencillo componente UI.
Backing beans:	Javabeans especializados que coleccionan valores de componentes UI e implementan métodos oyentes de eventos. También pueden mantener referencias a componentes UI.
Conversores:	Convierte el valor de un componente a/desde un String para visualizar. Un componente UI puede ser asociado con un convertidor sencillo.
Eventos y Oyentes	JSF utiliza el modelo evento/oyente de javabeans (también utilizado por Swing). Los componentes UI (y otros objetos) generan eventos, y pueden registrarse oyentes para manejar esos eventos.
Mensajes e internacionalización:	La información que se visualiza de vuelta al usuario. Cualquier parte de la aplicación (beans de respaldo, validadores, convertidores, etc) pueden generar información

	o mensajes de error que pueden ser mostrados al usuario.
Navegación:	La habilidad de moverse de una página a la siguiente. JSF tiene un potente sistema de navegación que está integrado con oyentes de evento especializados.

Tabla 1.13: Elementos de JSF.
Fuente: Kito D. Mann, JavaServer Faces in Action, 2005 pág. 3, modificado por los autores.

En la Figura 1.13 podemos apreciar la interacción entre los diferentes elementos que conforman la tecnología JSF.

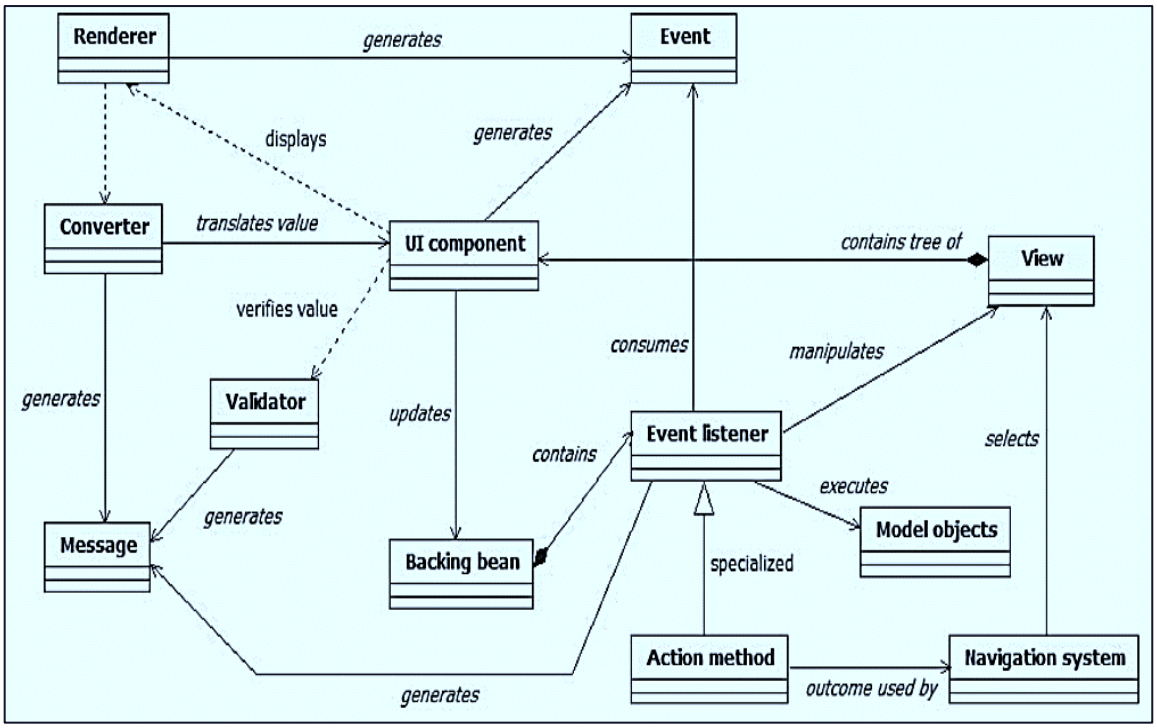


Fig. 1.13: Interacción entre los elementos la de tecnología JSF.
Fuente: Kito D. Mann, JavaServer Faces in Action, 2005 pág. 39, modificado por los autores.

1.3.3.4 Ciclo de vida

El ciclo de vida de una aplicación JavaServer Faces comienza cuando el cliente realiza una petición HTTP de una página y termina cuando el servidor responde con una página adecuada, traducida a HTML.

El ciclo de vida puede dividirse en dos fases principales, ejecutar y representar. La fase de ejecución se subdivide en las siguientes fases. [14]

Restore View: crea o restaura un árbol de componentes del lado del servidor para presentar la información UI de un cliente.

Apply Request Values: actualiza los componentes del lado del servidor con datos nuevos del cliente. Los nuevos valores son almacenados localmente en el componente.

Process Validations: realiza todas las conversiones y validaciones de nuevos datos.

Update Model Values: actualiza cualquier objeto del modelo del lado del servidor con nuevos datos.

Invoke Application: invoca a cualquier lógica de la aplicación para cumplir con una solicitud y navegar a una nueva página si fuera necesario.

RenderResponse: invoca las propiedades de codificación de los componentes guardando el estado respondiendo al cliente solicitante.

En la Figura 1.14 se puede apreciar la el ciclo de vida correspondiente a la tecnología JSF.

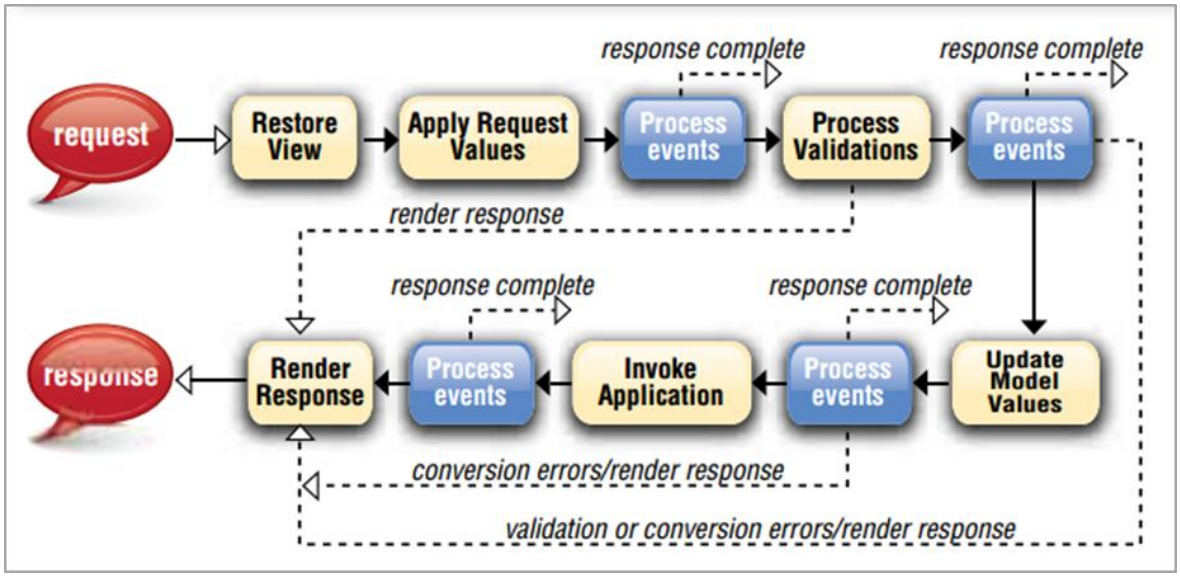


Fig. 1.14: Ciclo de vida JSF.
Fuente: Cay Horstmann, JSF Overview-JavaServer Faces 2.0, www.dzone.com.

1.3.3.5 Ventajas

- Es una tecnología Open Source y tiene un fuerte apoyo de empresas grandes como Oracle, IBM, JBoss, entre otros.
- Permite que los desarrolladores se enfoquen en su parte del proceso de desarrollo, proporcionando un modelo de programación que integra todas las partes de forma sencilla gracias a que la lógica esta separada de la presentación.
- Permite crear y reutilizar con facilidad los componentes de una interfaz de usuario, esto permite mejorar la productividad y consistencia de la aplicación.
- Permite desarrollar componentes a medida, ya que es extensible, además puede modificar los comportamientos del framework mediante APIs que controlan el funcionamiento
- Tiene un fuerte apoyo para Expression Language[3] que permite mejorar la legibilidad de código de interface de usuario.
- Resuelve eventos, validaciones, conversiones, mensajes de error, internacionalización y accesibilidad.

[3] http://docs.oracle.com/javaee/6/tutorial/doc/gjddd.html

1.3.4 POSTGRESQL

Es el líder de tecnología en base de datos objeto-relacional de código abierto, ofrece tecnología clave para el entorno empresarial, como vistas, disparadores, procedimientos almacenados y seguridad, está distribuido bajo licencia BSD (Berkeley Software Distribution), como se llamó a las distribuciones de código fuente desarrolladas en la Universidad de Berkeley en California mismas que en un origen eran extensiones de UNIX, permitiendo que evolucione rápidamente y gracias al apoyo de una gran comunidad de desarrolladores que trabajan de forma libre o apoyados por organizaciones comerciales. [15] [16]

1.3.4.1 Características

Posea un gran conjunto de características que se han ido mejorando e incrementando versión tras versión, a continuación se listan dichas características.

- Soporta consultas con EXCEPT UNION, UNION ALL, OUTER JOINS, Sub-selects.
- Cumple con ANSI SQL (implementando el estándar SQL92 y SQL99).
- Integridad referencial.
- Duplica bases de datos maestras en diferentes sitios de réplica.
- Dispone de interfaces nativas para JDBC, ODBC, C++, C, PHP, TCL, Python, Ruby, Perl, TCL, ECPG.
- Se puede hace uso de reglas, vistas, store procedure, triggers, unicode, secuencias.
- Incluye herencia entre tablas por ello se lo clasifica como gestor objeto relacional.
- Una API abierta.
- Soporte nativo SSL.
- Lenguajes procedurales.
- Respaldo en caliente.
- Bloqueo a nivel.
- Índices parciales y funcionales.
- Autentificación Kerberos nativa.

- Extensiones para MD5, SHA1, XML
- Herramientas para generar SQL portable para compartir con otros sistemas compatibles con SQL.
- Sistema de tipos de datos extensible para proveer tipos de datos definidos por el usuario, y rápido desarrollo de nuevos tipos.
- Funciones de compatibilidad para ayudar en la transición desde otros sistemas menos compatibles con SQL.
- Soporta distintos tipos de datos
- Posee funciones de manejo de fechas, geométricas, operaciones con redes.
- Cumple completamente con ACID.
- Permite alta disponibilidad con consistencia sobre múltiples servidores.
- Soporta correctamente el ordenamiento por lenguaje en las bases de datos, tablas o columnas.
- Mantiene consistentes múltiples transacciones concurrentes sin el uso de bloqueos, usando verdadera serialización.
- Ejecuta actualizaciones multi-fases complejas en una simple consulta.
- Despliega seguridad de nivel militar y control de acceso mandatorio.
- Utiliza índices basados en distancias para consultas rápidas de ubicación y búsquedas de texto. [17]

1.3.4.2 Arquitectura

PostgreSQL usa un modelo cliente/servidor. Una sesión de PostgreSQL se compone de un programa servidor llamado PostgreSQL, el cual puede atender exclusivamente a un solo cliente; es decir, hacen falta tantos procesos servidor PostgresSQL como clientes haya. También dispone de diversos programas cliente como Pgaccess (un cliente gráfico) y Psql (un cliente en modo texto). Los clientes pueden ejecutarse en el mismo sitio o en equipos remotos conectados por TCP/IP.

En la Figura 1.15 se muestra el esquema arquitectónico de PostgresSQL.

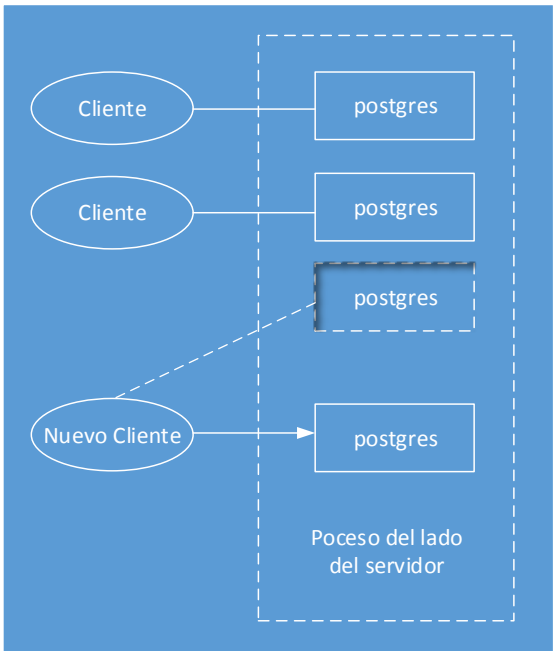


Fig. 1.15: Arquitectura PostgreSQL.
Fuente: Dataprix, http://www.dataprix.com/72-arquitectura-postgresql, modificado por los autores.

1.3.4.3 Componentes

- Aplicación cliente: Aplicación cliente que utiliza PostgreSQL como administrador de bases de datos.
- Demonio postmaster: Proceso principal de PostgreSQL que se encarga de la ejecución de un nuevo servidor cada uno de los clientes que soliciten una conexión.
- Ficheros de configuración: PostgresSQL utiliza los siguientes archivos de configuración, pg_hba.conf y pg_ident.conf, postgresql.conf.
- Procesos hijos postgres: Permite autentificar clientes, gestionar consultas y enviar los resultados a las aplicaciones clientes.
- PostgreSQL share buffer cache: Memoria compartida que utiliza PostgreSQL para almacenamiento de datos en caché.
- Write-Ahead Log (WAL): Componente del sistema que garantiza integridad de los datos.
- Kernel disk buffer cache: Caché de disco del sistema operativo.

- Disco: Unidad física donde se almacenan datos y la información necesaria para que PostgreSQL se ejecute.
- Storage Manager: Colección de módulos encargados de la administración general del almacenamiento de datos, memoria, transacciones y concurrencia además se encarga de controlar la consistencia de la información.
- Libpq: Responsable de manipular las comunicaciones entre la aplicación cliente y el postmaster. [18]

En la Figura 1.16 se observa los componentes de funcionamiento de PostgresSQL.

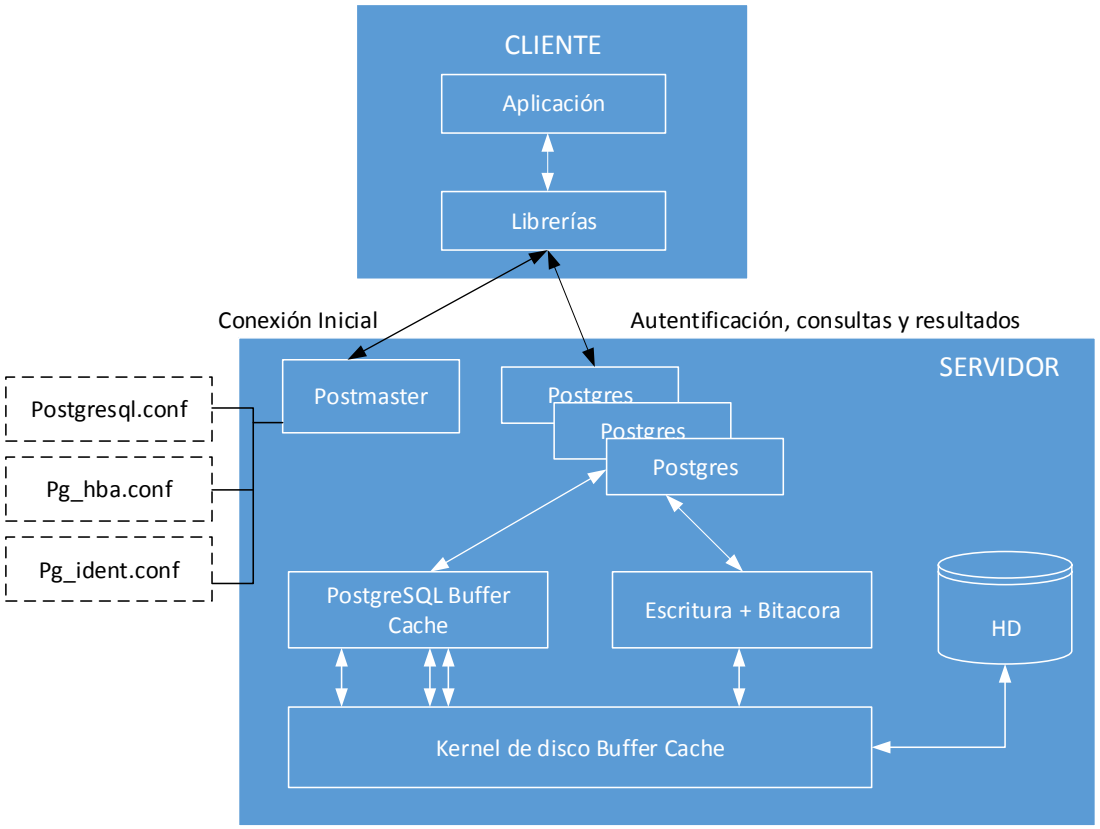


Fig. 1.16: Componentes PostgreSQL.
Fuente: PostgreSQL-es, http://www.postgresql.org.es/sobre_postgresql, modificado por los autores.

1.3.4.4 Ventajas

- Licencia: 100% open Source.
- Excelente cumplimiento del estándar SQL, siguiendo el último SQL: 2011.

- Provee integridad de datos, nunca pierda una transacción o viola una clave.
- Altamente configurable y extensible para muchos tipos de aplicación.
- Cuenta con un optimizador de consultas sofisticado, adecuado para inteligencia de negocios.
- Altamente confiable con características extensivas para durabilidad y alta disponibilidad.
- Aporta fiabilidad ya que lleva 15 años de desarrollo y está respaldado por una gran comunidad de profesionales que brindan un soporte excelente.
- Altamente escalable debido a la enorme cantidad de datos que puede manejar y al número de usuarios concurrentes que puede acomodar.
- Sumamente seguro, utiliza SSL para realizar conexiones TCP/IP, funciones criptográficas, se integra con infraestructuras de seguridad empresariales como LDAP o GSSAPI, incluye permisos para grupos y usuarios.

2 CAPÍTULO 2: DESARROLLO DEL SISTEMA UTILIZANDO SCRUM Y XP

2.1 ESPECIFICACIÓN DE REQUISITOS – HISTORIAS DE USUARIO (PRODUCT BACKLOG)

2.1.1 HISTORIAS DE USUARIO

Con el fin de capturar los requerimientos del sistema se pretende usar historias de usuario las mismas que representan los deseos del propietario del sistema.

Por medio de las reuniones establecidas entre el Product Owner y los involucrados en el desarrollo del sistema, se han establecido las siguientes historias de usuario:

<table>
<tr><td colspan="2">Historias de usuario</td></tr>
<tr><td>Numero: 1</td><td>Nombre: Alquiler de espacios para eventos</td></tr>
<tr><td colspan="2">Usuario: Administrador</td></tr>
<tr><td>Prioridad en el negocio(alta, media, baja): alta</td><td>Riesgo en el desarrollo(alta, media, baja):alta</td></tr>
<tr><td colspan="2">Responsable:
Leonardo, Wilson</td></tr>
<tr><td colspan="2">Descripción:
El aplicativo contará con un módulo para administrar el alquiler de espacios para eventos, el mismo que podrá registrar el costo, si existe algún saldo y una descripción del alquiler. Además se podrá actualizar, buscar y eliminar dichos registros.
El alquiler se realizará para eventos de tipo externo principalmente, para los eventos de tipo interno no se realiza el registro del alquiler. Las normas antes establecidas pueden regirse o no dependiendo de disposiciones de las autoridades competentes ante los eventos que se realicen en las instalaciones de la EPN.
Un evento interno puede registrar un alquiler solo si tiene autorización del Rector de la Escuela Politécnica Nacional.
Un evento externo no puede registrar un alquiler solo si tiene autorización del Rector de la Escuela Politécnica Nacional.</td></tr>
<tr><td colspan="2">¿Cómo comprobarlo?
El administrador podrá visualizar fechas disponibles para hacer uso del espacio mediante una consulta a la base de datos
El administrador ingresará información relaciona al alquiler de evento.
El administrador comprobará que se guardó la información del alquiler de evento</td></tr>
</table>

mediante mensajes de comprobación

Historias de usuario	
Numero: 2	**Nombre:** Cobranzas de alquiler de espacios para eventos
Usuario: Administrador	
Prioridad en el negocio(alta, media, baja): alta	**Riesgo en el desarrollo(alta, media, baja): alta**
Responsable: Leonardo Wilson	
Descripción: El aplicativo permite llevar un registro del alquiler de espacios para eventos que contiene los pagos de cada alquiler para lo cual se establece un monto, un saldo y una fecha de pago. Este registro se puede actualizar, buscar y eliminar.	
¿Cómo comprobarlo? • El usuario puede ingresar el monto del alquiler de un espacio para evento, procederá a guardarlo. • El saldo y la fecha de pago se guardan automáticamente por medio del sistema. • A continuación se presenta un mensaje para confirmar su registro exitoso	

Historias de usuario	
Numero: 3	**Nombre:** Agenda de espacios
Usuario: Administrador	
Prioridad en el negocio(alta, media, baja): alta	**Riesgo en el desarrollo(alta, media, baja): alta**
Responsable: Leonardo Wilson	
Descripción: El sistema contará con una agenda para verificar la disponibilidad y capacidad máxima de personas de un determinado espacio en el que se desea realizar un evento	
¿Cómo comprobarlo? • El administrador puede verificar los espacios disponibles para realizar eventos.	

Historias de usuario	
Numero: 4	**Nombre:** Asignación de espacios a ocupantes
Usuario: Administrador	
Prioridad en el negocio(alta, media, baja): alta	**Riesgo en el desarrollo(alta, media, baja): alta**
Responsable: Leonardo Wilson	
Descripción: El aplicativo permite asignar espacios del campus politécnico a uno o varios ocupantes, por ejemplo asignar oficinas para profesores o personal administrativo.	
¿Cómo comprobarlo? • El usuario puede seleccionar un espacio, verifica su disponibilidad, capacidad. • Escoger un horario de uso y días. • El usuario registra un único ocupante en dicho espacio para un determinado día y horario. • El usuario constata el registro mediante un mensaje de guardado exitoso.	

Historias de usuario	
Numero: 5	**Nombre:** Designación de espacios
Usuario: Administrador	
Prioridad en el negocio(alta, media, baja): alta	**Riesgo en el desarrollo(alta, media, baja): alta**
Responsable: Leonardo Wilson	
Descripción: El aplicativo permite seleccionar la ubicación de un espacio dentro del campus politécnico, además permite establecer su capacidad y un costo diario de uso si lo tuviera.	
¿Cómo comprobarlo? • El usuario puede buscar un edificio de campus politécnico, seleccionar un piso del edificio y asignar un nuevo espacio a dicho piso. • El usuario puede verificar los espacios en un plano. • El usuario puede marcar el nuevo espacio en el plano, registrar su capacidad y su costo. • Si el espacio se encuentra registrado anteriormente, no se realizará el ingreso del	

nuevo espacio en el plano y se genera un mensaje de advertencia en la pantalla.
- El usuario constata el registro del espacio en el plano mediante un mensaje de guardado es exitoso.

Historias de usuario	
Numero: 6	**Nombre:** Carga de planos arquitectónicos
Usuario: Administrador	
Prioridad en el negocio(alta, media, baja): media	**Riesgo en el desarrollo(alta, media, baja): media**
Responsable: Leonardo, Wilson	
Descripción: El aplicativo permitirá ingresar imágenes de los planos arquitectónicos de los espacios físicos con los que cuente el campus politécnico.	
¿Cómo comprobarlo? • El usuario selecciona un plano, agrega un nombre, una descripción y un estado • El usuario guarda el registro. • El nuevo plano es guardado cuándo el mensaje de guardado sea exitoso.	

Historias de usuario	
Numero: 7	**Nombre:** Elaboración de reportes
Usuario: Administrador	
Prioridad en el negocio(alta, media, baja): alta	**Riesgo en el desarrollo(alta, media, baja): alta**
Responsable: Leonardo, Wilson	
Descripción: El aplicativo permitirá generar tres tipos de reportes. La información prestada en los reportes con sus características pertenece a: • registros de espacios Nombre de espacio, descripción, referencia de la ubicación en el plano, tipo de espacio. • registros de ocupantes Nombre de edificio, nombre de piso, espacio físico, tipo de espacio, estado del espacio, nombre del ocupante, horario, días de ocupación. • registros de eventos Nombre del evento, responsable del evento, ubicación del evento, estado del evento, fecha y hora de inicio, fecha y hora de fin. El usuario podrá seleccionar que podrá elegir el usuario. Estos reportes podrán ser impresos o exportados a archivos de Excel.	

¿Cómo comprobarlo? • El usuario selecciona un reporte de un conjunto de reportes. • El usuario ingresa un rango de fechas para generar el reporte. • El usuario verifica la información generada en el reporte.

Historias de usuario	
Numero: 8	**Nombre:** Búsqueda de espacios en general
Usuario: Administrador	
Prioridad en el negocio(alta, media, baja): media	**Riesgo en el desarrollo(alta, media, baja): media**
Responsable: Leonardo Wilson	
Descripción: El aplicativo permite realizar búsquedas de los diferentes espacios físicos con los que cuanta el campus politécnico a través de los siguientes parámetros de búsqueda: • Tipo de espacio. • Ubicación (piso y edificio). • Costo diario. • Capacidad.	
¿Cómo comprobarlo? • El usuario escoge un criterio. • El usuario procede a realizar la búsqueda. • El usuario verifica el resultado de la búsqueda en una tabla que muestra algunas características del espacio buscado.	
Historias de usuario	
Numero: 9	**Nombre:** Ceder espacio para actividades académicas
Usuario: Administrador	
Prioridad en el negocio(alta, media, baja): media	**Riesgo en el desarrollo(alta, media, baja): media**
Responsable: Leonardo Wilson	
Descripción: El aplicativo permite seleccionar espacios en diferentes lugares dentro de campus politécnico para actividades académicas.	
¿Cómo comprobarlo? • El usuario selecciona un edificio • El usuario selección un piso	

Historias de usuario	
Numero: 10	**Nombre:** Asignación de Eventos
Usuario: Administrador	
Prioridad en el negocio(alta, media, baja): alta	**Riesgo en el desarrollo(alta, media, baja): alta**
Responsable: Leonardo Wilson	
Descripción: El aplicativo permite crear un evento y asignar el mismo a un espacio del campus politécnico dicho evento tendrá una fecha de inicio, una fecha de fin y una fecha de reserva. El evento puede tener un horario fijo durante las fechas de inicio y de fin o puede tener diferentes horarios durante el lapso de tiempo establecido sobre la fechas de inicio y fin.	
¿Cómo comprobarlo? • El usuario selecciona un espacio para realizar el evento. • El usuario verifica la disponibilidad del espacio seleccionado mediante la fecha y hora en la cual se va a realizar la reserva. • Si existe disponibilidad el usuario reserva el espacio. • Con el espacio reservado, el usuario ingresa los siguientes datos del evento: o Nombre del evento. o Descripción de uso del evento. o Tipo de evento (INTERNO, EXTERNO). o Estado del evento (Reservado, Confirmado). o Responsable del evento. • El usuario establece un horario fijo, o puede detallar varios horarios de uso para los diferentes días entre las fechas de inicio y fin ingresadas. • El usurario guarda el registro. • El usuario se asegura que el registro fue exitoso al presentarse un mensaje de guardado éxito.	
• El usuario selecciona un espacio disponible • El usuario registra el uso que se le dará a dicho espacio • El usuario guarda el registro. • Se comprueba el registro tras un mensaje de guardado exitoso.	

Historias de usuario

Numero: 11	**Nombre:** Búsqueda de eventos
Usuario: Administrador	
Prioridad en el negocio(alta, media, baja): media	**Riesgo en el desarrollo(alta, media, baja): media**
Responsable: Leonardo Wilson	
Descripción: El aplicativo permite realizar consultas sobre los eventos registrados en el sistema de acuerdo a diferentes criterios, por ejemplo, buscar evento por día, semana, mes, buscar eventos pasados y futuros, buscar eventos por responsable, buscar eventos por nombre, buscar eventos por tipo, buscar eventos por costo de alquiler.	
¿Cómo comprobarlo? • El usuario escoge uno o varios de los siguientes criterios de búsqueda: Nombre del evento. Responsable del evento. Tipo del evento. Estado del evento. Fecha inicio del evento. Fecha fin del evento. • El usuario procede a realizar la búsqueda. • El usuario verifica el resultado de la búsqueda en una tabla que muestra las siguientes características del evento buscado: Nombre del evento, responsable, ubicación, nombre del piso, tipo de evento, estado de evento, fecha de inicio y fin.	

Historias de usuario	
Numero: 12	**Nombre:** Agenda de eventos
Usuario: Administrador	
Prioridad en el negocio(alta, media, baja): media	**Riesgo en el desarrollo(alta, media, baja): media**
Responsable: Leonardo Wilson	
Descripción: El aplicativo permite llevar una agenda de los eventos a realizarse dentro del campus politécnico, esta agenda mostrará información como fecha de inicio, fecha de fin fecha de reserva, nombre del responsable, tipo y estado de eventos.	
¿Cómo comprobarlo? • El usuario puede verificar los distintos eventos a realizarse en un cronograma,	

por medio de un calendario de eventos, observando en cada día del evento que se lleva a cabo.

Historias de usuario	
Numero: 13	**Nombre:** Asignación de Horarios
Usuario: Administrador	
Prioridad en el negocio(alta, media, baja): alta	**Riesgo en el desarrollo(alta, media, baja): alta**
Responsable: Leonardo Wilson	
Descripción: El aplicativo permite registrar diferentes horarios de uso en diferentes días para los espacios que conforman el campus politécnico.	
¿Cómo comprobarlo? • El usuario puede seleccionar una hora de inicio y una hora de fin para el uso de un determinado espacio.	

Historias de usuario	
Numero: 14	**Nombre:** Responsable De Eventos
Usuario: Administrador	
Prioridad en el negocio(alta, media, baja): media	**Riesgo en el desarrollo(alta, media, baja): media**
Responsable: Leonardo Wilson	
Descripción: El aplicativo permite registrar a los responsables de eventos a desarrollarse en el campus politécnico.	
¿Cómo comprobarlo? • El usuario registra información del responsable como nombre, apellido, email, cédula de identificación definida por 10 dígitos y teléfono sea celular de 10 dígitos o convencional de 7. • El usuario guarda el registro. • El usuario comprueba su registro tras un mensaje de guardado exitoso.	

Historias de usuario	
Numero: 15	**Nombre:** Seguimiento De Cobranzas Y Arrendamiento
Usuario: Administrador	
Prioridad en el negocio(alta, media, baja): alta	**Riesgo en el desarrollo(alta, media, baja): alta**
Responsable: Leonardo Wilson	
Descripción: El aplicativo permite realizar búsquedas para verificar saldos por alquiler de eventos, verificar si el evento es gratuito o cobrado, además los espacios que han sido alquilados.	
¿Cómo comprobarlo? • El usuario ingresara el evento a consultar. • Si el evento no se muestra como un resultado de consulta, el evento es gratuito. • Si se muestra con detalles, el evento es con cobros y es un espacio con alquiler.	

Historias de usuario	
Numero: 16	**Nombre:** Especificar Tarifas Para Arrendar Espacios
Usuario: Administrador	
Prioridad en el negocio(alta, media, baja): alta	**Riesgo en el desarrollo(alta, media, baja): alta**
Responsable: Leonardo	

Wilson
Descripción: El aplicativo permite establecer la tarifa de un alquiler de acuerdo al tipo de espacio y capacidad o de acuerdo al tipo de arrendatario.
¿Cómo comprobarlo? • El usuario ingresa a la página de Tipo de Espacios para modificar un espacio con el tipo especificado. • El usuario puede modificar el costo del espacio dependiendo del tipo al que pertenece. • Otra manera de establecer la tarifa es estableciendo una tarifa específica para un espacio al momento de crearlo si es el caso de no establecer uno predefinido en el tipo de espacio.

Historias de usuario	
Numero: 17	**Nombre:** Autorización De Cobros
Usuario: Administrador	
Prioridad en el negocio(alta, media, baja): media	**Riesgo en el desarrollo(alta, media, baja): media**
Responsable: Leonardo, Wilson	
Descripción: El aplicativo permite cambiar el estado de un evento, gratuito o cobrado.	
¿Cómo comprobarlo? • El usuario determinara si el evento está establecido con cobros o no cuando se determina el tipo de evento interno o externo o dependiendo de normativas u ordenanzas externas. Sin embargo un evento interno también puede ser alquilado dependiendo de la disposición de autoridades.	

Historias de usuario	
Numero: 18	**Nombre:** Ubicación De Los Espacios En Planos
Usuario: Administrador	
Prioridad en el negocio(alta, media, baja): alta	**Riesgo en el desarrollo(alta, media, baja): alta**
Responsable: Leonardo Wilson	
Descripción: El aplicativo permite realizar la selección de los espacios en planos arquitectónicos	

ingresados por el usuario a través de coordenadas.
¿Cómo comprobarlo? • El usuario selecciona un espacio en el plano • Si el espacio no está seleccionado en el plano se guardan sus coordenadas caso contrario aparece un mensaje de que no puede seleccionar dicho espacio.

Historias de usuario	
Numero: 19	**Nombre:** Notificación de reserva de eventos vía mail.
Usuario: Administrador	
Prioridad en el negocio(alta, media, baja): media	**Riesgo en el desarrollo(alta, media, baja): alta**
Responsable: Leonardo Wilson	
Descripción: El aplicativo permite realizar el envió de correos electrónicos para informar a los usuarios de la reserva, cancelación o confirmación de un evento. *Nota: los eventos que no se confirmen en 72 horas serán cancelados automáticamente por el aplicativo.*	
¿Cómo comprobarlo? • El usuario recibe un correo electrónico cuando reserva un evento. • El usuario recibe un correo electrónico cuando cancela un evento. • El usuario recibe un correo electrónico cuando confirma un evento. *Nota: el correo recibido será de la dirección infraestructura.fisica@epn.edu.ec.*	

2.1.2 Pila del Producto Inicial (Product Backlog)

En base a las historias de usuario planteadas anteriormente se procede a realizar la pila del producto que es el conjunto de todas las características que darán funcionalidad a nuestro sistema. Tabla 2.1.

No	Tareas	Complejidad	Prioridad
1	Diseño de Diagrama Entidad-Relación	5	3
2	Generación modelo Físico	3	3
3	Generación código SQL para PostgreSQL	3	2

4	Definición Roles y Perfiles de Usuario	2	3
5	Refinamiento y consistencia de la Base de Datos	5	2
6	Control de calidad	1	4
7	Afinamiento de la base de datos	2	4
8	Creación de entidades desde base de datos.	3	2
9	Creación e implementación de interfaz edificio (CRUD).	3	3
10	Creación backingbean edificio.	3	3
11	Creación página JSF edificio.	3	4
12	Creación e implementación de interfaz plano (CRUD).	3	3
13	Creación backingbean plano.	3	3
14	Creación página JSF plano.	3	4
15	Creación e implementación de interfaz piso (CRUD).	3	3
16	Creación backingbean piso.	3	3
17	Creación página JSF piso.	3	4
18	Creación e implementación de interfaz lugar_espacio(CRUD).	3	3
19	Creación backingbean lugar_espacio.	3	3
20	Creación página JSF lugar_espacio.	3	4
21	Creación e implementación de interfaz espacio (CRUD).	3	3
22	Creación backingbean espacio.	3	3
23	Creación página JSF espacio.	3	4
24	Creación e implementación de interfaz tipo espacio (CRUD).	3	3
No	**Tareas**	**Complejidad**	**Prioridad**
25	Creación backingbean tipo espacio.	3	3
26	Creación página JSF tipo espacio.	3	4
27	Acoplamiento de planos en páginas JSF	4	3
28	Marcado de espacios de plano en JSF	4	3
29	Casos de Prueba	2	3
30	Control de calidad	1	4
31	Afinamiento	2	4
32	Creación e implementación de interfaz horario (CRUD).	3	3
33	Creación backingbean horario.	3	3
34	Creación página JSF horario.	3	4
35	Creación e implementación de interfaz horario_espacio (CRUD).	3	3
36	Creación backingbean horario_espacio.	3	3
37	Creación página JSF horario_espacio.	3	4
38	Creación e implementación de interfaz estados de espacio (CRUD).	3	3
39	Creación backingbean estados de espacio.	3	3
40	Creación página JSF estados de espacio.	3	4
41	Creación e implementación de interfaz ocupante (CRUD).	3	3

42	Creación backingbean ocupante.	3	3
43	Creación página JSF ocupante.	3	4
44	Creación e implementación de interfaz evento (CRUD).	3	3
45	Creación backingbean evento.	3	3
46	Creación página JSF evento.	3	4
47	Creación e implementación de interfaz estado de evento (CRUD).	3	3
48	Creación backingbean estado de evento.	3	3
49	Creación página JSF estado de evento.	3	4
50	Creación e implementación de interfaz tipo de evento (CRUD).	3	3
51	Creación backingbean tipo de evento.	3	3
52	Creación página JSF tipo de evento.	3	4
53	Casos de Prueba	2	3
54	Control de calidad	1	4
55	Afinamiento	2	4
56	Creación e implementación de interfaz responsable de evento (CRUD).	3	3
57	Creación backingbean responsable de evento.	3	3
58	Creación página JSF responsable de evento.	3	4
59	Creación e implementación de interfaz alquiler (CRUD).	3	3
60	Creación backingbean alquiler.	3	3
No	**Tareas**	**Complejidad**	**Prioridad**
61	Creación página JSF alquiler.	3	4
62	Creación e implementación de interfaz estado alquiler (CRUD).	3	3
63	Creación backingbean estado alquiler.	3	3
64	Creación página JSF estado alquiler.	3	4
65	Creación e implementación de interfaz pagos alquiler (CRUD).	3	3
66	Creación backingbean pagos alquiler.	3	3
67	Creación página JSF pagos alquiler.	3	4
68	Reportes de planos	3	3
69	Reportes de horarios	3	3
70	Reportes de eventos	3	3
71	Reportes de espacios	3	3
72	Reportes de ocupantes	3	3
73	Reportes de responsables de evento	3	3
74	Reportes de alquiler de eventos	3	3
75	Casos de Prueba	2	3
76	Control de calidad	1	4
77	Afinamiento	2	4

78	Recopilación de Datos de Inicio	3	3
79	Instalación del Sistema	3	5
80	Carga de Datos y Clientilización	3	5
81	Análisis de los Resultados	3	5
82	Prueba Final del Sistema	3	5
83	Control de calidad	1	4
84	Afinamiento	2	4

Tabla 2. 1: Product Backlog. Elaborado por los Autores

2.2 PLANIFICACIÓN DE LOS SPRINTS (SPRINTS BACKLOG)

En la Figura 2.1 se puede apreciar las fechas establecidas y el número de Sprints que se pretende realizar para el desarrollo del sistema mediante jornadas laborares de cuatro horas.

Proyecto			
Desarrollo de un Sistema Web para la Administración de la Infraestructura Física de la Escuela Politécnica Nacional			
Nº de sprint	Inicio	Días	Jornada
1	17-mar.-14	7	4
2	26-mar.-14	20	4
3	24-abr.-14	20	4
4	23-may.-14	18	4
5	18-jun.-14	20	4
TAREAS		EQUIPO	FESTIVOS
TIPOS	ESTADOS		
Análisis	Pendiente	Leonardo	3-mar.
Codificación	En Curso	Wilson	4-mar.
Prototipado	Terminada	Leonardo, Wilson	18-abr.
Pruebas	Eliminada	Sandra	19-abr.
Reunión		Geovanna	20-abr.
Documentación		Andrea	1-may.
			24-may.

Fig. 1.17: Planificación de los Sprints

Fuente: Elaborado por los Autores. Adaptado de www.navegapolis.net

2.2.1 PRIMER SPRINT

Para la ejecución del primer Sprint se ha tomado desde la tarea 1 a la tarea 7 del Product BackLog, estas tareas tienen que ver con la creación de la base de datos y la definición de perfiles y roles dentro del sistema.

El primer Sprint inicia el 17 de marzo del 2014 y finaliza el 25 de marzo del 2014, teniendo una duración de 7 días.

La Tabla 2.2 presenta las tareas seleccionadas para este Sprint así como el tiempo estimado para cada tarea, este Sprint tiene una duración estimada de 56 horas.

No	Tarea	Tiempo estimado

1	Diseño de Diagrama Entidad-Relación	24
2	Generación modelo Físico	4
3	Generación código SQL para PostgreSQL	4
4	Definición Roles y Perfiles de Usuario	8
5	Refinamiento y consistencia de la Base de Datos	8
6	Control de calidad	4
7	Afinamiento de la Base de Datos	4
	TOTAL	56

Tabla 2. 2: Selección de Tareas del Primer Sprint. Elaborado por los Autores

En la Figura. 2.2 podemos apreciar el listado de las tareas seleccionadas, su distribución en el tiempo asignado, sus estados y los responsables de las mismas.

		SPRINT	INICIO	DURACIÓN			L	M	X	J	V	L	M
		1	17-mar.-14	7									
							17-mar.	18-mar.	19-mar.	20-mar.	21-mar.	24-mar.	25-mar.
				Tareas pendientes			1	1	1	2	1	1	2
				Horas de trabajo pendientes			8	8	8	8	8	8	8
		PILA DEL SPRINT					ESFUERZO						
Backlog	Tarea			Tipo	Estado	Responsable							
1	Diseño de Diagrama Entidad-Relacion			Análisis	Terminada	Leonardo, Wilson	8	8	8				
2	Generacion modelo Fisico			Codificación	Terminada	Leonardo, Wilson				4			
3	Generacion codigo SQL para PostgreSQL			Prototipado	Terminada	Leonardo, Wilson				4			
4	Definición Roles y Perfiles de Usuario			Codificación	Terminada	Leonardo, Wilson					8		
5	Refinamiento y consistencia de la Base de Datos			Análisis	Terminada	Leonardo, Wilson						8	
6	Control de calidad			Codificación	Terminada	Geovanna							4
7	Afinamiento de la base de datos			Codificación	Terminada	Leonardo, Wilson							4

Fig. 1.18: Tareas del Primer Sprint. Elaborado por los Autores.

En la Figura 2.3 se muestra el esfuerzo y avance de las tareas para la ejecución del Primer Sprint, definiendo al esfuerzo como el trabajo colectivo o individual de los recursos (personas) para la finalización de cada tarea.

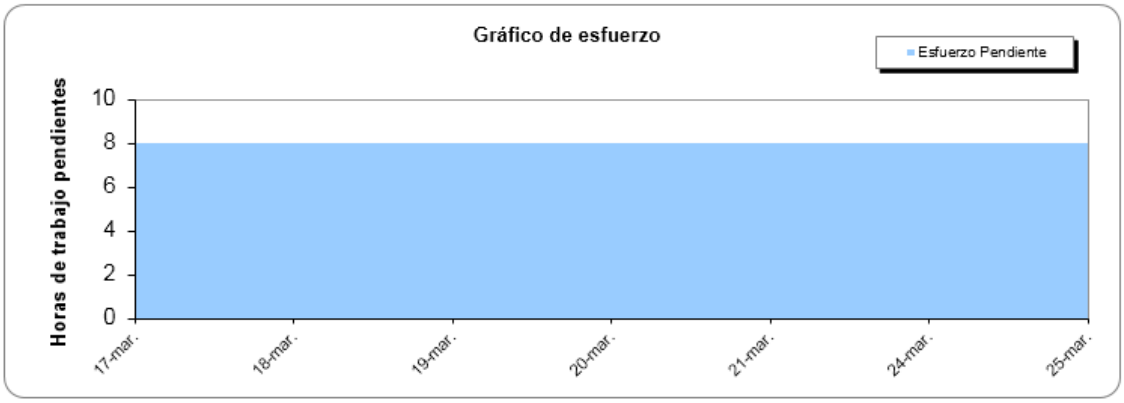


Fig. 1.19: Esfuerzo realizado en el Primer Sprint. Elaborado por los Autores.

Fuente: www.navegapolis.net

En la Figura 2.4 se muestra el avance de las tareas en cada intervalo de tiempo, donde se puede apreciar una fecha inicial y final las diferentes tareas del Primer Sprint a ser realizadas por el equipo de trabajo.

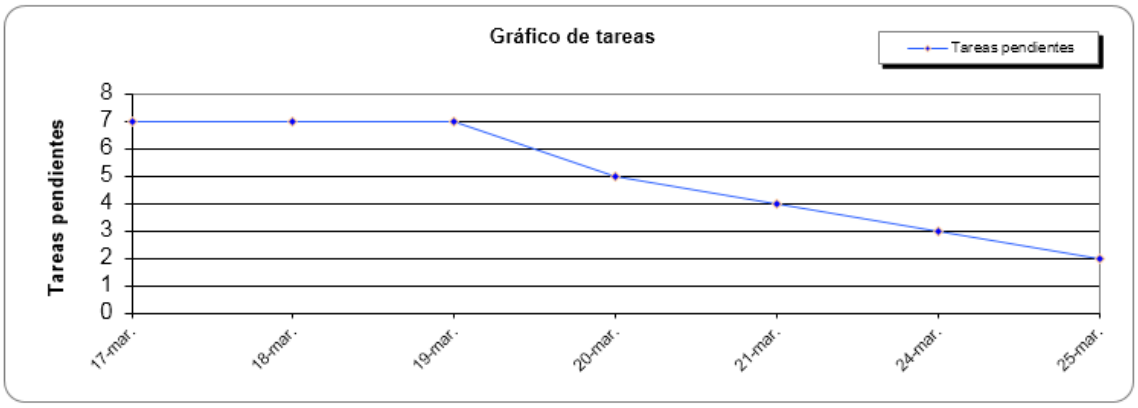


Fig. 1.20: Avance de tareas en el Primer Sprint. Elaborado por los Autores.

Fuente: www.navegapolis.net

2.2.2 SEGUNDO SPRINT

Para el Segundo Sprint se ha seleccionado las tareas 8 a la 31, que tiene como fecha de inicio el día 26 de marzo y finaliza el 23 de abril del 2014, teniendo una duración de 20 días.

En este Sprint se pretende realizar la creación de clases, interfaces, servicios y páginas JSF para el desarrollo del sistema. En la Tabla 2.3 se puede apreciar

cada una de las tareas seleccionadas que tienen una duración estimada de 160 horas.

No	Tarea	Tiempo estimado
1	Creación de entidades desde base de datos.	8
2	Creación e implementación de interfaz edificio (CRUD).	4
3	Creación backingbean edificio.	8
4	Creación página JSF edificio.	8
5	Creación e implementación de interfaz plano (CRUD).	4
6	Creación backingbean plano.	8
7	Creación página JSF plano.	8
8	Creación e implementación de interfaz piso (CRUD).	4
9	Creación backingbean piso.	8
10	Creación página JSF piso.	8
11	Creación e implementación de interfaz lugar_espacio (CRUD).	4
12	Creación backingbean lugar_espacio.	8
13	Creación página JSF lugar_espacio.	8
14	Creación e implementación de interfaz espacio (CRUD).	4
15	Creación backingbean espacio.	8
16	Creación página JSF espacio.	8
17	Creación e implementación de interfaz tipo espacio (CRUD).	4
18	Creación backingbean tipo espacio.	8
19	Creación página JSF tipo espacio.	8
20	Acoplamiento de planos en páginas JSF	4
21	Marcado de espacios de plano en JSF	4
22	Casos de Prueba	8
23	Control de calidad	8
24	Afinamiento	8
	TOTAL	160

Tabla 2. 3: Selección de Tareas del Segundo Sprint. Elaborado por los Autores

En la Figura 2.5 se muestran las tareas del Segundo Sprint, con su duración, estados y responsables.

Fig. 1.21 Tareas del Segundo Sprint. Elaborado por los Autores

En la Figura 2.6 se muestra el esfuerzo y avance de las tareas para la ejecución del Segundo Sprint, definiendo al esfuerzo como el trabajo colectivo o individual de los recursos (personas) para la finalización de cada tarea.

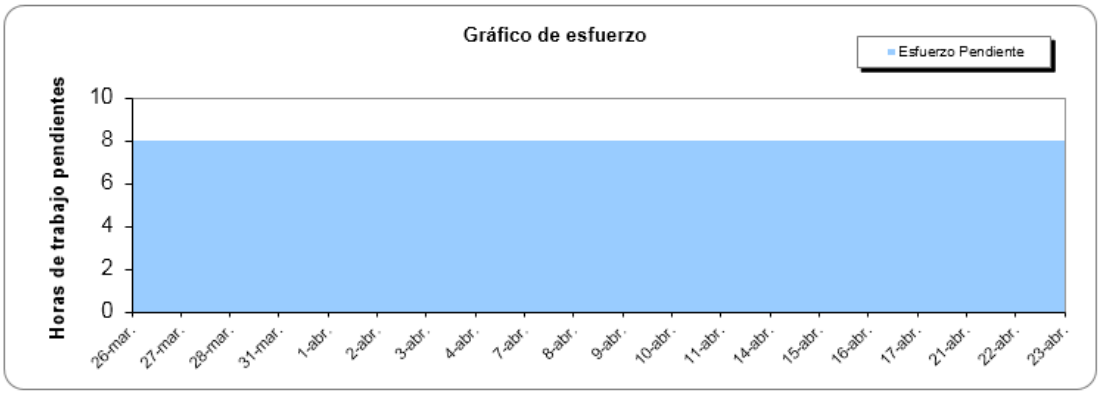


Fig. 1.22: Esfuerzo realizado en el Segundo Sprint. Elaborado por los autores.

Fuente: www.navegapolis.net

En la Figura 2.7 se muestra el avance de las tareas en cada intervalo de tiempo, donde se puede apreciar una fecha inicial y final las diferentes tareas del Segundo Sprint a ser realizadas por el equipo de trabajo.

Fig. 1.23: Avance de tareas en el Segundo Sprint. Elaborado por los autores.

Fuente: www.navegapolis.net

2.2.3 TERCER SPRINT

El tercer Sprint comprende las tareas 32 a la 55 del Product Backlog, que continúa con la creación de clases, interfaces, servicios y páginas JSP, que comprende la administración de horarios, asignación de ocupantes y el módulo para administrar eventos.

Este Sprint empieza el 24 de abril y termina el 22 de mayo del 2014, y tiene una duración de 20 días de 160 horas estimadas, en la Tabla 2.4 se muestra las tareas seleccionadas para este Sprint.

No	Tarea	Tiempo estimado
1	Creación e implementación de interfaz horario (CRUD).	4
2	Creación backingbean horario.	8
3	Creación página JSF horario.	8
4	Creación e implementación de interfaz horario_espacio (CRUD).	4
5	Creación backingbean horario_espacio.	8

6	Creación página JSF horario_espacio.	8
7	Creación e implementación de interfaz estados de espacio (CRUD).	4
8	Creación backingbean estados de espacio.	8
9	Creación página JSF estados de espacio.	8
10	Creación e implementación de interfaz ocupante (CRUD).	4
11	Creación backingbean ocupante.	8
12	Creación página JSF ocupante.	8
13	Creación e implementación de interfaz evento (CRUD).	3
14	Creación backingbean evento.	8
15	Creación página JSF evento.	8
16	Creación e implementación de interfaz estado de evento (CRUD).	3
17	Creación backingbean estado de evento.	8
18	Creación página JSF estado de evento.	8
19	Creación e implementación de interfaz tipo de evento (CRUD).	2
20	Creación backingbean tipo de evento.	8
21	Creación página JSF tipo de evento.	8
22	Casos de Prueba	8
23	Control de calidad	8
24	Afinamiento	8
	TOTAL	160

Tabla 2. 4: Selección de tareas del Tercer Sprint. Elaborado por los Autores.

En la Figura 2.8 se puede apreciar las tareas del Tercer Sprint, su tiempo de duración, estado y responsables.

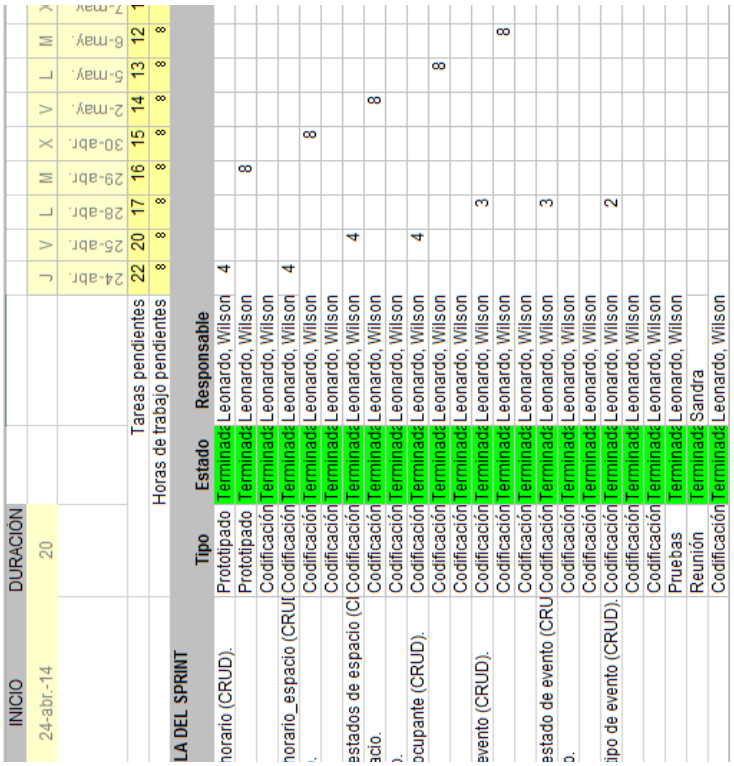


Fig. 1.24: Tareas del Tercer Sprint. Elaborado por los Autores.

En la Figura 2.9 se muestra el esfuerzo y avance de las tareas para la ejecución del Tercer Sprint, definiendo al esfuerzo como el trabajo colectivo o individual de los recursos (personas) para la finalización de cada tarea.

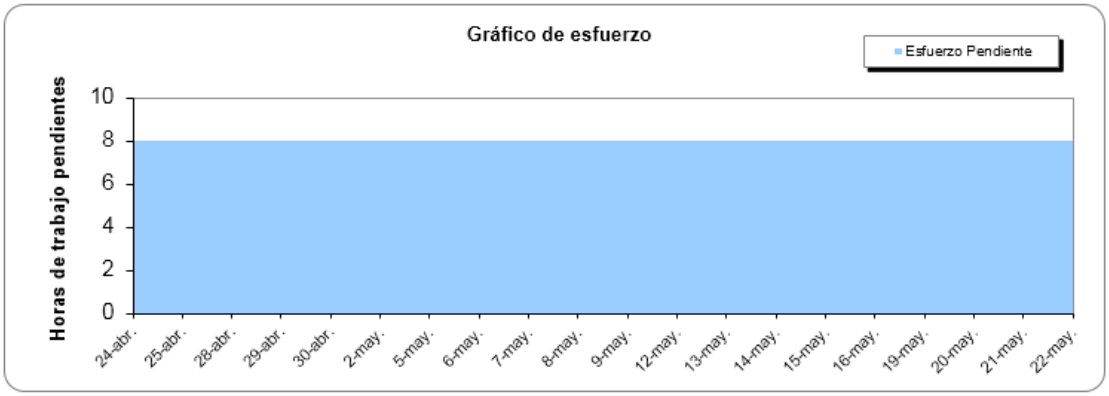


Fig. 1.25: Esfuerzo realizado en el Tercer Sprint. Elaborado por los Autores.

Fuente: www.navegapolis.net

En la Figura 2.10 se muestra el avance de las tareas en cada intervalo de tiempo, donde se puede apreciar una fecha inicial y final las diferentes tareas del Tercer Sprint a ser realizadas por el equipo de trabajo.

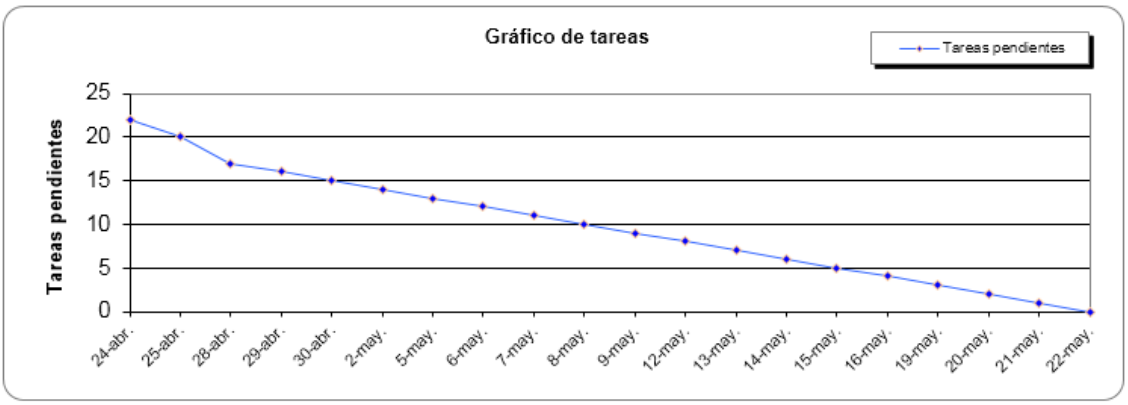


Fig. 1.26: Avance de tareas en el Tercer Sprint. Elaborado por los Autores.

Fuente: www.navegapolis.net

2.2.4 CUARTO SPRINT

El cuarto Sprint corresponde a las tareas 56 a la 77 del Product Backlog, este Sprint inicia el 23 de mayo y finaliza el 17 de junio del 2014, este Sprint tiene una duración de 18 días. En este Sprint se continúa con la creación de clases, interfaces, servicios y páginas JSF restantes para completar el sistema, además de la generación de reportes.

En la Tabla 2.5 se puede apreciar las tareas seleccionadas para este Sprint.

No	Tarea	Tiempo estimado
1	Creación e implementación de interfaz responsable de evento (CRUD).	4
2	Creación backingbean responsable de evento.	8
3	Creación página JSF responsable de evento.	8
4	Creación e implementación de interfaz alquiler (CRUD).	4
5	Creación backingbean alquiler.	8
6	Creación página JSF alquiler.	8
7	Creación e implementación de interfaz estado alquiler (CRUD).	4
8	Creación backingbean estado alquiler.	8
9	Creación página JSF estado alquiler.	8
10	Creación e implementación de interfaz pagos alquiler (CRUD).	4
11	Creación backingbean pagos alquiler.	8
12	Creación página JSF pagos alquiler.	8
13	Reportes de planos	8
14	Reportes de horarios	8
15	Reportes de eventos	8
16	Reportes de espacios	4
17	Reportes de ocupantes	4
18	Reportes de responsables de evento	4
19	Reportes de alquiler de eventos	4
20	Casos de Prueba	8
21	Control de calidad	8
22	Afinamiento	8
	TOTAL	144

Tabla 2. 5: Selección de Tareas del Cuarto Sprint. Elaborado por los Autores

En la Figura 2.11 se puede apreciar las tares del Cuarto Sprint con su duración, estado y responsables.

Fig. 1.27 Tareas del Cuarto Sprint. Elaborado por los Autores.

En la Figura 2.12 se muestra el esfuerzo y avance de las tareas para la ejecución del Cuarto Sprint, definiendo al esfuerzo como el trabajo colectivo o individual de los recursos (personas) para la finalización de cada tarea.

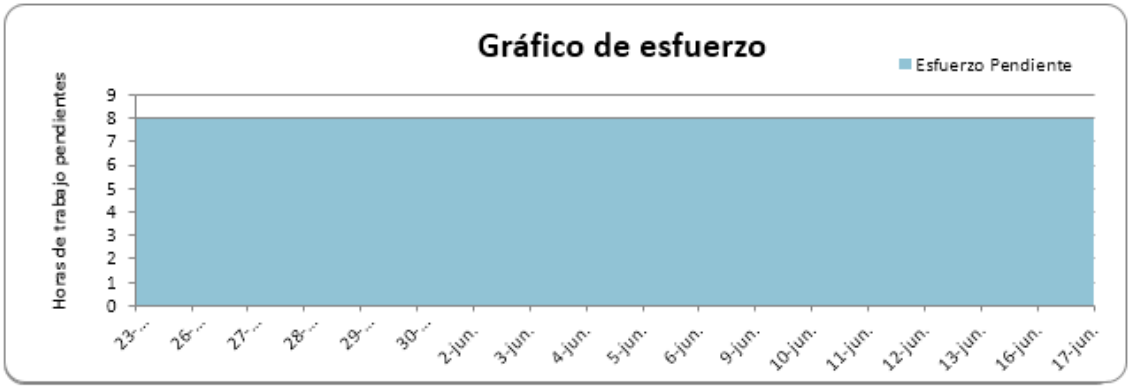


Fig. 1.28: Esfuerzo realizado en el Cuarto Sprint. Elaborado por los Autores.

Fuente: www.navegapolis.net

En la Figura 2.13 se muestra el avance de las tareas en cada intervalo de tiempo, donde se puede apreciar una fecha inicial y final las diferentes tareas del Cuarto Sprint a ser realizadas por el equipo de trabajo.

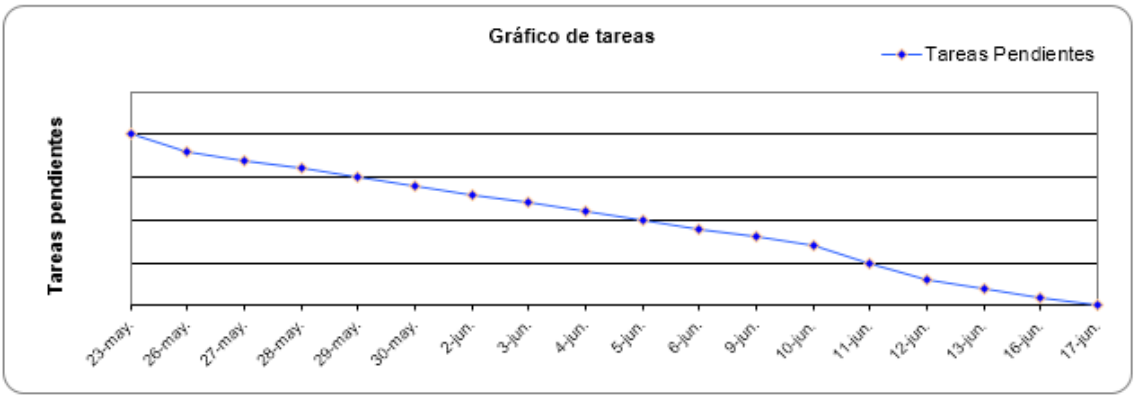


Fig. 1.29: Avance de tareas en el Cuarto Sprint. Elaborado por los Autores.

Fuente: www.navegapolis.net

2.2.5 QUINTO SPRINT

El quinto Sprint corresponde a las tareas 78 a la 84 que inicia el 18 de junio y termina el 15 de julio del 2014.

El objetivo de este Sprint es poner en funcionamiento el sistema, cargando datos, realizando pruebas de su funcionalidad global, debido a que el sistema debe ejecutarse en el ambiente de pruebas donde funcionan los sistemas pertenecientes a la Escuela Politécnica Nacional a cargo de la Dirección de Gestión de Información y Procesos.

Posteriormente se analizarán los resultados obtenidos.

En las Tabla 2.6 se muestran las tareas finales para la culminación de proyecto.

No	Tarea	Tiempo estimado
1	Recopilación de Datos de Inicio	24
2	Instalación del Sistema	16
3	Carga de Datos y Clientelización	40
4	Análisis de los Resultados	32
5	Prueba Final del Sistema	15
6	Control de calidad	14
7	Afinamiento	19
	TOTAL	160

Tabla 2. 6: Selección de Tareas del Quinto Sprint. Elaborado por los Autores.

En la Figura 2.14 se presenta las tareas del Quinto Sprint con su duración, estados y responsables.

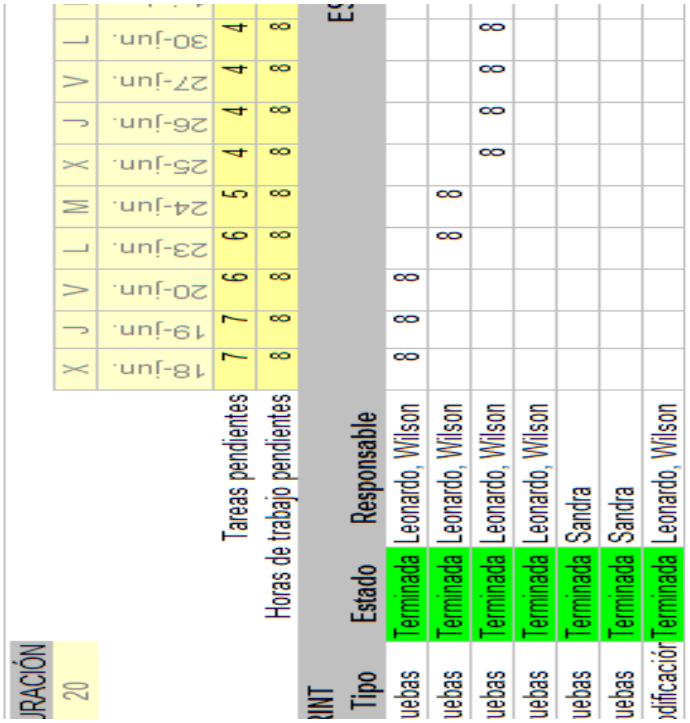


Fig. 1.30: Tareas del Quinto Sprint. Elaborado por los Autores.

En la Figura 2.15 se muestra el esfuerzo y avance de las tareas para la ejecución del Quinto Sprint, definiendo al esfuerzo como el trabajo colectivo o individual de los recursos (personas) para la finalización de cada tarea.

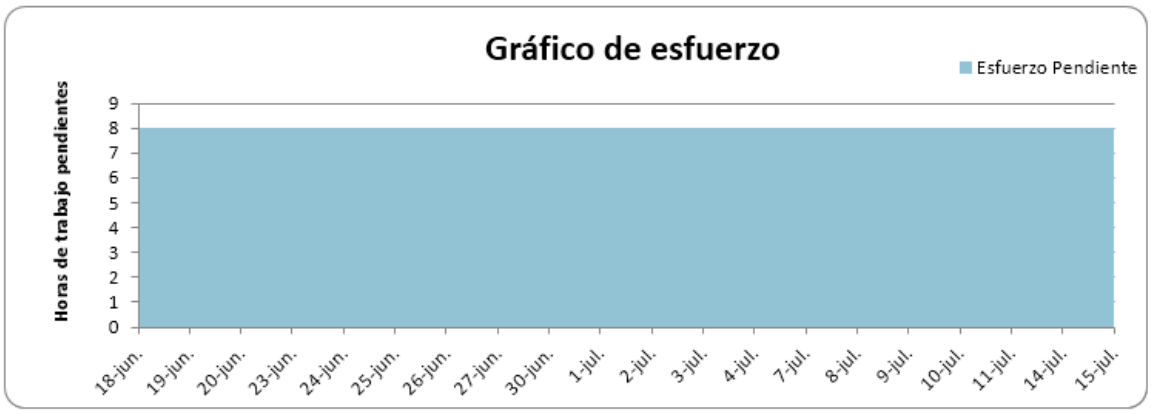


Fig. 1.31: Esfuerzo realizado en el Quinto Sprint. Elaborado por los Autores.

Fuente: www.navegapolis.net

En la Figura 2.16 se muestra el avance de las tareas en cada intervalo de tiempo, donde se puede apreciar una fecha inicial y final las diferentes tareas del Quinto Sprint a ser realizadas por el equipo de trabajo.

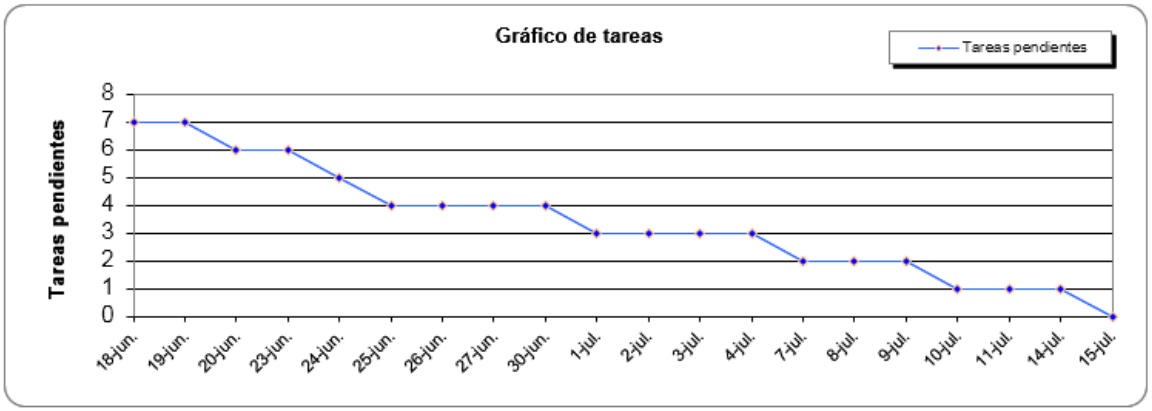


Fig. 1.32: Avance de tareas en el Quinto Sprint. Elaborado por los Autores.

Fuente: www.navegapolis.net

2.3 EJECUCIÓN DE LOS SPRINTS

2.3.1 ARQUITECTURA DE LA SOLUCIÓN

La implementación de los Sprints se basa en la creación del sistema, visualizando las interfaces web, la codificación, la base de datos y demás entregables de cada sprint.

Para el desarrollo del Sistema Web se ha utilizado la arquitectura MVC (Modelo – Vista - Controlador), la cual tiene un alto rendimiento con las herramientas y metodologías antes mencionadas.

En la Figura 2.17 se muestra claramente la arquitectura MVC.

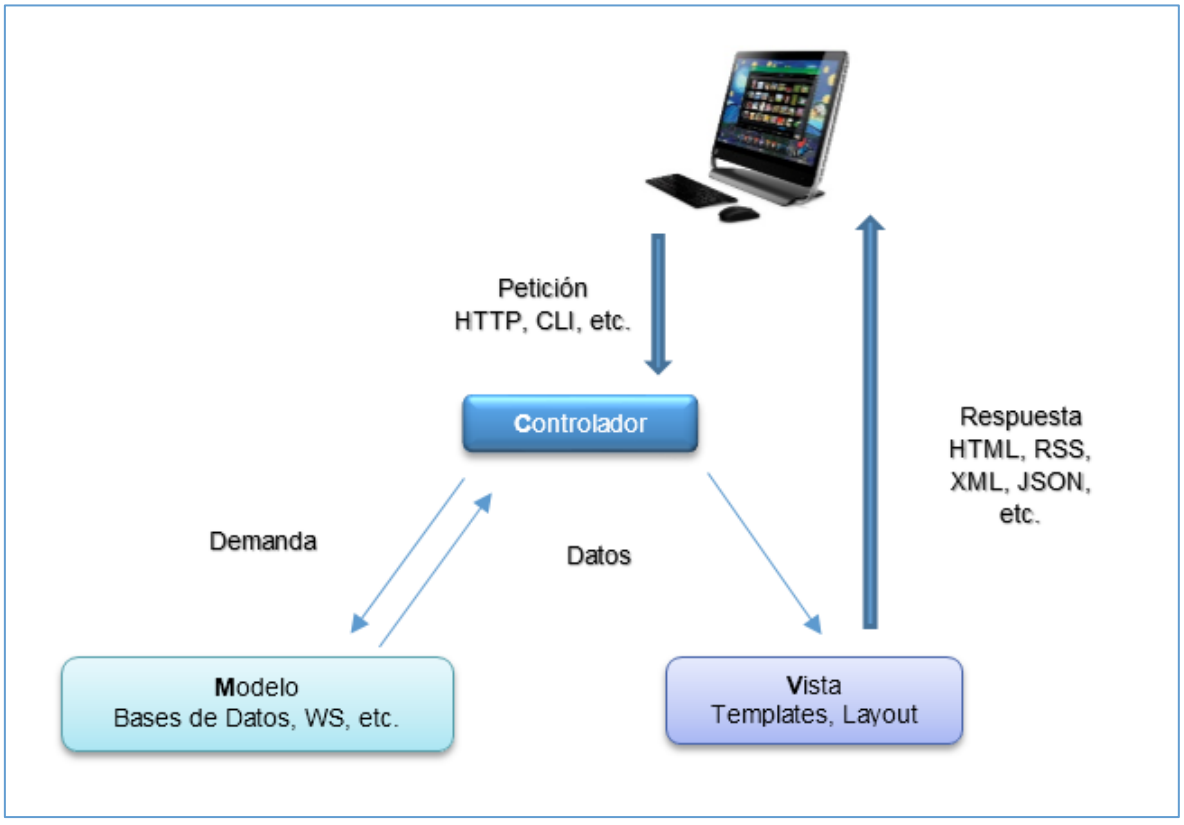


Fig. 1.33: Arquitectura de la Solución. Elaborado por los Autores.

2.3.1. EJECUCIÓN DEL PRIMER SPRINT

2.3.2.1. Diseño del modelo físico y lógico de la base de datos

En este Sprint se procede a realizar el diseño del modelo entidad relación en base a los requerimientos del sistema, el mismo es presentado en la Fig. 2.18.

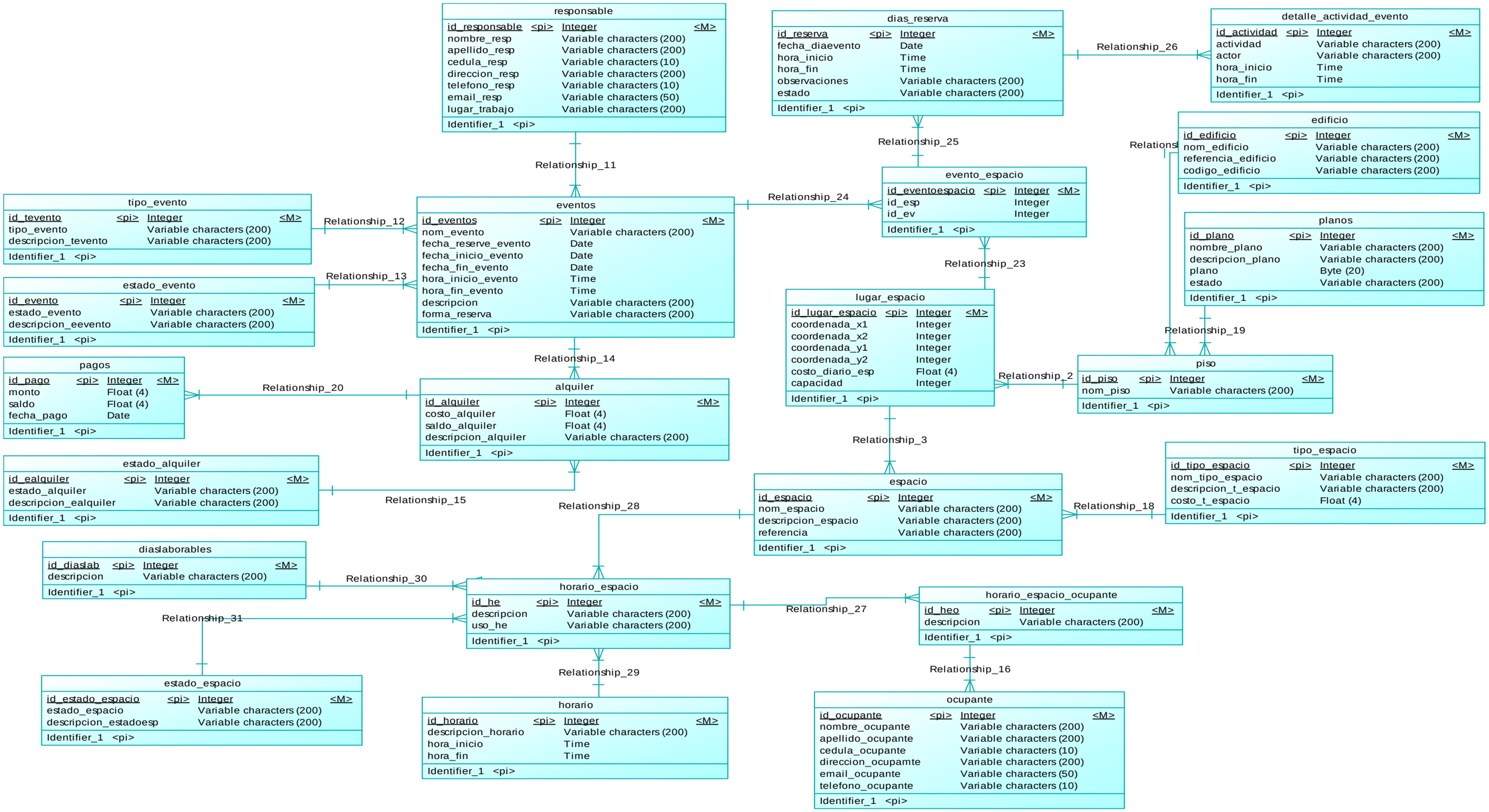


Fig. 1.34: Diagrama Entidad-Relación de la Base de Datos. Elaborado por los Autores.

Una vez realizado el modelo entidad relación se procede a generar el modelo físico que se muestra en la Fig. 2.19.

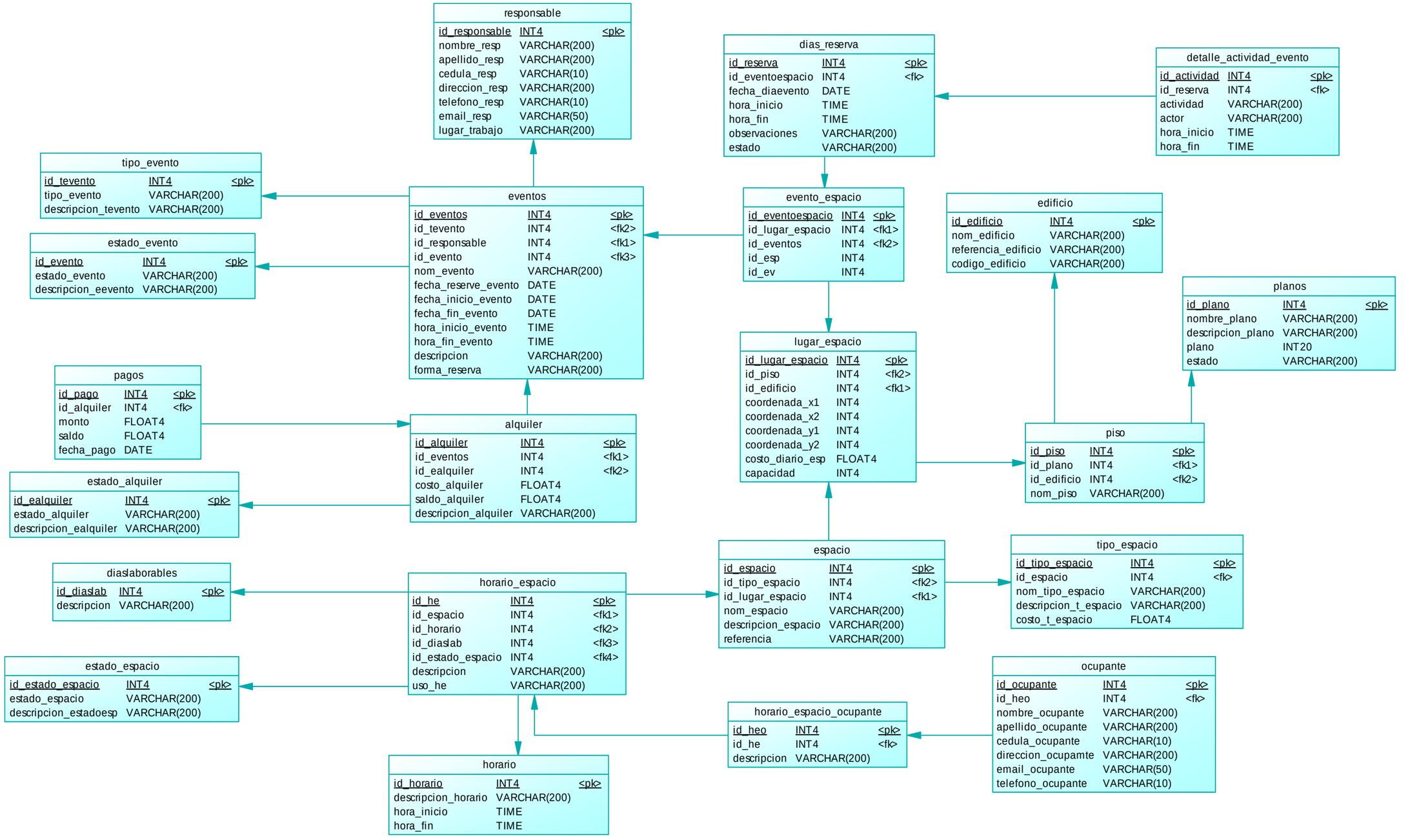


Fig. 1.35: Diagrama del Modelo Físico de la Base de Datos. Elaborado por los Autores

En la Tabla 2.7 se procede a describir cada entidad y atributos del modelo físico.

Nombre Tabla	Descripción
alquiler	id_alquiler (Clave Primaria) id_eventos (Clave Foránea) id_ealquiler (Clave Foránea) costo_alquiler saldo_alquiler descripcion_alquiler
dias_laborables	id_diaslab(Clave Primaria) descripcion
dias_reserva	id_reserva (Clave Primaria) id_eventoespacio (Clave Foránea) fecha_diaevento hora_inicio time hora_fin time estado observaciones
detalle_actividad_evento	id_actividad (Clave Primaria) id_reserva (Clave Foránea) actividad actor hora_inicio hora_fin
edificio	id_edificio (Clave Primaria) nom_edificio referencia_edificio codigo_edificio
espacio	id_espacio(Clave Primaria) id_tipo_espacio(Clave Foránea) id_lugar_espacio(Clave Foránea) nom_espacio descripcion_espacio referencia
estado_alquiler	id_ealquiler(Clave Primaria) estado_alquiler descripcion_ealquiler
estado_espacio	id_estado_espacio(Clave Primaria) estado_espacio descripcion_estadoesp
estado_evento	id_evento(Clave Primaria) estado_evento descripcion_eevento
eventos	id_eventos(Clave Primaria) id_tevento(Clave Foránea) id_responsable(Clave Foránea) id_espacio(Clave Foránea) id_evento(Clave Foránea) nom_evento fecha_reserve_evento

	fecha_inicio_evento fecha_fin_evento hora_inicio_evento hpra_fin_evento descripcion
Nombre Tabla	**Descripción**
evento_espacio	id_eventoespacio(Clave Primaria) id_lugar_espacio(Clave Foránea) id_eventos(Clave Foránea) id_esp, id_ev
horario	id_horario(Clave Primaria) descripcion_horario hora_inicio hora_fin
horario_espacio	id_horario_espacio(Clave Primaria) id_diaslab(Clave Foránea) id_espacio(Clave Foránea) id_horario(Clave Foránea) id_estado_espacio(Clave Foránea) descripcion_he uso_he
horario_espacio_ocupante	id_heo(Clave Primaria) id_horario_espacio(Clave Foránea) id_ocupante(Clave Foránea) descripcion
lugar_espacio	id_lugar_espacio(Clave Primaria) id_piso(Clave Foránea) coordenada_x1 coordenada_x2 coordenada_y1 coordenada_y2 costo_diario_esp capacidad
ocupante	id_ocupante(Clave Primaria) nombre_ocupante apellido_ocupante cedula_ocupante direccion_ocupamte email_ocupante telefono_ocupante
pagos	id_pago(Clave Primaria) id_alquiler(Clave Foránea) monto saldo fecha_pago
piso	id_piso(Clave Primaria) id_plano(Clave Foránea) id_edificio(Clave Foránea) nom_piso
planos	id_plano(Clave Primaria)

	nombre_plano descripcion_plano plano estado
Nombre Tabla	**Descripción**
responsable	id_responsable(Clave Primaria) nombre_resp apellido_resp cedula_resp direccion_resp telefono_resp, email_resp
tipo_espacio	id_tipo_espacio(Clave Primaria) nom_tipo_espacio descripcion_t_espacio, costo_t_espacio
tipo_evento	id_tevento(Clave Primaria) tipo_evento, descripcion_tevento

Tabla 2. 7: Descripción de Entidades de la Base de Datos. Elaborado por los Autores.

Los significados para cada una de las tablas y sus atributos se encuentran en el Anexo 7 – Diccionario de Base de Datos.

2.3.1.1 Definición de roles y perfiles de usuario

Súper Administrador:

- Tiene acceso a completamente toda la funcionalidad del Sistema.

Administrador:

- Personas encargadas de la administración física de la Escuela Politécnica Nacional o de ciertas instalaciones de la misma.
- Tienen habilitada las acciones para registrar, actualizar, eliminar y buscar información en el sistema generalmente.
- Podrá hacer uso de las siguientes opciones:
    - Formularios.
    - Administración de Eventos.
    - Administración Infraestructura Física.
    - Administración de Alquileres.
    - Asignación de Horarios y Ocupantes a Espacios.
    - Generación de Reportes.

Usuario Normal:

- Pueden ser estudiantes, profesores, personal administrativo.
- Puede realizar una reservación al administrador, por medio de tres vías de comunicación: telefónica, visita personal y correo electrónico, describiendo los espacios que necesita y los horarios respectivos.
- Solo Puede consultar información registrada en el sistema.
    - Referente a la ubicación de edificios, pisos y espacios que lo conforman como aulas oficinas, laboratorios, entre otros.
    - Referente a eventos, asignación de horarios y ocupantes a espacios.
    - Información adicional como: ubicación de las diferentes personas que ocupan un espacio con su respectivo horario y días.

Usuario Externo:

Son las personas externas a la Escuela Politécnica Nacional.

- Puede realizar una reservación al administrador por medio de tres vías de comunicación: telefónica, visita personal y correo, describiendo los espacios que necesita y los horarios respectivos.
- Solo pude verificar información registrada en el sistema de forma más resumida que un usuario normal.

2.3.2 EJECUCIÓN DEL SEGUNDO SPRINT

Para el Segundo Sprint se inicia con el mapeo de entidades de la base de datos con clases java. Al usar tecnología J2EE existen varias formas de realizar este procedimiento, sin embargo para el desarrollo del sistema se usara JPA.

Esta API nos permite usar un modelo de persistencia basado en Pojos, que facilita el mapeo de la base de datos con java.

A continuación se presenta un ejemplo del mapeo para la entidad Edificio de nuestra base de datos.

```
package ec.edu.epn.infraestructura.entidades;

import java.io.Serializable;
import javax.persistence.*;
import java.util.List;
/**
 * The persistent class for the edificio database table.
 *
 */
@Entity
@Table(name="edificio" , catalog = "bddcorpepn", schema =
"`Infraestructura`")
public class Edificio implements Serializable {
	private static final long serialVersionUID = 1L;

	@Id
	@Column(name="id_edificio")
	private Integer idEdificio;

	@Column(name="codigo_edificio")
	private String codigoEdificio;

	@Column(name="nom_edificio")
	private String nomEdificio;

	@Column(name="referencia_edificio")
	private String referenciaEdificio;

	//bi-directional many-to-one association to Piso
	@OneToMany(mappedBy="edificio")
	private List<Piso> pisos;

	public Edificio() {
	}

	public Integer getIdEdificio() {
		return this.idEdificio;
	}

	public void setIdEdificio(Integer idEdificio) {
		this.idEdificio = idEdificio;
	}

	public String getCodigoEdificio() {
		return this.codigoEdificio;
	}

	public void setCodigoEdificio(String codigoEdificio) {
		this.codigoEdificio = codigoEdificio;
	}

	public String getNomEdificio() {
		return this.nomEdificio;
```

```
	}

	public void setNomEdificio(String nomEdificio) {
		this.nomEdificio = nomEdificio;
	}

	public String getReferenciaEdificio() {
		return this.referenciaEdificio;
	}

	public void setReferenciaEdificio(String referenciaEdificio) {
		this.referenciaEdificio = referenciaEdificio;
	}

	public List<Piso> getPisos() {
		return this.pisos;
	}

	public void setPisos(List<Piso> pisos) {
		this.pisos = pisos;
	}

	public Piso addPiso(Piso piso) {
		getPisos().add(piso);
		piso.setEdificio(this);

		return piso;
	}

	public Piso removePiso(Piso piso) {
		getPisos().remove(piso);
		piso.setEdificio(null);

		return piso;
	}}
```

Como se puede apreciar muestra notaciones como @Entity, @Table, @Column, entre otras, las cuales permiten realizar la persistencia entre la entidad de nuestra tabla con la base de datos.

El mismo procedimiento es aplicado a las demás entidades que conforman la base de datos.

En el siguiente código podemos apreciar el mapeo de las mismas en un archivo ".xml" llamado persitence, el cual es generado al mapear cada una de las entidades.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0"
	xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd">
```

```
      <persistence-unit name="ServicioInfraestructura"
            transaction-type="JTA">
            <provider>org.hibernate.ejb.HibernatePersistence</provider>
            <jta-data-source>java:jboss/datasources/SeguridadEPNDS</jta-
data-source>
            <class>ec.edu.epn.infraestructura.entidades.Alquiler</class>
      <class>ec.edu.epn.infraestructura.entidades.Dias_Reserva</class>
            <class>ec.edu.epn.infraestructura.entidades.Edificio</class>
            <class>ec.edu.epn.infraestructura.entidades.Espacio</class>
      <class>ec.edu.epn.infraestructura.entidades.EstadoAlquiler</class>
      <class>ec.edu.epn.infraestructura.entidades.EstadoEspacio</class>
      <class>ec.edu.epn.infraestructura.entidades.EstadoEvento</class>
            <class>ec.edu.epn.infraestructura.entidades.Evento</class>
      <class>ec.edu.epn.infraestructura.entidades.EventoEspacio</class>
            <class>ec.edu.epn.infraestructura.entidades.Horario</class>
      <class>ec.edu.epn.infraestructura.entidades.LugarEspacio</class>
            <class>ec.edu.epn.infraestructura.entidades.Ocupante</class>
            <class>ec.edu.epn.infraestructura.entidades.Pago</class>
            <class>ec.edu.epn.infraestructura.entidades.Piso</class>
            <class>ec.edu.epn.infraestructura.entidades.Plano</class>
      <class>ec.edu.epn.infraestructura.entidades.Responsable</class>
      <class>ec.edu.epn.infraestructura.entidades.TipoEspacio</class>
      <class>ec.edu.epn.infraestructura.entidades.TipoEvento</class>
      <class>ec.edu.epn.infraestructura.entidades.Diaslaborable</class>
      <class>ec.edu.epn.infraestructura.entidades.HorarioEspacio</class>
      <class>ec.edu.epn.infraestructura.entidades.HorarioEspacioOcupante<
/class>
      <class>ec.edu.epn.infraestructura.entidades.Detalle_Actividad_Event
o</class>
            <properties>
                  <property name="hibernate.dialect"
value="org.hibernate.dialect.PostgreSQLDialect" />
                  <property name="hibernate.hbm2ddl.auto" value="false"
/>
                  <property name="hibernate.show_sql" value="true" />
            </properties>
      </persistence-unit>
</persistence>
```

Una vez mapeadas las entidades en nuestras clases javas, se procede con la creación de interfaces en donde se establecen los métodos CRUD:

- Creación
- Lectura
- Actualización
- Borrado

Para esta iteración se trabajará con siguientes entidades:

- Edificio.java
- Plano.java

- LugarEspacio.java
- Espacio.java
- TipoEspacio.java

En el siguiente código podemos ver la clase java que representa la interfaz perteneciente a la entidad Edificio.

```
@Local
public interface EdificioDAO extends DaoGenerico<Edificio> {
	public int maxEdifico();

	public List<Edificio> buscarEdificioXNombre(String NombreEd);

	public List<Edificio> buscarEdificioXCodigo(String CodigoEd);

	int consultarId();

	List<Edificio> listarEdificio();

	List<Edificio> buscarEdificio(int edif);


	List<LugarEspacio> buscarEdificioXLugarEspacio(Edificio edificio);

}
```

En esta implementación se encuentran los métodos mencionados anteriormente, el mismo procedimiento se realiza con las demás clases seleccionadas dando lugar a las siguientes interfaces.

- EdificioDAO.java
- PlanoDAO.java
- LugarEspacioDAO.java
- EspacioDAO.java
- TipoEspacioDAO.java

La siguiente tarea es implementar los métodos de nuestras interfaces, a continuación se presenta un extracto de código para la implementación de interfaz EdificioDAO.java., el código completo lo podemos revisar en el Anexo 2-Codificación.

```
@Stateless
```

```
public class EdificioDAOImplement extends DaoGenericoImplement<Edificio>
            implements EdificioDAO {

      @Override
      public int maxEdifico() {
            Query q = getEntityManager().createQuery(
                        "SELECT MAX(edi.idEdificio) FROM Edificio edi");
            Integer idEdificio = null;

            try {
                  idEdificio = (Integer)
q.setMaxResults(1).getSingleResult();
            } catch (Exception e) {
                  e.printStackTrace();
                  idEdificio = null;
            }
            if (idEdificio == null) {
                  idEdificio = 1;
            } else {
                  ++idEdificio;
            }
            return idEdificio;
      }

      @Override
      public int consultarId() {
            Query q = getEntityManager().createQuery(
                        "SELECT MAX(edif.idEdificio)FROM Edificio edif");
            Integer idReq = null;
            try {
                  idReq = (Integer) q.setMaxResults(1).getSingleResult();
            } catch (Exception e) {

                  e.printStackTrace();
                  idReq = null;

            }
            if (idReq == null) {
                  idReq = 1;

            } else {
                  ++idReq;

            }

            return idReq;

      }

      @Override
      public List<Edificio> listarEdificio() {
            StringBuilder queryString = new StringBuilder(
                        "SELECT edif FROM Edificio edif");
            Query query =
getEntityManager().createQuery(queryString.toString());

            return query.getResultList();
      }

      @Override
```

```
	public List<Edificio> buscarEdificio(int edif) {
		StringBuilder queryString = new StringBuilder(
				"SELECT edif FROM Edificio edif where
edif.idEdificio=?1");
		Query query =
getEntityManager().createQuery(queryString.toString());
		query.setParameter(1, edif);

		return query.getResultList();
	}

	@Override
	public List<Edificio> buscarEdificioXNombre(String nombreEdificio)
{
		StringBuilder queryString = new StringBuilder(
				"SELECT edif FROM Edificio edif where
edif.nomEdificio like ?1");
		Query query =
getEntityManager().createQuery(queryString.toString());
		query.setParameter(1, "%" + nombreEdificio + "%");

		return query.getResultList();
	}

	@Override
	public List<Edificio> buscarEdificioXCodigo(String codEdificio) {
		StringBuilder queryString = new StringBuilder(
				"SELECT edif FROM Edificio edif where
edif.codigoEdificio like ?1");
		Query query =
getEntityManager().createQuery(queryString.toString());
		query.setParameter(1, "%" + codEdificio + "%");

		return query.getResultList();
	}

	@Override
	public List<LugarEspacio> buscarEdificioXLugarEspacio(Edificio
edificio) {
		StringBuilder queryString = new StringBuilder(
				"SELECT edif FROM LugarEspacio edif where
edif.edificio.idEdificio=?1");
		Query query =
getEntityManager().createQuery(queryString.toString());
		query.setParameter(1, edificio.getIdEdificio());

		return query.getResultList();
	}

}
```

La implementación de nuestras interfaces da como resultado las siguientes clases:

- EdificioDAOImplement.java
- PlanoDAOImplement.java

- LugarEspacioDAOImplement.java
- EspacioDAOImplement.java
- TipoEspacioDAOImplement.java

El siguiente paso para este Sprint es la creación de nuestras clases BackingBean, las que actuarán como controladores de nuestro patrón MVC. A continuación se presenta un extracto de código para la implementación del BackingBean perteneciente a la entidad edificio. El código completo lo podemos revisar en el Anexo 2-Codificación.

```
* METODOS CRUD BACKING_BEAN EDIFICIO*/

// Guardar Edificio
public void guardar() {
//ec.edu.epn.infraestructura.entities.Edificio edi = new
ec.edu.epn.infraestructura.entities.Edificio();
            edi.setIdEdificio(edificioI.maxEdifico());
            edi.setNomEdificio(nom_edificio);
            edi.setReferenciaEdificio(referencia_edificio);
            edi.setCodigoEdificio(codigo_edificio);
            try {
                  edificioI.save(edi);
                  borrarRastros();
                  System.out.print(MSJ_SAVE);
                  final FacesMessage msg = new FacesMessage(
                              FacesMessage.SEVERITY_INFO, ED,
                              ED.concat(" ").concat(SAVE));
                  FacesContext.getCurrentInstance().addMessage(null,
msg);
            } catch (Exception e) {
                  System.out
                              .print(EXEPCIONES);
                  e.printStackTrace();
                  final FacesMessage msg = new FacesMessage(
                              FacesMessage.SEVERITY_INFO, ED,
ERROR_SAVE);
                  FacesContext.getCurrentInstance().addMessage(null,
msg);
            }

      }
// Eliminar Edificio
public void eliminar() {
            try {
                  edificioI.delete(edificioSelect);
                  System.out.print(MSJ_DELETE);
                  final FacesMessage msg = new FacesMessage(
                              FacesMessage.SEVERITY_INFO, ED,
                              ED.concat(" ").concat(DELETE));
                  FacesContext.getCurrentInstance().addMessage(null,
msg);
            } catch (Exception e) {
                  System.out
```

```
                        .print(EXEPCIONES);
                e.printStackTrace();
                final FacesMessage msg = new FacesMessage(
                        FacesMessage.SEVERITY_INFO,ED,
                    ERROR_DELETE);
                FacesContext.getCurrentInstance().addMessage(null,
msg);
            }
      }
// Actualizar Edificio
public void actualizar() {
            try {
                edificioI.update(edificioSelect);
                System.out.print(MSJ_UPDATE);
                final FacesMessage msg = new FacesMessage(
                        FacesMessage.SEVERITY_INFO, ED,
                        ED.concat(" ").concat(UPDATE));
                FacesContext.getCurrentInstance().addMessage(null,
msg);
            } catch (Exception e) {
                System.out
                        .print(EXEPCIONES);
                e.printStackTrace();
                final FacesMessage msg = new FacesMessage(
                        FacesMessage.SEVERITY_INFO, ED,
                        ERROR_UPDATE);
                FacesContext.getCurrentInstance().addMessage(null,
msg);
            }
      }
```

Los BackingBean creados para esta iteración son los siguientes:

- EdificioBackingBean.java
- PlanoBackingBean.java
- PisoBackingBean.java
- LugarEspacioBackinngBean.java
- EspacioBackingBean.java
- TipoEspacioBackingBean.java

Una vez creadas clases BackingBean procedemos a realizar las ultimas tereas del Product Backlog que tienen que ver con implementación de páginas JSF, las cuales representan las vistas de nuestro modelo MVC.

Para facilitar el desarrollo de las mismas se utilizará tecnología PrimeFaces la cual se basa en un conjunto de componentes JSF, que brindan una gran variedad de componentes UI y funcionalidades.

En la Figura 2.20 se presenta el diseño del prototipo para la página JSF perteneciente al registro de edificios.

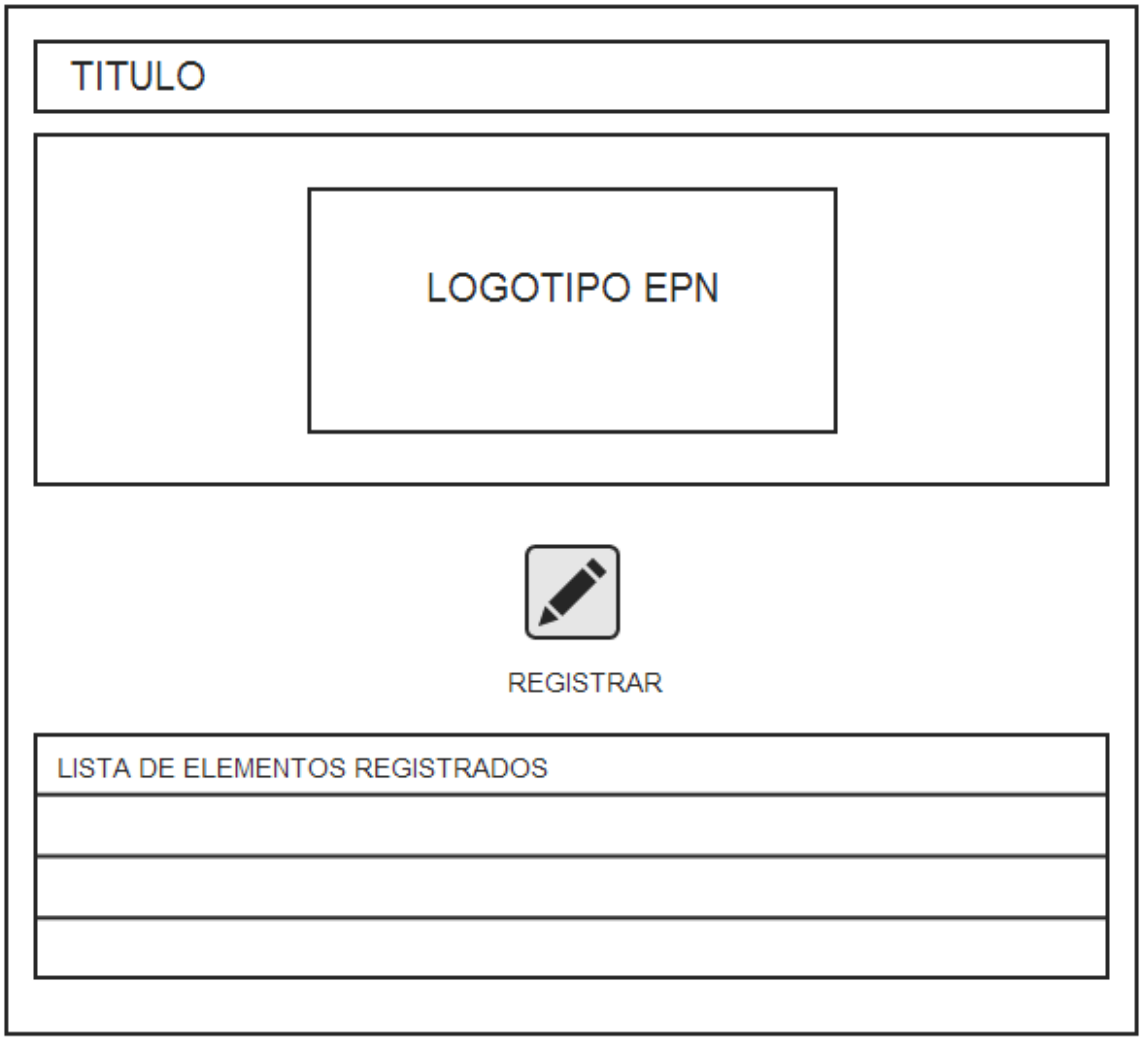


Fig. 1.36: Prototipo Página JSF para el registro de Edificios. Elaborado por los Autores.

En el siguiente código se muestra una parte de la implementación de la página JSF perteneciente al registro de edificios. El código completo lo podemos revisar en el Anexo 2-Codificación.

```
<div align="center">
<!-- Panel de registro de nuevo elemento-->
<h:panelGrid columns="1">
<p:commandLink id="btnnuevoEdificio" onclick="pnlnEdi.show()"
process="@this" update="@this">
```

```
<center>
<h:graphicImage value="/images/nuevo3d.png"
style="float:center;height : 50px; width : 45px;" />
</center>
<br />
<h:outputText value="Nuevo Edificio"
style="FONT-SIZE: small; FONT-WEIGHT: bold;" />
</p:commandLink>
</h:panelGrid>
</div><br />
<!-- Tabla que muestra los nuevos ingresos-->
<p:dataTable id="edificio" width="735px"
rowsPerPageTemplate="5,10,15,20" PáginatorPosition="bottom"
PáginatorTemplate="{FirstPageLink} {PreviousPageLink} {PageLinks}
{NextPageLink} {LastPageLink} {RowsPerPageDropdown}"
Páginator="true" rows="10"
emptyMessage="No hay Edificios Registrados"
value="#{edificioBack.listaEdificio}" var="lstedif">
<p:column headerText="Nombre">
<h:outputText value="#{lstedif.nomEdificio}" />
</p:column>
<p:column headerText="Referencia">
<h:outputText value="#{lstedif.referenciaEdificio}">
</h:outputText>
</p:column>
<p:column headerText="Código">
<h:outputText value="#{lstedif.codigoEdificio}">
</h:outputText>
</p:column>
<p:column headerText="Acciones" width="60">
<p:commandLink id="btneliminarEdificio" onclick="ei.show()"
process="@this, edificio"
update="@([id$=mesajeEliminaEdificio])">
<h:graphicImage value="/images/eliminar24.ico"
title="Eliminar Edificio" style="border:0px solid #CAD6E0;"
height="20" width="20">
</h:graphicImage>
<f:setPropertyActionListener
target="#{edificioBack.edificioSelect}" value="#{lstedif}" />
</p:commandLink>
<p:commandLink id="btnactualizarTE" onclick="pnlaedif.show()"
update="@([id$=pngactedificio])" process="@this, edificio">
<h:graphicImage value="/images/EDITAR2.png"
title="Editar Edificio" style="border:0px solid #CAD6E0;"
height="20" width="20">
</h:graphicImage>
<f:setPropertyActionListener
target="#{edificioBack.edificioSelect}" value="#{lstedif}" />
</p:commandLink>
</p:column></p:dataTable>
```

Siguiendo el mismo esquema para la creación de las páginas JSF, en esta iteración se han creado las siguientes páginas JSF:

- registroEdificio.JSF
- registrarPlanos.JSF
- registroPiso.JSF

- registroLugarEspacio.JSF
- registroEspacio.JSF
- registroTipoEspacio.JSF

2.3.3 EJECUCIÓN DEL TERCER SPRINT

En el Tercer Sprint se continúa con la creación de interfaces y su implementación, clases BackingBean y páginas JSF.

Para esta iteración nos centraremos básicamente en la gestión de horarios, ocupantes, y eventos, al igual que el anterior Sprint iniciamos con la creación de interfaces. A continuación podemos observar una parte del código perteneciente la interfaz de la entidad Evento. El código completo lo podemos revisar en el Anexo 2-Codificación.

```
@Local
public interface EventoDAO extends DaoGenerico<Evento> {
    int consultarId();

    List<Evento> buscarxparametros(Evento evento, Responsable reponsable,
            TipoEvento tipoEvento, EstadoEvento estadoEvento);

    List<Evento> listarEvento();

    List<Evento> listarSoloEventos();

    List<Evento> listarEventoSINAlquiler();

    List<Evento> buscarEvento(int evento);

    List<Evento> buscarEventoXResponsable(String resp);

    List<Evento> buscarEventoXNombre(String evento);

    List<Evento> buscarEventoXTipo(String tipoEvento);

    List<Evento> buscarEventoXEstado(String estado);

    List<Evento> buscarEventoXEstadoNoConfirmado();

    List<Evento> buscarEventoXFechaInicio(Evento ev);

    List<Evento> buscarEventoXFechaFin(Evento ev);

    List<Evento> buscarEventoXFechaReserva(Evento ev);

    List<EventoEspacio> reporteEventoXFechas(Date fechaI, Date fechaF,
            TipoEvento tipoevento, Responsable responsable);

    boolean buscarEventoXFechas(Evento ev);
```

```
    List<Espacio> listaEspacios();

    List<TipoEspacio> listaTipoEspacio();

    List<EstadoEspacio> listaEstadoEspacio();

    List<Edificio> listaEdificio();

    List<Piso> listaPiso();

    Evento buscarEventoXFIFFNombre(Date fechaI, Date fechaF, String
nombre);

    // Busqueda de eventos sin alquiler

    List<Evento> buscarEventoXNombreSA(String evento);

    List<Evento> buscarEventoXResponsableSA(String resp);

    List<Evento> buscarEventoXTipoSA(String tipoEvento);

    List<Evento> buscarEventoXEstadoSA(String estado);

    List<Evento> buscarEventoXFechaFinSA(Evento ev);

    List<Evento> buscarEventoXFechaInicioSA(Evento ev);
}
```

Las interfaces creadas para este Sprint son los siguientes:

- HorarioDAO.java
- HorarioEspacioDAO.java
- EstadoEspacioDAO.java
- OcupanteDAO.java
- EventoDAO.java
- EstadoEventoDAO.java
- TipoEventoDAO.java

Ahora se procede con la implementación de las interfaces. La siguiente parte de código presenta un ejemplo de la implementación de la interfaz EventoDAO.java. El código completo lo podemos revisar en el Anexo 2-Codificación.

```
@Stateless
public class EventoDAOImplement extends DaoGenericoImplement<Evento>
implements
            EventoDAO {

      @Override
      public int consultarId() {
```

```
            Query q = getEntityManager().createQuery(
                        "SELECT MAX(evento.idEventos) FROM Evento
evento");

            Integer idReq = null;
            try {
                  idReq = (Integer) q.setMaxResults(1).getSingleResult();
            } catch (Exception e) {

                  e.printStackTrace();
                  idReq = null;

            }
            if (idReq == null) {
                  idReq = 1;

            } else {
                  ++idReq;

            }
            return idReq;

      }

      @Override
      public List<Evento> listarEvento() {
            StringBuilder queryString = new StringBuilder(
                        "SELECT eves FROM Evento eves");
            Query query =
getEntityManager().createQuery(queryString.toString());
            return query.getResultList();
      }

      @Override
      public List<Evento> listarSoloEventos() {
            StringBuilder queryString = new StringBuilder(
                        "SELECT ev FROM Evento ev");
            Query query =
getEntityManager().createQuery(queryString.toString());
            return query.getResultList();
      }

      @Override
      public List<Evento> buscarEvento(int evento) {
            StringBuilder queryString = new StringBuilder(
                        "SELECT evento FROM Evento evento where
evento.idEventos=?1");
            Query query =
getEntityManager().createQuery(queryString.toString());
            query.setParameter(1, evento);
            return query.getResultList();
      }

      @Override
      public List<Evento> buscarEventoXNombre(String evento) {
            StringBuilder queryString = new StringBuilder(
                        "SELECT evento FROM Evento evento where
evento.nomEvento like ?1");
            Query query =
getEntityManager().createQuery(queryString.toString());
```

```
            query.setParameter(1, "%" + evento + "%");
            return query.getResultList();
      }

      @Override
      public List<Evento> buscarEventoXResponsable(String resp) {
            StringBuilder queryString = new StringBuilder(
                        "SELECT evento FROM Evento evento where
evento.responsable.cedulaResp like ?1");
            Query query =
getEntityManager().createQuery(queryString.toString());
            query.setParameter(1, resp + "%");
            return query.getResultList();
      }

      @Override
      public List<Evento> buscarEventoXTipo(String tipoEvento) {
            StringBuilder queryString = new StringBuilder(
                        "SELECT evento FROM Evento evento where
evento.tipoEvento.tipoEvento=?1");
            Query query =
getEntityManager().createQuery(queryString.toString());
            query.setParameter(1, tipoEvento);
            return query.getResultList();
      }

      @Override
      public List<Evento> buscarEventoXEstado(String estado) {
            StringBuilder queryString = new StringBuilder(
                        "SELECT evento FROM Evento evento where
evento.estadoEvento.estadoEvento=?1");
            Query query =
getEntityManager().createQuery(queryString.toString());
            query.setParameter(1, estado);
            return query.getResultList();
      }
```

La implementación de las interfaces para esta iteración son los siguientes:

- HorarioDAOImplement.java
- HorarioEspacioDAOImplement.java
- EstadoEspacioDAOImplement.java
- OcupanteDAOImplement.java
- EventoDAOImplement.java
- EstadoEventoDAOImplement.java
- TipoEventoDAOImplement.java

El siguiente paso es la creación de los BackingBean, la siguiente parte de código muestra la implementación del método guardar en la clase BackingBean para la

entidad Evento. El código completo lo podemos revisar en el Anexo 2-Codificación.

```
public void guardar() {
            try {
                  if (eventoI.buscarEventoXFechas(Evento) == false) {
                        final FacesMessage msg = new FacesMessage(
                                    FacesMessage.SEVERITY_FATAL,
                                    "No puede guardar el Evento",
                                    "Eventos encontrados con fechas
similares.!");
                  FacesContext.getCurrentInstance().addMessage(null,
msg);
                  } else {
                        verificarHorario();
                        if (!this.match.equals("SI")) {
                              if (responsableSelect == null) {
                                    final FacesMessage msg = new
FacesMessage(

      FacesMessage.SEVERITY_WARN, "Responsable",
                                                "Debe seleccionar un
Responsable del Evento.!");

      FacesContext.getCurrentInstance().addMessage(null, msg);
                              } else if (tipoEventoSelect == null) {
                                    final FacesMessage msg = new
FacesMessage(

      FacesMessage.SEVERITY_WARN, "Tipo Evento",
                                                "Debe seleccionar un Tipo
de Evento.!");
      FacesContext.getCurrentInstance().addMessage(null, msg);
                              } else if (espacioSelect == null) {
                                    final FacesMessage msg = new
FacesMessage(

      FacesMessage.SEVERITY_WARN, "Espacio",
                                                "Debe escoger un Espacio
para el Evento.!");
      FacesContext.getCurrentInstance().addMessage(null, msg);
                              } else if (estadoEventoSelect == null) {
                                    final FacesMessage msg = new
FacesMessage(

      FacesMessage.SEVERITY_WARN, "Estado Evento",
                                                "Debe escoger el Estado
del Evento.!");
      FacesContext.getCurrentInstance().addMessage(null, msg);
                              } else {
      Evento.setResponsable(responsableSelect);

      Evento.setTipoEvento(tipoEventoSelect);
                                    Evento.setEspacio(espacioSelect);

      Evento.setEstadoEvento(estadoEventoSelect);

      Evento.setIdEventos(eventoI.consultarId());
```

```
                                    Evento.setHoraInicioEvento(new
Time(Evento

      .getFechaInicioEvento().getTime()));
                                    Evento.setHpraFinEvento((new
Time(Evento

      .getFechaFinEvento().getTime())));
                                    Evento.setFechaReserveEvento(new
Date());
                                    eventoI.guardar(Evento);
                                    final FacesMessage msg = new
FacesMessage(
      FacesMessage.SEVERITY_INFO, "Evento",
                                                "Evento Guardado.!");

      FacesContext.getCurrentInstance().addMessage(null, msg);

      RequestContext.getCurrentInstance().execute(
                                                "eventDialog.hide();");
                                    borrarRastros();
                                    diasMatch = "";
                                    mensaje = "";
                              }
                        } else {
                              // Imprimimos el Map con un Iterador para
mostrar en
                              // mensaje de error en rojo
                              diasMatch = "";
                              for (Map<String, String> ma : lista) {
                                    Iterator it = ma.keySet().iterator();
                                    while (it.hasNext()) {
                                          String key =
it.next().toString();
                                          System.out.println(" Usaurio: "
+ key + " Dias: "
                                                      + ma.get(key) +
"\n");
                                          diasMatch += " Dia: " +
ma.get(key);
                                    }
                              }
                              //
RequestContext.getCurrentInstance().execute(
                              // "pnlMatch.show();");
                              mensaje = "Cruce de Horarios";
                              // final FacesMessage msg = new
FacesMessage(
                              // FacesMessage.SEVERITY_ERROR, "Horarios",
                              // "Cruce de Horarios.!");
                              //
FacesContext.getCurrentInstance().addMessage(null, msg);
                        }
                  }
            } catch (Exception ex) {
                  ex.printStackTrace();
                  final FacesMessage msg = new FacesMessage(
                              FacesMessage.SEVERITY_FATAL, "Evento",
"Error al Guardar.!");
```

```
                    FacesContext.getCurrentInstance().addMessage(null,
msg);
                    System.out.println("ERROR EN eventos:guardar");
                }
```

Los BackingBean creados para esta iteración son los siguientes:

- eventos.java
- HorarioBackingBean.java
- OcupanteBackingBean.java
- tipoevento.java
- estadoEvento.java
- EstadoEspacioBackingBean.java

Ahora realizamos las últimas tareas relacionadas con la creación de páginas JSF, En la Figura 2.21 podemos apreciar el prototipo del diseño de la página JSF perteneciente al registro de eventos.

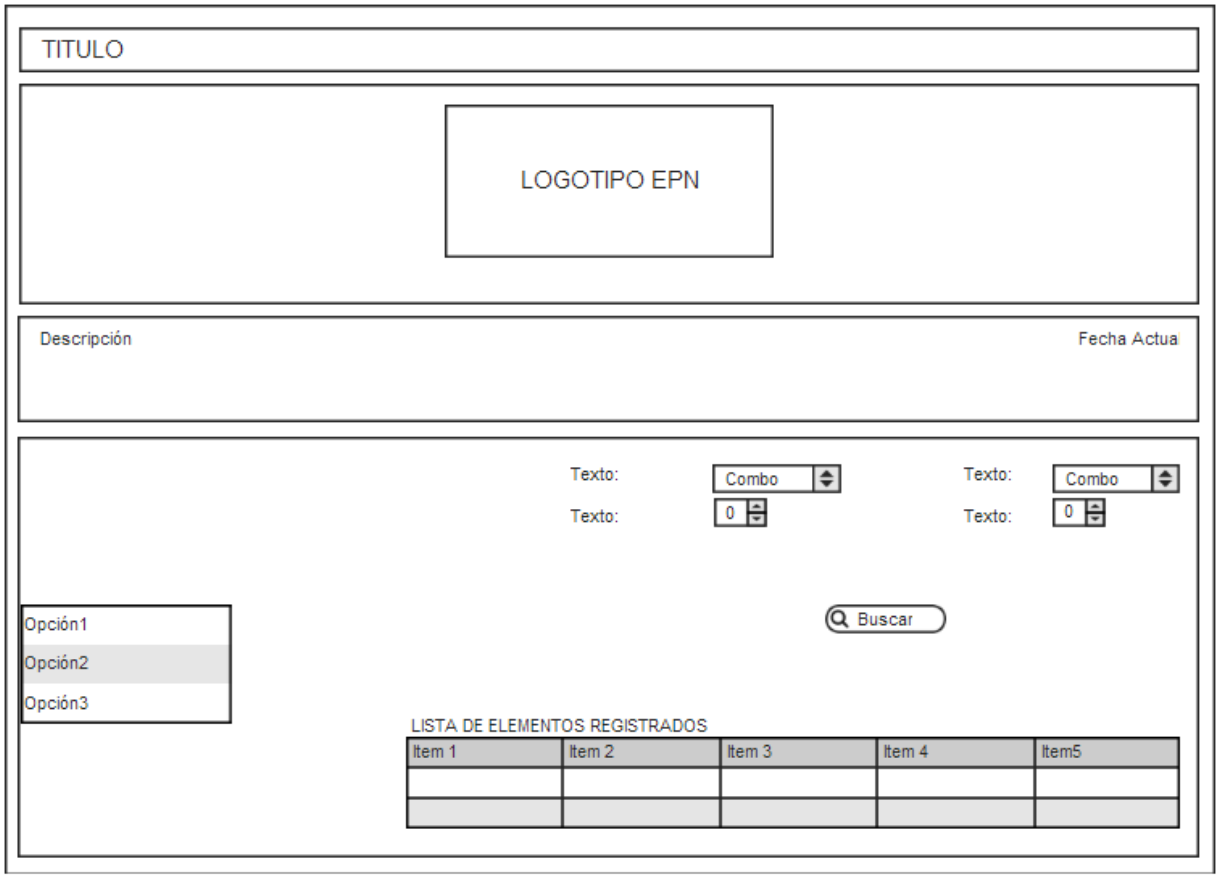


Fig. 1.37: Prototipo Página JSF Registro de Eventos. Elaborado por los Autores.

A continuación presentamos ciertas partes de la implementación del código para la página JSF perteneciente al registro de eventos. El código completo lo podemos revisar en el Anexo 2-Codificación.

```
<!-- LISTADO DE EVENTOS CALENDARIZADOS -->
<!-- ************************************* -->
<p:schedule id="schedule" value="#{eventosb.eventModel}"
editable="true" widgetVar="myschedule" locale="es">
<p:ajax event="dateSelect" listener="#{eventosb.onDateSelect}"
process="@this,schedule"
update="@([id$=eventDetails], [id$=messages], [id$=espacioSHDetalles])"
oncomplete="eventDialog.show()" />

<p:ajax event="eventSelect" listener="#{eventosb.onEventSelect}"
process="@this,schedule"
update="@([id$=eventDetails], [id$=messages], [id$=espacioSHDetalles])"
oncomplete="eventDialog.show()" />

<p:ajax event="eventMove" listener="#{eventosb.onEventMove}"
process="@this,schedule"
update="@([id$=eventDetails], [id$=messages], [id$=espacioSHDetalles])"
oncomplete="eventDialog.show()" />

<p:ajax event="eventResize" listener="#{eventosb.onEventResize}"
process="@this,schedule"
update="@([id$=eventDetails], [id$=messages])"
oncomplete="eventDialog.show()" />

</p:schedule>
<p:separator />
```

Para esta iteración los JSF resultantes son las siguientes:

- eventos.JSF
- registroHorario.JSF
- registroOcupante.JSF
- registrarTipoEvento.JSF
- registrarEstadoEvento.JSF
- registroEstadoEspacio.JSF

2.3.4 EJECUCIÓN DEL CUARTO SPRINT

En esta iteración se procede a realizar la creación de interfaces, implementación, clases BackingBean y páginas JSF pertenecientes a la administración de pagos de alquiler para eventos y la administración de responsables de eventos. Además se realiza la generación de reportes.

Como en los anteriores Sprints iniciamos con la creación de interfaces, las interfaces creadas son las siguientes:

- ResponsableDAO.java
- AlquilerDAO.java
- EstadoAlquilerDAO.java

A continuación presentamos parte del código de la interfaz perteneciente a la entidad alquiler. El código completo lo podemos revisar en el Anexo 2-Codificación.

```
@Local

public interface AlquilerDAO extends DaoGenerico<Alquiler> {

    int consultarId();

    List<Alquiler> listarAlquiler();

    List<Alquiler> listaAlquilerSINPago();

    List<Alquiler> buscarAlquiler(int alq);

    List<Alquiler> buscarAlquilerXEvento(String nombreEvento);

    List<Alquiler>                    buscarAlquilerXTipoAlquiler(String
nombreTipoAlquiler);

    boolean buscarEventoAlquilado(int idevento);

}
```

Después procedemos a crear su implementación, como ejemplo presentamos parte del código para la implementación de la interfaz alquiler. El código completo lo podemos revisar en el Anexo 2-Codificación.

```
@Stateless
public class AlquilerDAOImplement extends DaoGenericoImplement<Alquiler>
            implements AlquilerDAO {

      @Override
      public int consultarId() {
            Query q = getEntityManager().createQuery(
                        "SELECT MAX(alq.idAlquiler)FROM Alquiler alq");
            Integer idReq = null;
            try {
                  idReq = (Integer) q.setMaxResults(1).getSingleResult();
```

```
            } catch (Exception e) {

                  e.printStackTrace();
                  idReq = null;

            }
            if (idReq == null) {
                  idReq = 1;

            } else {
                  ++idReq;

            }

            return idReq;

      }

      @Override
      public List<Alquiler> listarAlquiler() {
            StringBuilder queryString = new StringBuilder(
                        "SELECT alq FROM Alquiler alq");
            Query query =
getEntityManager().createQuery(queryString.toString());

            return query.getResultList();
      }

      @Override
      public List<Alquiler> buscarAlquiler(int alq) {
            StringBuilder queryString = new StringBuilder(
                        "SELECT alq FROM Alquiler alq where
alq.idAlquiler=?1");
            Query query =
getEntityManager().createQuery(queryString.toString());
            query.setParameter(1, alq);

            return query.getResultList();
      }

      @Override
      public List<Alquiler> buscarAlquilerXEvento(String nombreEvento) {
            StringBuilder queryString = new StringBuilder(
                        "SELECT alq FROM Alquiler alq where
alq.evento.nomEvento=?1");
            Query query =
getEntityManager().createQuery(queryString.toString());
            query.setParameter(1, nombreEvento);

            return query.getResultList();
      }

      @Override
      public List<Alquiler> buscarAlquilerXTipoAlquiler(String
nombreTipoAlquiler) {
            StringBuilder queryString = new StringBuilder(
                        "SELECT alq FROM Alquiler alq where
alq.estadoAlquiler.estadoAlquiler=?1");
            Query query =
getEntityManager().createQuery(queryString.toString());
```

```
            query.setParameter(1, nombreTipoAlquiler);

            return query.getResultList();
      }

      @Override
      public List<Alquiler> listaAlquilerSINPago() {
            StringBuilder queryString = new StringBuilder(
                        "select alq from Alquiler alq where
alq.idAlquiler not IN (select pag.alquiler.idAlquiler from Pago pag)");
            Query query =
getEntityManager().createQuery(queryString.toString());

            return query.getResultList();
      }

      @Override
      public boolean buscarEventoAlquilado(int idevento) {
            StringBuilder queryString = new StringBuilder(
                        "select alq from Alquiler alq where
alq.evento.idEventos = ?1");
            Query query =
getEntityManager().createQuery(queryString.toString());
            query.setParameter(1, idevento);
            if (query.getResultList().size() > 0)
                  return true;
            else
                  return false;

      }

}
```

Las interfaces implementadas para esta iteración son las siguientes:

- ResponsableDAOImplement.java
- AlquilerDAOImplement.java
- EstadoAlquilerDAOImplement.java

Siguiendo las tareas del Product Backlog, ahora implementamos las clases BackingBean, que para este Sprint son las siguientes:

- alquiler.java
- responsable.java
- estadoAlquiler.java

El siguiente código es una parte del BackingBean perteneciente al alquiler de eventos, en donde se implementa el método guardar. El código completo lo podemos revisar en el Anexo 2-Codificación.

```
public void guardar() {
            try {
                  alquiler.setIdAlquiler(alquilerI.consultarId());
                  alquiler.setEstadoAlquiler(estadoAlquiler);
                  alquiler.setEvento(eventoAlquiler);
                  alquilerI.guardar(alquiler);

                  pagos.setIdPago(pagoI.consultarId());
                  pagos.setAlquiler(alquiler);
                  pagos.setFechaPago(new Date());

                  pagoI.save(pagos);
                  borrarRastros();
                  final FacesMessage msg = new FacesMessage(
                              FacesMessage.SEVERITY_INFO, "Alquiler",
                              "Alquiler de Evento Guardado !.");
                  FacesContext.getCurrentInstance().addMessage(null,
msg);
            } catch (Exception ex) {
                  ex.printStackTrace();
                  final FacesMessage msg = new FacesMessage(
                              FacesMessage.SEVERITY_FATAL, "Alquiler",
                              "Error al Guardar.");
                  FacesContext.getCurrentInstance().addMessage(null,
msg);
            }
      }

      public void borrarRastros() {
            try {
                  this.alquiler = new Alquiler();
                  this.alquilerSelect = new Alquiler();
                  this.eventoAlquiler = new Evento();
                  this.estadoAlquiler = new EstadoAlquiler();
                  this.pagos = new Pago();
                  listaAlquiler = alquilerI.listarAlquiler();
                  listaEAlquiler = estadoalquilerI.listarEAlquiler();
                  listaAlquilerSINPago =
alquilerI.listaAlquilerSINPago();


            } catch (Exception e) {
                  e.printStackTrace();
            }
      }
```

La siguiente tarea es la creación de páginas JSF, que para este Sprint las comprenden las siguientes:

- registrarEstadoAlquiler.JSF
- registrarAlquiler.JSF
- registrarResponsable.JSF

En la Figura 2.22 podemos apreciar el prototipo del diseño de la página JSF perteneciente al registro de alquiler para eventos.

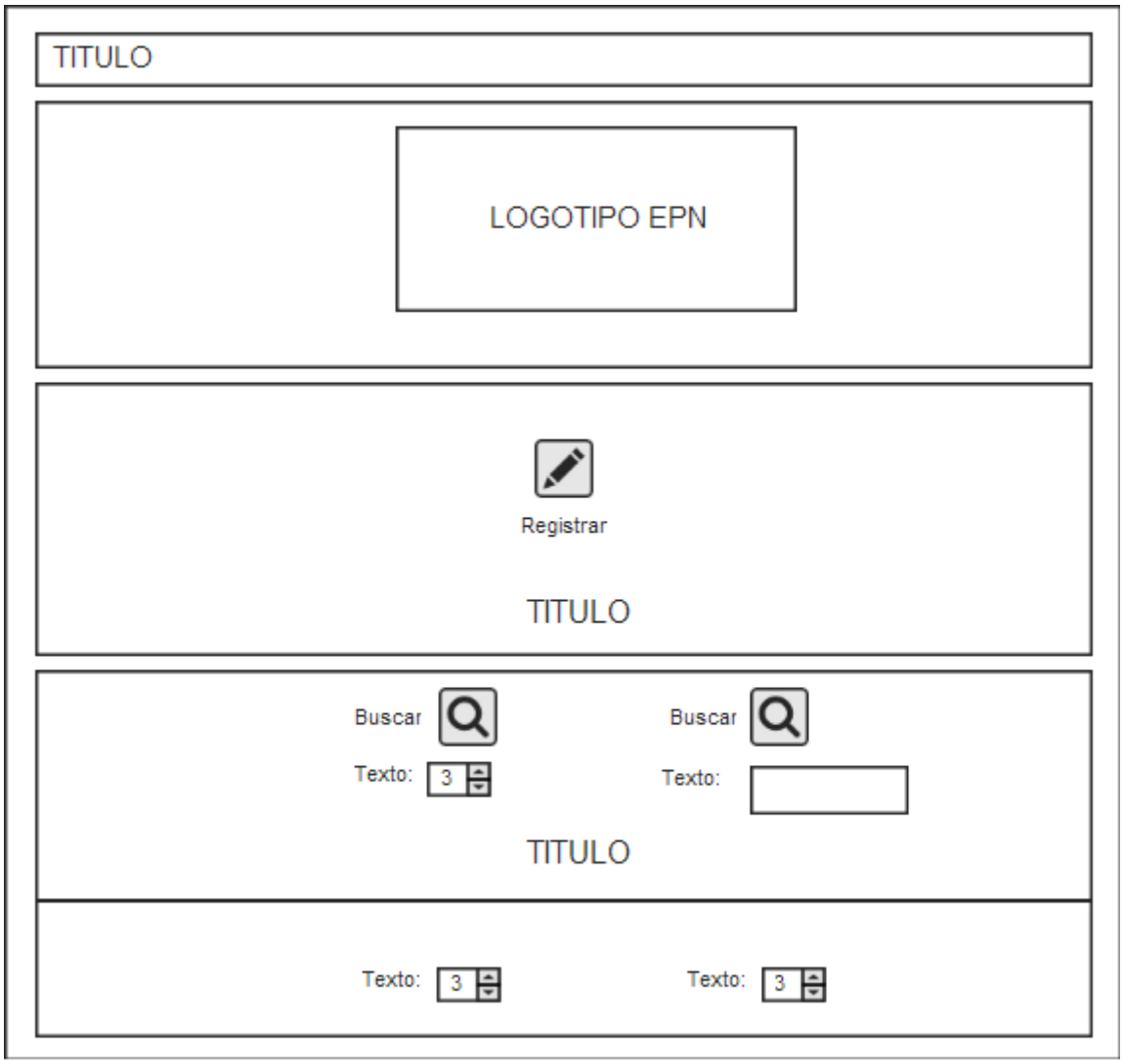


Fig. 1.38: Prototipo Página JSF Registro Alquiler para Eventos. Elaborado por los Autores.

A continuación se presenta partes del código para la elaboración de la página JSF perteneciente al registro de alquiler para eventos. El código completo lo podemos revisar en el Anexo 2-Codificación.

```
<!-- LISTADO DE ALQUILERES -->
<!-- ************************************* -->
<p:dataTable id="alquiler" width="735px"
rowsPerPageTemplate="5,10,15,20" PáginatorPosition="bottom"
PáginatorTemplate="{FirstPageLink} {PreviousPageLink} {PageLinks}
{NextPageLink} {LastPageLink} {RowsPerPageDropdown}"
Páginator="true" rows="10"
emptyMessage="No hay Alquileres Registrados"
value="#{alquilerb.listaAlquiler}" var="la">

<p:column headerText="Evento">
<h:outputText value="#{la.evento.nomEvento}" />
</p:column>
<p:column headerText="Estado Alquiler">
<h:outputText value="#{la.estadoAlquiler.estadoAlquiler}" />
</p:column>
<p:column headerText="Costo">
<h:outputText value="#{la.costoAlquiler}" />
</p:column>
<p:column headerText="Saldo">
<h:outputText value="#{la.saldoAlquiler}" />
</p:column>
<p:column headerText="Descripción">
<h:outputText value="#{la.descripcionAlquiler}" />
</p:column>
<p:column headerText="Acciones" width="60">
<p:commandLink id="btneliminarA" onclick="ei.show()"
process="@this, alquiler" update="@([id$=mesajeEliminoA])">
<h:graphicImage value="/images/eliminar24.ico"
title="Eliminar Alquiler" style="border:0px solid #CAD6E0;"
height="20" width="20">
</h:graphicImage>
<f:setPropertyActionListener
target="#{alquilerb.alquilerSelect}" value="#{la}" />
</p:commandLink>
<p:commandLink id="btnacA" onclick="pnlaa.show()"
update="@([id$=datosA])" process="@this,alquiler,btnacA">
<h:graphicImage value="/images/EDITAR2.png"
title="Editar Alquiler" style="border:0px solid #CAD6E0;"
height="20" width="20">
</h:graphicImage>
<f:setPropertyActionListener
target="#{alquilerb.alquilerSelect}" value="#{la}" />
</p:commandLink>
</p:column>
</p:dataTable>
```

Una vez terminada toda la implementación de clases, backingbeans, interfaces y páginas JSF, concluimos con el desarrollo de los reportes.

En la Figura 2.23 se puede apreciar el prototipo de la pantalla para presentar los reportes del sistema, cabe mencionar que existen tres tipos de reportes pero a continuación se muestra el prototipo del reporte de espacios.

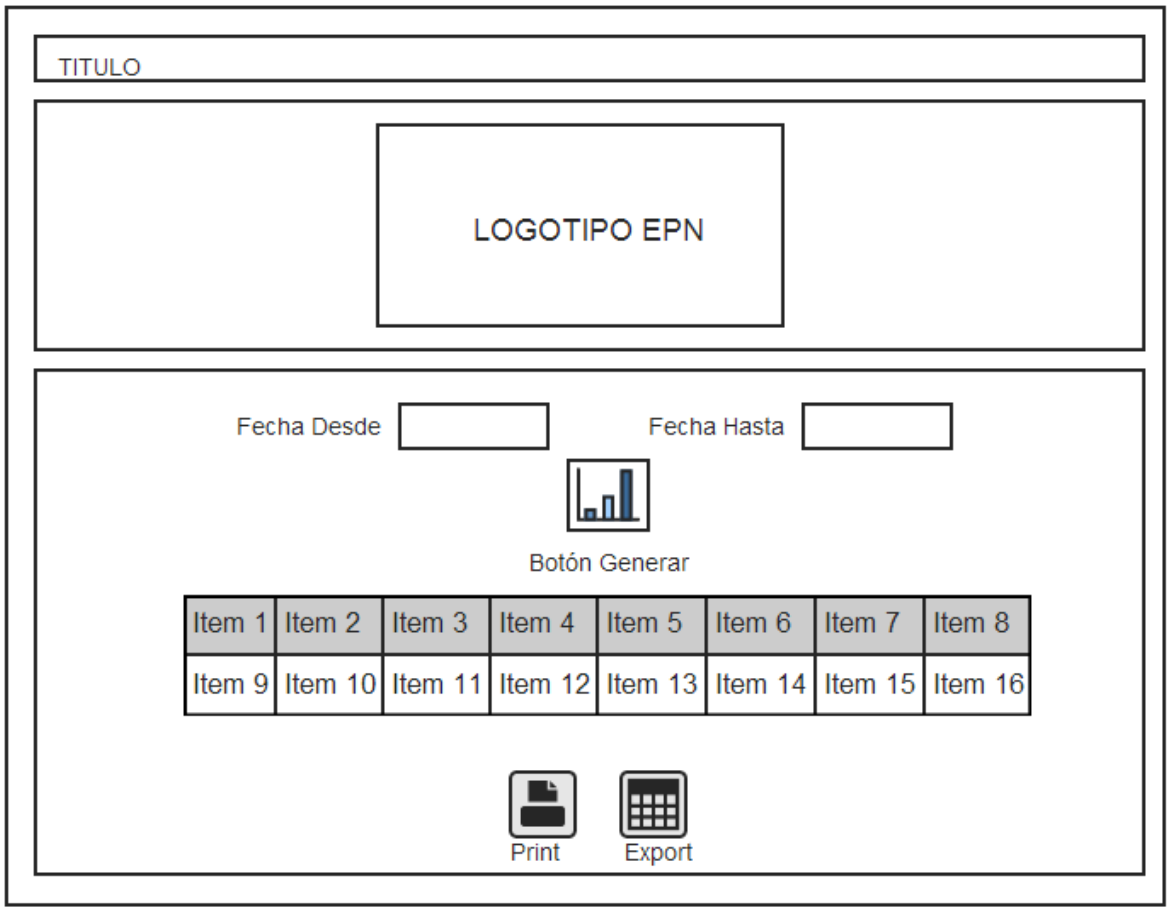


Fig. 1.39: Prototipo Página JSF Reportes. Elaborado por los Autores.

A continuación se presenta parte del código de la página JSF perteneciente a los reportes de espacios. El código completo lo podemos revisar en el Anexo 2-Codificación.

```
<div align="center">
```

```
                <p:panel>
                    <h:panelGrid columns="4" cellpadding="4"
cellspacing="4"  id="buscarEspacio">
                        <h:outputText value="Espacios: "
                            style="FONT-FAMILY: arial; FONT-SIZE: 10px;
FONT-WEIGHT: bold; " />
                        <p:selectOneMenu id="esp"
                           value="#{espacioBack.parametroReporteEspacio}"
                            style="FONT-FAMILY: arial; width:240px;"
                            requiredMessage="Seleccionar un Parámetro">
                            <f:selectItem itemLabel="Por favor
seleccione" itemValue="" />
                            <f:selectItem itemLabel="Con Evento"
itemValue="evento"          style="font-size:10px;" />
                            <f:selectItem itemLabel="Con Ocupante"
itemValue="ocupante"    style="font-size:10px;" />
                        </p:selectOneMenu>
                    </h:panelGrid>
                    <br />
                    <p:commandLink process="@this, buscarEspacio"
                        update="@([id$=espacios])"
                        actionListener="#{espacioBack.reporteEspacios}">
                        <div align="center">
                            <h:graphicImage width="50" height="50"
value="/images/report.png"
                                title="Generar Reporte">
                            </h:graphicImage>
                        </div>
                        <center>
                            <h:outputText
                                style="FONT-FAMILY: arial; FONT-SIZE:
10px; FONT-WEIGHT: bold;"
                                value="Generar" />
                        </center>
                    </p:commandLink>
                    <p:separator />
                    <p:dataTable id="espacios" width="735px"
                        rowsPerPageTemplate="5,10,15,20"
PáginatorPosition="bottom"
                        PáginatorTemplate="{FirstPageLink}
{PreviousPageLink} {PageLinks} {NextPageLink} {LastPageLink}
{RowsPerPageDropdown}"
                        Páginator="true" rows="10"
                        tableStyle="FONT-FAMILY: arial; FONT-SIZE: 10px;"
                        emptyMessage="No hay Espacios Registrados"
                        value="#{espacioBack.reporteEspacio}"
var="lpisoxEdificio">
                        <f:facet name="header">
                            <h:outputText value="Lista de espacios en el
Piso" />
                        </f:facet>
                        <p:column headerText="Espacio">
                            <h:outputText
value="#{lpisoxEdificio.nomEspacio}" />
                        </p:column>
                        <p:column headerText="Descripción">
                            <h:outputText
value="#{lpisoxEdificio.descripcionEspacio}" />
                        </p:column>
                        <p:column headerText="Referencia en Plano">
```

```
                              <h:outputText
value="#{lpisoxEdificio.referencia}" />
                           </p:column>
                           <p:column headerText="Tipo">
                              <h:outputText

value="#{lpisoxEdificio.tipoEspacio.nomTipoEspacio}" />
                           </p:column>

                        </p:dataTable>
                     </p:panel>
                  </div>
```

2.3.5 EJECUCIÓN DEL QUINTO SPRINT

La ejecución de Quinto Sprint se centra en la puesta en marcha e instalación de nuestro sistema, para lo cual se debe realizar una recopilación de datos de inicio, cargar dichos datos y ver el funcionamiento del sistema para su respectiva evaluación de resultado.

Los datos de inicio serán únicamente los necesarios para el funcionamiento del sistema y deberán ser ingresados en la base de datos por un administrador con acceso a dichas actividades de registro.

En la Tabla 2.8 se muestran las principales tablas en la base de datos que necesitan datos de inicio:

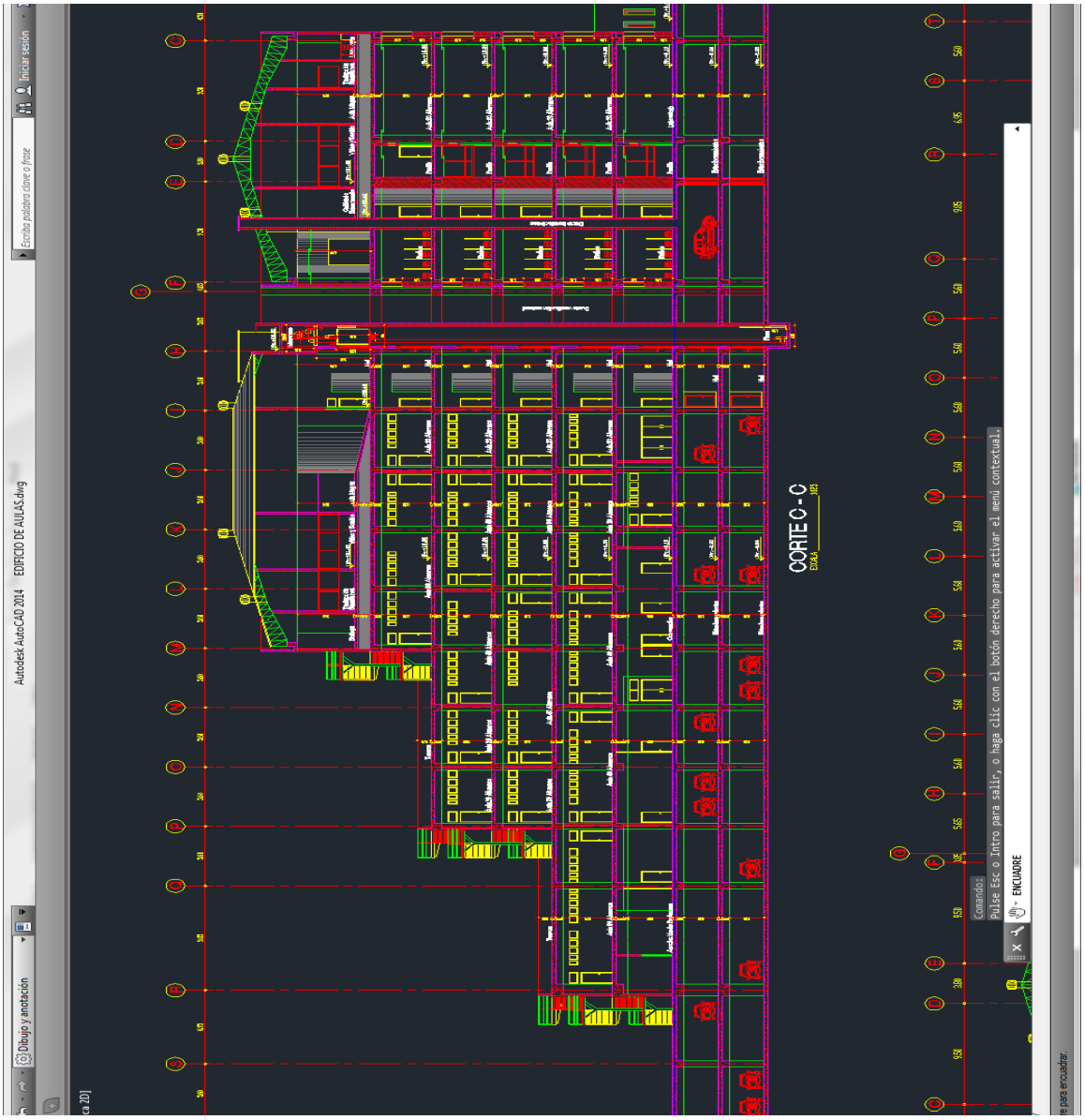
Tabla en la base de datos	Información	Observaciones
Tipo de Evento	Interno Externo	Interno por medio de Memo. Externo con número de factura
Estado de Evento	Reservado Confirmado Iniciado Terminado	
Tipo de Espacio	Aula Oficina Bodega Salas de Lectura	Los tipos de espacio dependerán de los administradores de cada edificio añadiendo los que sean necesarios
Estado de Espacio	Libre	

	Ocupado	
Estado de Alquiler	Pagado Con deuda	

Tabla 2. 8: Tablas en la Base de Datos necesarias para inicio del Sistema. Elaborado por los Autores.

La carga de datos de inicio se basará en la recopilación de imágenes de los diferentes planos de cada piso de los edificios del campus, para lo cual se ha trabajado conjuntamente con la Dirección de Planificación de la EPN para poder obtener los archivos más actuales de los planos arquitectónicos de la EPN, dichos archivos se ejecutarán bajo la plataforma de Autodesk Autocad 2014 para la extracción de las imágenes de los diferentes pisos, por medio de la extracción de podemos cargar las gráficas al sistema y usarlas para la visualización y marcaje de espacios.

En la Figura 2.24 se muestra el edificio EARME en una vista lateral con ayuda de Autocad, donde podemos observar el número de pisos y las diferentes estructuras que contiene el edificio.



Para la extracción de los planos se procede a exportar dichos planos a formato pdf, para obtener las imágenes que luego serán usadas en el Sistema Web. En la Figura 2.25 se muestra la anterior vista del edificio EARME, pero ahora exportado

Fig. 1.40: Plano del Edifico EARME - Vista Lateral. Elaborado por los Autores.

a pdf.

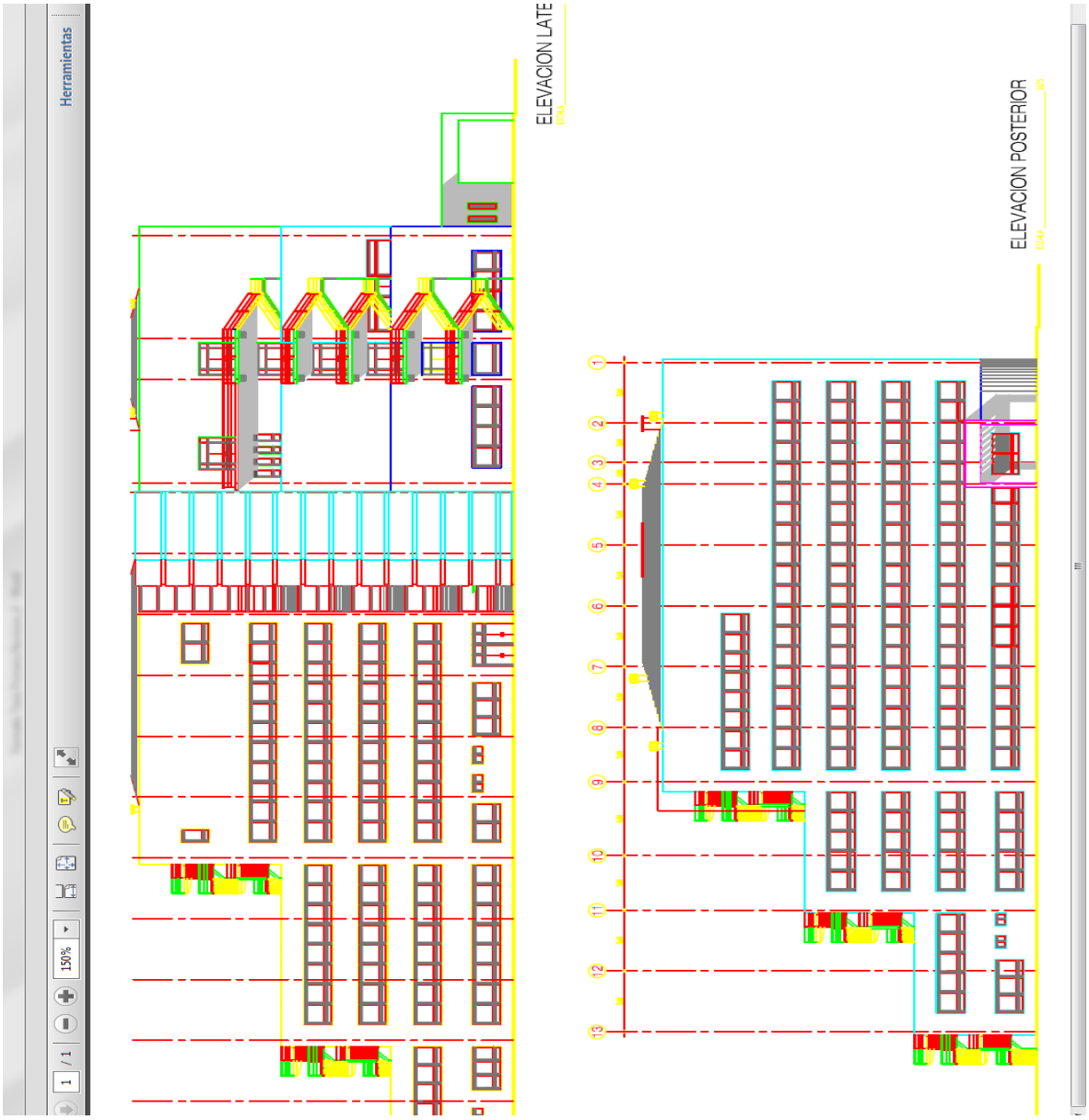


Fig. 1.41: Plano del Edifico EARME - Vista Lateral - formato pdf. Elaborado por los Autores.

Con ayuda del archivo exportado a formato pdf, se puede limitar las observaciones del plano, lo cual significa que nosotros podemos establecer que se muestra y que no en la imagen que se utilizará posteriormente para la ubicación de espacios.

En la Figura 2.26 se puede observar las diferentes opciones que tiene al archivo para mostrar en el plano, las casillas que se eliminarán serán aquellas que no brinden aporte a la visualización del plano como son: medidas, cuadriculado, paneles, dimensiones, etc.

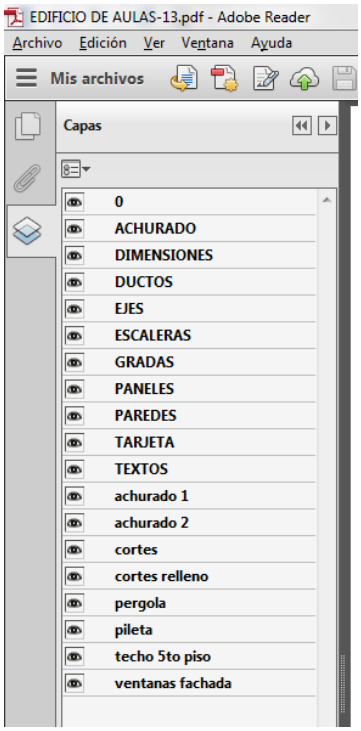


Fig. 1.42: Opciones de visualización para plano. Elaborado por los Autores.

Una vez establecidas las opciones u objetos que se mostrarán en el plano, se procede a la captura y guardado de la imagen desde pdf para así obtener el plano en formato png o jpg que se usará posteriormente en la carga del plano para el uso en el Sistema.

La Fig. 2.27 muestra el plano procesado sin objetos innecesarios para el uso en el Sistema.

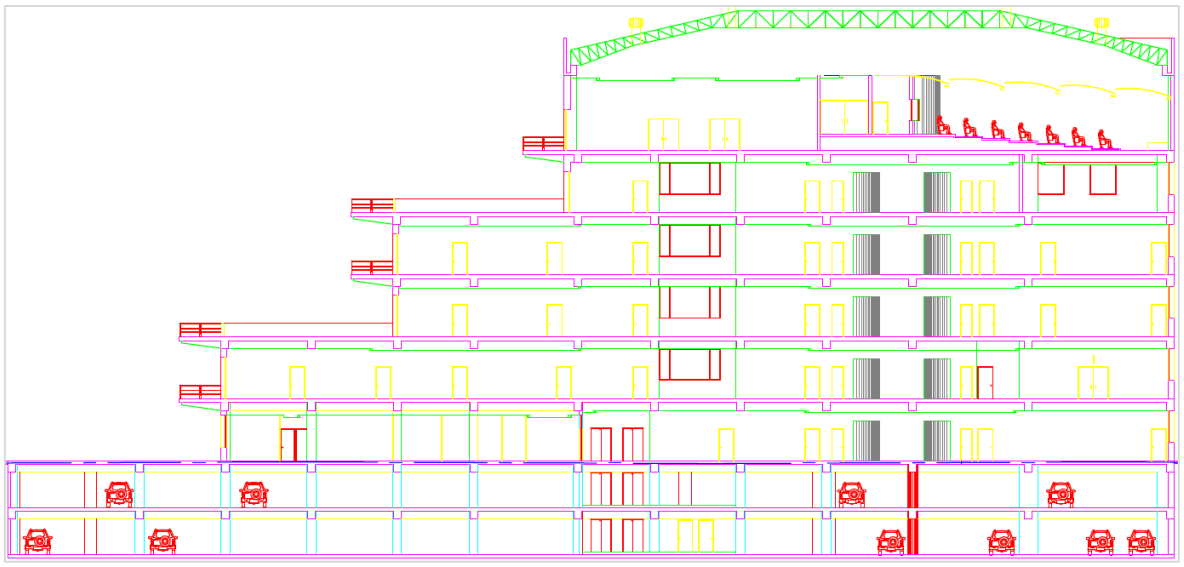


Fig. 1.43: Plano procesado sin objetos. Elaborado por los Autores.

Todo el proceso de captura de imágenes se debe realizará para todos y cada uno de los pisos de los edificios de la EPN.

2.4 ENTREGA DE LOS SPRINTS

2.4.1 PRIMER SPRINT

El Primer Sprint se desarrolla con normalidad, pudiendo así completar cada una de las tareas asignadas en el tiempo establecido y fechas acordadas. En la Figura. 2.28 podemos apreciar con estado terminado a las tareas establecidas.

SPRINT	INICIO	DURACIÓN
1	17-mar.-14	7

Tareas pendientes
Horas de trabajo pendientes

PILA DEL SPRINT

Backlog	Tarea	Tipo	Estado	Responsable
1	Diseño de Diagrama Entidad-Relacion	Análisis	Terminada	Leonardo, Wilson
2	Generacion modelo Fisico	Codificación	Terminada	Leonardo, Wilson
3	Generacion codigo SQL para PostgreSQL	Prototipado	Terminada	Leonardo, Wilson
4	Definición Roles y Perfiles de Usuario	Codificación	Terminada	Leonardo, Wilson
5	Refinamiento y consistencia de la Base de Datos	Análisis	Terminada	Leonardo, Wilson
6	Control de calidad	Codificación	Terminada	Geovanna
7	Afinamiento de la base de datos	Codificación	Terminada	Leonardo, Wilson

Fig. 1.44: Cumplimiento de Tareas-Primer Sprint. Elaborado por los Autores.

El resultado final de esta iteración es la generación de Scripts de la base de datos, los mismos que posteriormente son utilizados en la creación de la base de datos en la herramienta PostgreSQL.

En la Figura 2.29 se muestra el esquema creado en la base de datos con el nombre Infraestructura.

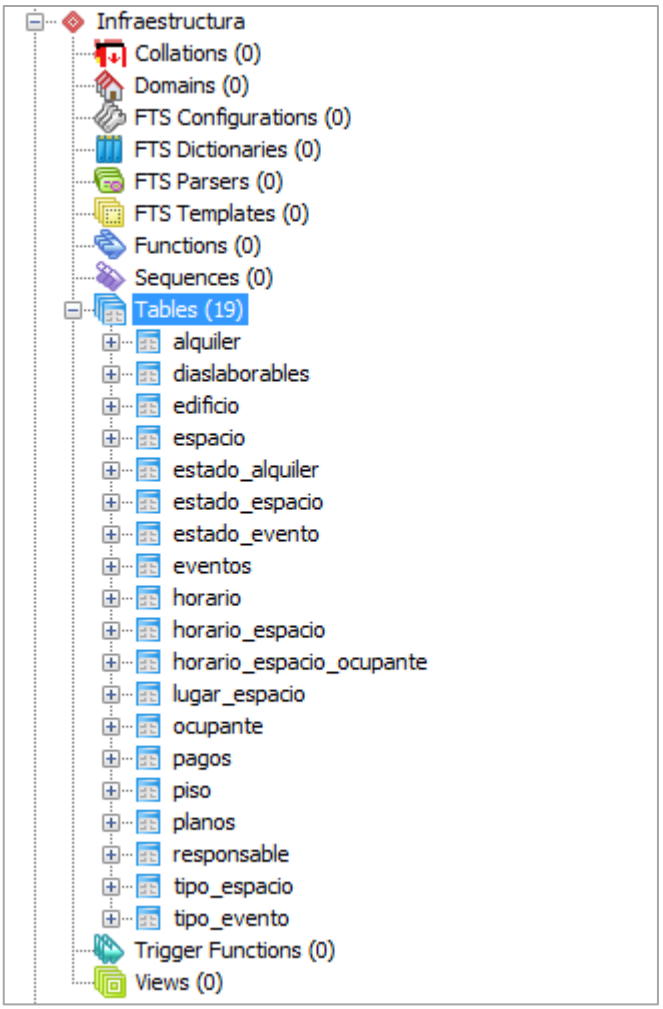


Fig. 1.45: Esquema BDD Infraestructura-PostgreSQL. Elaborado por los Autores

2.4.2 SEGUNDO SPRINT

En la Figura. 2.30 podemos apreciar que también se consigue culminar con todas las tareas asignadas para este Sprint.

SPRINT	INICIO	DURACION
2	26-mar.-14	20

Tareas pendientes
Horas de trabajo pendientes

PILA DEL SPRINT

Backlog ID	Tarea	Tipo	Estado	Responsable
8	Creación de entidades desde base de datos.	Codificación	Terminada	Leonardo, Wilson
9	Creación e implementación de interfaz edificio (CRUD).	Codificación	Terminada	Leonardo, Wilson
10	Creación backingbean edificio.	Codificación	Terminada	Leonardo, Wilson
11	Creación página jsf edificio.	Codificación	Terminada	Leonardo, Wilson
12	Creación e implementación de interfaz plano(CRUD).	Codificación	Terminada	Leonardo, Wilson
13	Creación backingbean plano.	Codificación	Terminada	Leonardo, Wilson
14	Creación página jsf plano.	Codificación	Terminada	Leonardo, Wilson
15	Creación e implementación de interfaz piso(CRUD).	Codificación	Terminada	Leonardo, Wilson
16	Creación backingbean piso.	Codificación	Terminada	Leonardo, Wilson
17	Creación página jsf piso.	Codificación	Terminada	Leonardo, Wilson
18	Creación e implementación de interfaz lugar_espacio(CRUD	Codificación	Terminada	Leonardo, Wilson
19	Creación backingbean lugar_espacio.	Codificación	Terminada	Leonardo, Wilson
20	Creación página jsf lugar_espacio.	Codificación	Terminada	Leonardo, Wilson
21	Creación e implementación de interfaz espacio(CRUD).	Codificación	Terminada	Leonardo, Wilson
22	Creación backingbean espacio.	Codificación	Terminada	Leonardo, Wilson
23	Creación página jsf espacio.	Codificación	Terminada	Leonardo, Wilson
24	Creación e implementación de interfaz tipo espacio(CRUD).	Codificación	Terminada	Leonardo, Wilson
25	Creación backingbean tipo espacio.	Codificación	Terminada	Leonardo, Wilson
26	Creación página jsf tipo espacio.	Codificación	Terminada	Leonardo, Wilson
27	Acoplamineto de planos en paginas jsf	Codificación	Terminada	Leonardo, Wilson
28	Marcado de espacios de plano en jsf	Codificación	Terminada	Leonardo, Wilson
29	Pruebas unitarias TDD	Pruebas	Terminada	Leonardo, Wilson
30	Control de calidad	Reunión	Terminada	Sandra
31	Afinamiento	Codificación	Terminada	Leonardo, Wilson

Fig. 1.46: Cumplimiento de Tareas-Segundo Sprint. Elaborado por los Autores.

Como parte de la presentación de los entregables al funcional, se presenta el módulo para la historia de usuario carga de planos arquitectónicos la cual es mostrada en la Figura 2.31.

ESCUELA POLITECNICA NACIONAL
Campus José Rubén Orellana Ricaurte

Guardar

Ingreso de Nuevos Planos

Nombres del Plano: Servicios Generales

Descripcion: Planta Baja-Taller Servicios Genrales

Buscar Plano:

Visualización del Nuevo Plano

Plano Arquitectónico

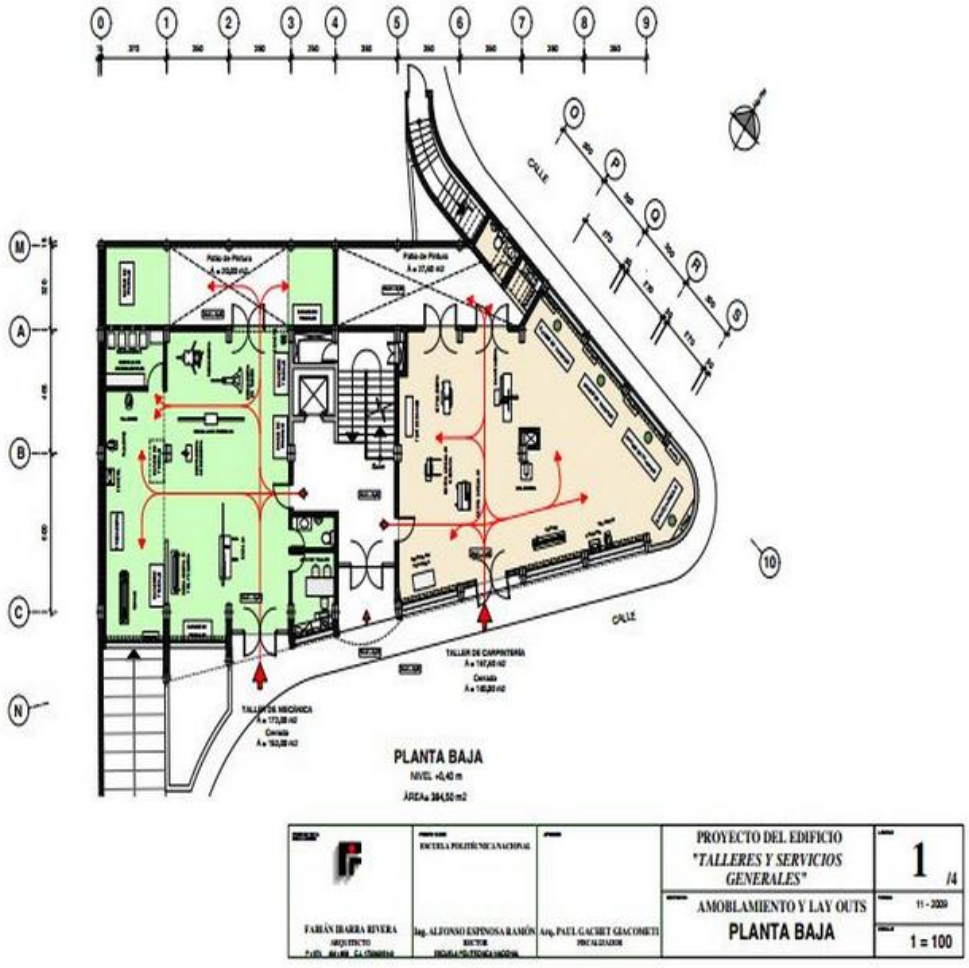


Fig. 1.47: Página JSF, Registro de Planos. Elaborado por los Autores.

También se presenta el módulo para la historia de usuario Ubicación de espacios en plano, Figura. 2.32 y Figura. 2.33.

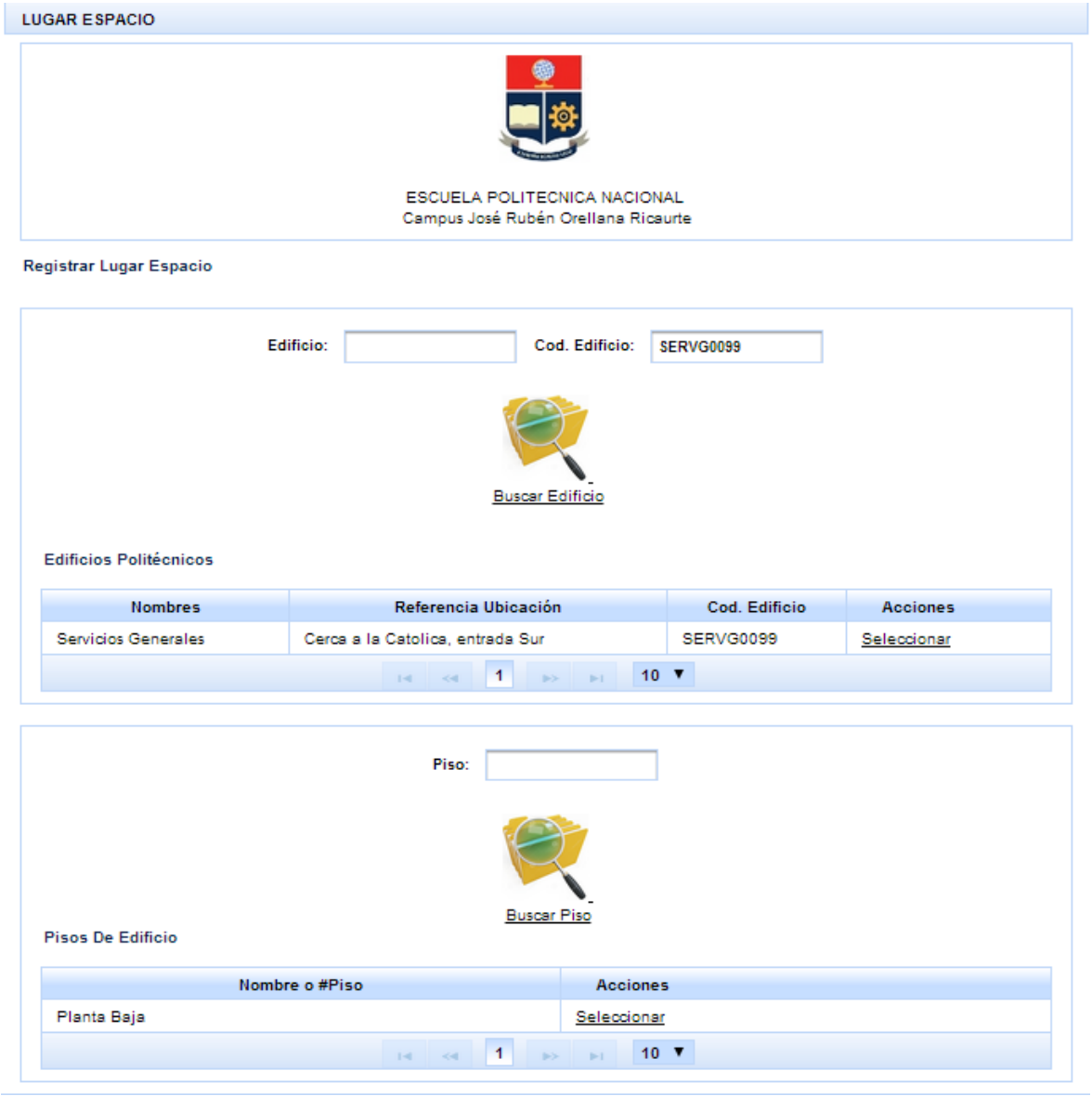


Fig. 1.48: Página JSF, Registro de Lugar Espacio. Elaborado por los Autores.

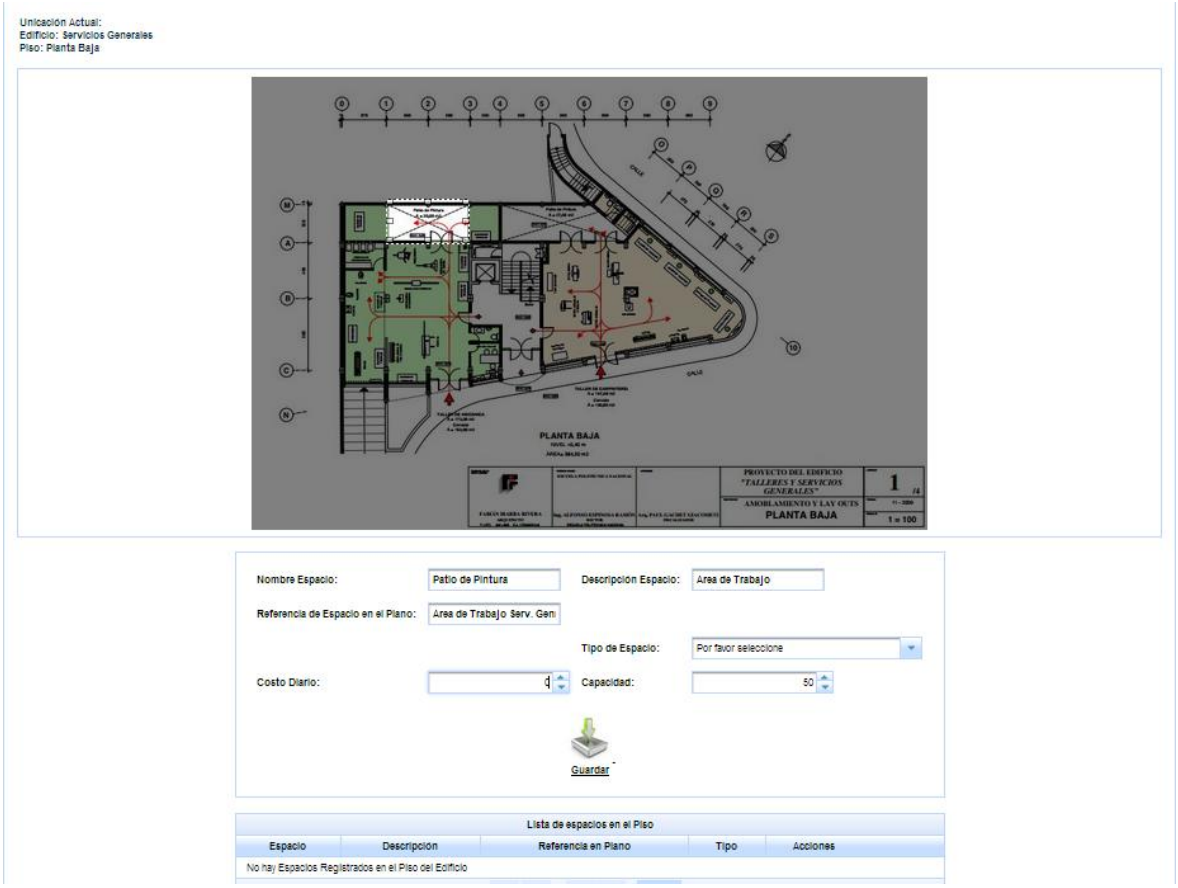


Fig. 1.49: Página JSF, Registro de Espacio. Elaborado por los Autores.

Los casos de prueba para de esta iteración se presentan en el Anexo 3-Casos de Prueba. Como ejemplo se presenta el caso de prueba perteneciente al registro de un nuevo plano Tabla 1.9.

Caso de Prueba	
Número caso de Prueba: 1	**Historia de Usuario:** Carga de planos arquitectónicos
Nombre: Registrar nuevo plano	
Descripción: Se probará que los datos ingresados y la imagen del plano, se guarden correctamente.	
Precondiciones: El usuario con perfil Administrador Espacios ingresa al sistema.	
Entrada: • El usuario ingresa al sistema y selecciona el menú Infraestructura, dentro la opción Ingresar Planos. • El usuario ingresa el nombre y descripción del plano. • El usuario presiona en el botón Buscar Plano. • El usuario elije una imagen en formato png, jpg, gif o jpeg.	

• Se despliega una pre visualización del plano a ingresar. • El usuario presiona en la imagen Guardar.
Resultado esperado: • Se muestra un mensaje exitoso. • Se guarda en la base de datos.
Resultado obtenido: Cumple satisfactoriamente lo esperado.

Tabla 2. 9: Historia de Usuario: Carga de planos arquitectónicos. Elaborado por los Autores.

2.4.3 TERCER SPRINT

El Tercer Sprint tampoco presenta ningún inconveniente en su culminación, En la Figura 2.34 se aprecia las tareas terminadas.

		SPRINT	INICIO	DURACIÓN		
		3	24-abr.-14	20		
						Tareas pendientes
						Horas de trabajo pendientes
		PILA DEL SPRINT				
Backlog ID	Tarea			Tipo	Estado	Responsable
32	Creación e implementación de interfaz horario (CRUD).			Prototipado	Terminada	Leonardo, Wilson
33	Creación backingbean horario.			Prototipado	Terminada	Leonardo, Wilson
34	Creación página jsf horario.			Codificación	Terminada	Leonardo, Wilson
35	Creación e implementación de interfaz horario_espacio (CRUI			Codificación	Terminada	Leonardo, Wilson
36	Creación backingbean horario_espacio.			Codificación	Terminada	Leonardo, Wilson
37	Creación página jsf horario_espacio.			Codificación	Terminada	Leonardo, Wilson
38	Creación e implementación de interfaz estados de espacio (CI			Codificación	Terminada	Leonardo, Wilson
39	Creación backingbean estados de espacio.			Codificación	Terminada	Leonardo, Wilson
40	Creación página jsf estados de espacio.			Codificación	Terminada	Leonardo, Wilson
41	Creación e implementación de interfaz ocupante (CRUD).			Codificación	Terminada	Leonardo, Wilson
42	Creación backingbean ocupante.			Codificación	Terminada	Leonardo, Wilson
43	Creación página jsf ocupante.			Codificación	Terminada	Leonardo, Wilson
44	Creación e implementación de interfaz evento (CRUD).			Codificación	Terminada	Leonardo, Wilson
45	Creación backingbean evento.			Codificación	Terminada	Leonardo, Wilson
46	Creación página jsf evento.			Codificación	Terminada	Leonardo, Wilson
47	Creación e implementación de interfaz estado de evento (CRU			Codificación	Terminada	Leonardo, Wilson
48	Creación backingbean estado de evento.			Codificación	Terminada	Leonardo, Wilson
49	Creación página jsf estado de evento.			Codificación	Terminada	Leonardo, Wilson
50	Creación e implementación de interfaz tipo de evento (CRUD).			Codificación	Terminada	Leonardo, Wilson
51	Creación backingbean tipo de evento.			Codificación	Terminada	Leonardo, Wilson
52	Creación página jsf tipo de evento.			Codificación	Terminada	Leonardo, Wilson
53	Pruebas unitarias TDD			Pruebas	Terminada	Leonardo, Wilson
54	Control de calidad			Reunión	Terminada	Sandra
55	Afinamiento			Codificación	Terminada	Leonardo, Wilson

Fig. 1.50: Cumplimiento de Tareas-Tercer Sprint. Elaborado por los Autores.

Como parte de la presentación de los entregables al funcional, se presenta el módulo para la historia de usuario Asignación de Eventos la cual es mostrada En la Figura 2.35.

Fig. 1.51: Página JSF, Registro de Eventos. Elaborado por los Autores.

También se presenta el módulo para la historia de usuario Asignación de espacios a ocupantes, Fig. 2.36.

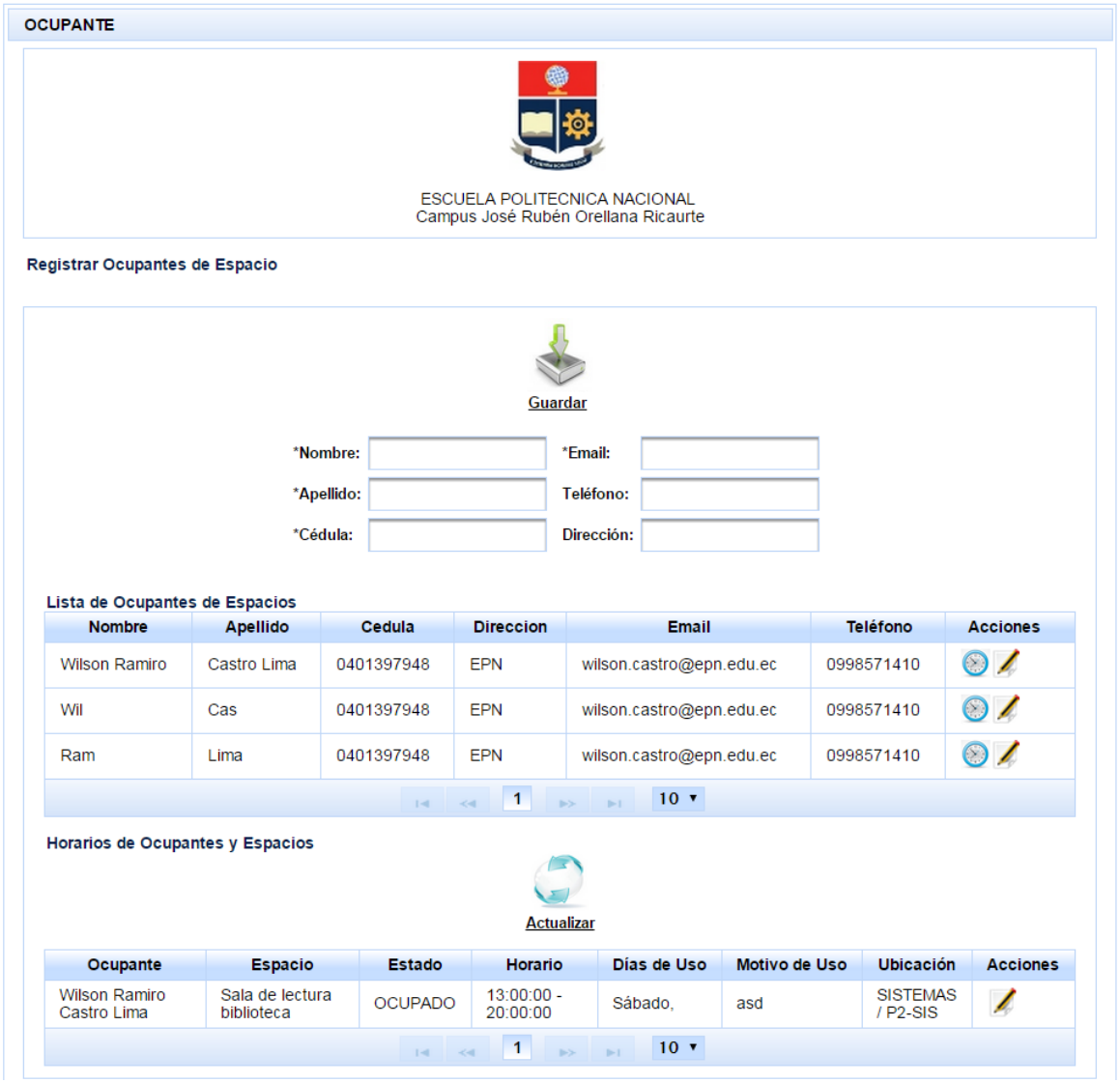


Fig. 1.52: Página JSF, Registro de Ocupantes. Elaborado por los Autores.

Los casos de prueba para de esta iteración se presentan en el Anexo 3-Casos de Prueba. Como ejemplo se presenta el caso de prueba perteneciente al registro de un nuevo ocupante Tabla 1.10.

Caso de Prueba	
Número caso de Prueba: 24	**Historia de Usuario:** Asignación de espacios a ocupantes
Nombre: Registrar nuevo Ocupante	
Descripción: Se probará el registro de un nuevo Ocupante.	
Precondiciones: El usuario con perfil Administrador Espacios ingresa al sistema.	

Entrada:
• El usuario ingresa al sistema y selecciona el menú Ocupantes y Horarios, dentro la opción Registro Ocupante.
• El usuario ingresa información del Ocupante:
  o Nombre: con caracteres alfanuméricos de hasta 200.
  o Email: con caracteres alfanuméricos de hasta 50, con las características necesarios de ingresar con el símbolo "@" y la terminación ".com" o ".ec".
  o Apellido: con caracteres alfanuméricos de hasta 200.
  o Teléfono: con caracteres numéricos máximo de 10 dígitos.
  o Cedula: con caracteres numéricos de máximo 10 dígitos imponiendo un control de cédula si el usuario ingresa mal el dato.
  o Dirección: con caracteres alfanuméricos de hasta 200
• El usuario presiona en la imagen Guardar.

Resultado esperado: • Se muestra un mensaje exitoso. • Se guarda en la base de datos.
Resultado obtenido: Cumple satisfactoriamente lo esperado.

Tabla 2. 10: Historia de Usuario: Asignación de espacios a ocupantes. Elaborado por los Autores.

2.4.4 CUARTO SPRINT

El Cuarto Sprint no presenta ningún percance, teniendo prácticamente finalizadas todas las funcionalidades del sistema. La Figura. 2.37 muestras las tareas terminadas para este Sprint.

SPRINT	INICIO	DURACION
4	23-may.-14	18

Tareas pendientes
Horas de trabajo pendientes

PILA DEL SPRINT

Backlog ID	Tarea	Tipo	Estado	Responsable
56	Creación e implementación de interfaz responsable de evento (CRUD).	Codificació	Terminada	Leonardo, Wilson
57	Creación backingbean responsable de evento.	Codificació	Terminada	Leonardo, Wilson
58	Creación página jsf responsable de evento.	Codificació	Terminada	Leonardo, Wilson
59	Creación e implementación de interfaz alquiler (CRUD).	Codificació	Terminada	Leonardo, Wilson
60	Creación backingbean alquiler.	Codificació	Terminada	Leonardo, Wilson
61	Creación página jsf alquiler.	Codificació	Terminada	Leonardo, Wilson
62	Creación e implementación de interfaz estado alquiler (CRUD).	Codificació	Terminada	Leonardo, Wilson
63	Creación backingbean estado alquiler.	Codificació	Terminada	Leonardo, Wilson
64	Creación página jsf estado alquiler.	Codificació	Terminada	Leonardo, Wilson
65	Creación e implementación de interfaz pagos alquiler (CRUD).	Codificació	Terminada	Leonardo, Wilson
66	Creación backingbean pagos alquiler.	Codificació	Terminada	Leonardo, Wilson
67	Creación página jsf pagos alquiler.	Codificació	Terminada	Leonardo, Wilson
68	Reportes de planos	Codificació	Terminada	Leonardo, Wilson
69	Reportes de horarios	Codificació	Terminada	Leonardo, Wilson
70	Reportes de eventos	Codificació	Terminada	Leonardo, Wilson
71	Reportes de espacios	Codificació	Terminada	Leonardo, Wilson
72	Reportes de ocupantes	Codificació	Terminada	Leonardo, Wilson
73	Reportes de responsables de evento	Codificació	Terminada	Leonardo, Wilson
74	Reportes de alquiler de eventos	Codificació	Terminada	Leonardo, Wilson
75	Pruebas unitarias TDD	Codificació	Terminada	Leonardo, Wilson
76	Control de calidad	Codificació	Terminada	Sandra
77	Afinamiento	Codificació	Terminada	Leonardo, Wilson

Fig. 1.53: Cumplimiento de Tareas-Cuarto Sprint. Elaborado por los Autores.

Como parte de la presentación de los entregables al funcional, se presenta el módulo para la historia de usuario Alquiler de espacios para eventos la cual es mostrada en la Figura 2.38.

Fig. 1.54: Página JSF, Registro de Alquiler de Eventos. Elaborado por los Autores.

También se presenta la página JSF para generar reportes para los Espacios Físicos de la EPN. Fig. 2.39.

Fig. 1.55: Página JSF, Reporte de Espacios Físicos. Elaborado por los Autores.

Los casos de prueba para de esta iteración se presentan en el Anexo 3-Casos de Prueba. Como ejemplo se presenta el caso de prueba perteneciente al registro de un nuevo alquiler de evento Tabla 2.11 y el de generación de reportes Tabla 2.12.

Caso de Prueba	
Número caso de Prueba: 57	**Historia de Usuario:** Alquiler de espacios para Eventos
Nombre: Registrar nuevo Alquiler de Evento	
Descripción: Se probará el registro de un nuevo Alquiler de Evento.	
Precondiciones: El usuario con perfil Administrador Espacios ingresa al sistema.	
Entrada: • El usuario ingresa al sistema y selecciona el menú Alquileres, dentro la opción Registro Alquiler Evento. • El usuario ingresa el Estado de Alquiler, un Costo con valores decimales mayores a cero, el valor a pagar mayor a cero y menor o igual al costo y una Descripción. El costo del pago del alquiler se genera automáticamente por medio del sistema, pero cuando se genera un valor cero de costo el	

administrador debe establecer un costo definido para el alquiler. • El usuario presiona en la imagen Guardar.
Resultado esperado: • Se muestra un mensaje exitoso. • Se guarda en la base de datos.
Resultado obtenido: Cumple satisfactoriamente lo esperado.

Tabla 2. 11: Historia de Usuario: Alquiler de espacios para Eventos. Elaborado por los Autores.

Caso de Prueba	
Número caso de Prueba: 66	**Historia de Usuario:** Elaboración de reportes
Nombre: Reporte de Espacios.	
Descripción: Se probará la generación de reportes pertenecientes a espacios.	
Precondiciones: El usuario con perfil Administrador Espacios ingresa al sistema.	
Entrada: • El usuario ingresa al sistema y selecciona el menú Reportes, dentro la opción Espacios. • El usuario puede seleccionar un criterio para generar el reporte como espacios con ocupantes o espacios con eventos. • El usuario presiona en la imagen Generar. • Se despliega el reporte de acuerdo al criterio seleccionado anteriormente, mostrando así las características como: nombre, descripción, referencia de ubicación el plano arquitectónico y el tipo al cual pertenece el espacio. • El usuario pude exportar el reporte a un archivo Excel o imprimir el mismo.	
Resultado esperado: • Se muestra el reporte generado. • Se pude imprimir o exportar a Excel.	
Resultado obtenido: Cumple satisfactoriamente lo esperado.	

Tabla 2. 12: Historia de Usuario: Elaboración de reportes. Elaborado por los Autores.

2.4.5 QUINTO SPRINT

Con el sistema terminado, este sprint tampoco presenta ningún problema en su desarrollo, en la Figura. 2.40 muestras las tareas terminadas para este Sprint.

		SPRINT	INICIO	DURACIÓN		
		5	18-jun.-14	20		
						Tareas pendientes
						Horas de trabajo pendientes
PILA DEL SPRINT						
Backlog ID	Tarea			Tipo	Estado	Responsable
78	Recopilación de Datos de Inicio			Pruebas	Terminada	Leonardo, Wilson
79	Instalación del Sistema			Pruebas	Terminada	Leonardo, Wilson
80	Carga de Datos y Clientilización			Pruebas	Terminada	Leonardo, Wilson
81	Análisis de los Resultados			Pruebas	Terminada	Leonardo, Wilson
82	Prueba Final del Sistema			Pruebas	Terminada	Sandra
83	Control de calidad			Pruebas	Terminada	Sandra
84	Afinamiento			Codificación	Terminada	Leonardo, Wilson

Fig. 1.56: Cumplimiento de Tareas-Quinto Sprint. Elaborado por los Autores.

El desarrollo de este sprint se lo puede apreciar de forma más detallada en el Capítulo 3.

3 CAPÍTULO 3: INSTALACIÓN Y PUESTA EN MARCHA

En el ciclo de vida del Desarrollo del Sistema se han usado diversas herramientas, lenguajes de programación, bases de datos y diversas herramientas para lograr tener el producto final, por lo cual, para la instalación del Sistema necesitaremos de las siguientes características en el computador que deseemos instalarlo.

- Conexión a internet.
- Servidor Web.
- Sistema gestor de bases de datos (recomendado postgresql 8.4 o mayores).
- Máquina virtual java.
- Infraestructura de redes.

Para iniciar con el procedimiento se necesita saber que el Sistema no es instalable en cualquier ambiente, sino que, se necesita de una máquina virtual java, PorstgreSQL y un Servidor Web JBoss, una vez con estos requisitos primordiales, es necesario seguir los siguientes procedimientos:

A. Restauración de los esquemas en la base de datos del Servidor Web.

Es necesario tener instalado PostgreSQL, para posteriormente restaurar los backups del sistema.

En la Figura 3.1 se muestran los esquemas public e infraestructura los cuales serán utilizados para el funcionamiento del sistema.

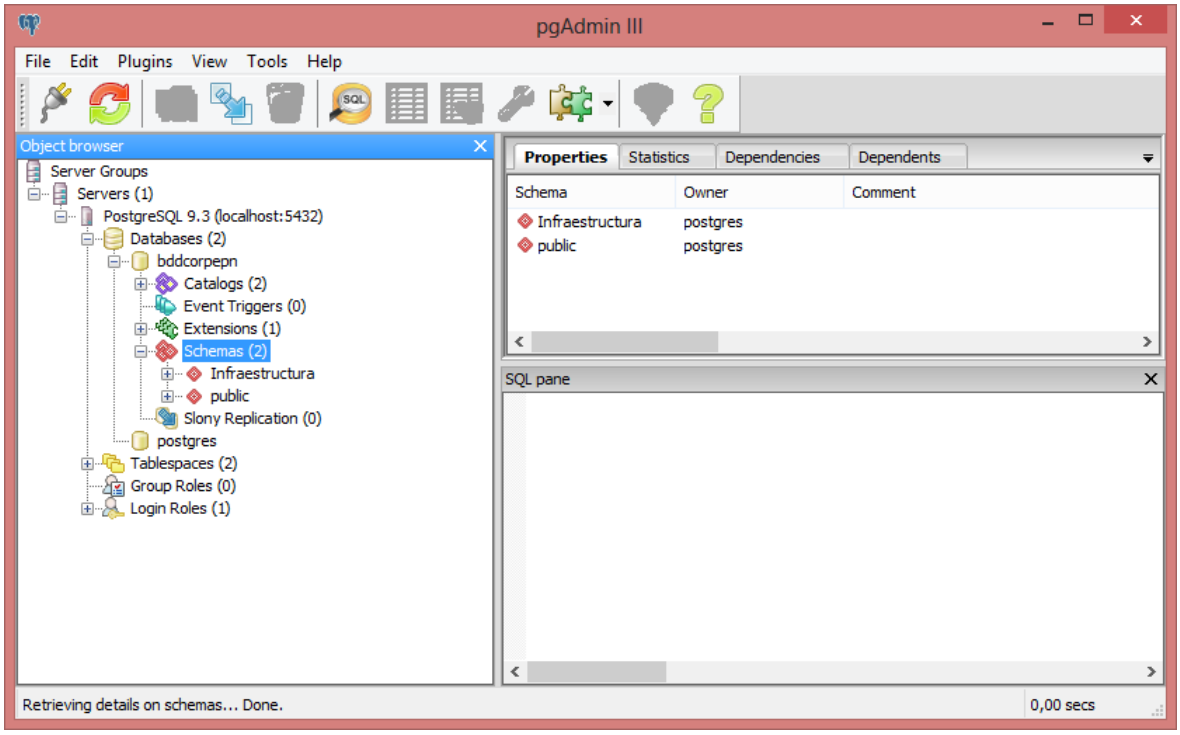


Fig. 1.57: Esquemas en la base de datos. Elaborado por los Autores.

B. Una vez completada la instalación de la base de datos, seguimos con la el archivo de extensión ".war" de nuestro proyecto web, el cual contiene el Sistema desarrollado. Para lo cual nos dirigimos a nuestro servidor y colocamos el archivo con formato ".war" en la carpeta de proyectos desarrollados del servidor (deployments).

En la Figura 3.2 se muestra el servidor con los archivos ".war" implementados en el mismo. Entre los archivos se puede observar el driver de conexión para la base de datos, un Cas para autentificación y control de seguridad y un ".war" donde se ubican los servicios y EJB del Sistema, cabe recalcar que los EJB son todos los servicios que se prestan por parte del sistema, dichos servicios se alojan en el archivo ServicioInfraestructura.jar

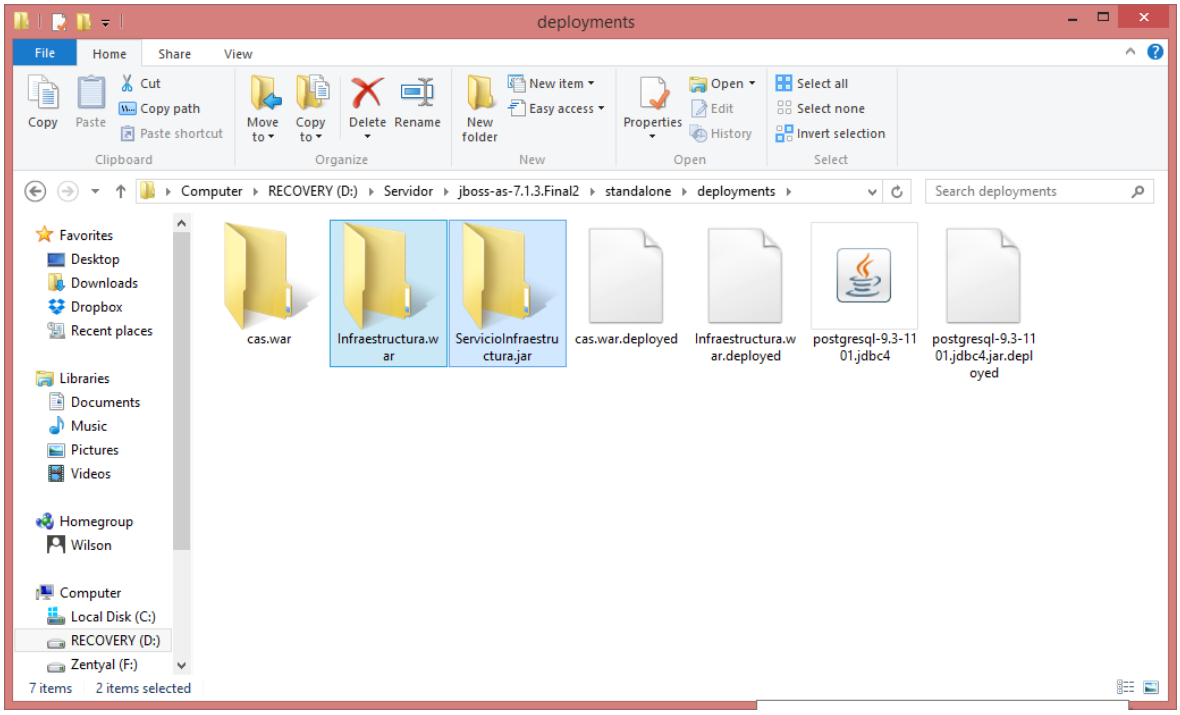


Fig. 1.58: Archivos con formato .war del Sistema ubicados en el Servidor Web. Elaborado por los Autores.

Con los archivos ".war" alojados en los proyectos desarrollados del servidor web, se procede a ingresar al sitio web por medio de la siguiente dirección.

- https://app.epn.edu.ec/Infraestructura

Una vez en el Sitio Web procedemos a ingresar con el usuario y clave otorgado a las personas que harán uso del sistema y de acuerdo a sus determinados perfiles de uso.

En la Figura 3.3 se muestra la pantalla de inicio del sistema donde el usuario deberá autentificarse.

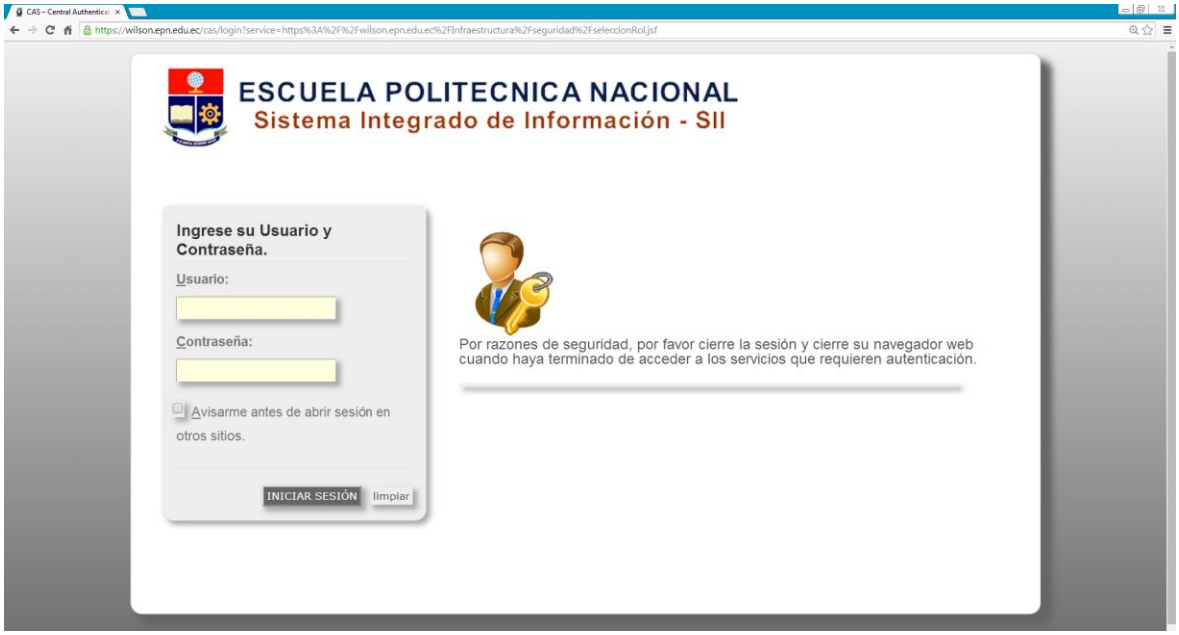


Fig. 1.59: Pantalla de inicio del Sistema. Elaborado por los Autores.

En al Fig.3.4 se muestra la selección del perfil de usuario con el cual podrá hacer uso de las diferentes actividades y facilidades que otorga el sistema.

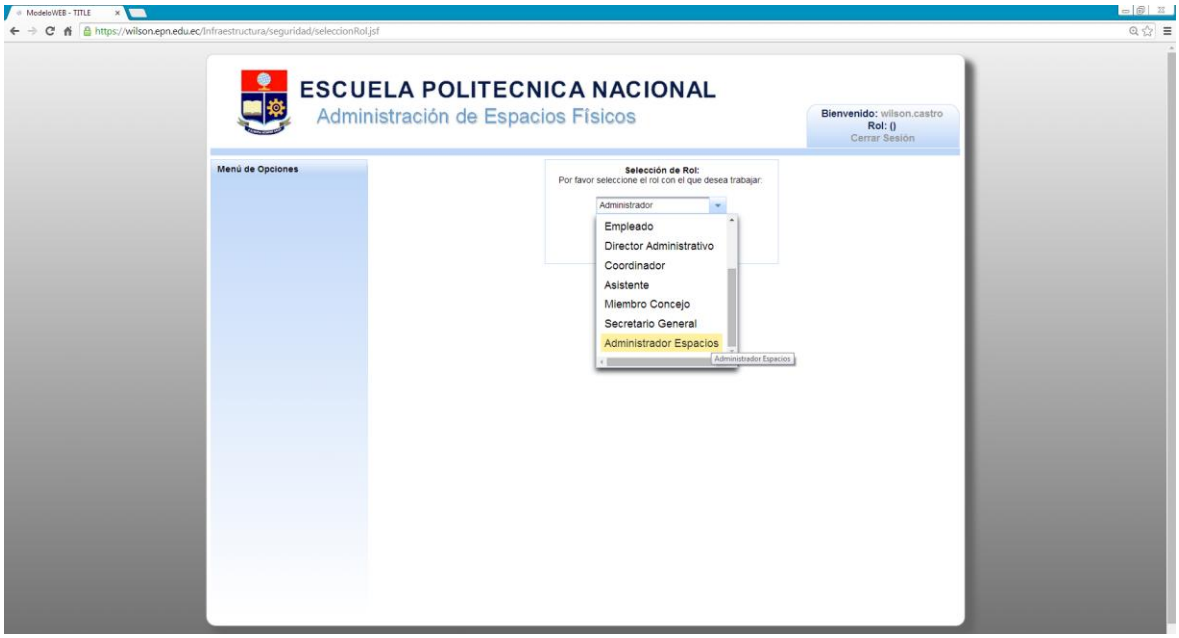


Fig. 1.60: Selección del Perfil de Usuario. Elaborado por los Autores.

3.1 RECOPILACIÓN DE DATOS DE INICIO

Para el inicio del Sistema se debe tener planos de los diferentes pisos de cada edificio de la EPN, cabe recalcar que los planos de los pisos deben ser lo más legítimos y legibles posibles, ya que el sistema, si bien es cierto admite cualquier tipo de plano en formato foto sean estos: jpg y png. Por manejo de visibilidad es recomendable usar fotos grandes y claras.

En la Figura 3.5 se muestra los planos de los pisos del Edificio EARME que se usan para la ubicación de espacios.

Fig. 1.61: Imágenes de planos - Edificio EARME. Elaborado por los Autores.

Las imágenes que se han recolectado a lo largo del desarrollo del sistema están ubicadas en el Anexo 4 Imágenes Pisos de Edificios

Los registros de las tablas señaladas en el punto 2.3.6 EJECUCION DEL QUINTO SPRINT se las ha realizado satisfactoriamente mediante el Sistema Web.

Tabla Tipo de Evento.

Fig. 1.62: Registros Tipo de Evento. Elaborado por los Autores.

Tabla Estado de Evento.

Fig. 1.63: Registros Estado de Evento. Elaborado por los Autores.

Tabla Estado de Espacio

Fig. 1.64: Registros Estado de Espacios. Elaborado por los Autores.

Tabla Tipo de Espacio

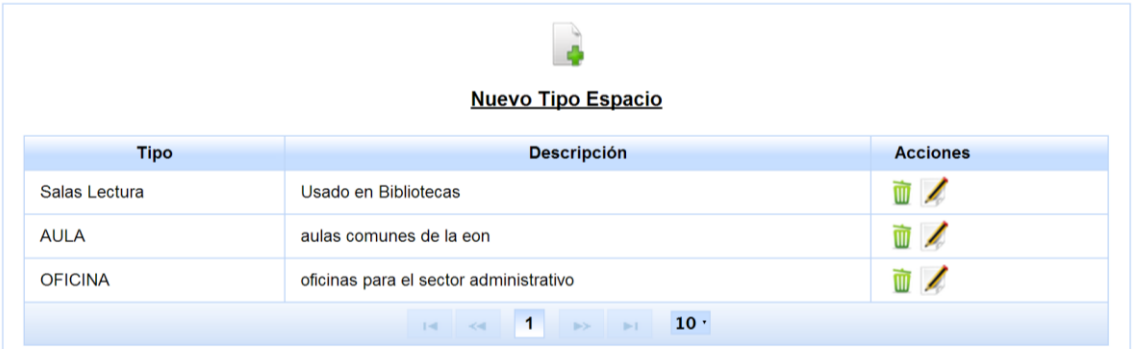


Fig. 1.65: Registros Tipo de Espacios. Elaborado por los Autores.

Tabla Estado de Alquiler

Fig. 1.66: Registros Estado de Alquiler. Elaborado por los Autores.

3.2 CARGA DE DATOS Y CLIENTELIZACIÓN

Para la carga de datos del Sistema se ha tomado como inicio al Edificio EARME el cual está administrado en temas de infraestructura y espacios por la Ing. Andrea Plaza.

Como primer paso se crea un Edifico el cual tendrá posteriormente pisos y planos de cada uno de estos. Para este registro se debe establecer un nombre, referencia del edificio donde puede ser ubicación, color del edificio o el que se vea más conveniente para el reconocimiento del edificio y un código.

En la Figura 3.11 se muestra como se ha creado el edifico y se ha almacenado en la base de datos, listando al nuevo edificio con los demás existentes en la base de datos.

Fig. 1.67: Registro Nuevo Edifico. Elaborado por los Autores.

Posteriormente como segundo paso se debe crear el plano del piso estableciendo un nombre para el plano, una descripción y la imagen que se va a usar como base para referenciar los espacios del piso. En al Fig.3.12 se muestra los datos del plano a ser creado y posteriormente de su registro se puede observar la existencia del nuevo plano.

Fig. 1.68: Datos del Plano a ser Creado. Elaborado por los Autores.

A continuación se debe registrar un piso que hará uso del plano antes registrado, los parámetros de registro para un piso son: nombre del piso, edifico al cual pertenecerá el piso y el plano que se usara para referenciarlo.

En la Figura 3.13 se muestra el registro del piso con el debido edificio y plano, se puede observar una zona de visualización del plano donde el usuario puede ver que plano ha escogido en caso de no estar seguro el nombre del mismo.

Fig. 1.69: Registro de Pisos. Elaborado por los Autores.

Ahora como último paso para registrar los espacios que tiene un piso se debe escoger un edificio y un piso del edificio seleccionado para marcar los espacios deseadas en el piso.

En el caso de que el piso no tenga espacios registrados en el piso se mostrarán mensajes de confirmación de espacio libre para ser usado, caso contrario se mostraran mensajes de que el espacio está ocupado.

En la Figura3.14 se muestra la asignación de un aula al piso creado recientemente en el Edificio del EARME.

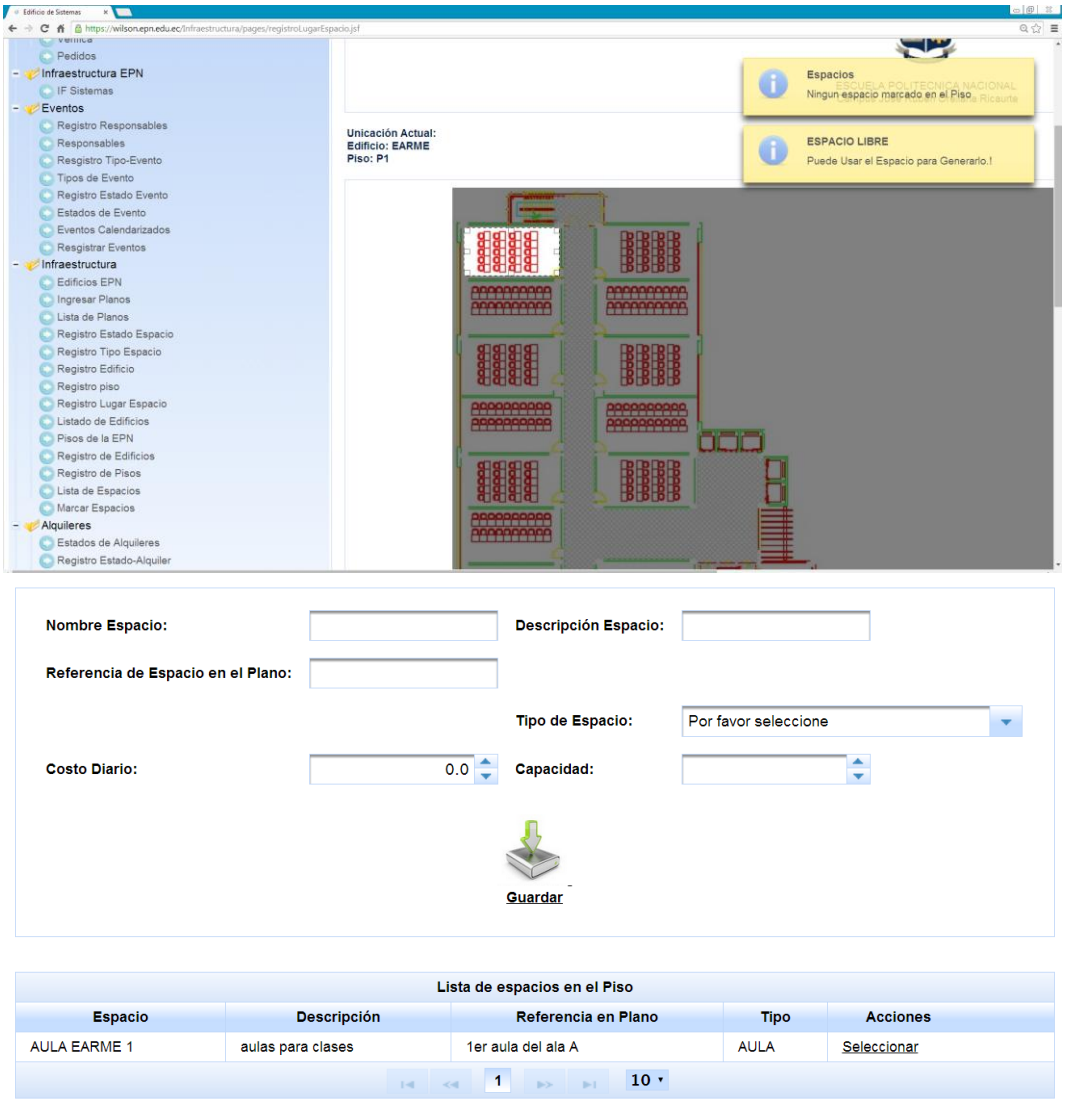


Fig. 1.70: Asignación de Lugar en Piso. Elaborado por los Autores.

Todo el proceso antes realizado se lo debe ejercer en todos y cada uno de los pisos de los Edificios de la EPN por parte de los administradores de los mismos, estableciendo precios para los espacios, capacidad y referencia de ubicación en el plano.

La primera presentación de la realizó el día jueves 18 de junio del 2014 a la Ing. Andrea Plaza quien es el la persona que más usará el sistema y quien fue la persona fundamental para el desarrollo del sistema debido al campo laboral y expertise en el problema que soluciona el desarrollo del Sistema, obteniendo varias observaciones y puntos de vista además de las labores cotidianas que

dicha persona realiza a diario y confronta con los problemas de los espacios del EARME.

Una segunda presentación del Sistema se lo realizó el día martes, 26 de agosto de 2014, con presencia de Msc. Ing. Cristian Villarroel Director de la Dirección de Gestión de la Información y Procesos, la Tlga. Geovanna Saltos Llíder del Área de Desarrollo de Soluciones Informáticas de la DGIP y la Ing. Andrea Plaza Administradora del Edificio EARME.

Cada una de las personas antes mencionadas se vieron conformes con el desarrollo del Sistema y reconocieron la mejora de los procesos que maneja el mismo ante la actual forma de manejarlos, en especial la Ing. Andrea Plaza quien día a día cruza las diferentes problemáticas de la administración de un edificio y los espacios físicos que este conlleva. De parte de la Tlga. Geovanna Saltos mostró un grato agradecimiento a los desarrolladores de la presente tesis, por haber culminado tal trabajo en los tiempos y la planificación estimados.

Cada miembro en la presentación destacó la facilidad de localizar los diferentes espacios físicos con ayuda de un mapa referencial de cada piso de las instalaciones de la EPN.

3.3 ANÁLISIS DE LOS RESULTADOS

Para el análisis de los resultados procedemos a realizar una simulación de toda la funcionalidad del sistema y la utilización por parte de los clientes.

3.3.1 ANÁLISIS DE FUNCIONALIDAD.

Con la simulación de la funcionalidad del Sistema, se ha determinado acertados puntos de uso del mismo, así como posibles recomendaciones y posibles cambios al sistema para su mejora.

En la simulación se ha tomado en cuenta toda la funcionalidad, estableciendo como funcional y no funcional en cada opción del sistema, lo cual se muestra en la Tabla 3.1.

MENÚ	ESTADO	SUBMENUS
Reservaciones	Función correcta	Función correcta
Eventos	Función correcta	Función correcta
Infraestructura	Función correcta	Función correcta
Alquileres	Función correcta	Función correcta
Ocupantes y horarios	Función correcta	Función correcta
Reportes	Función correcta	Función correcta

Tabla 3. 1: Análisis de resultados de Funcionalidad. Elaborado por los Autores

Los 6 menús son accesibles, y de las 34 opciones, trabajan de manera correcta todas y cada una de ellas. En las opciones Registrar Evento y Marcar Espacios en un Plano, se tienen las funcionalidades de envío de correo electrónico para confirmación, el cual funciona correctamente con un lapso de demora de 2 minutos máximo y la funcionalidad de marcar espacios demora máximo 30 segundos en hacer efectiva la acción.

3.3.2 ANÁLISIS DE USABILIDAD.

Hemos definido la usabilidad con parámetros de: aceptación, tiempo de ejecución de las actividades, aceptación de los usuarios, errores y visualización. Para el tiempo de ejecución se han definido valores como bajo, medio y alto para tener en cuenta la demora de cada actividad.

En la Tabla 3.2 se muestra la aceptación de los usuarios, definiendo los valores de bueno, muy bueno y malo para cada uno de los diferentes menús del Sistema.

MENÚ	TIEMPO DE EJECUCIÓN	ACEPTACIÓN
Reservaciones	Medio	Muy bueno
Eventos	Medio	Muy bueno
Infraestructura	Bajo	Muy bueno
Alquileres	Medio	Muy bueno
Ocupantes y horarios	Alto	Muy bueno
Reportes	Medio	Muy bueno

Tabla 3. 2: Análisis de resultados Ejecución. Elaborado por los Autores.

Los errores en las opciones y los menús de cada página se los ha tomado como errores al momento de realizar un ingreso de datos o cuando ha ocurrido un error en una actividad, los errores más comunes fue el manejo de mail para las notificaciones, ya que no aceptaban correos con terminación “.ec” y tampoco se podía hacer uso de dicho recurso sin tener un uso del sistema dentro de la intranet de la EPN. La visualización es un aspecto que llama la atención de los usuarios por lo cual también afecta en la aceptación del sistema, los resultados del análisis realizado con respecto a este parámetro arrojó los resultados mostrados en la Tabla 3.3.

PARÁMETRO	**ACEPTACIÓN**
Interfaces	Muy buena
Facilidad de operación	Buena
Gama de colores	Buena

Tabla 3. 3: Análisis de resultados Visualización. Elaborado por los Autores.

4 CAPÍTULO 4: CONCLUSIONES Y RECOMENDACIONES

4.1 CONCLUSIONES

- El Sistema Web para la Administración de la Infraestructura Física de la Escuela Politécnica Nacional, permite automatizar los procesos de gestión actuales en cuanto a espacios físicos se refiere estableciendo un control adecuado, consulta y fácil acceso a información actual de los mismos. Además permite agilitar procesos de gestión de reserva para eventos y alquiler siendo de gran ayuda para los administradores encargados de estas actividades dando así por terminado el uso de hojas electrónicas tal y como se lo venía realizando anteriormente.

- Se constató que ciertos espacios físicos de la Escuela Politécnica Nacional no se encuentran plenamente identificados sin embargo con el apoyo del sistema web desarrollado se obtendrá un debido control y así llevar un registro adecuado que garantice la existencia de los mismos de tal forma que personas que trabajan, estudian o visitan la EPN puedan apreciar la infraestructura física con la que cuenta la misma.

- El entorno de desarrollo J2EE permitió implementar fácilmente el modelo vista controlador, logrando que la creación de código fuente sea gestionada de forma organizada. Además usar este esquema en capas permite realizar modificaciones en el código rápidamente permitiendo así tener un sistema mantenible a futuro.

- Un pilar fundamental al aplicar XP y SCRUM en conjunto fue mantener interacción directa con el cliente, porque se garantizó el entendimiento de los requerimientos y a su vez con los entregables se apreció la satisfacción del mismo.

- Se apreció total compromiso por parte del equipo de desarrollo en cada Sprint, su experiencia y puntos de vista ayudó en la estimación de esfuerzo para cumplir los requerimientos en el plazo indicado, lo que permitió garantizar la entrega de un producto funcional y operativo.

- De acuerdo a los resultados obtenidos se pudo constatar que el producto de software final cumple los requisitos planteados por el usuario en cuanto a funcionalidad (Tabla 3.1), ejecución (Tabla 3.2) y visualización (Tabla 3.3) debido a que en el desarrollo se pudo realizar los ajustes y correcciones necesarios en cada iteración de cara a la entrega del producto final.

4.2 RECOMENDACIONES

- La información almacenada por el sistema podría ser usada como constancia de la infraestructura con la que cuenta la EPN para futuras evaluaciones de acreditación de universidades.

- Los Centros de Educación Superior deberían contar en su mayoría con sistemas desarrollados en herramientas libres ya que a su vez se minimizan costos y se tienen mayor flexibilidad que con herramientas propietarias.

- Para el seguimiento del desarrollo de un sistema amplio y sensible a cambios es necesario mantener una metodología apta para dichas funciones, además de que el usuario conozca cada paso que se debe adoptar para poder mejorar el desarrollo y la metodología a la par.

- Mayormente el uso de instalaciones en instituciones se mantiene como una actividad sin necesidad de seguimiento, pero cuando se tiene una amplia cantidad de instalaciones, se ve necesario mantener dichas instalaciones o infraestructura controlada de alguna forma y que mejor con un sistema que administre cada punto de dichas instalaciones.

- Para el manejo de los espacios que maneja una institución, se debe tener en cuenta que las oficinas pueden cambiar de lugar si son construcciones modulares y no con paredes fijas, esto afecta a la creación o actualización de datos en la ubicación de los espacios en un plano de infraestructura, como los utilizados en esta tesis.

- La actualización de planos por parte de una persona encargada de la infraestructura de una institución deben ser debidamente documentada y siempre accesible para las personas que lo necesiten.

- Un grupo de personas encargada de la administración de las instalaciones de una institución es primordial para dar a conocer como está estructurado y ubicado cada espacio, a personas propias y ajenas a la institución.

5 BIBLIOGRAFÍA

[1] EPN, «Escuela Politécnica Nacional,» 2010. [En línea]. Available: http://www.epn.edu.ec/.

[2] «ACTA DE RESOLUCIONES DE LA SESION ORDINARIA DECONSEJO POLITECNICO DE 23 DE JUNIO DEL 2014,» Quito, 2014.

[3] E. Oguejiofor, K. Childers, D. Dhuyvetter y P. Hood, de *IBM WebSphere and Microsoft .NET Interoperability*, 2006, pp. 32-38.

[4] « An Introduction to Java Platform, Enterprise Edition,» Oracle, 2012. [En línea]. Available: http://goo.gl/eKq0hl.

[5] G. P. Mosquera Soto y J. D. Bracho Baquero, «Aplicaciones empresariales usando plataforma J2EE,» 2005. [En línea]. Available: http://goo.gl/TdFzkq.

[6] E. Jendrock, R. Cervera Navarro, I. Evans, K. Haase y W. Markito, «The Java EE 6 Tutorial,» 2013. [En línea]. Available: http://goo.gl/ZNbcPZ.

[7] J. M. Ordax Cassá y P. A. Ocaña Díaz-Ufano, «Programación Web en Java,» [En línea]. Available: http://goo.gl/5af5Il.

[8] «Understanding JBoss EAP 6,» 2014. [En línea]. Available: http://goo.gl/JesXsW.

[9] P. Johnson y J. Jamae, de *JBoss in Action: Configuring the JBoss Application Server*, 2009.

[10] F. Maciá Pérez, de *Administración de servicios de Internet: De la teoría a la práctica*, 2008.

[11] «EcuRed,» EcuRed, [En línea]. Available:

http://www.ecured.cu/index.php/Java_Management_Extensions.

[12] D. Allen, «Jaxenter,» 2012. [En línea]. Available: http://jaxenter.com/7-reasons-to-love-jboss-as7-104408.html.

[13] K. D. MANN, de *JavaServer Faces in Action*, United States of America, Manning Publications Co, 2005.

[14] E. Burns y C. Schalk, de *JavaServer Faces 2.0-Complete Reference*, McGraw-Hill, 2010.

[15] R. M. Lerner, «Open-Source Databases, Part II: PostgreSQL,» [En línea]. Available: http://www.acm.org/.

[16] B. Momjian, «PostgreSQL Performance Tuning,» [En línea]. Available: http://www.acm.org/.

[17] F. Kasián y N. Reyes, *Búsquedas por similitud en PostgreSQL,* Argentina.

[18] «PostgreSQL,» 2010. [En línea]. Available: http://www.postgresql.org.es/sobre_postgresql.

[19] J. Palacio, de *Scrum I Gestion Tecnica de Proyectos con Scrum*, 2013.

[20] C. L. B. V. Pete Deemer Gabrielle Benefield, de *Scrum Primer*, 2012.

[21] G. M. Peñalver Romero , S. J. García de la Puente y A. Meneses Abad, de *SXP, Metodología de Desarrollo de Software*, 2010.

[22] J. Ramonet, de *RUP y SCRUM y La Gestion de Proyectos WEB La Gestion de Proyectos WEB*, Barcelona, 2010.

[23] J. Palacio, de *Scrum Manager Manual Gestión de Proyectos*, 2008.

[24] H. Kniberg, «Scrum y XP desde las Trinchieras,» 2007. [En línea]. Available: http://goo.gl/SiF56t.

[25] J. Palacio, de *Flexibilidad con Scrum*, 2007.

[26] P. Letelier y C. Penadés, de *Metodologias agiles para el desarrollo de software: eXtreme Programming (XP)*, Valencia, 2006.

[27] G. Fernandez Escribano, de *Programacion Extrema*, 2002.

[28] K. Beck, de *Extreme Programming Explained*, 1999.

6 GLOSARIO DE TÉRMINOS

ACID: Atomicity **C**onsistency **I**solation **D**urability, en español Atomicidad, Consistencia, Aislamiento y Durabilidad, propiedades que garantizan que las transacciones de base de datos se realicen de forma confiable.

ANSI SQL: American **N**ational **S**tandards **I**nstitute **S**tructured **Q**uery **L**anguage, español Instituto Nacional de Estándares Americano de Lenguaje de Consultas Estructurado.

API: Application **P**rogramming **I**nterface, en español Interfaz de programación de aplicaciones que es un protocolo de programación para comunicarse con un software en particular.

Applets: Pequeña aplicación escrita en Java la cual se difunde a través de la red en orden de ejecutarse en el navegador cliente.

AT&T: American **T**elephone and **T**elegraph**,** es una compañía estadounidense de telecomunicaciones. Provee servicios de voz, video, datos, e internet a negocios, clientes y agencias del gobierno.

AWT: Abstract **W**indow **T**oolkit, en español Kit de Herramientas de Ventana Abstracta, es un conjunto de gráficos, interfaz de usuario, y sistema de ventanas independiente de la plataforma original de Java.

Accesibilidad: Se define como el grado en que el hardware o el software puedan ser utilizados por diferentes usuarios sea cual sea su diversidad funcional.

Beans: Componente hecho en software que se puede reutilizar y que puede ser manipulado visualmente por una herramienta de programación en lenguaje Java.

C: Lenguaje de programación creado en 1972 por Dennis M. Ritchie en los Laboratorios Bell como evolución del anterior lenguaje B, a su vez basado en BCPL.

C++: lenguaje de programación diseñado a mediados de los años 1980 por Bjarne Stroustrup. La intención de su creación fue el extender al exitoso lenguaje de programación C con mecanismos que permitan la manipulación de objetos.

CAS: aplicación web para implementar un procedimiento de autenticación que habilita a un usuario para acceder a distintas aplicaciones web (en distintos dominios y en distintos servidores) con hacer login una única vez.

CEAACES: Consejo de **E**valuación, **A**creditación y **A**seguramiento de la **C**alidad de la **E**ducación **S**uperior.

EARME: Edificio de **A**ulas y **R**elación con el **M**edio **E**xterno.

ECPG: herramienta de programación que permite al usuario integrar sentencias de SQL a un programa fuente de alto nivel.

EIS: Enterprise **I**nformation **S**ystem, en español capa de sistemas de información empresarial en la arquitectura J2EE.

Escalabilidad: propiedad de aumentar la capacidad de trabajo o de tamaño de un sistema sin comprometer su funcionamiento y calidad normales.

Expression Language: Lenguaje de programación de propósito especial utilizado principalmente en aplicaciones web en Java para incrustar expresiones en páginas web.

Extensible: Propiedad de un sistema para adecuarse a otros sistemas.

FacesContext: Es una clase que sirve al Bean de puente al exterior, ya que le permite acceder no solo al contexto JSF sino también al contexto HTTP.

Flexibilidad: Propiedad que permite establecer en qué medida un sistema es susceptible a ser cambiado.

Framework entorno de trabajo o conjunto de bibliotecas orientadas a la reutilización a muy gran escala de componentes software para el desarrollo rápido de aplicaciones.

GSSAPI: Es un API para usar sistemas de seguridad de forma genérica.

HTML: Hyper **T**ext **M**arkup **L**anguage, en español lenguaje de marcas de hipertexto y hace referencia al lenguaje de marcado que permite elaborar páginas web.

HTTP: Hyper **T**ext **T**ransfer **P**rotocol en español protocolo de transferencia de hipertexto, es el protocolo usado en cada transacción de la World Wide Web.

J2EE6 Web Profile: Versión 6 de J2EE para servidores que implementan un subconjunto de características como la especificación de EJB, JPA y JTA**.**

JCP: Java **C**ommunity **P**rocess o en español El Proceso de la Comunidad Java, es un proceso formalizado el cual permite a las partes interesadas a involucrarse en la definición de futuras versiones y características de la plataforma Java.

JDBC: Java **D**atabase **C**onnectivity es un API usado para enviar comandos SQL hacia una base de datos relacional.

JMX: Java **M**anagement e**X**tensions, tecnología que define una arquitectura de gestión, para los patrones de diseño, y los servicios para la monitorización/administración de aplicaciones basadas en Java.

JVM: Máquina **V**irtual **J**ava que es el entorno en el que se ejecutan los programas Java, su misión principal es la de garantizar la portabilidad de las aplicaciones Java.

Kerberos: Protocolo de autenticación de redes de ordenador creado por el MIT que permite a dos ordenadores en una red insegura demostrar su identidad mutuamente de manera segura.

LDAP: Lightweight **D**irectory **A**ccess **P**rotocol en español Protocolo compacto de acceso a directorios, es un protocolo estándar que permite administrar directorios al acceder a bases de información de usuarios de una red mediante protocolos TCP/IP.

Licencia BSD: La licencia BSD cubre las distribuciones de software **Berkeley Software Distribution,** además de otros programas. Esta es una licencia considerada permisiva ya que impone pocas restricciones sobre la forma de uso, alteraciones y redistribución del software.

Middleware: Software que asiste a una aplicación para interactuar o comunicarse con otras aplicaciones, software, redes, hardware y/o sistemas operativos.

Modular: Característica que permite desarrollar por partes un mismo sistema e integrar los módulos en la medida en que lo impongan las reglas de negocio.

ODBC: Open **D**ataBase **C**onnectivity, es un estándar de acceso a bases de datos que utilizan los sistemas Microsoft. Las siglas significan.

Perl: Practical **E**xtracting and **R**eporting **L**anguaje, lenguaje de programación que permite extraer información de archivos de texto y generar informes a partir del contendido de los ficheros.

Persistencia: Capacidad que tienen los objetos java para guardarse y recuperarse desde un medio de almacenamiento.

PHP: Lenguaje de programación de uso general de código del lado del servidor originalmente diseñado para el desarrollo web de contenido dinámico.

Portabilidad: Propiedad que poseen los sistemas para funcionar independientemente de la plataforma donde se estén ejecutado.

Python: Lenguaje de programación interpretado cuya filosofía hace hincapié en una sintaxis que favorezca un código legible.

REST: REpresentational **S**tate **T**ransfer, es un tipo de arquitectura de desarrollo web que se apoya totalmente en el estándar HTTP.

Ruby: Lenguaje de programación interpretada, reflexiva y orientada a objetos, creado por el programador japonés Yukihiro "Matz" Matsumoto, está enfocado en la simplicidad y productividad.

Serialización: Consiste en un proceso de codificación de un objeto en un medio de almacenamiento con el fin de transmitirlo a través de una conexión en red como una serie de bytes o en otro formato.

SOAP: Simple **O**bject **A**ccess **P**rotocol, se trata de un protocolo que te permite la comunicación entre aplicaciones a través de mensajes por medio de Internet.

SL: Software **L**ibre.

SSL: Secure **S**ockets **L**ayer es un protocolo diseñado para permitir que las aplicaciones para transmitir información de ida y de manera segura hacia atrás.

Stand-Alone: Significa sin conexión por ejemplo equipos que tienen su sistema operativo, aplicaciones y todo lo necesario para funcionar, pero sin conexión a una red.

Swing: Es un API para proporcionar una interfaz gráfica de usuario (GUI) para programas de Java.

Triggers: Es un procedimiento que se ejecuta cuando se cumple una condición establecida al realizar una operación, utilizada principalmente en base de datos

UI: User **I**nterface, interfaz de usuario es el medio con que el usuario puede comunicarse con una máquina, un equipo o una computadora.

UNIX: es un sistema operativo portable, multitarea y multiusuario; desarrollado, en principio, en 1969, por un grupo de empleados de los laboratorios Bell de AT&T, entre los que figuran Ken Thompson, Dennis Ritchie y Douglas McIlroy.

XHTML: e**X**tensible **H**ypertext **M**arkup **L**anguage, en español lenguaje extensible de marcación hipertexto es un lenguaje similar a HTML, pero con algunas diferencias que lo hacen más robusto y aconsejable para la modelación de páginas web.

7 ANEXOS

Los anexos que se numeran a continuación se encuentran en el disco adjunto:

ANEXO 1 – DATOS DE EDIFICIOS.

ANEXO 2 – CODIFICACION.

ANEXO 3 – CASOS DE PRUEBA.

ANEXO 4 – IMÁGENES PISOS DE EDIFICIOS.

ANEXO 5 – MANUAL DE USUARIO.

ANEXO 6 - DECRETO 1014 SOFTWARE LIBRE EN ECUADOR.

ANEXO 7 – DICCIONARIO DE BASE DE DATOS.