

**UNIVERSIDAD MAYOR DE SAN SIMÓN**

**DIRECCIÓN DE PLANIFICACIÓN ACADÉMICA**

**PROGRAMA DE TITULACIÓN DE ALUMNOS ANTIGUOS NO GRADUADOS**



**FACULTAD DE CIENCIAS Y TECNOLOGÍA  
CARRERAS DE LICENCIATURA EN INFORMÁTICA E INGENIERÍA DE SISTEMAS**

**“SISTEMA DE INFORMACIÓN PARA LA AUTOMATIZACIÓN DE LOS PROCESOS ADMINISTRATIVOS DE LA ESCUELA POPULAR DE ARTES (EPA).”**

TRABAJO DIRIGIDO INTERDISCIPLINARIO PARA OPTAR EL GRADO ACADÉMICO DE LICENCIATURA EN INFORMÁTICA E INGENIERÍA DE SISTEMAS

**AUTORES:     LUIS FERNANDO CAMACHO CAMACHO**

**ELMER ALVARADO PONCE**

**LEONARDO S. MENDIETA AZCUI**

**TUTOR:           LIC. JORGE ORELLANA**

|  |
| --- |
| **TEC25TD019** |

**COCHABAMBA – BOLIVIA**

**2014**

**AGRADECIMIENTOS**

**DEDICATORIAS**

**ÍNDICE DE CONTENIDO**

**CAPÍTULO 1:  
Introducción**

El presente proyecto consiste en la implementación de un sistema de información para resolver el problema de la ineficiencia del control manual de asistencia y generación de planilla de pagos, para ello se aplicarán  tecnologías web 2.0 y Scrum como método de desarrollo (PTAANG 2014).

**Antecedentes Generales**

El desarrollo de software en la actualidad marca ciertas tendencias importantes, una de ellas es completar el ciclo de vida de desarrollo en tiempos cortos, para así poder entregar un Software funcionando de manera frecuente; otra de ellas es hacer que el usuario quede involucrado en el proceso de desarrollo porque aún se ven casos en que la falta de comunicación con el usuario final (cliente), lleva a entregar un producto que es sujeto a muchos cambios, esto da como resultado la elevación del costo del producto final, lo que se pretende evitar es que el usuario final interactúa sólo con el producto terminado, sino que sea parte del proceso de desarrollo, es ahí donde aparece como una alternativa el uso de metodologías ágiles de desarrollo de Software como Scrum (Scrum Alliance, manifiesto Agile).

Gran parte del desarrollo de software está enfocado a aplicaciones web, como versión inicial la Web mostraba páginas de contenido estático estas se las conoce como Web 1.0. Ésta demostró ofrecer pocas opciones de interactuar con el usuario, además que tenía un bajo rendimiento al intentar añadir interactividad, el término web 2.0  se refiere a aplicaciones web interactivas con el usuario y que a su vez le permiten a éste interactuar con otros usuarios, como por ejemplo las wikis, blogs, redes sociales, etc. siendo así una evolución a las páginas de contenido estático. con esto llega la tendencia del  desarrollo de aplicaciones web usando JavaScript para el poblado de las páginas HTML debido a que JavaScript se ejecuta en el navegador ofreciendo así un mejor rendimiento y tiempo de respuesta comparado con las clásicas aplicaciones cliente  servidor (Wikipedia-2014. Web 2.0.)

Una de las técnicas más comunes para el desarrollo de aplicaciones Web 2.0 es la creación de un servidor REST (La Transferencia de Estado Representacional) que interactúa con la base de datos, recibiendo y enviando la información hacia el navegador el cual determinará cómo mostrará al usuario el contenido de la página y así el servidor evitará generar contenido HTML dinámico.

**Antecedentes Específicos**

La Escuela Popular de Artes desde sus inicios viene utilizando métodos manuales para su administración. La forma de trabajo tiene como objetivo la adaptación de horarios acorde a las necesidades de los estudiantes, es así que no existen fechas fijas para la inscripción, sino que lo puede realizar durante todo el mes.

Existen diferentes modalidades de clases, primer grupo que pasa clases 2 veces a la semana, un segundo grupo que consta de 3 clases por semana y los grupos regulares (todos los días).

Los estudiantes tienen un cronograma de clases, que se crea al momento de la inscripción, pero ellos pueden reprogramar sus clases para otros horarios según sea su necesidad, esto requiere una coordinación de la disponibilidad del profesor para evitar cruces de horarios con otros grupos.

La asistencia de los estudiantes es controlada al ingreso y se marca de forma manual en su respectivo kardex, los profesores también tienen un kardex en el que se registran las horas de clases dictadas, a partir del cual se preparan sus planillas de pagos.

Durante el periodo de clases es frecuente que los padres de familia vengan a requerir el número de clases tomadas por el estudiante, esta recopilación de información es obtenida del kardex del estudiante en forma manual.

Antes de concluir el periodo de clases, la administración notifica al estudiante que el curso está por terminar, para dar plazo de una reinscripción o un cierre de curso.

**Descripción del problema.**

La gestión del plantel académico de la Escuela Popular de Artes es ineficiente debido a que el manejo de la información referente al plantel académico  y el proceso de inscripción toman mucho tiempo por ser procesos manuales y susceptibles a errores, así también el control de asistencia tomando en cuenta que si una clase dictada no fue registrada, el estudiante aún tendrá  una clase pendiente en su kardex pero el pago que  se realiza por dicha clase al profesor correrá por parte de la escuela, dando la posibilidad de disminuir las utilidades.

**Identificación del Problema**

Procesos manuales en el manejo de la información de inscripciones y control de asistencia, provocan un tiempo excesivo en la administración de la escuela.

**Objetivo General**

Automatizar los procesos en el manejo de la información en la administración de la escuela para agilizarlos y reducir el tiempo utilizado

**Objetivos Específicos.**

1.    Investigar las tecnologías RIA (Rich Internet Applications) para aplicarlas al sistema de información.

2.    Agilizar el proceso de inscripción facilitando el registro de estudiantes nuevos y regulares.

3.    Dar mayor precisión al control de asistencia de estudiantes y profesores.

4.    Obtener información de estudiantes y/o profesor de forma inmediata, creando reportes automáticos de la asistencia a las clases.

5. Notificar la culminación de curso por estudiante, creando alertas automáticas.

**Justificación.**

El Sistema de información ayudará a la Escuela Popular de Artes en el  proceso de inscripción de estudiantes nuevos y regulares, como también  la asignación de horarios Profesor-estudiante, puesto que estos procesos son manuales, toman mucho tiempo y que carecen de información. Así también ayudará en generación de planilla de sueldos, este proceso conlleva mucho tiempo por ser un trabajo moroso, pues es necesario revisar todas las clases dictadas almacenadas en un libro físico. Con el sistema de información todos estos procesos se harán de manera automática y más eficiente.

El proyecto es de carácter investigativo por el hecho de que usará Scrum en el proceso de desarrollo y además tecnología recientes en el área de Web 2.0 como ser EmberJs, ExpressJs ,NodeJs y REST, este último es un servicio genérico.

**Límites y Alcance**

Alcances

* El producto final será una aplicación web 2.0
* El producto final podrá ser usado los siguientes navegadores (Chrome, Internet Explorer, Safari)
* La Aplicación estará enfocada al Gerente y Administrador, no así para los estudiantes.

Límites

* El software entregado no estará diseñado para usarse en dispositivos móviles
* El proceso de control de asistencia no tendrá registro biométrico
* No se implementara una parte contable para este proyecto
* El presente proyecto no tendrá el módulo de ventas ni inventarios
* No tendrá el módulo de facturación.

**Diseño Metodológico**

|  |  |  |  |
| --- | --- | --- | --- |
| Sprint o Hito | Tareas a realizar | Fecha inicio | Fecha fin |
| Recolección de información | visita y entrevista al usuario final  -árbol de problemas y árbol de objetivos | 23/10/2014 | 30/10/2014 |
| Análisis de la problemática | -Identificar requerimientos funcionales y no funcionales  -Creación de un acuerdo sobre el alcance del sistema de información | 30/10/2014 | 07/11/2014 |
| conocer las tecnologías a utilizar en el sistema de información | Investigar y hacer una prueba  de concepto de la creación de un servicio REST usando Nodejs, Expressjs, Emberjs | 07/11/2014 | 14/11/2014 |
| Diseño del sistema | Diseño de la arquitectura del sistema de información(Base de Datos, REST, Navegación) | 15/11/2014 | 21/11/2014 |
| Implementación del entorno de desarrollo | Implementación del servicio REST  Implementación básica  de estructura del proyecto web | 22/11/2014 | 06/12/2014 |
| Sprint 1 | Crear el Modulo de inscripción de alumnos antiguos y nuevos | 07/12/2014 | 07/01/2015 |
| Sprint 2 | Crear  el módulo del control de asistencia de estudiantes y profesor | 08/01/2015 | 08/02/2015 |
| Sprint 3 | Desarrollar reportes definidos para la planilla de sueldos del plantel docente. | 08/02/2015 | 08/03/2015 |
| Sprint 4 | Desarrollar reportes de asistencia, inicio y fin del módulo del plantel estudiantil. | 08/03/2015 | 08/04/2015 |

**CAPÍTULO 2:  
LA ESCUELA POPULAR DE ARTES (EPA)**

1. **EPA**
   1. **Introducción**

La EPA ofrece una capacitación musical integral que dota a los estudiantes de todas las herramientas, tanto técnicas como teóricas, en la práctica musical y sus distintas formas y géneros, el proceso de capacitación comprende, el aprendizaje de un instrumento, así también entendimiento de la historia, teoría y los diversos conceptos que definen al arte, además de enseñar a los estudiantes que la música es una forma de expresión de la sociedad.

* 1. **La escuela**
  2. **Cursos que se ofertan**

La Escuela Popular de Artes (EPA) ofrece una variedad de Programas, estos programas son:

* Programa regular
* Cursos Libres
* Programa Interactivo Niños
* Asignaturas técnico – teóricas
  + 1. **Programa regular.**

El programa regular tiene como objetivo la formación de un intérprete profesional, este programa de música comprende los Ciclos de básico a intermedio, el cual tiene una duración de 5 años comprende las siguientes áreas:

* Clases individuales de un instrumento de especialidad
* Clases de:
  + Solfeo Básico
  + Lectoescritura Rítmica
  + Teoría Musical
  + Dictado Rítmico-Melódico
* Prácticas Grupales (Coros, Orquestas y Ensambles)
* Conciertos Anuales
* Sesiones de Video, Talleres y Clases Magistrales
  + 1. **Cursos Libres**

Los Cursos Libres de Instrumento están abiertos durante todo el año, brindando horarios flexibles para aquellas personas que desean capacitarse en un instrumento y no cuentan con la disponibilidad horaria para realizar los cursos regulares de música

* + 1. **Programa Interactivo Niños**

El Programa para Niños de la EPA está abierto durante todo el año y brinda un innovador sistema pedagógico en música utilizando juegos interactivos, software especializado y métodos de expresión corporal diseñados para un mejor aprendizaje de los más pequeños

* + 1. **Asignaturas técnico – teóricas**

La Escuela Popular de Artes brinda la opción de tomar las materias Técnico-Teóricas de forma libre a aquellos músicos que desean reforzar alguna rama teórica en su formación artística

* 1. **Profesores**

El plantel docente de la EPA se encuentra constituido por músicos nacionales e internacionales.

Los Profesores tienen diversas especialidades y pueden ser asignados a más de un instrumento o área, cada Profesor puede dictar varias clases durante el día sobre uno o más instrumentos. Los salarios de los profesores son calculados de acuerdo a la cantidad de horas de clase dictadas.

* 1. **Horarios**

Los horarios de clase son a elección del estudiante.

* 1. **Administración**
     1. Administrador

El Administrador tiene la función realizar la inscripción, el cobro de pensiones, el seguimiento del registro de clases dictadas, registrar las clases suspendidas y así también la reprogramación de clases suspendidas.

* 1. **Inscripciones**

El Proceso de inscripción se puede realizar cualquier día de la semana. Se busca la disponibilidad del profesor y de preferencia se busca el horario más cercano a otras clases dictadas por el profesor tratando que todas las clases de ese profesor sean adyacentes o próximas.

El profesor puede suspender la clase, pero dicha clase será sustituida en otro momento en acuerdo con el estudiante.

* 1. **Seguimiento de asistencia**
     1. **Clases.**

Las clases son personalizadas: un profesor por un estudiante, que son programadas de acuerdo a la necesidad de los estudiantes y disponibilidad de los profesores, es decir que no necesariamente todas las sesiones de clases serán dictadas en el mismo horario todos los días, cada clase tiene que ser registrada y/o marcada como dictada para poder llevar el control de asistencia.

* + 1. **Registro de Asistencia.**

El seguimiento de la asistencia se realiza de forma manual, el administrador registra en un kardex personal cada vez que una sesión de clase es completada

* + 1. **Notificación de fin de curso.**

Antes de finalizar el ciclo de clases la administración advierte al estudiante que tiene que éste está por terminar, dando así posibilidad de registrarse para el nuevo ciclo de clases, en caso de no volver a registrarse para un nuevo ciclo el curso quedará suspendido.

* + 1. **Postergación de clases.**

Se podrá postergar las clases según requerimiento tanto del estudiante como del profesor, cuando esto ocurre el administrador deberá registrar la fecha como clase suspendida y consensuar un nuevo horario con el estudiante y el profesor.

* 1. **Profesores.**

El salario de los profesores se calcula según las horas de clase dictadas y son calculadas al final de cada mes, el administrador saca de un total de las horas clases dictadas.

**CAPÍTULO 3:  
TECNOLOGÍAS DE LA INFORMACIÓN (**Marco Teórico)

1. **Tecnologías a usar en la Ingeniería del sistema**A continuación se detallan algunos aspectos sobre la ingeniería del sistema
   1. **Enfoque (Approach)**

Un análisis basado en los requerimientos de los usuarios, quienes deben poder acceder al sistema desde diferentes ubicaciones, considera que desarrollar la aplicación en un entorno Web será lo más práctico

* + 1. **Web2.0**
       1. **Evolución de la Web**

La Web ha sufrido muchos cambios relacionados con la infraestructura, mantenibilidad e interacción con el usuario, inició con páginas estáticas de contenido difícil de actualizar, que han ido cambiando para permitir su actualización constante y brindar una mayor interacción con el usuario gracias a nuevas y revolucionarias tecnologías.

* + - 1. **Web 1.0 vs. Web 2.0**

La implementación original de la Web (Web1.0) se basaba en páginas con documentos de hipertexto híper vinculados que se transferían por medio de internet, con el uso de un navegador era posible ver texto, imágenes y demás archivos multimedia, a través de un clic el usuario podía navegar entre las distintas páginas y el usuario solo podía ver dicha información.

Con el paso del tiempo las llamadas “Tecnologías Web” han evolucionado para cambiar esas páginas estáticas y dar paso a las páginas dinámicas en las que la información se genera directa o indirectamente por los usuarios y es compartida por los sitios Web de diferentes maneras, es así que el usuario tiene la posibilidad de aportar contenidos al interactuar y colaborar con la página, abriendo paso a novedosas aplicaciones como por ejemplo las conocidas como “Redes Sociales”.

* + - 1. **Aplicaciones de la Web 2.0**

La Web2.0 ofrece un ambiente propicio para una variedad de aplicaciones y herramientas:

"Mientras que la primera ola de la Web estaba unida muy de cerca al navegador, la segunda ola extiende las aplicaciones por toda la Web y hace posible una nueva generación de servicios y oportunidades de negocio." (Tim O'Reilly, 2004)

Algunos ejemplos de las aplicaciones de estas páginas web se describen a continuación:

* + **Blog**

Es un sitio Web en el que los autores publican fragmentos en orden cronológico, del más reciente al más antiguo. Los blogs permiten que los usuarios hagan comentarios en las publicaciones de otros autores, pero no dejan que cambien nada que no haya sido publicado por ellos mismos.

* + **Micro-blog**

Es similar a un blog pero con la limitación de que en el contenido de mensajes se debe tener una longitud de 140 caracteres como máximo.

* + **Wiki**

El sitio permite la creación y edición de páginas con contenido relacionado unas de otras, sin requerir conocimientos de HTML. El sitio guarda versiones de las páginas lo que permite rastrear los cambios realizados en la edición y así volver a una versión anterior cuando sea necesario. El sitio además realiza un control de concurrencia cuando varios usuarios intentan hacer cambios a una misma página de modo que no existan conflictos y un usuario pueda terminar su trabajo antes de dar paso al siguiente.

* + **Herramientas de colaboración para Oficina**

Aplicaciones de procesamiento de texto, planilla de cálculo, o de presentación que funcionan en un navegador y permiten que varios usuarios tengan acceso simultáneo a un mismo documento y puedan trabajar juntos en tiempo real.

* + **Etiqueta de contenido y Redes sociales**

Cada usuario puede “Etiquetar” los contenidos asociando palabras clave para clasificar la información de distintas formas y de esta manera administrar la información digital.

* + - 1. **RIA - Rich Internet application**

Las aplicaciones de Internet Enriquecidas (Rich Internet applications) son aplicaciones basadas en Web que funcionan como las aplicaciones tradicionales de escritorio, sin embargo, éstas aplicaciones requieren un navegador web (o clientes) para ejecutarse y dar acceso, pero a diferencia de las aplicaciones tradicionales, no se requiere la instalación del software, no obstante, dependiendo de la aplicación, a menudo será necesario tener instaladas en la máquina cliente tecnologías como ActiveX, Java, Flash u otros complementos similares que agregan las características adicionales.

Las RIA buscan ser una combinación de las ventajas que ofrecen las aplicaciones web y las aplicaciones tradicionales tratando de mejorar la experiencia del usuario y aportar a su productividad.

El objetivo es reducir la recarga continua de páginas en las aplicaciones web ya que cada vez que el usuario usa un enlace, se produce un tráfico muy alto entre el cliente y el servidor llegando a menudo a descargar la misma página con prácticamente el mismo contenido sin cambio. Por otro lado, en las RIA no se producen recargas de página dado que toda la aplicación se carga desde el inicio y solo se genera tráfico de comunicación con el servidor el momento que se requieren datos externos ya sea de una base de datos o de otros ficheros externos.

* + - 1. **REST**
         1. **Servicio Web**

Se conoce como servicio Web a la interacción de las aplicaciones o tecnologías entre sí en una red con la finalidad de ofrecer un servicio;

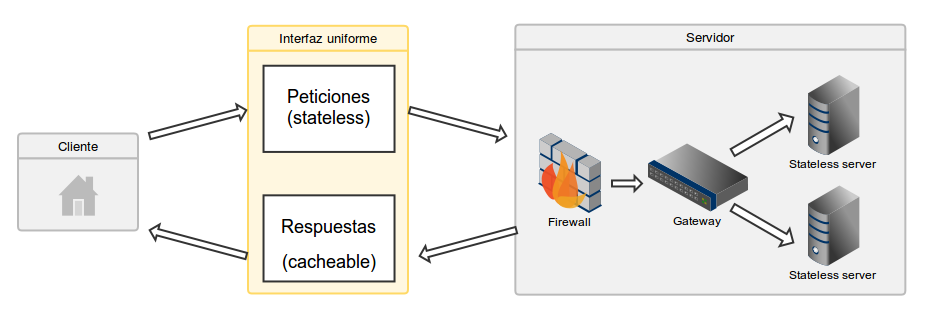
Con el objetivo de proveer información dinámica, ésta interacción trabaja sobre un modelo Cliente-Servidor mediante mensajes XML.

SOAP (Simple Object Access Protocol) es un estándar que define cómo dos objetos en diferentes procesos pueden comunicarse por medio de un intercambio de datos. [2]“Este protocolo deriva de un protocolo creado por [David Winer En](http://www.ecured.cu/index.php?title=David_Wineren&action=edit&redlink=1) [1998](http://www.ecured.cu/index.php/1998), llamado [XML-RPC](http://www.ecured.cu/index.php?title=XML-RPC&action=edit&redlink=1). SOAP fue creado por [Microsoft](http://www.ecured.cu/index.php/Microsoft).”   SOAP tiene la ventaja de que no está asociado a ningún Lenguaje, además tampoco está asociado a ningún protocolo de transporte.

Los mensajes en SOAP son transmitidos unidireccionalmente desde un emisor a un receptor, pero a medida que crecían las capacidades de SOAP también crecía su complejidad, muchas veces convirtiéndose en algo difícil de manejar haciendo que se implemente algo que no siempre es usado. A diferencia de ello REST da una propuesta más simple: no pretende dar soluciones a todo, eso evita la complejidad y evita implementar algo que no se vaya a usar.

REST (Representation State Transfer) es una de las arquitecturas más conocidas de Software que nació años atrás, pronto este se volvió popular porque presentaba un nuevo estilo en lo que se refiere servicios Web, se apoya en el estándar HTTP, para poder entenderlo mejor mencionaremos sus principales características:

* Arquitectura Cliente-Servidor: consiste en tener 2 agentes para el intercambio de información Cliente/ Servidor, estos tienen que ser totalmente independientes lo cual permite una flexibilidad alta.
* Stateless (Sin estado): El servidor no tiene que almacenar ningún dato del cliente para mantener un estado del mismo.
* Cacheable: El servidor debe definir un modo de “cachear” (almacenar temporalmente) las peticiones del cliente esto con el objetivo de aumentar rendimiento y escalabilidad, HTTP implemente en su cabecera un “Cache-control”  que dispone de parámetros para controlar la “cacheabilidad” de las respuestas.
* Sistema por capas: El cliente no debería tener conocimiento de por qué capas se transmite la información,  de esta manera el cliente mantiene independencia.
* Interfaz uniforme: Tener definida una interfaz uniforme ayuda en la escalabilidad debido a que la interfaz de comunicación entre Cliente-Servidor no es dependiente del servidor al cual se hacen las peticiones.



Para la comunicación, el propósito es simular las llamadas hechas por el protocolo HTTP, usando llamadas como ser GET, PUT, POST, DELETE. De esta manera se limita a interactuar con los recursos con estado, y no así con operaciones o mensajes.

|  |  |  |  |
| --- | --- | --- | --- |
| **Acción** | **HTTP** | **SQL** | **Copy&Paste** |
| Create | PUT | Insert | Pegar |
| Read | GET | Select | Copiar |
| Update | POST | Update | Pegar después |
| Delete | DELETE | Delete | Cortar |

El protocolo HTTP define claramente los términos de un servidor y un navegador. Esto permite que la implementación de estos sea independiente, basándose en el concepto cliente/servidor. Cuando utilizamos REST, HTTP no tiene estado. Cada mensaje contiene toda la información necesaria para comprender la petición cuando se combina el estado en el recurso. Como resultado, ni el cliente ni el servidor necesita mantener ningún estado en la comunicación. Cualquier estado mantenido por el servidor debe ser modelado como un recurso.

* + - 1. **MongoDB**.

MongoDB (de la palabra en inglés “humongous” que significa enorme) es un sistema de [base de datos](http://es.wikipedia.org/wiki/Base_de_datos) [NoSQL](http://es.wikipedia.org/wiki/NoSQL) orientado a documentos, desarrollado bajo el concepto de [código abierto](http://es.wikipedia.org/wiki/C%C3%B3digo_abierto).

MongoDB forma parte de la nueva familia de sistemas de base de datos NoSQL. En vez de guardar los datos en tablas como se hace en las base de datos relacionales, los guarda estructuras de datos en documentos tipo [JSON](http://es.wikipedia.org/wiki/JSON) con un esquema dinámico (que es llamado formato [BSON](http://es.wikipedia.org/wiki/BSON)), haciendo que la integración de los datos en ciertas aplicaciones sea más fácil y rápida.

Es una base de datos ágil que permite a los esquemas cambiar rápidamente cuando las aplicaciones evolucionan, proporcionando siempre la funcionalidad que los desarrolladores esperan de las bases de datos tradicionales, tales como índices secundarios, un lenguaje completo de búsquedas y consistencia estricta a continuación mencionamos sus características más importantes:

* Escalabilidad: tiene servidor aislado a arquitecturas distribuidas de grandes fragmentos que permite repartir, de manera transparente para el usuario, nuestra base de datos, esto aumenta el rendimiento del procesamiento de los datos.
* Marco de Agregación (Aggregation Framework): Procesamiento por lotes de datos para cálculos agrupados usando operaciones nativas de MongoDB.
* Auto-balanceado de carga: Maneja varios fragmentos de la DB. El balanceador decide cuándo y a qué fragmento migrar los datos, para que estén uniformemente distribuidos entre todos los servidores del racimo.
* Replicación nativa: sincronización de datos entre servidores
* Seguridad: autenticación, autorización
* Monitorización: [MongoDB Management Service](https://mms.mongodb.com/) (MMS) vigila continuamente la base de datos y los aspectos más importantes del hardware de nuestro servidor, a través de un panel Web configurable podemos seguir y optimizar nuestra base de datos
* Backup: Respaldo continuo con recuperación point-in-time de las distintas réplicas de fragmentos en el racimo, guarda múltiples copias de cada respaldo en distintos “data centers” (distribuidos geográficamente y tolerantes al fallo)
  1. **Metodología de desarrollo**
     1. **Metodologías Ágiles (Agile)**
        1. **Metodología Scrum**

Es una metodología de desarrollo de aplicaciones Informáticas, que se basa en el trabajo en equipo, la relación con el cliente y entregas periódicas.

"Scrum se basa en la teoría de control de procesos empírica o empirismo, la cual  asegura que el conocimiento procede de la experiencia y de tomar decisiones basándose en lo que se conoce. Scrum emplea un enfoque iterativo e incremental para optimizar la predictibilidad y el control del riesgo." (Schwaber y Sutherland, 2013)

* + - * 1. **Modelo de desarrollo de Scrum**

El proceso Scrum es iterativo e incremental, cada iteración se compone de varios ciclos de trabajo en los que se van incrementando funcionalidades o características al producto. La estructura de Scrum está conformada por:

**Roles**

Los roles en la metodología Scrum son:

* + Dueño del producto (Product Owner)

Es quien en representación de los clientes y usuarios debe enfocar y mantener una visión del producto. Es responsable de estimar los recursos para el proyecto, proveer la documentación relacionada con la Lista de Requerimientos del Producto (Product Backlog)

* + El equipo (Team)

Profesionales de distintas disciplinas conforman los equipos. El equipo debe ir convirtiendo la lista de requerimientos en funcionalidades. Un equipo debe tener Las siguientes características:

* Autosuficientes: no debe requerir participación de personas externas para poder operar sobre el proyecto.
* Auto-organizado: El equipo conoce la metodología y cada miembro es capaz de asumir responsabilidades según se haga necesario dentro del marco de trabajo.
* Funcionales: El equipo entiende los objetivos y requerimientos y va agregando características al producto de acuerdo al área de cada integrante.
  + Moderador (Scrum Master)

El moderador se ocupa de guiar el funcionamiento de la metodología Scrum adecuándola a las necesidades del proyecto. Las tareas de un Scrum Master son:

* Gestionar la lista de requisitos.
* Participar en las reuniones y orientar a los participantes y al equipo para que las reuniones sean provechosas.
* Guiar al equipo a lo largo del desarrollo.
* Generar un clima de trabajo adecuado.
* Lidiar con los problemas externos para que no afecten el ambiente interno.

**Artefactos**

La metodología Scrum utiliza los siguientes artefactos:

* + Lista de Requerimientos. Las características que posea el producto en el futuro y todas las tareas que deban asignarse y realizarse se encuentran en la Lista de Requerimientos del Producto (Product Backlog).

El Product Backlog está cambiando y creciendo constantemente ya que el cliente participa del equipo de desarrollo y su visión de los requisitos puede ir cambiando a lo largo del proceso de desarrollo, según el producto vaya tomando forma y el equipo, y él mismo, comprendan mejor el producto.

Las Historias de Usuario componen la Lista de Requerimientos, cada Historia de Usuario define un requerimiento y su importancia para el cliente. De acuerdo a la importancia se establece la prioridad con la que la historia de usuario será incluida en una iteración antes que otras.

* + Lista de Requerimientos de la Iteración (Sprint Backlog)

El Sprint Backlog incluye solo las historias de usuario a ser desarrolladas en cada iteración. Es un subconjunto del Product Backlog y debe incluir los requerimientos, la funcionalidad esperada y las tareas relacionadas a ser incluidas en la iteración actual.

Este documento debe cubrir los siguientes aspectos:

Definir todas las tareas que se van a realizar.

Las tareas deben tener una duración de entre 4 a 16 horas.

El equipo completo puede tener acceso a él.

* + Diagrama de Trabajo Realizado (Burn Down)

El diagrama Burn Down se va actualizando periódicamente y muestra la velocidad a la que se está avanzando en el trabajo y permite proyectar si el avance permitirá completar las tareas dentro del tiempo determinado.

* + Incremento (Increment)

Es el resultado de una iteración y debe ser un producto que funcione y se pueda entregar al cliente. Entonces el Incremento es una parte del producto que se ha terminado y probado en una iteración y es potencialmente entregable.

Actividades

La metodología Scrum contempla el siguiente conjunto de actividades y reuniones:

* + Planificación de la Iteración (Sprint Planning)

Consiste en definir y organizar las tareas que se llevaran a cabo en una iteración en particular. La reunión tiene dos etapas: en la primera etapa los miembros del equipo revisan y seleccionan las historias de usuario del Product Backlog que serán desarrolladas en la nueva iteración, haciendo modificaciones o aclaraciones según sea necesario. En la segunda etapa, se asignan las tareas y responsabilidades, se hace una estimación y evaluación de tiempos y riesgos para completar la lista de tareas.

* + Iteración (Sprint)

Una iteración debe tener una duración para su ejecución de entre dos semanas y un mes para ser útil y debe tener como resultado un producto que se pueda entregar al cliente.

* + Reunión diaria de Scrum (Scrum Daily Meeting)

Consiste en poner a los miembros del equipo al tanto de la situación del proyecto, todos deben conocer en qué se está trabajando y cuál es el objetivo, su duración debe ser de 15 a 20 minutos. Cada miembro deberá responder las siguientes preguntas:

¿Qué tareas realicé?

¿Qué problemas tuve?

¿Cuáles son mis tareas pendientes?

* + Demostración Final de la Iteración (Sprint Demo)

Se debe mostrar al cliente la nueva funcionalidad que se ha finalizado en la iteración.

* + Reunión Retrospectiva (Retrospective)

Se trata de repasar los errores y virtudes como experiencias para tomarlas en cuenta en iteraciones siguientes. Se comparan las estimaciones y los tiempos reales de las tareas, y si hubo elementos externos que afecten el desarrollo, si existen dificultades que no se habían tomado en cuenta. Si existen actividades o historias de usuario que no se pudieron completar, estas serán consideradas en la planificación en una iteración futura, y se les dará mayor prioridad con relación a las nuevas actividades de la siguiente iteración. Esta lista de actividades pendientes es llamada "Carry Over".

* + - * 1. **Integración Continua**

Se entiende por integración a la compilación y ejecución de pruebas de todo un proyecto. La Integración continua es una práctica de desarrollo de software que consiste en hacer integraciones automáticas de un proyecto lo más a menudo posible, para lo cual cada desarrollador de software sube su código trabajado a un repositorio compartido varias veces al día. Cada registro de código subido al repositorio es verificado por la máquina de compilación que permite verificar que el código es correcto y no rompe funcionalidades del producto, esto permite detectar tempranamente los problemas y fallos y así corregirlos cuanto antes.

Además permite generar pequeños productos entregables para el cliente final del producto.

La Integración continua tiene los siguientes beneficios:

* Mantener un único repositorio de código fuente.
* Automatizar la construcción del proyecto.
* Hacer que la construcción del proyecto ejecute sus propias pruebas de funcionalidad.
* Subir código al repositorio principal.
* Construir la línea principal en la máquina de integración.
* Mantener una ejecución rápida de la construcción del proyecto.
* Probar una réplica del producto en un entorno de producción.
* Hacer que todo el mundo pueda obtener el último producto ejecutable de forma fácil.
* Publicar lo que está pasando con el producto.
* Ejecución de pruebas unitarias
* Ejecución de pruebas de integración
* Ejecución de pruebas de aceptación
* Obtención de métricas de calidad de código y despliegues automáticos

Jenkins

Jenkins es un software de [Integración continua](http://es.wikipedia.org/wiki/Integraci%C3%B3n_continua) de código abierto escrito en [Java](http://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)), está basado en el proyecto [Hudson](http://es.wikipedia.org/wiki/Hudson_(software)) y liberado bajo licencia MIT. Es un sistema que corre en un servidor contenedor de servlets y soporta una variedad de herramientas de [control de versiones](http://es.wikipedia.org/wiki/Sistema_de_control_de_versi%C3%B3n) y puede ejecutar proyectos basados en [Apache Ant](http://es.wikipedia.org/wiki/Apache_Ant) y [Apache Maven](http://es.wikipedia.org/wiki/Apache_Maven), así como secuencias de comandos y programas por lotes de Windows. Jenkins proporciona integración continua para el desarrollo de software, es decir, permite configurar y agendar la ejecución de tareas conteniendo todas y/o cada una de las partes que conforman el ciclo de vida de un proyecto. Para dar una idea de esto, basta pensar en la famosa Generación Nocturna del Producto, la cual consiste en compilar, validar y liberar una versión beta que realizan algunos sistemas durante las noches de forma automática sobre lo que han ido trabajando los desarrolladores durante el día.

Es una pieza importante en lugares donde a veces una corrección de 5 minutos demora en ver la luz una semana debido a que tiene que pasar por toda una serie de procesos manuales de control, creación de instaladores y en su posterior distribución para los usuarios finales.

Por qué Usar Jenkins como máquina de compilación de código

* Es fácilmente accesible con un par de clics desde cualquier parte ya que se controla dentro de un marco Web.
* Simplifica significativamente los procesos involucrados en el ciclo de vida de un proyecto, ya sean  simples o repetitivos, largos y complejos como ser de generación de productos entregables integrando el trabajo de varios desarrolladores, ejecución de Pruebas, generación de reportes de estado, creación de instaladores, publicación en página web, notificación a clientes de versiones beta, etc. sin la necesidad y el tiempo de una persona.
* Da un marco de simplicidad sobre la integración continua gracias a las diferentes posibilidades de notificaciones de éxitos y errores entre procesos, por lo que con la cultura adecuada sobre los mismos se puede asegurar una altísima calidad de software desde el primer día, evitando errores, demoras y a un mínimo coste.
* Debido a los múltiples sistemas de notificaciones por cual se puede optar (ya sea por mail, chat, gtalk, twitter, etc. ), es sencillo estar siempre informado y poder tomar decisiones rápidamente

Control de Versiones

El control de versiones es un sistema que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo que se puedan recuperar versiones específicas más adelante.

[Sistemas de control de versiones locales](http://git-scm.com/book/es/v1/Empezando-Acerca-del-control-de-versiones#Sistemas-de-control-de-versiones-locales) (VCSs) [Sistemas de control de versiones locales](http://git-scm.com/book/es/v1/Empezando-Acerca-del-control-de-versiones#Sistemas-de-control-de-versiones-locales) (VCSs)

Para Chacon, S., & Straub (2014)

Un método de control de versiones usado por mucha gente es copiar los archivos a otro directorio (quizás indicando la fecha y hora en que lo hicieron). Este enfoque es muy común porque es muy simple, pero también tremendamente propenso a errores. Es fácil olvidar en qué directorio te encuentras, y guardar accidentalmente en el archivo equivocado o sobrescribir archivos que no querías.

Para hacer frente a este problema, los programadores desarrollaron hace tiempo VCSs locales que contenían una simple base de datos en la que se llevaba registro de todos los cambios realizados sobre los archivos (véase Figura 1-1).

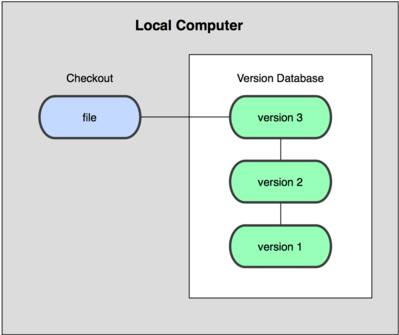


Figura 1-1. Diagrama de control de versiones local.

Una de las herramientas de control de versiones más popular fue un sistema llamado RCS (Revision Control System), que todavía podemos encontrar en muchos de los ordenadores actuales. Hasta el famoso sistema operativo Mac OS X incluye el comando RCS cuando se instalan las herramientas de desarrollo. Esta herramienta funciona básicamente guardando conjuntos de parches (es decir, las diferencias entre archivos) de una versión a otra en un formato especial en disco; puede entonces recrear cómo era un archivo en cualquier momento sumando los distintos parches. (p.28)

[Sistemas de control de versiones centralizados](http://git-scm.com/book/es/v1/Empezando-Acerca-del-control-de-versiones#Sistemas-de-control-de-versiones-centralizados) (CVCSs) [Sistemas de control de versiones centralizados](http://git-scm.com/book/es/v1/Empezando-Acerca-del-control-de-versiones#Sistemas-de-control-de-versiones-centralizados) (CVCSs).

Para Chacon, S., & Straub (2014)

El siguiente gran problema que se encuentra la gente es que necesitan colaborar con desarrolladores en otros sistemas. Para solucionar este problema, se desarrollaron los sistemas de control de versiones centralizados Estos sistemas, como CVS, Subversión, y Perforce, tienen un único servidor que contiene todos los archivos versionados, y varios clientes que descargan los archivos desde ese lugar central. Durante muchos años éste ha sido el estándar para el control de versiones (véase Figura 1-2).

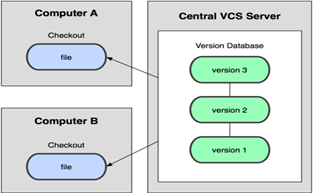


Figura 1-2. Diagrama de control de versiones centralizado.

Esta configuración ofrece muchas ventajas, especialmente frente a VCSs locales. Por ejemplo, todo el mundo puede saber (hasta cierto punto) en qué están trabajando los otros colaboradores del proyecto. Los administradores tienen control detallado de qué puede hacer cada uno; y es mucho más fácil administrar un CVCS que tener que lidiar con bases de datos locales en cada cliente.

Sin embargo, esta configuración también tiene serias desventajas. La más obvia es el punto único de fallo que representa el servidor centralizado. Si ese servidor se cae durante una hora, entonces durante esa hora nadie puede colaborar o guardar cambios versionados de aquello en que están trabajando. Si el disco duro en el que se encuentra la base de datos central se corrompe, y no se han llevado copias de seguridad adecuadamente, se pierde absolutamente todo —toda la historia del proyecto salvo aquellas instantáneas que la gente pueda tener en sus máquinas locales. Los VCSs locales sufren de este mismo problema— cuando se tiene toda la historia del proyecto en un único lugar, el riesgo es perderlo todo. (p.29)

[Sistemas de control de versiones distribuidos](http://git-scm.com/book/es/v1/Empezando-Acerca-del-control-de-versiones#Sistemas-de-control-de-versiones-distribuidos)(DVCSs)

Para Chacon, S., & Straub (2014)

En un Sistema de Control de Versiones Distribuido (Distributed Version Control Systems o DVCSs en inglés) (como GIT, Mercurial, Bazaar o Darcs), los clientes no sólo descargan la última instantánea de los archivos: replican completamente el repositorio. Así, si un servidor muere, y estos sistemas estaban colaborando a través de él, cualquiera de los repositorios de los clientes puede copiarse en el servidor para restaurarlo. Cada vez que se descarga una instantánea, en realidad se hace una copia de seguridad completa de todos los datos (véase Figura 1-3).

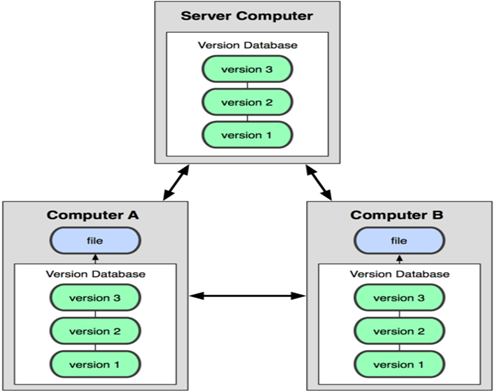


Figura 1-3. Diagrama de control de versiones distribuido.

Muchos de estos sistemas se las arreglan bastante bien teniendo varios repositorios con los que trabajar, por lo que puedes colaborar con distintos grupos de gente simultáneamente dentro del mismo proyecto. Esto te permite establecer varios flujos de trabajo que no son posibles en sistemas centralizados, como pueden ser los modelos jerárquicos. (p.30)

GIT

GIT es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente. Al principio, GIT se pensó como un motor de bajo nivel sobre el cual otros pudieran escribir la interfaz de usuario o front end como Cogito o StGIT. Sin embargo, GIT se ha convertido desde entonces en un sistema de control de versiones con funcionalidad plena. Hay algunos proyectos de mucha relevancia que ya usan GIT, en particular, el grupo de programación del núcleo Linux.

Características de GIT

Fuerte apoyo al desarrollo no lineal, por ende rapidez en la gestión de ramas y mezclado de diferentes versiones. GIT incluye herramientas específicas para navegar y visualizar un historial de desarrollo no lineal. Una presunción fundamental en GIT es que un cambio será fusionado mucho más frecuentemente de lo que se escribe originalmente, conforme se pasa entre varios programadores que lo revisan.

Gestión distribuida. Al igual que Darcs, BitKeeper, Mercurial, SVK, Bazaar y Monotone, GIT le da a cada programador una copia local del historial del desarrollo entero, y los cambios se propagan entre los repositorios locales. Los cambios se importan como ramas adicionales y pueden ser fusionados en la misma manera que se hace con la rama local.

Los almacenes de información pueden publicarse por HTTP, FTP, rsync o mediante un protocolo nativo, ya sea a través de una conexión TCP/IP simple o a través de cifrado SSH. GIT también puede emular servidores CVS, lo que habilita el uso de clientes CVS pre-existentes y módulos IDE para CVS pre-existentes en el acceso de repositorios GIT.

Los repositorios Subversion y svk se pueden usar directamente con GIT-SVN.

Gestión eficiente de proyectos grandes, dada la rapidez de gestión de diferencias entre archivos, entre otras mejoras de optimización de velocidad de ejecución.

Todas las versiones previas a un cambio determinado, implican la notificación de un cambio posterior en cualquiera de ellas a ese cambio (denominado autenticación criptográfica de historial).

Resulta algo más caro trabajar con ficheros concretos frente a proyectos, eso diferencia el trabajo frente a CVS, que trabaja con base en cambios de fichero, pero mejora el trabajo con afectaciones de código que concurren en operaciones similares en varios archivos.

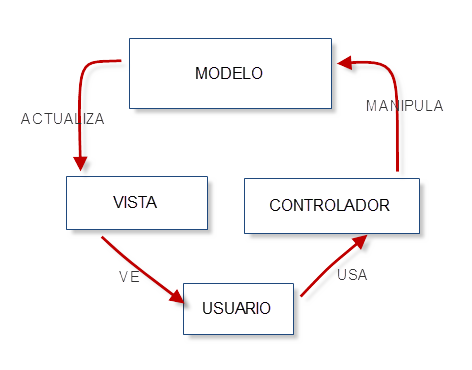
Los renombrados se trabajan basándose en similitudes entre ficheros, aparte de nombres de ficheros, pero no se hacen marcas explícitas de cambios de nombre con base en supuestos nombres únicos de nodos de sistema de ficheros, lo que evita posibles, y posiblemente desastrosas, coincidencias de ficheros diferentes en un único nombre.

Re-almacenamiento periódico en paquetes (ficheros). Esto es relativamente eficiente para escritura de cambios y relativamente ineficiente para lectura si el reempaquetado (con base en diferencias) no ocurre cada cierto tiempo.

* + - 1. Arquitectura
         1. El patrón Modelo-Vista-Controlador El patrón Modelo-Vista-Controlador

El patrón de arquitectura de software “Modelo-Vista-Controlador” es usado para implementar interfaces de usuario. Consiste en dividir una aplicación de software en tres partes conectadas una con la otra, pero que a su vez separan las representaciones internas de la información de la forma en que ésta información es presentada al usuario.

El patrón del Modelo-Vista-Controlador, así como otros patrones de software, se enfoca en el "núcleo de la solución" de un problema permitiendo al mismo tiempo que se pueda adaptar a cada sistema. Una implementación de la arquitectura MVC en un caso particular puede llegar a ser significativamente diferente de la definición que se acaba de presentar.



La parte central en este patrón es el Modelo, el cual se encarga de manejar los datos, la lógica y las reglas de la aplicación, de forma independiente a la interfaz de usuario.

La Vista es la parte encargada de la representación de la información, como por ejemplo, tablas o cuadros que son diferentes formas de mostrar la misma información.

La otra parte es el Controlador, encargada de recibir instrucciones y enviar los comandos al Modelo.

* + 1. Plataforma de desarrollo
       1. EmberJS

Es un marco de aplicaciones web de código abierto para el lado del cliente en JavaScript basado en el patrón de arquitectura Modelo Vista Controlador permite a los desarrolladores crear aplicaciones escalables de página única.

Se basa en que “el poder de la Web radica en su habilidad de referenciar y compartir URLs” [1], considera que las URLs son la característica clave que da a las aplicaciones web la facilidad de compartir y colaborar.

* + - * 1. Conceptos Básicos
* Templates  
  Un template describe la interfaz de usuario de la aplicación y está respaldado por un modelo, así el template se actualiza automáticamente si es que el modelo cambia.
* Router  
  El router traduce las URLs en una serie de templates anidados, cada uno respaldado por un modelo. si el template o el modelo que se muestran al usuario cambiarán, las URLs se mantendrán actualizadas automáticamente.
* Componentes  
  Un componente es una etiqueta HTML personalizada cuyo comportamiento es implementado usando JavaScript, permite crear controles reutilizables que pueden simplificar los templates.
* Modelos  
  Un modelo es un objeto que almacena un estado persistente. los templates son los responsables de mostrar el modelo al usuario al convertirlo en HTML.
* Ruta  
  Una ruta es un objeto que le indica al template que modelo es el que debe mostrar.
* Controladores  
  Un controlador es un objeto que almacena el estado de la aplicación. un template opcionalmente puede tener un controlador además de un modelo y puede recuperar sus propiedades de ambos.
  + - 1. Node JS

JavaScript fue creado para funcionar en los navegadores del lado del cliente, pero su rendimiento y características únicas lo llevaron de estar del lado cliente hacia el servidor, Node JS tiene como propósito ejecutar código Java script del lado del servidor

Node JS creado en 2009 por Ryan Dahl, es un intérprete de Java Script del lado del servidor, tiene como núcleo de su arquitectura motor JavaScript V8, su meta es crear aplicaciones altamente escalables y escribir código que maneje grandes cantidades de conexiones simultáneas en una sola máquina física.

* + - 1. Programación asíncrona.

NodeJS fue pensado desde el primer momento para potenciar los beneficios de la programación asíncrona, La programación asíncrona da la posibilidad de que algunas operaciones devuelvan el control al programa llamante antes que hayan terminado mientras siguen operando en segundo plano.

Hacer que los programas no bloqueen la línea de ejecución de código con respecto a entradas y salidas, de modo que los ciclos de procesamiento se queden disponibles cuando se está esperando a que se completen tales acciones. Esto se implementa a partir de "callbacks".

Programación orientada por eventos. A muchos programadores se les ha hecho creer que la programación orientada a objetos es el diseño perfecto de programación y que no deben usar nada más. Node JS utiliza lo que se conoce como modelo de programación orientado por eventos, es por eso que debido a que no hay estructuras complicadas Orientadas a Objetos, tampoco hay interfaces, simplemente esperar un evento escribir y una función para la devolución del llamado, a largo plazo esto es lo que lo vuelve más fácil de codificar y de mantener.

* + - 1. Express JS

Express JS, es un framework de desarrollo de aplicaciones web minimalista y flexible para Nodejs. Está inspirado en [Sinatra](http://www.sinatrarb.com/), además es robusto, rápido, flexible y muy simple. Entre otras características, ofrece Router de URL (Get, Post, Put, Delete)

1. **IMPLEMENTACION DEL ENTORNO DE DESARROLLO**
   1. **Introducción**  
      Un entorno de desarrollo es un conjunto de recursos necesarios para, precisamente, desarrollar el proyecto, estos recursos incluyen: los programas, hardware y software, servidores, etc. Que son herramientas necesarias para poder llevar a cabo las tareas.
   2. **Equipo del Proyecto**

EL equipo de trabajo está compuesto por las siguientes entidades:

* Cliente: Colegio EPA
* Programadores:
  + Luis Fernando Camacho Camacho
  + Leonardo Mendieta Azcui
  + Elmer Alvarado Ponce
* Pruebas:
  + Luis Fernando Camacho Camacho
  + Leonardo Mendieta Azcui
  + Elmer Alvarado Ponce
  1. **Duración del Proyecto**

Para completar los requerimientos identificados, se trabajará en iteraciones de dos semanas, por el tiempo de duración del proyecto, se completarán 5 iteraciones, en la Figura 4.1 se puede ver el cronograma que se seguirá para el Sprint Backlog.

* + 1. **Instalación de ambientes**

Uno de los hitos marcados para el desarrollo del sistema es el corresponde a la instalación del ambiente de desarrollo y a la investigación de las tecnologías a ser aplicadas, durante la instalación de ambientes se han identificado las siguientes tareas, como se puede ver en tabla 4.2.

.

|  |  |
| --- | --- |
| **Recurso** | **Tareas** |
|
| Instalar controlador de versiones | Implementar controlador de versiones |
| Revisión y actualización del documento |
| Crear interface de usuario con EmberJS | Crear controles de UI |
| Revisar ejemplos |
| Implementar REST Service | Crear documento técnico |
| Definir la Base de Datos |
| Implementar el servicio REST |
|  |
| Investigar EmberJS, el patrón MVC y RIA | Crear documento técnico |
| Investigar EmberJS |
|  |
| Investigar sobre integración continua | Crear documento técnico |
| Investigar integración continua |
| Investigar REST, MongoDB, NodeJS, ExpressJS | Crear documento técnico |
| Investigar MongoDB |
| Investigar NodeJS, ExpresJS |

## Asignación de las tareas

Una vez revisada la lista de tareas requeridas para la instalación del ambiente de trabajo y la investigación sobre las tecnologías a ser utilizadas, las tareas son revisadas por el equipo y aceptadas para su realización.

* + - * 1. **Tarea #1**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Recurso para Ambiente de trabajo** | | | | | |
| **Número:** | 1 | **Nombre:** | Instalar controlador de versiones | | |
| **Prioridad:** | Alta |  |  | | |
| **Descripción:** | | | | | |
| Instalar un controlador de versiones que pueda hacer lo siguiente:   * Verificar que el código no este roto * Enviar Correos en caso de que falle * Compilar Código Manera Automática (Configurar a cierta hora) | | | | | |
| **Criterio de Aceptación:** | | | | | |
| Se dará como completada la tarea. al entregar un versionador de código funcionando, con la debida documentación correspondiente | | | | | |
| **Verificación y pruebas** | | | | | |
| Verificación de la instalación del controlador de versiones | | | | | |
| **Asignacion:** |  | | | **Estado:** |  |

* + - * 1. **Tarea #2**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Recurso para Ambiente de trabajo** | | | | | |
| **Número:** | 2 | **Nombre:** | Crear interface de usuario con EmberJS | | |
| **Prioridad:** | Alta |  |  | | |
| **Descripcion:** | | | | | |
| Implementar controles de UI usando EmberJs usando MVC. Se deben de crear los sigtes. controles:  Controles Básicos: • Buttons • Radio Buttons • Check box • Text Field • Text Area • Links • Label Controles Complejos: • List • Menus • Tables | | | | | |
| **Criterio de Aceptación:** | | | | | |
| Se dará como completada la tarea. al entregar un versionador de código funcionando, con la debida documentación correspondiente | | | | | |
| **Verificación y pruebas** | | | | | |
| Se hacen pruebas para cada control a ser utilizado de acuerdo a ejemplos de código. | | | | | |
| **Asignacion:** |  | | | **Estado:** |  |

* + - * 1. **Tarea #3**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Recurso para Ambiente de trabajo** | | | | | |
| **Número:** | 3 | **Nombre:** | Implementar REST Service | | |
| **Prioridad:** | Alta |  |  | | |
| **Descripción:** | | | | | |
| Implementar un REST service usando Express JS, en un proyecto de prueba donde el proyecto deberá contemplar el manejo de Dase de Datos. En el caso de Mongodb se deberá tomar los siguientes criterios de relación:  Relación 1 - 1 Relación 1 - n Relación n - n se recomienda como caso de uso 3 tablas en la base de Datos. | | | | | |
| **Criterio de Aceptación:** | | | | | |
| entregar un proyecto de prueba funcionando que tenga implementado un "REST service" documentado, y que contenga lo mencionado en la descripción | | | | | |
| **Asignacion:** |  | | | **Estado:** |  |

* + - * 1. **Tarea #4**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Recurso para Ambiente de trabajo** | | | | | |
| **Número:** | 4 | **Nombre:** | Investigar EmberJS, el patrón MVC y RIA | | |
| **Prioridad:** | Alta |  |  | | |
| **Descripción:** | | | | | |
| Investigar la tecnología Ember Js para el manejo de controles de UI, además del patrón MVC con Ember Js además de investigar que es RIA. | | | | | |
| **Criterio de Aceptación:** | | | | | |
| Presente un Doc. técnico que contenga mencionado en la descripción, con referencias | | | | | |
| **Asignacion:** |  | | | **Estado:** |  |

* + - * 1. **Tarea #5**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Recurso para Ambiente de trabajo** | | | | | |
| **Número:** | 5 | **Nombre:** | investigar sobre integración continua | | |
| **Prioridad:** | Alta |  |  | | |
| **Descripción:** | | | | | |
| Investigar el uso de un servidor para versionar el código fuente (Build machine) y seguir el proceso de integración continua, de tal manera que este de alertas diarias sobre el estado del código, en caso que el código este roto mandar un mail de manera oportuna | | | | | |
| **Criterio de Aceptación:** | | | | | |
| Entrega de un Documento técnico explicando los detalles de la Investigación. | | | | | |
| **Asignación:** |  | | | **Estado:** |  |

* + - * 1. **Tarea #6**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Recurso para Ambiente de trabajo** | | | | | |
| **Número:** | 6 | **Nombre:** | Investigar REST, MongoDB, NodeJS, ExpressJS | | |
| **Prioridad:** | Alta |  |  | | |
| **Descripción:** | | | | | |
| investigar las tecnologías Express Js, Node Js y Mongo DB, describir y como inter actúan entre ellas | | | | | |
| **Criterio de Aceptación:** | | | | | |
| con la entrega de un documento técnico que contemple toda la información descrita | | | | | |
| **Asignacion:** |  | | | **Estado:** |  |

## Scrum Daily Meeting

La metodología seleccionada recomienda tener reuniones diarias para el equipo involucrado en el proyecto, pero por algunas restricciones en los horarios de los integrantes, las reuniones se programaron para los días miércoles y domingos, de 17:00 a 19:00. De acuerdo a la metodología, en la reunión cada uno de los integrantes debe responder a las siguientes preguntas:

* ¿En qué tareas estuve trabajando?
* ¿Tuve algún problema?
* ¿En qué tareas trabajaré hasta la siguiente reunión?

Estas reuniones permiten al equipo hacer el seguimiento del progreso de las tareas y poder identificar problemas que surgen y no fueron previstos durante la planificación, y pueden causar que algunas tareas no puedan completarse según lo planeado.

Para esta etapa no se transcribieron las reuniones, pero a continuación se detallan algunos puntos sobresalientes:

* Escasa documentación sobre EmberJS en la Web, fue necesario consultar con un experto.
* Falta de consenso en el uso de herramientas para compartir información, Word es más cómodo de usar, pero Docs permite la edición colaborativa en tiempo real.

## Demo

En esta etapa el demo fue con respecto al funcionamiento del ambiente de desarrollo y correcta operación de las herramientas de versionamiento y servicios, por lo que solo fue de interés del equipo de desarrollo.

## Retrospectiva

Habiéndose completado la instalación del ambiente de trabajo se realiza una reunión de retrospectiva en la cual se consideran algunos aspectos que pueden ayudar al equipo a mejorar los procesos, entre estos puntos se observan las cosas que salieron bien o fueron positivas para repetirlas o si se puede mejorarlas, las cosas que no salieron bien o fueron negativas, para buscar soluciones y aplicarlas:

* + - 1. **Qué salió bien**
* El equipo de trabajo ha empezado a tomar ritmo en el avance de las tareas asignadas.
* (Se completaron las tareas programadas, en prácticamente su totalidad)
* Se completó la implementación del controlador de versiones
* Se completó la implementación del servicio de integración continua
* Se completó la implementación del servicio REST.
* Se completaron las tareas de investigación para familiarizar al equipo con las tecnologías
* Se completó la implementación de funcionalidad del registro de estudiantes.
  + - 1. **Qué no salió bien**
* La tecnología propuesta para el desarrollo del sistema (EmberJS) es nueva y el equipo no tiene experiencia en su uso, el proceso de aprendizaje inicial fue autodidacta mediante investigaciones en la web, lo cual tomo más tiempo del previsto para esta etapa.
* El progreso en la elaboración de la documentación del proyecto es lenta debido a la necesidad de constantes revisiones y actualizaciones
* Existieron dificultades de comunicación debido a falta de coordinación en horarios en que los miembros el equipo estén disponibles.
* + - 1. **Mejoras a implementar en la próxima iteración**
* Tomar en cuenta el nivel de conocimiento en las tecnologías y cuando sea necesario buscar ayuda de expertos lo antes posible.
* Incluir en la planificación más tiempo para investigación y entrenamiento, de modo que las tareas programadas no queden incompletas
* Fomentar una comunicación más fluida entre miembros del equipo.

1. **CAPITULO 5: REGISTRO DE PROFESORES, ALUMNOS Y GENERACIÓN DE HORARIOS**
   1. **Introducción.**

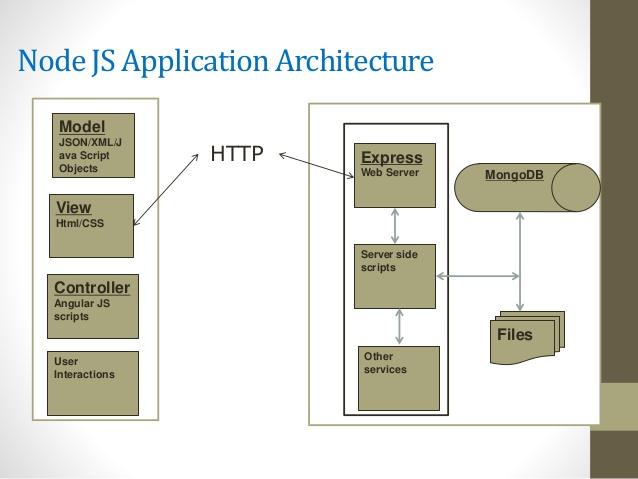
Para el desarrollo de este proyecto se tomó la decisión de usar Scrum como metodología de desarrollo, siguiendo ésta metodología se debe comenzar por escribir la lista de requerimientos y ponerlos en un Product Backlog. Después El Administrador del proyecto que en este caso es el dueño de la EPA escoge y da prioridad para este sprint a los requerimientos más importantes según las necesidades de su negocio.

Posteriormente se procede a convertir cada requerimiento en una o varias historias de usuario, hacer un poker planning para poder estimar el esfuerzo de cada historia de usuario y escoger la cantidad de historias de usuario a implementar para esta iteración, para después proceder a seguir las siguientes prácticas por cada historia de usuario:

* Análisis
* Diseño
* Implementación
* Pruebas
  1. **Arquitectura del sistema**

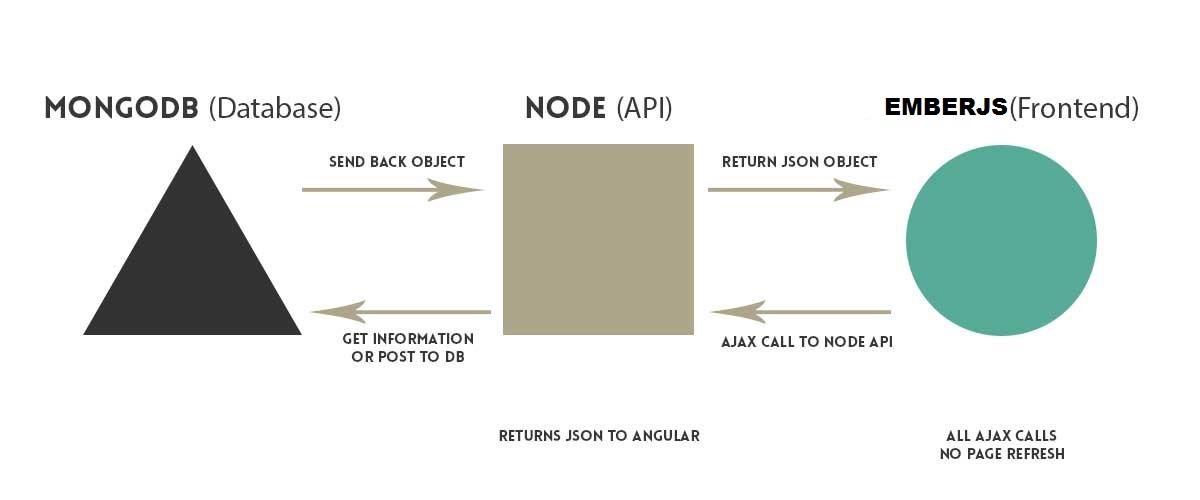
El Sistema de información para la automatización de los procesos administrativos de la Escuela Popular de Artes será implementado usando tecnologías web 2.0, para eso se procederá a utilizar la arquitectura Cliente-Servidor y seguir el patrón Modelo-Vista-Controlador en la aplicación del cliente para mejorar la cohesión y reducir el acoplamiento, lo cual permite al sistema ser mantenible y legible.

Además de un Servicio REST, el cual tiene la responsabilidad de ejecutar operaciones CRUD para la lógica del negocio.



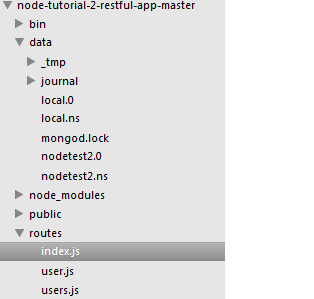
Las tecnologías elegidas para la implementación del servicio REST y la aplicación del cliente se muestran a continuación:

* 1. **Servicio REST**



Para la implementación del sistema de información se decidió usar ExpressJs como framework para el desarrollo del Servicio REST que de manera sencilla nos permite la creación de URIs y su respectiva publicación para las operaciones CRUD de la lógica del negocio en formato JSON. Además de éste, usaremos una librería para conectarnos y crear una base de datos no-relacional implementada en mongodb, que nos permite crear de manera dinámica la base de datos en formato JSON y poder cambiar dinámicamente según nuestros requerimientos sin ninguna pérdida de datos y ninguna ejecución de scripts extras de base de datos después de algún tipo de modificación en la estructura de datos.

Todo el servicio REST será implementado en el lenguaje javascript y la estructura de directorio del servicio REST será la siguiente:



* + 1. **Cliente**

Para la implementación del sistema información el grupo decidió usar tecnologías más recientes (desarrolladas estos 2 últimos años) basadas en web 2.0 utilizando Nodejs y Expressjs para la parte del servidor del sistema de información y como administrador de comunicaciones realizadas entre el servidor y el cliente se usará la librería EmberJs, que permite manejar el patrón MVC usando javascript como lenguaje de programación, además de librerías *“calendar extensible 1.5”* que permite mostrar el calendario de clases por estudiante y profesor de nuestro sistema de información, y usar CSS3 para los estilos de las vistas del sistema de información.

* 1. **Equipo del Proyecto (esto ya se menciona en el punto 4.2.)**

EL equipo de trabajo está compuesto por las siguientes entidades:

* Cliente: Colegio EPA
* Programadores:
  + Luis Fernando Camacho Camacho
  + Leonardo Mendieta Azcui
  + Elmer Alvarado Ponce
* Pruebas:
  + Luis Fernando Camacho Camacho
  + Leonardo Mendieta Azcui
  + Elmer Alvarado Ponce
  1. Tipos de Usuario

El sistema cuenta con los siguientes tipos de usuarios:

* Profesores
* Administrador
* Contador
* Secretaria
* Propietario
  1. **Product Backlog**

Después de tener una reunión con el Administrador General y el Propietario de la EPA se logra obtener la información necesaria para definir la lista de los requerimientos que son necesarios para tener el sistema de información para la EPA.

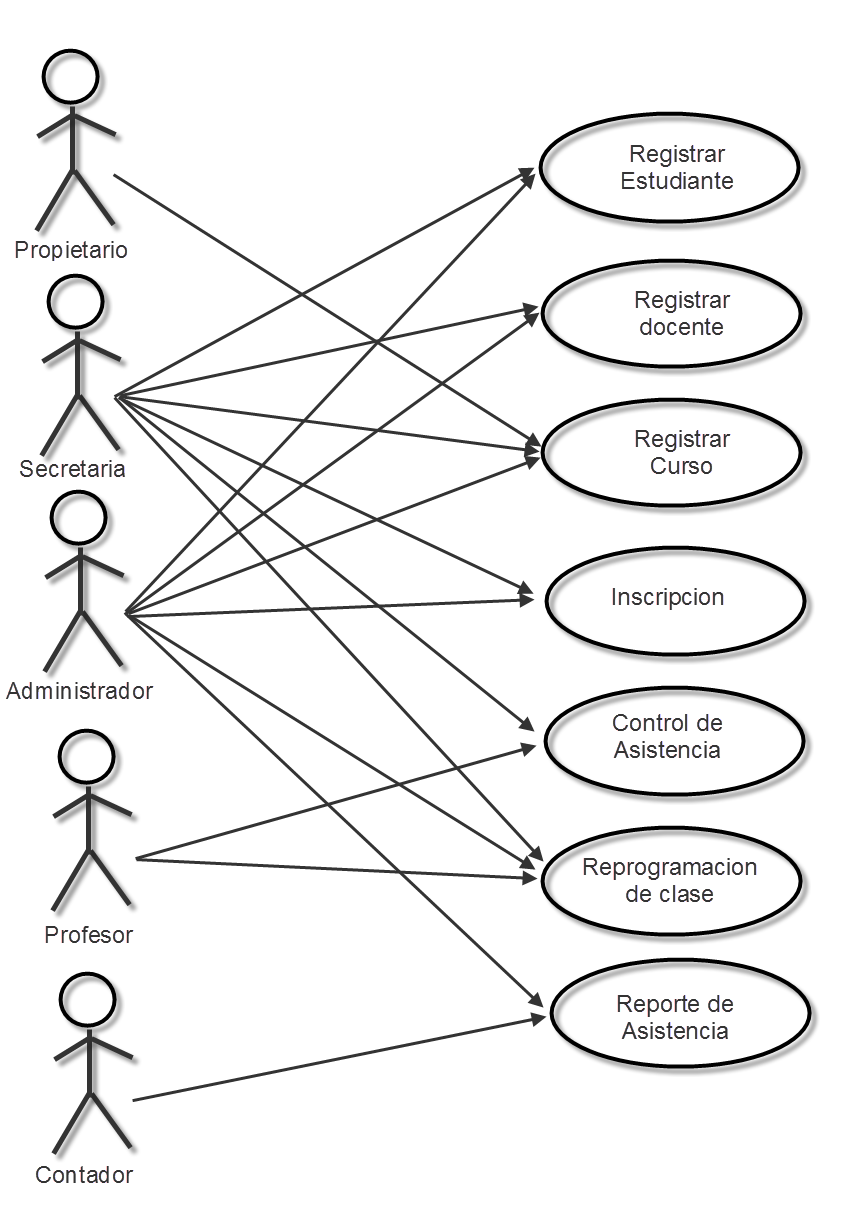
La Lista de requerimientos son los siguientes:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | prioridad | Tiempo estimado(Horas) | Descripción | Criterio de validación |
| REQ1 | ALTA |  | Registro de estudiantes | Se debe poder registrar a un nuevo estudiante o modificar los datos de uno ya registrado |
| REQ2 | ALTA |  | registro de docentes | Se debe poder registrar a un nuevo docente o modificar los datos de uno ya registrado |
| REQ3 | ALTA |  | Inscripciones | Se debe poder inscribir a un estudiante (nuevo o ya registrado) en un curso |
| REQ4 | ALTA |  | Control de asistencia | se debe poder marcar una clase como avanzada |
| REQ5 | ALTA |  | Reprogramación de clases | se debe poder modificar la fecha y hora de una clase |
| REQ6 | ALTA |  | Reportes de asistencia | Se debe tener reportes de asistencia a clases por estudiantes y por Profesores |
| REQ7 | ALTA |  |  |  |
| REQ8 | ALTA |  |  |  |

Tabla 4.1: Product Backlog

Fuente: PRIOLO Sebastián, 2009

Se confecciona también el siguiente diagrama de casos de uso para una mayor comprensión de la funcionalidad que debe ofrecer el sistema:



* + 1. **Sprint Planning** – **Primera iteración** - **Registro de estudiantes, profesores y generación de horarios**

Siguiendo el cronograma del Product Backlog, la Primera iteración corresponde al Registro de estudiantes, Profesores y la generación de horarios de clases, durante la planificación de la primera iteración se han identificado las siguientes historias de usuario, como se puede ver en tabla 4.2.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Backlog ID** | **Historia de Usuario** | **Tareas** | **Tipo** | **Estimación**  **(horas)** |
| REQ2 | US1 - Desarrollo del formulario de Registro de estudiantes | Definir los campos requeridos para registrar un estudiante | Análisis | 2 |
| Crear el formulario de registro de estudiante | Diseño y Desarrollo | 4 |
| Prueba del formulario de registro de estudiante | Prueba | 2 |
| US2 - Almacenamiento de los datos del estudiante | Crear la estructura de las base de datos | Diseño y Desarrollo | 2 |
| Implementar la funcionalidad del botón Registrar | Desarrollo | 6 |
| Implementar la funcionalidad del botón Cancelar | Desarrollo | 2 |
| Implementar la funcionalidad del botón Limpiar | Desarrollo | 2 |
| Prueba del botón Registrar para registro de estudiante | Prueba | 2 |
| Prueba del botón Cancelar para registro de estudiante | Prueba | 2 |
| Prueba del botón Limpiar para registro de estudiante | Prueba | 2 |
| US3 - Modificación de datos del estudiante | Implementar la funcionalidad del botón Guardar | Desarrollo | 8 |
| Implementar la funcionalidad del botón Cancelar | Desarrollo | 2 |
| Prueba del botón Editar para estudiantes | Prueba | 2 |
| Prueba del botón Cancelar para la modificación de datos del estudiante | Prueba | 2 |
| US4 - Eliminación de un estudiante | Implementar la funcionalidad del botón Eliminar | Desarrollo | 4 |
| Implementar la funcionalidad de cancelar eliminación | Desarrollo | 2 |
| Prueba de la eliminación de un estudiante | Prueba | 2 |
| Prueba de mensaje de información por no seleccionar un estudiante | Prueba | 1 |
| Prueba de la cancelación de eliminación de un estudiante | Prueba | 1 |
| US5 - Crear guía de usuario para registro de estudiantes | Documentar el registro, edición y eliminación de un estudiante | Documentación | 2 |
| Revisar la guía de usuario para para agregar un estudiante | Prueba | 2 |

## 

## Historias de Usuario

Una vez revisada la lista del backlog, al planificar la Segunda iteración, se detallan las historias de usuario correspondientes al registro de estudiantes, y las tareas relacionadas a cada una de éstas, a continuación se detallarán cada historia de usuario y sus tareas.

El esfuerzo de 1 punto = 1 día ideal de programación

1 historia de usuario = de 2 – 5 puntos (generalmente)

* + - * 1. **Historia de Usuario #1**

**Tarea#**

**Tarea #**

**Tarea #**

**Prueba #**

**Prueba #**

* + - * 1. **Historia de Usuario #2**

**Tarea #**

**Tarea #**

**Prueba #**

**Prueba #**

**Prueba #**

* + - * 1. **Historia de Usuario #3**
        2. **Historia de Usuario #**
        3. **Historia de Usuario #**
        4. **Historia de Usuario #**

## Scrum Daily Meeting

Las reuniones del equipo se programaron de forma más regular, utilizando llamadas y chat como herramientas para poder hacer el seguimiento del progreso de las tareas.

## Sprint Demo

La demostración de avances para la segunda iteración consiste principalmente en el funcionamiento básico del formulario de registro de estudiantes, se realiza con una transferencia de datos a un repositorio alternativo hasta poder completar la implementación del servicio REST.

## Retrospectiva

Cuando se ha completado la iteración se realiza una reunión de retrospectiva en la cual se consideran algunos aspectos que pueden ayudar al equipo a mejorar los procesos para la siguiente iteración, entre estos puntos se observan las cosas que salieron bien o fueron positivas para repetirlas o si se puede mejorarlas, las cosas que no salieron bien o fueron negativas, para buscar soluciones y aplicarlas:

* + 1. **Que salió bien**
* Las tareas asignadas se han completado satisfactoriamente
* Se completó la implementación del formulario de registro de estudiantes
* Se completó la implementación del formulario de registro de profesores
* Se pudo conectar con la base de datos y validar los servicios (REST) y el funcionamiento de acuerdo al patrón MVC
  + 1. **Qué no salió bien**
* La tecnología es complicada y requirió una reorganización de los archivos y las referencias, fue necesario consultar con un experto externo para completar la configuración
* + 1. **Mejoras a implementar en la próxima iteración**
* Fomentar una comunicación más fluida entre miembros del equipo.
* Fomentar la puntualidad en la asistencia a las reuniones

1. **CAPITULO 6 CREAR  EL MÓDULO DEL CONTROL DE ASISTENCIA DE ESTUDIANTES Y PROFESOR**
   1. **Introducción**
   2. **Análisis**
   3. **Diseño**
   4. **Implementación**
   5. **Pruebas**
   6. **Product backlog (revision)**
   7. **Sprint Planning**
   8. **Introducción**

Para el desarrollo de este proyecto se tomó la decisión de usar Scrum como metodología de desarrollo, siguiendo ésta metodología se debe comenzar por escribir la lista de requerimientos y ponerlos en un Product Backlog. Después El Administrador del proyecto que en este caso es el dueño de la EPA escoge y da prioridad para este sprint a los requerimientos más importantes según las necesidades de su negocio.

Posteriormente se procede a convertir cada requerimiento en una o varias historias de usuario, hacer un poker planning para poder estimar el esfuerzo de cada historia de usuario y escoger la cantidad de historias de usuario a implementar para esta iteración, para después proceder a seguir las siguientes prácticas por cada historia de usuario:

* Análisis
* Diseño
* Implementación
* Pruebas
  1. **Arquitectura del sistema**

El Sistema de información para la automatización de los procesos administrativos de la Escuela Popular de Artes será implementado usando tecnologías web 2.0, para eso se procederá a utilizar la arquitectura Cliente-Servidor y seguir el patrón Modelo-Vista-Controlador en la aplicación del cliente para mejorar la cohesión y reducir el acoplamiento, lo cual permite al sistema ser mantenible y legible.

Además de un Servicio REST, el cual tiene la responsabilidad de ejecutar operaciones CRUD para la lógica del negocio.

* 1. **Servicio REST**
     1. **Cliente**

Para la implementación del sistema información el grupo decidió usar tecnologías más recientes (desarrolladas estos 2 últimos años) basadas en web 2.0 utilizando Nodejs y Expressjs para la parte del servidor del sistema de información y como administrador de comunicaciones realizadas entre el servidor y el cliente se usará la librería EmberJs, que permite manejar el patrón MVC usando javascript como lenguaje de programación, además de librerías *“calendar extensible 1.5”* que permite mostrar el calendario de clases por estudiante y profesor de nuestro sistema de información, y usar CSS3 para los estilos de las vistas del sistema de información.

* 1. **Equipo del Proyecto**

EL equipo de trabajo está compuesto por las siguientes entidades:

* Cliente: Colegio EPA
* Programadores:
  + Luis Fernando Camacho Camacho
  + Leonardo Mendieta Azcui
  + Elmer Alvarado Ponce
* Pruebas:
  + Luis Fernando Camacho Camacho
  + Leonardo Mendieta Azcui
  + Elmer Alvarado Ponce
  1. Tipos de Usuario

El sistema cuenta con los siguientes tipos de usuarios:

* Profesores
* Administrador
* Contador
* Secretaria
* Propietario
  1. **Product Backlog**

Después de tener una reunión con el Administrador General y el Propietario de la EPA se logra obtener la información necesaria para definir la lista de los requerimientos que son necesarios para tener el sistema de información para la EPA.

La Lista de requerimientos son los siguientes:

* + 1. **Sprint Planning** – **Segunda iteración** – **Creación del módulo de control de asistencia de estudiantes y profesores**

Siguiendo el cronograma del Product Backlog, la Primera iteración corresponde al Registro de estudiantes, Profesores y la generación de horarios de clases, durante la planificación de la primera iteración se han identificado las siguientes historias de usuario, como se puede ver en tabla 4.2.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Backlog ID** | **Historia de Usuario** | **Tareas** | **Tipo** | **Estimación**  **(horas)** |
| REQ2 | US1 - Desarrollo del formulario de Registro de Asistencia | Definir los campos requeridos para registrar la asistencia de un estudiante a una clase | Análisis | 2 |
| Crear el formulario de registro de asistencia estudiante | Diseño y Desarrollo | 4 |
| Prueba del formulario de registro de asistencia estudiante | Prueba | 2 |
| US2 - Almacenamiento de los datos de asistencia | Crear la estructura de las base de datos | Diseño y Desarrollo | 2 |
| Implementar la funcionalidad del botón Registrar | Desarrollo | 6 |
| Implementar la funcionalidad del botón Cancelar | Desarrollo | 2 |
| Prueba del botón Registrar para registro de estudiante | Prueba | 2 |
| Prueba del botón Cancelar para registro de estudiante | Prueba | 2 |
| Implementar la funcionalidad del botón Guardar | Desarrollo | 8 |
| Implementar la funcionalidad del botón Cancelar | Desarrollo | 2 |
| US3 - Modificación de datos del estudiante | Prueba del botón Editar para estudiantes | Prueba | 2 |
| Prueba del botón Cancelar para la modificación de datos del estudiante | Prueba | 2 |
| Implementar la funcionalidad del botón Eliminar | Desarrollo | 4 |
| Implementar la funcionalidad de cancelar eliminación | Desarrollo | 2 |
| US4 - Eliminación de un estudiante | Prueba de la eliminación de un estudiante | Prueba | 2 |
| Prueba de mensaje de información por no seleccionar un estudiante | Prueba | 1 |
| Prueba de la cancelación de eliminación de un estudiante | Prueba | 1 |
| Documentar el registro, edición y eliminación de un estudiante | Documentación | 2 |
| Revisar la guía de usuario para para agregar un estudiante | Prueba | 2 |
| US5 - Crear el guía de usuario para registro de estudiantes |  |  |  |
|  |  |  |

## Historias de Usuario

Una vez revisada la lista del backlog, al planificar la Segunda iteración, se detallan las historias de usuario correspondientes al registro de estudiantes, y las tareas relacionadas a cada una de éstas, a continuación se detallarán cada historia de usuario y sus tareas.

El esfuerzo de 1 punto = 1 día ideal de programación

1 historia de usuario = de 2 – 5 puntos (generalmente)

* + - * 1. **Historia de Usuario #1**
        2. **Historia de Usuario #**
        3. **Historia de Usuario #**
        4. **Historia de Usuario #**

## Scrum Daily Meeting

Las reuniones del equipo se programaron de forma más regular, utilizando llamadas y chat como herramientas para poder hacer el seguimiento del progreso de las tareas.

## Sprint Demo

La demostración de avances para la segunda iteración consiste principalmente en el funcionamiento básico del formulario de registro de estudiantes, se realiza con una transferencia de datos a un repositorio alternativo hasta poder completar la implementación del servicio REST.

## Retrospectiva

Cuando se ha completado la iteración se realiza una reunión de retrospectiva en la cual se consideran algunos aspectos que pueden ayudar al equipo a mejorar los procesos para la siguiente iteración, entre estos puntos se observan las cosas que salieron bien o fueron positivas para repetirlas o si se puede mejorarlas, las cosas que no salieron bien o fueron negativas, para buscar soluciones y aplicarlas:

* + 1. **Que salió bien**
* Las tareas asignadas se han completado satisfactoriamente
* Se completó la implementación del formulario de registro de estudiantes
* Se completó la implementación del formulario de registro de profesores
* Se pudo conectar con la base de datos y validar los servicios (REST) y el funcionamiento de acuerdo al patrón MVC
  + 1. **Qué no salió bien**
* La tecnología es complicada y requirió una reorganización de los archivos y las referencias, fue necesario consultar con un experto externo para completar la configuración
* + 1. **Mejoras a implementar en la próxima iteración**
* Fomentar una comunicación más fluida entre miembros del equipo.
  1. Fomentar la puntualidad en la asistencia a las reuniones

1. **CAPITULO CONCLUSIONES Y RECOMENDACIONES**

## *Conclusiones*

La investigación realizada sobre las tecnologías RIA muestra la flexibilidad y ventajas [beneficios] que éstas ofrecen sobre un sistema de información al que se pueden aplicar, principalmente en el acceso desde diferentes lugares y la distribución del trabajo entre el cliente y el servidor.

Mediante la implementación del sistema se ha cumplido el objetivo de automatizar los procesos en el manejo de la información en la administración de la escuela, lo cual agiliza el trabajo de los encargados de las [distintas] tareas administrativas.

Con el sistema se ha conseguido que el proceso de registro de estudiantes nuevos y regulares sea ágil e intuitivo, así mismo su correspondiente inscripción a los [respectivos] cursos. El sistema también permite un control preciso, y brinda información de forma inmediata, de la asistencia de estudiantes y profesores a sus clases.

La estructura del sistema, construida en EmberJS y en los servicios REST, permitió que cada módulo sea implementado de forma independiente y luego se integre con facilidad al resto de la aplicación, esto ofrece mayor mantenibilidad y extensibilidad al sistema.

La administración de la EPA es notificada por el sistema cuando un estudiante está próximo a completar su curso, de este modo puede coordinar la inscripción al siguiente modulo y mantener continuidad.

## *Recomendaciones*

Durante el desarrollo del sistema se pudieron observar varios aspectos que merecen la pena revisar y tomar en cuenta

* La selección de la tecnología que se va a utilizar para desarrollar el producto debe basarse no solo en lo novedosa que sea o las ventajas aparentes que pueda ofrecer, sino también se debe considerar la experiencia y conocimientos del equipo de desarrollo sobre esa tecnología, y una evaluación de costo/beneficio sobre tiempo y recursos que implica el entrenamiento en algo nuevo o el uso de algo ya conocido que puede cubrir los requerimientos [a un costo menor].
* La comunicación entre los miembros del equipo debe ser fluida y mantener a cada uno al tanto de las actividades del resto. Para ello, se debe seleccionar un medio adecuado y de uso general para todos los miembros del equipo. El uso de varias herramientas hace que la información esté dispersa y que alguno de los miembros del equipo pueda omitir algo importante por no hacer seguimiento a todas ellas. La recomendación en este caso es tener una herramienta principal de comunicación
* El análisis de los requerimientos debe ser más detallado, y debe llevarse a cabo en coordinación con el cliente, aunque tome más tiempo, una adecuada interpretación de los requerimientos en un inicio evitara incrementar [o repetir] el trabajo en iteraciones posteriores
* Mantener actualizado el repositorio de código con el trabajo más reciente permite que se identifiquen oportunamente los errores y problemas y se les pueda dar solución. La integración continua es una buena práctica.
* La metodología Scrum recomienda reuniones regulares entre los miembros del equipo, esto permite que cada integrante esté al tanto de las tareas en las que se está trabajando y de los problemas que se han encontrado, y de ser necesario puedan aportar o ayudar en la solución lo antes posible, permitiendo así que el trabajo tenga un avance uniforme y los miembros del equipo se apoyen entre sí.

Referencias Bibliográficas.

* PTAANG (2014). Guía para la elaboración de trabajos dirigidos interdisciplinario. Programa de Titulación de Alumnos Antiguos No Graduados. PTAANG XXV
* Wikipedia. (2014). Web2.0. Octubre 2014, de Wikipedia Sitio web: HTTP://es.wikipedia.org/wiki/Web\_2.0
* scrumalliance  - Scrum, Principios de Scrum Valores del Manifiesto Ágil

Metodología agiles (© 2012 Scrum Alliance, Inc. )

* Model–view–controller. (2014, November 26). Retrieved November 27, 2014, from [HTTP://en.wikipedia.org/wiki/Model–view–controller](http://en.wikipedia.org/wiki/Model–view–controller)
* tutorialspoint. (2014). MVC Pattern. 27/11/2014, de tutorialspoint Sitio web: HTTP://www.tutorialspoint.com/design\_pattern/mvc\_pattern.htm
* Wikipedia. (2014). Rich Internet application. 27/11/2014, de Wikipedia Sitio web: HTTP://es.wikipedia.org/wiki/Rich\_Internet\_application
* Webopedia. (2014). Rich Internet application. 27/11/2014, de Webopedia Sitio web: HTTP://www.webopedia.com/TERM/R/Rich\_Internet\_Application.HTML
* Wikipedia. (2014). Ember.JS. 27/11/2014, de Wikipedia Sitio web: [HTTP://en.wikipedia.org/wiki/Ember.js](http://en.wikipedia.org/wiki/Ember.js)
* [1] Ember. (2014). Core Concepts. 27/11/2014, de Ember Sitio web: [HTTP://emberjs.com/guides/concepts/core-concepts/](http://emberjs.com/guides/concepts/core-concepts/)
* Jenkins. (2014, 10 de marzo). *Wikipedia, La enciclopedia libre*. Fecha de consulta: 00:16, diciembre 1, 2014 desde [HTTP://es.wikipedia.org/w/index.php?title=Jenkins&oldid=73085392](http://es.wikipedia.org/w/index.php?title=Jenkins&oldid=73085392).
* Chacon, S., & Straub, B. (2014). Getting Started. In *Pro GIT* (Second ed., Vol. 2014, p. 27,28,29,30). San Francisco, California, 94105: Apress.
* Integración continua. (2014, 14 de noviembre). *Wikipedia, La enciclopedia libre*. Fecha de consulta: 00:20, diciembre 1, 2014 desde
* SOAP  (accessed 12/04/2014). Protocolo simple de acceso a objetos

<http://www.ecured.cu/index.php/Protocolo_simple_de_acceso_a_objetos_(SOAP)>

* [2] Simple Object Access Protocol. (2014, 1 de diciembre). *Wikipedia, La enciclopedia libre*. Fecha de consulta: 00:00, diciembre 15, 2014 desde <http://es.wikipedia.org/w/index.php?title=Simple_Object_Access_Protocol&oldid=78471334>.
* REST, 12/04/2014, Servicios web RESTful con HTTP sitio Web:

<http://www.adwe.es/general/colaboraciones/servicios-web-restful-con-http-parte-i-introduccion-y-bases-teoricas>

* Mongo DB. JUAN ROY.  (12/04/2014)  sitio Web: http://www.mongodbspain.com/es/2014/08/17/mongodb-characteristics-future/
* Node JS,  Node Js hispano. 12/04/2014, Que es Node Js sitio web:

<http://www.nodehispano.com/2011/11/que-es-node-js-nodejs/>

* Manual Node js. Miguel Angel Alvarez. (12/04/2014) ,  Características destacables de Node.JS: <http://www.desarrolloweb.com/articulos/caracteristicas-nodejs.html>