Universidad Cenfotec

Proyecto de Ingeniería de Software IV

Plan General del Proyecto

PROFESORES

Armando Blanco

Álvaro Cordero Peña

INTEGRANTES

Juan Arias

Juan Parra

Adrián Rodríguez

Daniel Rodríguez

Dayana Vásquez

23 de febrero del 2020

I Cuatrimestre

Tabla de contenidos

Tabla de Figuras

Introducción

En el presente documento se describe el plan general del proyecto, donde se establecen los diferentes planes a seguir de cada uno de los miembros del equipo de trabajo, para la elaboración de la aplicación.

A continuación, se describen los apartados del documento:

En el primer apartado se hace referencia las definiciones, acrónimos, abreviaturas y siglas que son utilizados en el documento, esto tiene como finalidad la comprensión de los diferentes conceptos esenciales para entender el documento.

En el segundo apartado se explica el plan de administración de calidad, donde se pretende mostrar las herramientas utilizadas por los encargados de la aplicación de calidad en los diferentes entregables del proyecto.

En el tercer apartado se hace referencia al plan de administración de la configuración, donde se detalla la herramienta que se va a utilizar, el encargado de la administración de la herramienta, el manejo de versiones de los documentos y del software, y la cantidad de ambientes que serán manejados y por último el respaldo de la información pertinente al proyecto.

En el cuarto apartado se hace referencia a los estándares de desarrollado que serán utilizados en el desarrollo del proyecto, a nivel de interfaz gráfica, nomenclatura de código y bases de datos y estándares de documentación del código y de los objetos de base de datos.

En el quinto apartado se hace referencia al plan de administración de tiempo, donde se hace detalla el WBS además de las herramientas que serán utilizadas para el desarrollo de las minutas, el control de tareas y la administración del proyecto.

Finalmente, el último apartado se hace referencia a los anexos del documento.

Definiciones, acrónimos y abreviaturas

Open source: En español significa “código abierto” que se refiere a ciertos tipos de software que permite al usuario final utilizar el código fuente para realizar mejoras.

Sprints: son los ciclo o interacción que tiene un proyecto Scrum. Se tienen que programar ciertas funcionalidades en un tiempo límite. En este caso un sprint dura una semana, pero en la teoría lo normal es que duren entre dos a un máximo de dos meses.

Historia de usuario: son descripciones muy concretas de ciertas necesidades y funcionalidades que resumen la necesita de un usuario al utilizar un producto. Su función principal es proponer soluciones y estimar esfuerzo.

GitFlow. Es una extensión de Git, que permite tener un workflow de trabajo a través de diferentes ramas. Sus principales ramas son el Master, Develop, Features, Releases y HotFixes.

Plan de Administración de la Calidad

En esta sección de detallará el plan que llevarán a cabo los encargados de calidad. Primero se describe el control de calidad en los documentos el cual contiene tablas para revisar los estándares de los documentos. Estas tablas revisan detalladamente el formato de los documentos para que no errores en formato u ortografía. El formato lo tienen que seguir todos los integrantes del equipo en sus documentos, ya que para eso es la tabla de verificación de documentos.

También este documento busca la calidad en el producto de software final, ya que en los procesos de desarrollo se tienen que hacer diferentes pruebas para verificar que el producto final cumpla con los estándares establecidos por el cliente. En este documento se van a detallar las herramientas que se van a utilizar ya que estas son vitales para mantener la calidad en el desarrollo del proyecto.

Formato y detalle de las Listas de Verificación

Las listas de verificación son el listado de diferentes elementos que se van a tomar en cuenta para la revisión de los documentos. Esta tabla va a contar con el nombre del documento que se va a revisar, los puntos a revisar, el estado de la revisión y la fecha en la que se revisó el documento. Se van a manejar dos listas de verificación para los documentos que se van a hacer, estos documentos son el plan general del proyecto y la especificación de requerimientos de software. Es importante que se siga el formato en los documentos ya que de esta forma se van a presentar de una forma más organizada.

- **Plan General del Proyecto**

Lista de verificación [Plan general del proyecto]			
Nombre del documento:	**Plan general del proyecto**		
Evaluación	✓	✗	**Fecha de revisión**
¿Presenta el documento una portada?			
¿El documento tiene la portada con el formato correcto?			
¿El documento presenta una tabla de contenidos?			
¿El documento presenta una tabla de contenidos con el formato correcto?			
¿El documento presenta una tabla de contenidos con la numeración correcta?			
¿El documento presenta una tabla de figuras (cuando el documento presenta imágenes)?			
¿El documento presenta una tabla de figuras con el formato correcto?			
¿El documento presenta una introducción?			
¿El documento presenta el mismo tipo de letra (Arial)?			
¿El documento presenta el mismo tamaño de letra en los párrafos?			
¿El documento presenta el mismo tamaño de letra en los títulos?			
¿El documento presenta el espaciado correcto?			
¿El documento presenta referencias?			
¿El documento presenta referencias en formato APA?			
¿El documento está escrito en tercera persona?			
¿El documento no presenta errores ortográficos?			
¿El documento presenta con todos los apartados del documento?			
¿El documento presenta una numeración de páginas?			
¿El documento presenta el encabezado correcto?			

- **Especificación de Requerimientos de Software**

Lista de verificación [Especificación de Requerimientos de Software]			
Nombre del documento:	**Plan general del proyecto**		
Evaluación	✓	✗	**Fecha de revisión**
¿Presenta el documento una portada?			
¿El documento tiene la portada con el formato correcto?			
¿El documento presenta una tabla de contenidos?			
¿El documento presenta una tabla de contenidos con el formato correcto?			
¿El documento presenta una tabla de contenidos con la numeración correcta?			
¿El documento presenta una tabla de figuras (cuando el documento presenta imágenes)?			
¿El documento presenta una tabla de figuras con el formato correcto?			
¿El documento presenta una introducción?			
¿El documento presenta el mismo tipo de letra (Arial)?			
¿El documento presenta el mismo tamaño de letra en los párrafos?			
¿El documento presenta el mismo tamaño de letra en los títulos?			
¿El documento presenta el espaciado correcto?			
¿El documento presenta referencias?			
¿El documento presenta referencias en formato APA?			
¿El documento está escrito en tercera persona?			
¿El documento no presenta errores ortográficos?			
¿El documento presenta con todos los apartados del documento?			
¿El documento presenta una numeración de páginas?			
¿El documento presenta el encabezado correcto?			

Diseño y especificación de los casos de prueba

Las pruebas de la aplicación van a ser manuales, ya que no tienen una curva de aprendizaje, son rápidas y efectivas. Para comprobar que todas las historias de usuario estén correctas, se tienen que cumplir todos los criterios de aceptación, si algún requerimiento de la historia de usuario no pasa, esta historia se le devolverá al desarrollador hasta que esta historia haya sido finalizada. Para la revisión de las pruebas manuales se va a utilizar la siguiente tabla, que va dirigida a los encargados del rol de calidad.

Nombre del caso de prueba		
ID	*Identificador del caso de prueba*	
Módulo	*Módulo al que pertenece la historia de usuario*	
Método	*El método que se ejecutó para hacer la prueba*	
Autor	*Desarrollador del caso de prueba*	
QA	*Quien hizo la prueba*	
Fecha	*Fecha de ejecución de la prueba*	
Procedimiento	**Resultado Esperado**	**Resultado Obtenido**
Pasos a seguir para ejecutar la prueba	*Datos o estados que se esperan al finalizar la prueba*	*Resultados obtenidos después de ejecutar la prueba*

Diseño y especificación de las pruebas de rendimiento

Para realizar las pruebas de rendimiento, se utilizará la herramienta JMeter. Las pruebas de rendimiento se harán cuando el producto esté bastante avanzado, porque si se hacen cuando el desarrollo es muy reciente esto va a causar problemas ya que el producto va a tener muchos cambios en el proceso de desarrollo y también la cantidad de datos que se manejan a va a aumentar.

JMeter de Apache, es una herramienta que fue inicialmente diseñada para pruebas de estrés en aplicaciones web, pero con el pasar del tiempo se le han agregado más funcionalidades. JMeter tiene muchos elementos que la hacer una buena herramienta como por ejemplo su interfaz amigable, su documentación que es fácil de comprender y además la aplicación es *"open source"*, estos elementos hacen que la aplicación sea atractiva para los desarrolladores.

Lo que se puede hacer con JMeter es hacer pruebas automatizadas y funcionales para la aplicación, dentro de estas pruebas se incluyen las pruebas de rendimiento y disponibilidad. También monitorea los servidores (servidores de base de datos, el servidor web, entro otros).

La forma en que JMeter funciona es que simula un grupo de usuarios que mandan solicitudes a un servidor en específico, y devuelve los resultados que muestran el rendimiento del servidor/aplicación a través de diagramas. Estas es una explicación muy básica de como JMeter funciona ya que tiene muchas otras funcionalidades que procesos más extensos. La siguiente imagen muestra una forma gráfica en como JMeter funciona.

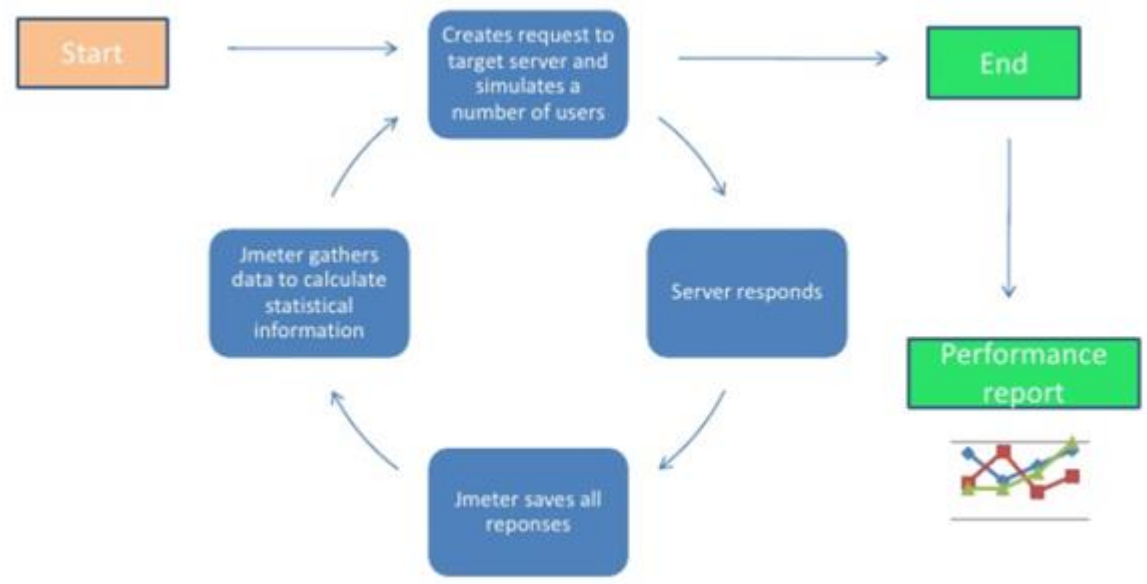

Figura 1: Funcionamiento de JMeter

Para descargar la herramienta hay que seguir los siguientes pasos:

1. Ingresar al sitio de “The Apache Software Foundation”
2. Buscar JMeter y descargar el archivo comprimido.
3. Descomprimir el archivo.
4. Abrir el ejecutable que viene en la carpeta bin/.
5. Finalmente siguiendo estos pasos la IU de JMeter se tuvo que ejecutar.

Plan de administración de la configuración

En esta sección se mostrará el plan de administración de la configuración, el cual se tiene como objetivo administrar los recursos del proyecto de una manera exitosa; con el fin de garantizar la seguridad, privacidad y orden a través de todos los entregables del proyecto.

El plan se utilizará para establecer las reglas y estructura del uso de los recursos del proyecto para poder administrarlos entre todos los integrantes del equipo. Se detalla la integración de Google Drive para el manejo de documentación, el control de versiones para el código fuente de la aplicación.

Responsabilidades

A continuación, se especificará las responsabilidades que tendrá cada integrante del equipo.

- **Adrián Rodríguez.** Principal tarea será la administración de control de versiones. Garantizar que cada miembro del equipo tenga instalado las herramientas adecuadas, de explicar el uso de GitFlow, para un mejor orden a la hora de programar. De generar los entregables para cada ciclo del desarrollo.
- **Daniel Rodríguez.** Encargado de darle estructura a las carpetas para la administración de documentos. De configurar y darle acceso a cada uno de los miembros del equipo a Google Drive. También deberá garantizar que todos los documentos trabajados durante en el proyecto se encuentren en Google Drive, así como sus respaldos.
- **Dayana Vásquez.** Se encargará de estar revisando constante todas las partes del código fuente, especialmente después de que cada uno de los integrantes hagan cambios o agreguen nuevo código.

- **Juan Arias.** Administrar la herramienta Trello, y del manejo de las tareas para cada miembro del equipo. Debe administrar los tiempos de cada ciclo y la asignación de trabajo.
- **Juan Parra.** Configurar el control de versiones para el código fuente. Debe garantizar seguridad y privacidad a los repositorios de la aplicación. Se encargará de darle acceso y compartir el código fuente a todos los integrantes usando la plataforma GitHub.

Descripción de las herramientas

- **Git.** Es un software para el manejo de control de versiones, que no depende de un solo repositorio central, más bien permite la distribución del código alrededor del mundo o en cada computadora de los integrantes de la aplicación.
- **GitHub.** Es un sistema de gestión de proyectos y control de versiones de código. permite trabajar en colaboración con otras personas de todo el mundo, planificar proyectos y realizar un seguimiento del trabajo.
- **Google Drive.** Es un tipo de almacenamiento en la nube desarrollado por Google, que te permite compartir, crear archivos y administrar todos tus archivos desde cualquier dispositivo. Es gratis al tener una cuenta de Google.
- **Android Studio e Intellij IDEA.** Son entornos de desarrollo que te permiten desarrollar de una forma más rápida, depurar errores, realizar pruebas. Soportan varios lenguajes de programación, aunque Java es su principal lenguaje.
- **MySQL**. Es una de las bases de datos más populares y de las más usadas del mundo. Sus principales características son su continua actualización, multiusuario y multihilos.
- **CMS.** Es un sistema para manejo de contenidos, permite la creación y administración de contenidos principalmente para las aplicaciones web.
- **Trello.** Es una herramienta de colaboración continua para la gestión de proyectos. Permite la administración de tareas de una forma clara y sencilla.

Administración del repositorio

- **GitFlow.** Para una mejor gestión de control de versiones, el equipo deberá definir un workflow de trabajo que nos permita trabajar en conjunto e

individualmente. Es aquí donde entra GitFlow, que nos va a permitir integrar una serie de ramas en nuestro proyecto para una administración eficiente en nuestros repositorios. GitFlow se organizará en dos principales ramas:

- **Master:** Cualquier commit que pongamos en esta rama debe estar preparado para subir a producción.
- **Develop:** Rama donde se encontrará todo aquel código en desarrollo y que no está listo para el próximo entregable.

- **Ramas auxiliares:**
  - **Feature:** Se crea a partir de la rama Develop y se utiliza para desarrollar nuevas características de la aplicación que, una vez terminadas, se incorporan a la rama Develop.
  - **Release:** Se origina a partir de la rama Develop, e incorporará con la rama Master. Es utilizada para preparar el siguiente código a producción.
  - **HotFix:** Se origina a partir de la rama Master y se incorpora a la rama Master y Develop. se utilizan para corregir errores y bugs en el código en producción.

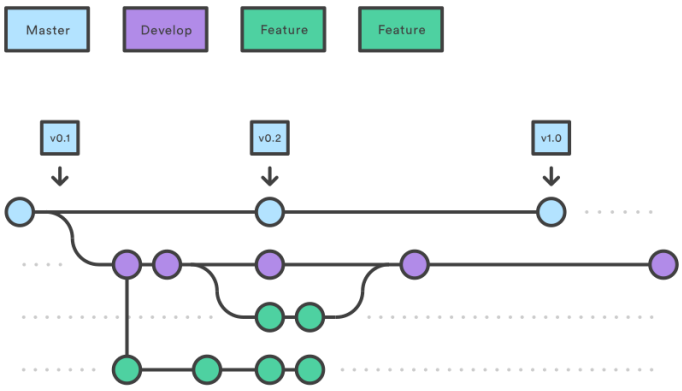

Figura 2:Workflow con GitFlow

Plan de administración de estándares

Estándares de documentación del código

A continuación, se detalla con profundidad los estándares de codificación a seguir del equipo de desarrollo.

Paquetes

El nombramiento de paquetes será de la siguiente manera: los nombres separados por puntos, y en caso de que haya nombres compuestos, se utiliza el primer nombre en minúscula, pero a partir del segundo nombre, la letra inicial deberá ir en mayúscula y así consecutivamente.

Ejemplo:

```
com.nombreEmpresa.nombreCompletoAplicacion.modulo
```

Actividades

Cada actividad debe contener:

- El nombre del autor
- La fecha creada (yyyy-mm-dd)
- Y una descripción breve de su funcionalidad.

Ejemplo:

```
/*
* @author Juan V
* @since  2020-03-31
* Esta actividad es la que se encarga de...
*/
```

Todo nombre de actividad deberá iniciar en mayúscula, aún en casos de actividades con nombres compuestos.

Ejemplo:

```
MainActivity.java
```

Métodos

- Los nombres de los métodos deberán ser significativos y en minúscula.
- Nombres que sean de dos o más palabras, deberán estar unidas y a partir de la primera palabra, el resto de las palabras deberá ir con la primera letra en mayúscula.

Ejemplo:

```
public void iniciarSession(String[] args) {
  ...
}

public void metodoTresPalabras(String[] args) {
  ...
}
```

Cada método deberá estar comentado antes de crearlo, describiendo brevemente su funcionalidad.

Ejemplo:

```
* Comentario sobre el método
* @params recibe un parámetro de tipo String
* @returns retorna un resultado de tipo String
*/
public String metodo(String param) {
  ...
  return "String";
}
```

Sangría

Se establecen **dos** caracteres como unidad de sangría.

Longitud y división de líneas

La longitud de línea no debe superar los 80 caracteres por motivos de visualización e impresión. Cuando una expresión ocupe más de una línea, esta se podrá romper o dividir en función de los siguientes criterios:

- Tras una coma.
- Antes de un operador.
- Se recomienda las rupturas de nivel superior a las de nivel inferior.
- Alinear la nueva línea con el inicio de la expresión al mismo nivel que la línea anterior.

Ejemplo:

```
metodoEjemploConParametros(parametro1, parametro2, parametro3,
  parametro4);

if ((criterio1 && criterio2) || (criterio3 && criterio4)
  || !(criterio5 && criterio6) || (criterio7 == true)) {
  metodoEjemploSinParametros();
}
```

Declaraciones de variables

Se hará uso de una declaración por línea, ya que se simplifica así el uso de comentarios de ser necesario.

Ejemplo:

```
String nombreUsuario; // Primer comentario sobre esta variable
String[] arregloDePermisos; // Segundo comentario sobre esta variable
```

Toda variable local tendrá que ser inicializada en el momento de su declaración, salvo que su valor inicial dependa de algún valor que tenga que ser calculado previamente.

Ejemplo:

```
String nombreUsuario = "juanv";
String[] arregloDePermisos = generarPermisos(); // Se generan los
permisos dependiendo del tipo de usuario
```

Las declaraciones deben situarse al principio de cada bloque de código en el que se utilicen, y nunca en el momento de su uso.

Ejemplo:

```
public void metodoEjemplo() {
  int contador = 0; // inicio del método
  ... // cuerpo restante del método
}
```

Se debe evitar el uso de declaraciones de variables globales que no sean constantes o finales.

Ejemplo:

```
int tasaDePorcentaje = 1;

public void metodoEjemplo() {
  if (criterio == true) {
    int tasaDePorcentaje = 0; // se altero la variable
    ...
  }
}

public double calcularResultado(double valor) {
  metodoEjemplo(); // el valor de tasaDePorcentaje es 0
  double resultado = valor / tasaDePorcentaje; // se tira una excepción
  return resultado;
}
```

Constantes y otras variables que sean declaradas con la connotación “final” deberán tener un valor inicial asignado al tiempo de declaración.

Ejemplo:

```
int POSICION_FINAL = 1000;
final int valorQueNoCambia = 1;
```

Sentencias

Las sentencias que pertenezcan a un bloque de código deberán estar tabuladas un nivel más a la derecha con respecto a la sentencia que las contiene. El carácter inicio de bloque “{“debe situarse al final de la línea que inicia el bloque. El carácter final de bloque “}” debe situarse en una nueva línea tras la última línea del bloque y alineada con respecto al primer carácter de dicho bloque. Todas las sentencias de un bloque deben encerrarse entre llaves “{}”, aunque el bloque conste de una única sentencia. Esta práctica permite añadir código sin cometer errores accidentalmente al olvidar añadir las llaves.

Ejemplo:

```
int contador = 0;

if (criterio == true) {
  contador++;
}
```

La sentencia “try/catch” se utilizará para rescatar al sistema de errores que se espera que provengan del sistema de software y que puedan ser probados o monitoreados. En el bloque “try” vendrá la lógica que pueda ocasionar un error de sistema, mientras que en el bloque “catch” siempre se imprimirá una traza de error indicando el tipo de excepción generada y posteriormente se elevará dicha excepción al código que está invocando, salvo que la lógica de ejecución de la aplicación no lo requiera.

Finalmente, en el bloque “finally”, se utilizará para concluir de procesos que puedan ocasionar derrames de memoria y serán de uso si solo si son necesarios. La sentencia “try/catch” siempre debe tener el formato siguiente:

Ejemplo:

```
try {
  calcularResultado(1000);
} catch (ClaseException ex) {
  System.out.println(ex.getMessage());
} finally {
  cerrarProcesos();
}
```

Espacios en blanco

Las líneas y espacios en blanco mejoran la legibilidad del código permitiendo identificar las secciones de código relacionadas lógicamente. Se utilizarán espacios en blanco en los siguientes casos:

- Entre una palabra clave y un paréntesis (esto permite que se distingan las llamadas a métodos de las palabras clave).

Ejemplo:

```
if (criterio == false) {
  ...
}
  // espacio en blanco
do {
  ...
} while(criterio == true);
  // otro espacio en blanco
public void metodoEjemplo(String parametro) {
  ...
}
```

Una coma seguida de un espacio en blanco tras dos o varios argumentos.

Ejemplo:

```
objeto.metodoEjemplo(argumentoA, argumentoB, argumentoC);
```

Nunca se utilizarán espacios entre los operadores unarios (ej. “++” o “--”) y sus operadores.

Ejemplo:

```
int total = var1 + var2;

double tasaDePorcentaje = var1 / var2;

contador++;
```

Para separar las expresiones incluidas en la sentencia “for”.

Ejemplo:

```
for (int i = 0; i > 10; i++) {
  ...
}
```

Variables

variables se escribirán siempre en minúsculas. Las variables compuestas tendrán la primera letra de cada palabra componente en mayúsculas. Las variables nunca podrán comenzar con caracteres especiales (ej. “_” o “$”). Los nombres de variables deben ser cortos y significativos, o sea tienen que expresar con suficiente claridad la función que desempeñan en el código. Debe evitarse el uso de nombres de variables con un sólo carácter, excepto para variables temporales (ej. en secuencias “for”).

Ejemplo:

```
Usuario usuario;
int totalSuma = suma1 + suma2 + suma3;
```

Constantes

Todos los nombres de constantes tendrán que escribirse en mayúsculas. Cuando los nombres de constantes sean compuestos las palabras se separarán entre sí mediante el carácter de subrayado “_”.

Ejemplo:

```
int POSICION_FINAL = 1000;
int PAGINA_DE_INICIO = 1;
```

Documentación interna

A continuación, se detallan los estilos de documentación interna de código del equipo de desarrollo.

Documentación: Javadoc

Se deberá seguir el estándar de documentación JavaDoc, dado por Oracle. Toda clase de código fuente (ej. Ejemplo.java) deberá comenzar con un comentario que incluya el nombre de la clase, la fecha de creación, el nombre del implementador y una descripción concisa pero completa del elemento del API. Las descripciones deberán ser cortas, concisas pero descriptivas.

Las etiquetas **@autor**, **@deprecated**, **@param**, **@return**, **@throws**, **@since,** son las que comúnmente se van a utilizar cuando estas apliquen en interfaces y clases. Descripciones de variables, funcionalidades de prácticas comunes de java o el tipo de dato de estas no son necesarias de que sean comentadas (ej. setters y getters). La longitud de línea no debe superar los 80 caracteres por motivos de visualización y legibilidad de código o documentación.

Ejemplo:

```java
/**
* Esta clase representa una abstracción de tipo Usuario
*@class Usuario
*@autor Juan V
*@since 2020-01-30
**/
public class Usuario {

  private String nombre; // No es necesario documentar

  /**
  * Este método es responsable de crear una instancia de tipo Usuario
  *@param name of type String
  *@return una instancia de tipo Usuario
  **/
  public User(String nombre) {
    this.nombre = nombre;
  }

  // NO es necesario documentar
  public void setNombre(String nombre) {
    this.nombre = nombre;
  }

  // NO es necesario documentar
  public String getNombre() {
    return this.nombre;
  }
}
```

Plan de administración del tiempo: Cronograma

En el presente apartado, se detalla el plan de administración de tiempo que será implementado por el equipo de trabajo para el desarrollo del producto final, dicho apartado está comprendido por los siguientes subapartados:

- **WBS:** Es un diagrama que a través de una jerarquía en sus distintos componentes permite observar el trabajo que debe realizarse para completar los resultados de un proyecto.
- **Diagrama de red:** Es un diagrama que permite obtener una representación gráfica de todas las tareas y responsabilidades.
- **Cronograma final:** Es la distribución de entregables o tareas a realizar por parte del equipo para ir cumpliendo con los entregables del proyecto que será desarrollado.

WBS

Ver Anexo # 1

Diagrama de red

Ver Anexo # 2

Cronograma final

Tarea	Tipo de tarea	Encargado	Estado	Fecha de inicio	Fecha de finalización	Fecha de calidad
Plan General del Proyecto	Documentación	Todo el equipo	Completado	17/02/2020	23/03/2020	23/03/2020
Especificación de Requerimientos de Software	Documentación	Todo el equipo	En progreso	24/02/2020	01/03/2020	01/03/2020
Creación del backlog del proyecto	Planeamiento	Juan Arias	No iniciado	28/02/2020	29/02/2020	-
Asignación de puntaje de historias de usuario	Planeamiento	Todo el equipo	No iniciado	01/03/2020	01/03/2020	-
Asignación de historias de usuario / Retrospective - Sprint # 1	Planeamiento	Juan Arias	No iniciado	01/03/2020	01/03/2020	-
Sprint # 1	Desarrollo	Todo el equipo	No iniciado	02/03/2020	08/03/2020	08/03/2020
Asignación de historias de usuario / Retrospective - Sprint # 2	Planeamiento	Juan Arias	No iniciado	08/03/2020	08/03/2020	-
Sprint # 2	Desarrollo	Todo el equipo	No iniciado	9/3/2020	15/3/2020	15/3/2020
Asignación de historias de usuario / Retrospective - Sprint # 3	Planeamiento	Juan Arias	No iniciado	15/3/2020	15/3/2020	-
Sprint # 3	Desarrollo	Todo el equipo	No iniciado	16/3/2020	22/3/2020	22/3/2020
Asignación de historias de usuario / Retrospective - Sprint # 4	Planeamiento	Juan Arias	No iniciado	22/3/2020	22/3/2020	-
Sprint # 4	Desarrollo	Todo el equipo	No iniciado	23/3/2020	29/3/2020	29/3/2020
Asignación de historias de usuario / Retrospective - Sprint # 5	Planeamiento	Juan Arias	No iniciado	29/3/2020	29/3/2020	-
Sprint # 5	Desarrollo	Todo el equipo	No iniciado	30/3/2020	5/4/2020	5/4/2020
Asignación de historias de usuario / Retrospective - Sprint # 6	Planeamiento	Juan Arias	No iniciado	5/4/2020	5/4/2020	-
Sprint # 6	Desarrollo	Todo el equipo	No iniciado	6/4/2020	12/4/2020	12/4/2020
Asignación de historias de usuario / Retrospective - Sprint # 7	Planeamiento	Juan Arias	No iniciado	12/4/2020	12/4/2020	-
Sprint # 7	Desarrollo	Todo el	No iniciado	13/4/2020	19/04/2020	19/04/2020

		equipo				
Asignación de historias de usuario / Retrospective - Sprint # 8	Planeamiento	Juan Arias	No iniciado	19/04/2020	19/04/2020	-
Sprint # 8	Desarrollo	Todo el equipo	No iniciado	20/04/2020	26/04/2020	26/04/2020

Herramientas para control de minutas, tareas y administración del proyecto.

Administración de minutas

La administración de minutas se llevará a cabo mediante archivos realizados en Word y se utilizará el control de estas mediante la herramienta que proporciona el sitio https://jam.wisembly.com/

Administración de tareas

La administración de las tareas se llevará a cabo mediante la herramienta de control de actividades proporcionada por el sitio https://trello.com/

Administración del proyecto

La administración del proyecto se llevará a cabo mediante la aplicación online Monday, accesado mediante el siguiente enlace: https://ucenfotec-group.monday.com/

Anexos

Anexo # 1: Proyecto Swapy – WBS

- Ubicación de archivo: Mismo folder que el presente documento con el nombre: *Proyecto Swapy - WBS.png*

Anexo # 2: Proyecto Swapy - Diagrama de red

- Ubicación de archivo: Mismo folder que el presente documento con el nombre: *Proyecto Swapy - Diagrama de red.png*

Referencias

Cordero, A. (2019). Plantilla general del proyecto. San José: Universidad Cenfotec

NOTA 98