**TP Integrador, Laboratorio 3: Vanguard PIM**

Yatel Nehuen Espasandin, Máximo Alessandro Camesana

Universidad Tecnologica Nacional, Facultad Regional Mar del Plata

Comision 4: Laboratorio 3

Gabriel Chaldu

<https://github.com/maxkmesana/Final-Espasandin-Camesana>

Nota de autores

Se agradece la fundamentación teórica-contable de la Lic. Carmen Martucci.

Contenido

[TP Integrador, Laboratorio 3: Vanguard PIM 4](#_Toc170038559)

[Investigación tecnologica 4](#_Toc170038560)

[Interfaz gráfica de usuario1 4](#_Toc170038561)

[Model-view-controller2 4](#_Toc170038562)

[Vista. 4](#_Toc170038563)

[Modelo 5](#_Toc170038564)

[Controlador 5](#_Toc170038565)

[Control de versionado (Git/Github)3 5](#_Toc170038566)

[JavaFx4 6](#_Toc170038567)

[Scene Builder 6](#_Toc170038568)

[Maven 7](#_Toc170038569)

[JBcrypt5 7](#_Toc170038570)

[Nuestro flujo de trabajo (Git/Github) 7](#_Toc170038571)

[Flujo de Controllers y Vistas 9](#_Toc170038572)

[Lenguaje 9](#_Toc170038573)

[Referencias 10](#_Toc170038574)

Resumen

Para este proyecto se utilizan una gran variedad de tecnologías distintas, todas las cuales fueron imperativas para el desarrollo y flujo de trabajo del mismo, cada una de ellas será explicada en brevedad en este informe. Adicionalmente, se explica como se aplicaron algunas de estas tecnologías al presente proyecto y su utilidad. También podremos encontrar algunos diagramas informativos sobre el proyecto y su desarrollo.

TP Integrador, Laboratorio 3: Vanguard PIM

Para este trabajo se usaron diferentes tecnologías, todas con distintos propósitos y usos reales. Desde hasheo de contraseñas y control de versionado hasta patrones arquitectónicos de software. En el siguiente apartado se explica cada una de ellas.

# Investigación tecnologica

## Interfaz gráfica de usuario1

Su principal objetivo es proporcionarle un entorno visual al usuario para poder interactuar con diversos tipos de sistemas informáticos y mediante ella, poder accionar distintos mecanismos de las aplicaciones. Nuestro proyecto cuenta con una gran complejidad de interacciones gráficas mediante una interfaz llamada JavaFx, de la cual se hablará mas en detalle en partes posteriores de este informe.

## Model-view-controller2

Es un patrón arquitectónico de software basado en la construcción de tres componentes distintos, cuyo objetivo es separar responsabilidades, reutilizar código y dividir el código en distintas áreas conceptuales.

### Vista.

Es la presentación de la aplicación lógica que existe detrás, es con lo que el usuario interactúa de manera directa y su responsabilidad es la prestación y recolección de información hacia y desde el exterior. Es lo que conocemos como interfaz de usuario (UI) o interfaz gráfica de usuario (GUI). En nuestro proyecto se implementa una GUI mediante tecnologías que serán mencionadas posteriormente vía el presente informe.

### Modelo

Es la representación de la información con la que cuenta el sistema, su responsabilidad se trata del manejo sistemático de la misma, lo cual incluye consultas, actualizaciones, etc. Se comunica con la vista a través del Controlador. En nuestro proyecto este Modelo se ve evidenciado en el manejo de distintos Repositorios.

### Controlador

Responde a eventos y maneja la lógica completa del sistema, encargándose de integrar y comunicar la lógica de la Vista y el Modelo. Su responsabilidad se basa en ser un intermediario entre estos dos. En nuestro trabajo implementamos una amplia variedad de controladores, cada uno con distintas responsabilidades y comunicaciones secuenciales entre ellos.

## Control de versionado (Git/Github)3

Es un sistema que registra cambios que se realizan en un archivo o en un conjunto de archivos a lo largo del tiempo mientras recauda información sobre estos cambios. Esto nos permite, entre muchas otras cosas, recuperar versiones anteriores de archivos o incluso mantener dos versiones del mismo archivo (separadas) al mismo tiempo.

Git es la herramienta predilecta de control de versiones del programador y, a la misma vez, Github es la herramienta predilecta para el control de versiones en la nube y la colaboración de programadores mediante internet.

En nuestro proyecto usamos ambas herramientas, tanto para el versionado local y la distribución de Features o Características como para la colaboración rápida y eficaz entre los miembros del grupo.

## JavaFx4

JavaFx es una interfaz y plataforma que permite a los programadores usar funcionalidades de cualquier biblioteca de Java en implementaciones gráficas de una manera que es fácil y genérica para el programador. Tiene amplia variedad de funcionalidades, desde implementaciones gráficas básicas hasta gráficos vectoriales, animaciones, sonidos, implementación de CSS, etc.

Funciona a través de Stages (Escenarios) y Scenes (Escenas). Los escenarios actúan como contenedor de nivel superior (como ventanas) que contienen a las escenas (y pueden contener multiples escenas), mientras que las escenas contienen todos los elementos graficos, la disposición y el estilo. Nuestro proyecto hace uso de ambos de estos conceptos de distintas maneras para lograr ditintos tipos de ventanas y flujos de lógica.

Las vistas de esta interfaz se generan mediante Scene Builder y son traducidas a archivos .fxml, los cuales tienen una arquitectura de .xml con pequeños cambios para adaptarse a las necesidades de JavaFx e interfaces graficas de usuario. En estos archivos de vista se contienen jerarquías de elementos gráficos que, luego de desplegada la lógica, pueden ser visualizados y se puede interactuar con ellos.

## Scene Builder

Es una herramienta con interfaz gráfica que permite crear escenas y escenarios de JavaFx de una manera visual e intuitiva, con ciertas capacidades que facilitan el código de JavaFx. La más grande función que presenta esta herramienta es la de generar vistas complejas en formato .fmxl (requerido por JavaFx).

## Maven

Es una potente herramienta de gestión de proyectos que se utiliza para la fácil y rapida gestión de dependencias, como herramienta de compilación e incluso como herramienta de documentación. Algunas de sus funciones clave son: descargar e instalar modulos, paquetes y herramientas de dependencias, compilar el código fuente de la aplicación y empaquetar el código en distintos formatos de archivo. En nuestro proyecto, se usan todas estas funciones de Maven.

## JBcrypt5

JBcrypt es una implementación en Java de Bcrypt, una función de hash (un algoritmo unidireccional de cifrado) para contraseñas. Blowfish, un algoritmo creado por Bruce Schneier, es la principal característica de JBcrypt ya que se basa en subir el costo computacional de softwares diseñados para el crackeo de contraseñas, tanto así que su costo computacional es parametrizado para poder ser incrementado mientras avanza el tiempo y la tecnología. Considerando todos estos puntos, creimos correcto utilizar JBcrypt para el hasheo de contraseñas en la creación de un sistema de cuentas y log in para el proyecto.

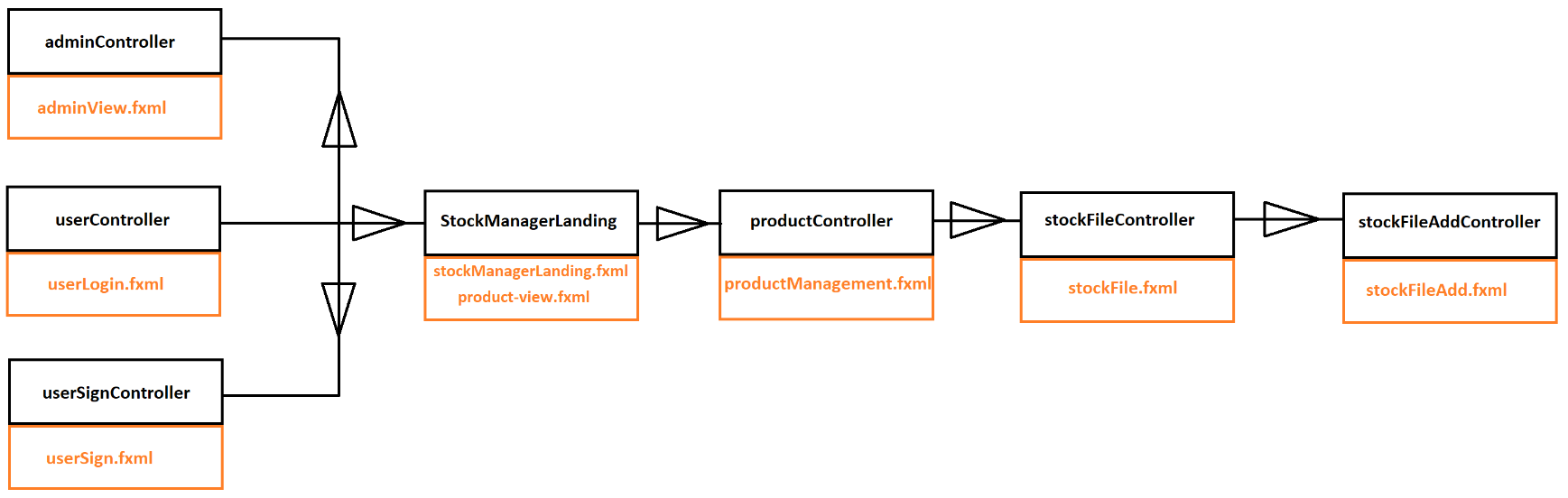
# Nuestro flujo de trabajo (Git/Github)

Para poder mantener un flujo de trabajo concurrente y eficaz, en el equipo dividimos todas las distintas funcionalidades del programa en Features y las asignamos como responsabilidades a los integrantes. Teniendo cada uno sus responsabilidades e implementaciones aisladas de demás partes del proyecto (a ser construidas) nos pudimos asegurar en tener una buena planeación de commits y branches. A continuación podemos observar un la línea de tiempo de nuestro repositorio y los commits hechos (y a su vez, la curva de aprendizaje de git evidenciada en su máximo esplendor):



# Flujo de Controllers y Vistas

En un nuestro proyecto utilizamos un flujo de vistas y controllers lineal, en donde cada uno cuenta con una responsabilidad particular (cada responsabilidad es obvia cuando se usa la aplicación). A continuación podemos ver un diagrama de flujo de los controllers con sus respectivas vistas:



# Lenguaje

Contamos también como buena experiencia de práctica, desarrollo y aprendizaje que el proyecto en su totalidad se encuentra confeccionado en inglés. Desde el código, nombres de Entidades, Clases, variables y métodos hasta la interfaz gráfica.

# Diagrama UML de clases

Ya que el presente diagrama no pude ser visualizado cómodamente por este medio, adjuntamos a continuación un enlace donde podrá inspeccionarse cada parte del proyecto en formato UML.

<https://lucid.app/lucidchart/d9530737-32ad-40b4-b95b-aaeeeacc5233/edit?invitationId=inv_90b807dd-4810-4f96-a2fb-f541d0ef2781>

Referencias

1. GUI, Graphical User Interface: <https://es.wikipedia.org/wiki/Interfaz_gr%C3%A1fica_de_usuario>
2. Model-view-controller:

<https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador>

1. Control de versiones:

<https://git-scm.com/book/es/v2/Inicio---Sobre-el-Control-de-Versiones-Acerca-del-Control-de-Versiones>

1. JavaFx:

<https://dev.java/learn/javafx/>

1. JBcrypt:

<https://www.mindrot.org/projects/jBCrypt/>