

Universidad de Santiago de Chile  
Facultad de Ingeniería  
Departamento de Ingeniería Informática  
Paradigmas de Programación

# PARADIGMAS DE PROGRAMACIÓN

## PROYECTO SEMESTRAL DE LABORATORIO

Versión Preliminar - actualizada al 3/10/2020

Durante el semestre se trabajarán distintos paradigmas de programación, los cuales serán abordados mediante cuatro lenguajes de programación en cuatro entregas de laboratorio. Para el desarrollo satisfactorio del laboratorio del curso se requerirá desarrollar un sistema de foros similar a Stack Overflow.

El software en su versión final (Laboratorio N°4) deberá tener una interfaz gráfica que permita una interacción similar a la presentada en el sistema de preguntas y respuestas stackoverflow.

### 1. Descripción general

Varios aspectos del software quedarán abiertos a la creatividad de cada estudiante, sin embargo se debe cumplir con requerimientos mínimos obligatorios, los cuales se especificarán en las siguientes secciones.

Los primeros tres laboratorios tendrán interfaces claramente definidas (esto es, formatos de lectura, cabeceras de funciones, etc). **El no respetar estas interfaces/prototipos implicarán la obtención de la nota mínima y por consecuencia la reprobación del laboratorio.**

Para este semestre se ha definido hacer una simulación de un software de sistema de foros, tomando como ejemplo la implementación de **Stack Overflow**. Al final del semestre los alumnos de Paradigmas de Programación habrán implementado distintas versiones de esta simulación de software en distintos lenguajes de programación, según los paradigmas que los rigen. A continuación se listan de forma general características del software que serán abordadas de forma obligatoria o complementaria durante el semestre en los distintos proyectos. Los detalles para cada laboratorio se listan posteriormente.

Stack Overflow es un foro que construye una base de conocimiento sobre programación, permitiendo plantear preguntas que luego pueden ser respondidas por otros usuarios.

- Stack Overflow tiene los siguientes elementos claves a considerar: Etiquetas, Preguntas, respuestas y Usuarios.

La página principal del sitio, una vez se inicia sesión, se visualiza de la siguiente manera:



## 1.1 Etiquetas:

*“Una etiqueta es una palabra clave o un rótulo que categoriza tu pregunta con otras preguntas similares. El uso de las etiquetas correctas facilita que otros encuentren una pregunta y la respondan”.*

En Stack overflow se encuentran múltiples etiquetas, por ejemplo: [php](#), [javascript](#), [python](#), etc...

Cada etiqueta posee una **descripción** sobre en qué consiste que el usuario puede ver al situar el puntero sobre ella. Además, un usuario puede seguir una etiqueta si esta le interesa, lo que le permite ver en su inicio preguntas y respuestas relacionadas a esta.

## 1.2 Preguntas:

Las preguntas publicadas por los usuarios en el sitio tienen las siguientes características principales:

- Votos
- Respuestas (answers)

- Visualizaciones
- Etiquetas
- Título
- Contenido
- Fecha de publicación
- Fecha de última actividad (respuesta, modificación, etc)
- Autor (usuario)
- Estado
- Recompensa (Opcional): El usuario que plantea una pregunta puede ofrecer una recompensa en puntos de reputación, los cuales se deducen de su propia reputación.

Los usuarios pueden publicar una pregunta, editarla o eliminarla. Además, el usuario que publicó una pregunta puede seleccionar la que, a su gusto, es la mejor respuesta.

Otros usuarios pueden votar (positiva o negativamente) tanto por la pregunta como por sus respuestas.

- Las preguntas permiten ingresar texto con formato (negrita, cursiva), imágenes, enlaces y código.

### 1.3 Usuario y Reputación:

Otra de las partes esenciales de Stack Overflow son los usuarios. Estos son quienes plantean y responden las preguntas del sitio y, para asegurar la calidad del contenido, el sitio provee un sistema de reputación de usuarios.

Reputación: Se trata de una medida que representa cuánto confía la comunidad en un usuario en particular y se gana ofreciendo respuestas y haciendo preguntas útiles que contribuyan a generar conocimiento.

A continuación hay un listado de acciones en el sitio que afectan a la reputación del usuario:

Se gana reputación cuando:

- tu pregunta es votada a favor: +10
- tu respuesta es votada a favor: +10
- tu respuesta es marcada como "aceptada": +15 (y +2 para quien "acepta" la respuesta)
- una edición sugerida es aceptada: +2 (hasta +1000 en total por usuario)
- una recompensa es otorgada manualmente a tu respuesta: + el total del monto ofrecido como recompensa

Se pierde reputación cuando:

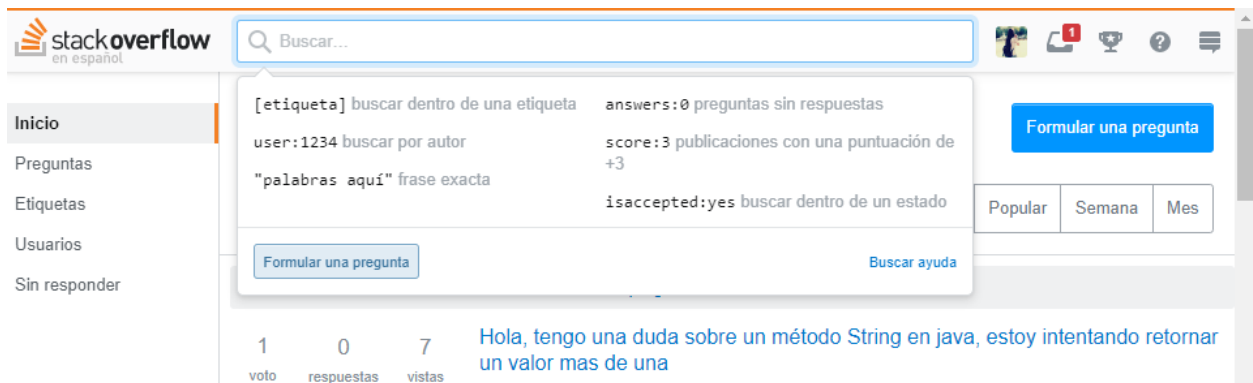
- tu pregunta es votada en contra: -2

- tu respuesta es votada en contra: -2
- tú votas en contra una respuesta de otro usuario: -1
- tú ofreces una recompensa sobre una pregunta: - el total del monto ofrecido como recompensa
- una publicación que has hecho recibe 6 reportes de spam u ofensivo: -100

## 1.4 Opciones de búsqueda:

Para buscar entre los posts de Stack Overflow, el buscador del sitio ofrece varias características interesantes que permiten encontrar información rápidamente.

- Buscar por etiquetas: Para buscar por etiquetas el usuario debe colocar la palabra entre corchetes. Ej: [javascript]
- Buscar por palabras exactas: Para buscar por palabras exactas el usuario debe colocar la palabra o frase a buscar entre comillas. Ej: "control de versiones".
- Buscar por cantidad de respuestas: para buscar por cantidad de respuestas el usuario debe escribir la palabra "answers" seguida de dos puntos y la cantidad de respuestas que desea que tengan los resultados. Ej: answers:0
- Buscar por puntaje: para buscar por puntaje el usuario debe escribir la palabra "score" seguida de dos puntos y el puntaje que desea que tengan los resultados. Ej: score: 2



**Comodines:** Durante el semestre cada alumno dispondrá de **CUATRO comodines (uno para cada laboratorio)** que podrá utilizar **solo en el siguiente caso:**

- **Si su calificación** en los laboratorios 1, 2, 3 y parcialmente en el 4 (Ya sea por requerimientos funcionales, no funcionales, informe) **es mayor igual que 2.0 e inferior a 4.0.**

El comodín permitirá al alumno entregar lo necesario (requerimientos funcionales, no funcionales y/o informe) para alcanzar la nota mínima requerida (4.0) en la correspondiente entrega de laboratorio para la aprobación al final del semestre. **Es fundamental que los estudiantes siempre entreguen un avance (dentro de los plazos establecidos para cada laboratorio) que permita alcanzar al menos una calificación igual o superior a 2.0 para poder optar al uso de comodines.** Recordar que para aprobar el laboratorio de esta asignatura, todos los laboratorios deben tener una calificación igual o mayor a 4.0.

Antes de proceder al uso de comodines, y si lo amerita, se recomienda a los alumnos que empleen las instancias de correcciones de laboratorio.

En caso de requerir el uso de comodín, podrá hacer entrega del correspondiente laboratorio corregido al final del semestre. La fecha específica y espacio de entrega estará disponible en CampusVirtual al final del semestre.

Finalmente, por temas de tiempo considerando el avance del semestre, en el laboratorio final se podrá hacer un uso limitado del comodín. Esto quedará a disposición de los profesores correctores dependiendo del nivel de los elementos que son calificados como insuficientes y la factibilidad de poder completarlos en los plazos para el cierre oficial del semestre dentro de las semanas lectivas 17.

# Instrucciones generales para la entrega de TODOS los laboratorios

Versión 1.0

(Cambios menores pueden incorporarse en futuras versiones a fin de aclarar o corregir errores)

(Sus dudas las puede expresar en este mismo enunciado, incluso puede responder a preguntas de compañeros en caso de que conozca la respuesta)

**Entregables:** Archivo ZIP con el siguiente nombre de archivo: labN\_rut\_ApellidoPaterno.zip (ej: lab1\_12123456\_Perez.zip). **Notar que no incluye dígito verificador.** Dentro del archivo se debe incluir:

1. Informe en formato Word o PDF.
2. Carpeta con código fuente.
3. Archivo leeme.txt para cualquier instrucción especial para ejecución u otro, según aplique.
4. Autoevaluación.txt donde debe incluir una lista completa de todos los requerimientos funcionales y no funcionales señalando una evaluación para cada uno según la siguiente escala:
  - a. 0: No realizado.
  - b. 0.25: Implementación con problemas mayores (funciona 25% de las veces o no funciona)
  - c. 0.5: Implementación con funcionamiento irregular (funciona 50% de las veces)
  - d. 0.75: Implementación con problemas menores (funciona 75% de las veces)
  - e. 1: Implementación completa sin problemas (funciona 100% de las veces)
5. Archivo repositorio.txt que incluya la URL de su repositorio privado en GitHub<sup>1</sup> el que deberá además ser compartido con los profesores y ayudantes al momento de la evaluación a la cuenta **paradigmasdiinf** de GitHub (no lo comparta antes ya que las invitaciones expiran después de un tiempo). El nombre final del repositorio a compartir debe ser el mismo del entregable en Campus Virtual.

El nombre del archivo con el código fuente debe seguir el siguiente formato: NombreArchivo\_rut\_Apellidos.extension (ej: lab\_git\_12123456\_PerezPeña.rkt)

**Se aplicarán descuentos en caso de no incluir estos datos**

---

<sup>1</sup> Ver tutorial de Git en Campus Virtual donde se explica como activa Student Pack para poder crear repositorios privados de forma gratuita.

**Informe del proyecto:** Extensión no debe superar las **10 planas** de contenido (se excluyen: portada, índice, referencias, anexos), para más detalle de requerimientos del informe [ver descripción en nuestro espacio en Campus Virtual](#). El informe debe incluir introducción, descripción breve del problema, análisis del problema, diseño de la solución (esquemática y explicada brevemente), consideraciones de implementación (ej: algoritmos, bibliotecas), **instrucciones con ejemplos claros de uso**, resultados obtenidos, evaluación completa y conclusiones.

**Repositorio en Git del Proyecto:** El laboratorio deberá ser desarrollado incluyendo el sistema de control de versiones Git y utilizando GitHub como servidor remoto. **El repositorio debe ser privado durante la elaboración del proyecto.** Cualquier inconsistencia entre lo entregado en Campus Virtual y la versión final del repositorio será evaluado con nota mínima. **Para el proceso de revisión se considerará la versión subida a CampusVirtual.**

**Evaluación:** El Informe (Inf), requerimientos funcionales (RF), requerimientos no funcionales (RNF), ejecución (Eje) se evalúan por separado. La nota final de este laboratorio (NL) se calcula de la siguiente forma considerando sólo la calificación base a partir del cumplimiento de los RF y RNF obligatorios.

```
if (Inf >= 4.0 && (RF + 1.0) >= 4.0 && (RNF + 1.0) >= 4.0) then
    NL = 0.1 · Inf + 0.7 · (RF + 1.0) + 0.2 · (RNF + 3.0)
else
    NL = min(Inf, RF + 1.0, RNF + 1.0)
```

# Laboratorio 1 (Paradigma Funcional - Lenguaje Scheme)

Versión 1.0

(Cambios menores pueden incorporarse en futuras versiones a fin de aclarar o corregir errores)

(Sus dudas las puede expresar en este mismo enunciado, incluso puede responder a preguntas de compañeros en caso de que conozca la respuesta)

**Fecha de Entrega:** 9-11-2020 (Vespertino)

**Objetivo del laboratorio:** Aplicar conceptos del paradigma de programación funcional usando el lenguaje de programación Scheme en la resolución de un problema acotado.

**Resultado esperado:** Programa que simula las funcionalidades y comportamientos de un sistema de preguntas y respuestas como stackoverflow.



**Requerimientos No Funcionales obligatorios. Algunos son ineludibles, esto quiere decir que al no cumplir con dicho requerimiento, su proyecto será evaluado con la nota mínima.**

1. **(obligatorio) Autoevaluación:** Incluir autoevaluación de cada uno de los requerimientos funcionales (obligatorios y extras) solicitados.
2. **(obligatorio) Lenguaje:** La implementación debe ser en el lenguaje de programación Scheme.
3. **(obligatorio) Versión:** Usar DrRacket versión 6.11 o superior
4. **(obligatorio) Standard:** Se deben utilizar funciones estándar de dicho lenguaje y no aquellas propias Racket. Si considera integrar su solución posteriormente con C# en el laboratorio 4 debe registrarse por standard R6RS.
5. **(obligatorio) No variables:** No hacer uso de función `define` **en combinación** con otras como `set!` (o similares) para emular el trabajo con variables.
6. **(1 pts) Documentación:** Todas las funciones deben estar debidamente comentadas. Indicando descripción de la función, argumentos y retornos. En caso de que la función sea recursiva, indicar el tipo de recursión utilizada y el porqué de esta decisión.
7. **(obligatorio) Dom->Rec:** Respetar la definición de función en términos de conjunto de salida (dominio) y llegada (recorrido) sin efectos colaterales, además del nombre de las mismas (respetar mayúsculas y minúsculas).
8. **(1 pts) Organización:** Estructurar su código en archivos independientes. Un archivo para cada TDA implementado y uno para el programa principal donde solo se dispongan sólo las funciones requeridas en el apartado de requerimientos obligatorios.
9. **(2.5 pts) Historial:** Historial de trabajo en Github tomando en consideración la evolución en el desarrollo de su proyecto en distintas etapas. Se requieren **al menos 10 commits** distribuidos en un periodo de tiempo **mayor o igual a 2 semanas (no espere a terminar la materia para empezar a trabajar en el laboratorio. Puede ir haciendo pequeños incrementos conforme avance el curso)**. Los criterios que se consideran en la evaluación de este ítem son: fecha primer commit, fecha último commit, total commits y máximo de commits diarios. A modo de ejemplo (y solo como una referencia), si hace todos los commits el día antes de la entrega del proyecto, este ítem tendrá 0 pts. De manera similar, si hace dos commits dos semanas antes de la entrega final y el resto los concentra en los últimos dos días, tendrá una evaluación del 25% para este ítem (0.375 pts). Por el contrario, si demuestra constancia en los commits (con aportes claros entre uno y otro) a lo largo del periodo evaluado, este ítem será evaluado con el total del puntaje.

10. **(obligatorio) Ejemplos:** Al final de su código incluir al menos 3 ejemplos de uso para cada una de las funciones correspondientes a requerimientos funcionales obligatorios y los extra. **Solo se revisarán aquellas funciones para las que existan los ejemplos provistos.**
11. **(obligatorio) Prerrequisitos:** Para cada función se establecen prerrequisitos. Estos deben ser cumplidos para que se proceda con la evaluación de la función implementada. Ej: Para evaluar la función login, debe estar implementada la función register.

**Requerimientos Funcionales. Para que el requerimiento sea evaluado, DEBE cumplir con el prerequisite de evaluación y requisito de implementación. En caso contrario la función no será evaluada.**

1. **(0.5 pts) TDAs.** Implementar abstracciones apropiadas para el problema. Recomendamos leer el enunciado completo a fin de que analice el problema y determine el o los TDAs y representaciones apropiadas para la implementación de cada uno. En particular, las funciones obligatorias asumen una estructura de repositorios lineal (solo master y por tanto sin ramas), mientras que algunas funciones extras se enfocan en el uso de ramas. Luego, planifique bien su enfoque de solución de manera que los TDAs y representaciones escogidos sean aplicables a ambos tipos de funciones.

Para la implementación debe registrarse por la estructura de especificación e implementación de TDA vista en clases: Representación, Constructores, Funciones de Pertenencia, Selectores, Modificadores y Otras Funciones. Procurar hacer un uso adecuado de esta estructura a fin de no afectar la eficiencia de sus funciones. En el resto de las funciones se debe hacer un uso adecuado de la implementación del TDA (ej: usar selectores, modificadores, constructores, según sea el caso. No basta con implementar un TDA y luego NO hacer uso del mismo). **Solo implementar las funciones necesarias dentro de esta estructura.**

Dejar claramente documentado con comentarios en el código aquello que corresponde a la estructura base del TDA. Una estructura base que deberá considerar para el resto de las funciones corresponde a "stack" que corresponde al contenedor de preguntas, respuestas, usuarios y nombre del usuario con sesión activa.

Debe contar además con representaciones para que los usuarios puedan tener información relativa a sus credenciales de acceso, reputación y otros elementos que considere relevante para abordar el problema.

De igual forma, las preguntas, además de información relativa a fecha de publicación, ID único incremental (se va generando un correlativo a partir de la última pregunta registrada), debe contar con información relativa a votos a favor, votos en contra, recompensas, número de visualizaciones, estado (cerrada/abierta), listado de respuestas, reportes de ofensa y otros datos que considere relevante para abordar el problema.

Para el caso de las respuestas, se registra el autor de la respuesta, fecha de respuesta, ID único incremental en el contexto de la pregunta (se va generando un correlativo a partir de la última respuesta registrada), votos a favor, votos en contra, estado de aceptación (si/no), reportes de ofensa y otros datos que considere relevante para abordar problema.

<b>Prerrequisitos para evaluación</b>	Ninguno
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"> <li>- Usar estructuras basadas en listas</li> <li>- Especificar representación de manera clara para cada TDA implementado (en el informe y en el código a través de comentarios). Luego implementar constructores, funciones de pertenencia, y según se requiera selectores, modificadores y otras funciones que pueda requerir para las otras funciones requeridas a continuación.</li> </ul>
<b>Ejemplo (Solo de referencia. Cada estudiante define sus propios TDAs)</b>	<p><i>(stack .....)</i> ; constructor de stack</p> <p><i>(questions stack)</i> ;selector preguntas desde el stack</p> <p><i>(answers stack questionID)</i> ;selector respuestas pregunta</p> <p><i>etc.</i></p>

2. **(0.5 pts) register:** Función que permite registrar a un nuevo usuario en el stack. Para esto se solicita el stack, nombre del usuario (identificador único, se debe verificar que no exista para su correcto registro) y contraseña. El retorno de la función es un stack con el nuevo usuario registrado.

<b>Prerrequisitos para evaluación</b>	Implementación de TDAs
<b>Requisitos de implementación</b>	- Emplear recursión natural
<b>Dominio</b>	stack X string X string
<b>Recorrido</b>	stack
<b>Encabezado</b>	<i>(register stack username password)</i>
<b>Ejemplos</b>	<i>(register stackOverflow "user" "pass")</i>

3. **(0.5 pts) login:** Función que permite autenticar a un usuario registrado iniciar sesión y junto con ello permite la ejecución de comandos concretos dentro del stack. Si la autenticación es válida (i.e., que el usuario existe). El retorno es una función currificada correspondiente a la operación (operation) parcialmente evaluada con el parámetro stack actualizado. La actualización del stack incorpora al usuario autenticado en sesión activa (fundamental para que el comando pueda funcionar). De lo contrario retorna solo operation.

<b>Prerrequisitos para evaluación</b>	Implementación de TDAs, función register
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"><li>- Usar recursión natural o de cola</li><li>- Debe emplear funciones de orden superior.</li><li>- Retorno es una función currificada</li><li>- Usar recursión natural o de cola (en alguna de las funciones debe hacer uso de alguno de los estilos)</li><li>- Emplear recursión de cola</li></ul>
<b>Dominio</b>	stack X string X string X function
<b>Recorrido</b>	function
<b>Recorrido final</b>	stack
<b>Encabezado</b>	<i>(login stack username password operation)</i>
<b>Ejemplos</b>	<i>(login stackOverflow "user" "pass" ask)</i>

4. **(0.5 pts) ask:** Función currficada que permite a un usuario con sesión iniciada en la plataforma realizar una nueva pregunta. Cada pregunta registra el autor de la misma (obtenido desde la sesión iniciada con login), fecha de publicación, la pregunta y 0 o más etiquetas (en este orden). El retorno final de la función es una versión actualizada del stack donde se registra la nueva pregunta y se elimina la sesión activa del usuario en el stack.

<b>Prerrequisitos para evaluación</b>	Implementación de TDAs, función login
<b>Requisitos de implementación</b>	<p>- Esta función no arroja ningún cambio en stack si no se utiliza como parámetro de entrada de la función login que es la que inicia la sesión del usuario.</p> <p>- La función debe estar currficada solo para los dos primeros parámetro. El retorno para la evaluación sobre el parámetro stack es una función que opera sobre el resto de los argumentos. Retorno es una función currficada.</p>
<b>Dominio</b>	stack
<b>Recorrido</b>	function: list ->stack X date X string X string list
<b>Recorrido final</b>	stack
<b>Encabezado</b>	<i>(ask stack)</i>
<b>Ejemplos</b>	<i>((login stackOverflow "user" "pass" ask) (date 30 10 2020)) "mi pregunta" "e1" "e2" "e3")</i>
<b>Tips</b>	<p>Para la definición de la función más interna que retorna ask y que opera sobre los argumentos pregunta y etiquetas, considere la siguiente definición:</p> <p style="text-align: center;"><i>(lambda (question . labels) ..... )</i></p>

5.

6. **(0.5 pto) reward:** Función currificada que permite a un usuario con sesión iniciada en la plataforma ofrecer una recompensa para una determinada pregunta. La recompensa puede ir dirigida a preguntas propias o de terceros. Las preguntas se refieren en base a su identificador único. No se puede ofrecer recompensas mayores a la reputación con la que cuenta el usuario. Una vez ofrecida la recompensa, el puntaje ofrecido queda retenido temporalmente y descontado de la reputación actual. De esta forma se resguarda que el usuario no pueda ofrecer más recompensas de las que podrá eventualmente pagar. Una vez que una respuesta es aceptada, la recompensa se descuenta definitivamente de la reputación del autor y se traspasa al usuario que entregó la respuesta. El resultado de la función es un stack donde se refleja el cambio en el estado de pregunta (recompensa), retenciones en la reputación del usuario que ofrece la recompensa y se elimina la sesión activa del usuario en el stack.

<b>Prerrequisitos para evaluación</b>	Implementación de TDAs, función ask
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"><li>- Esta función no arroja ningún cambio si no se utiliza como parámetro de entrada de la función login que es la que inicia la sesión del usuario.</li><li>- La función debe estar currificada solo para los tres primeros parámetros. El retorno para la evaluación sobre el parámetro stack es una función que opera sobre el resto de argumentos. Retorno es una función currificada.</li></ul>
<b>Dominio</b>	stack
<b>Recorrido</b> <b>Recorrido final</b>	function: integer (IDPregunta) X integer (recompensa) stack
<b>Encabezado</b>	<i>(reward stack)</i>
<b>Ejemplos</b>	<i>((login stackOverflow "user" "pass" reward) 123) 300)</i>



7. **(0.5 pts) answer:** Función currificada que permite a un usuario con sesión iniciada en la plataforma responder una pregunta. Cada respuesta registra al autor de la misma (obtenido desde la sesión iniciada con login), se vincula al ID de la pregunta, incluye fecha de publicación, la respuesta y 0 o más etiquetas (en este orden). El retorno final de la función es una versión actualizada del stack donde se registra la nueva respuesta vinculada a la pregunta y se elimina la sesión activa del usuario en el stack.

<b>Prerrequisitos para evaluación</b>	Implementación de TDAs, función reward
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"> <li>- Esta función no arroja ningún cambio si no se utiliza como parámetro de entrada de la función login que es la que inicia la sesión del usuario.</li> <li>- La función debe estar currificada solo para los tres primeros parámetros. El retorno para la evaluación sobre el parámetro stack es una función que opera sobre el resto de argumentos. Retorno es una función currificada.</li> </ul>
<b>Dominio</b>	stack
<b>Recorrido</b> <b>Recorrido final</b>	function: list -> stack X date X string X string list stack
<b>Encabezado</b>	<i>(answer stack)</i>
<b>Ejemplos</b>	<i>(((((login stackOverflow "user" "pass" answer) (date 25 10 2020)) 132) "mi respuesta" "e1" "e2" "e3"))</i>
<b>Tips</b>	Para la definición de la función más interna que retorna answer y que opera sobre los argumentos respuesta y etiquetas, considere la siguiente definición:  <i>(lambda (answer . labels) ..... )</i>

8. **(1 pts) accept:** Función currificada que permite a un usuario con sesión iniciada en la plataforma aceptar una respuesta a una de sus preguntas. La pregunta y la respuesta involucrada se refieren con sus respectivos IDs. Un usuario solo puede aceptar respuestas a sus propias preguntas. El resultado de la función es un stack donde se refleja el cambio en el estado de pregunta, respuesta, **actualización de puntajes** según indicaciones señaladas al comienzo de este enunciado (votos, recompensas vigentes, etc.) y se elimina la sesión activa del usuario en el stack. .

<b>Prerrequisitos para evaluación</b>	Implementación de TDAs, función answer
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"><li>- Esta función no arroja ningún cambio si no se utiliza como parámetro de entrada de la función login que es la que inicia la sesión del usuario.</li><li>- Función debe ser implementada de forma currificada</li></ul>
<b>Dominio</b>	stack
<b>Recorrido</b> <b>Recorrido final</b>	function: integer (IDPregunta) X integer (IDRespuesta) stack
<b>Encabezado</b>	<i>(accept stack)</i>
<b>Ejemplos</b>	<i>((login stackOverflow "user" "pass" accept) 10) 5)</i>

9. **(0.5 pts) stack->string:** Función que recibe un stack y entrega una representación del mismo como un string posible de visualizar de forma comprensible al usuario. Debe hacer uso del char ‘\n’ para los saltos de línea. **No utilice las funciones write y display dentro de esta función**, ya que debe retornar un string el cual pueda luego ser pasado como argumento a la función “write” o “display” para poder visualizarlo de forma comprensible al usuario.

<b>Prerrequisitos para evaluación</b>	Implementación de TDAs, funciones ask y answer.
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"><li>- El string resultante al ser impreso por la función write o display debe ilustrar con claridad los elementos de un stack (i.e., preguntas con sus respuestas, usuarios, sus valoraciones, etc.).</li><li>- El resultado de aplicar string-&gt;stack directamente sobre stack sin usar función login de por medio, imprime todo el stack.</li><li>- El resultado de aplicar string-&gt;stack a través de la función login entrega como resultado todos los antecedentes (preguntas formuladas con sus respectivas respuestas) junto a los detalles del usuario que inició sesión .</li></ul>
<b>Dominio</b>	stack
<b>Recorrido</b>	string
<b>Encabezado</b>	(stack->string stack)
<b>Ejemplos</b>	<i>(login stackOverflow “user” “pass” stack-&gt;string)</i> <i>(stack-&gt;string stackOverflow )</i>

**De las siguientes funciones implementar sólo 1 conforme al cumplimiento de los prerequisites. La nota máxima en total será un 7.0 aun cuando se desborde puntaje.**

10. **(2 pts) vote:** Función que permite a un usuario votar por una pregunta o respuesta. El voto puede ser positivo o negativo. Para esto se debe entregar la pregunta o respuesta sobre la que se desea efectuar el voto y el tipo de voto (positivo o negativo) y el usuario que efectúa el voto. Los puntajes a favor o en contra se deben ajustar a las indicadas presentadas al comienzo de este enunciado. El resultado de la función es un nuevo stack con los cambios producidos por el voto y se elimina la sesión activa del usuario en el stack.

<b>Prerrequisitos para evaluación</b>	Implementación de TDAs, funciones ask, reward, answer, accept y stack->string
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"> <li>- Para esta función debe implementar dos funciones complementarias (questions y answers). Estas funciones se emplean como argumento de vote para determinar donde se aplica el voto.</li> <li>- La función complementaria getQuestion permite obtener una pregunta en base a su ID. Por otro lado, getAnswer permite seleccionar la respuesta identificada con ID para una pregunta en base a su ID. Estas funciones deben estar currificadas para que sean empleadas internamente por vote.</li> <li>- La función vote debe implementarse de manera currificada</li> </ul>
<b>Dominio</b>	stack
<b>Recorrido</b>	function: function X integer (ID pregunta o respuesta) X boolean (true: positivo   false: negativo)
<b>Recorrido final</b>	stack
<b>Encabezado</b>	(vote stack)
<b>Ejemplos</b>	<p><i>;voto positivo para la pregunta 25</i></p> <p><i>(((((login stackOverflow "user" "pass" vote) getQuestion) 25) true)</i></p> <p><i>;voto negativo para la respuesta 4 de la pregunta 10</i></p> <p><i>(((((login stackOverflow "user" "pass" vote) (getAnswer 10)) 4) false)</i></p>

11. **(2 pts) reportOffense:** Función que permite a un usuario reportar preguntas o respuestas que contengan contenido ofensivo. Este contenido puede encontrarse en preguntas o respuestas. La función recibe como entrada el stack y el ID de pregunta o el ID de pregunta y respuesta que se desea reportar. El retorno de la función corresponde a un stack con el reporte de ofensa en la pregunta o respuesta, ajuste de puntaje asociado a reputación (en caso de que se hayan alcanzado 6 o más votos de ofensa, en cuyo caso se aplica descuento según indicaciones provistas al comienzo del enunciado) y se elimina la sesión activa del usuario en el stack.

<b>Prerrequisitos para evaluación</b>	Implementación de TDAs, funciones ask, reward, answer, accept y stack->string
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"><li>- Para esta función debe implementar dos funciones complementarias (questions y answers). Estas funciones se emplean como argumento de vote para determinar donde se aplica el voto.</li><li>- La función complementaria getQuestion permite obtener una pregunta en base a su ID. Por otro lado, getAnswer permite seleccionar la respuesta identificada con ID para una pregunta en base a su ID. Estas funciones deben estar currificada para que sean empleadas internamente por reportOffense.</li><li>- La función vote debe implementarse de manera currificada</li></ul>
<b>Dominio</b>	stack
<b>Recorrido</b> <b>Recorrido final</b>	function: function X integer (ID pregunta o respuesta)  stack
<b>Encabezado</b>	(reportOffense stack)
<b>Ejemplos</b>	<i>;reporte de ofensa para la pregunta 25</i>  <i>((login stackOverflow "user" "pass" reportOffense) getQuestion) 25)</i>  <i>;reporte de ofensa para la respuesta 4 de la pregunta 10</i>  <i>((login stackOverflow "user" "pass" reportOffense) (getAnswers 10)) 4)</i>

12. **(2 pts) ranking:** Función que retorna un ranking de las preguntas o respuestas de una pregunta según votación, pudiendo ser el orden ascendente (menos votado al mas votado) o descendente (más votado al menos votado). La función se puede usar directamente sin pasar por login. El retorno es una lista de elementos (preguntas ordenadas).

<b>Prerrequisitos para evaluación</b>	Implementación de TDAs, funciones ask, reward, answer, accept y stack->string
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"><li>- Para esta función debe implementar las siguientes funciones complementarias: questions, answers, users, asc, desc. Estas funciones se emplean como argumento de ranking para determinar que se quiere rankear y en qué orden.</li><li>- La función complementaria questions permite obtener las preguntas dentro de un stack. Por otro lado, getAnswer permite seleccionar la respuesta para una pregunta a través de su ID. Por último, la función users permite obtener el listado de usuarios. Estas funciones deben estar currificada para que sean empleadas internamente por ranking.</li><li>- La función ranking debe implementarse de manera currificada</li><li>- Puede usar funciones de ordenamiento de listas que ya vienen implementadas en Scheme o Racket.</li></ul>
<b>Dominio</b>	stack
<b>Recorrido</b>	function: function X function
<b>Recorrido final</b>	list
<b>Encabezado</b>	(ranking stack)
<b>Ejemplos</b>	<pre><i>;ranking ascendente de preguntas</i> <i>(( (ranking stackOverflow) questions) asc)</i>  <i>;ranking descendente de respuestas para pregunta 10</i> <i>(( (login stackOverflow "user" "pass" ranking) (answers 10)) desc)</i>  <i>;ranking ascendente de usuarios según reputación</i> <i>(( (ranking stackOverflow) users) asc)</i></pre>

13. **(2 pts) search:** Función que permite buscar preguntas o respuestas en base a una coincidencia parcial de texto. Se debe identificar el tipo de búsqueda a realizar (si se hará por cualquier texto dentro de la pregunta y/o respuestas incluyendo las etiquetas, o si la búsqueda será solo por etiqueta. La función se puede usar directamente sin pasar por login. El resultado de la búsqueda debe ser un listado de las preguntas y/o respuestas que cumplen con criterio de búsqueda.

<b>Prerrequisitos para evaluación</b>	Implementación de TDAs, funciones ask, reward, answer, accept y stack->string
<b>Requisitos de implementación</b>	- Para esta función debe implementar dos funciones complementarias (all y labels). Estas funciones se emplean como argumento de search para concentrar las búsquedas en todo el cuerpo de texto o solo en las etiquetas
<b>Dominio</b>	stack
<b>Recorrido</b> <b>Recorrido final</b>	function: function X string list
<b>Encabezado</b>	(search stack)
<b>Ejemplos</b>	<i>;búsqueda del texto “curry” en todo el cuerpo de texto</i> <i>;retorna preguntas o respuestas que contengan el substring “curry” (ej: currying, currificación)</i> <i>((search stackOverflow) all) “curry”</i>  <i>;búsqueda del texto “racket” en las etiquetas de preguntas y respuestas</i>  <i>((login stackOverflow “user” “pass” search) labels) “racket”</i>

14. **(2 pts) clean:** Función que automáticamente realiza un proceso de limpieza del stack borrando los N elementos según filtro. Los filtros pueden estar orientados a eliminar preguntas con baja votación (worstQuestions) y preguntas previas a una fecha F sin respuesta (before). Por otro lado, la función permite eliminar todas las preguntas reportadas como ofensivas (con más de 6 reportes de ofensa). La función clean se puede aplicar independientemente de login. El resultado de la función es un stack con el contenido eliminado.

<b>Prerrequisitos para evaluación</b>	Implementación de TDAs, funciones ask, reward, answer, accept y stack->string
<b>Requisitos de implementación</b>	- Para esta función debe implementar funciones complementarias que permitan aplicar el filtro.
<b>Dominio</b>	stack
<b>Recorrido</b> <b>Recorrido final</b>	function: function X string stack
<b>Encabezado</b>	(clean stack)
<b>Ejemplos</b>	<i>;Elimina las 5 preguntas con votación más baja</i> <i>((clean stack) worstQuestions) 5)</i> <i>;Elimina las 8 preguntas sin respuestas antes del 5/12/2019</i> <i>((clean stack) (before (date 5 12 2019))) 8)</i> <i>;Elimina todas las preguntas marcadas como ofensivas</i> <i>((clean stack) offenses)</i>



## Laboratorio 2 (Paradigma Lógico - Lenguaje Prolog)

Versión 2.0

(Cambios menores pueden incorporarse en futuras versiones a fin de aclarar o corregir errores)

(Sus dudas las puede expresar en este mismo enunciado, incluso puede responder a preguntas de compañeros en caso de que conozca la respuesta)

**Fecha de Entrega Vespertino:** 30-11-2020

**Fecha de Entrega Diurno:** 3-12-2020

**Objetivo del laboratorio:** Aplicar conceptos del paradigma de programación lógico usando el lenguaje de programación Prolog en la resolución de un problema acotado.

**Resultado esperado:** Programa que simula las funcionalidades y comportamientos de un sistema de preguntas y respuestas como stackoverflow.

**Requerimientos No Funcionales obligatorios. Algunos son ineludibles, esto quiere decir que al no cumplir con dicho requerimiento, su proyecto será evaluado con la nota mínima.**

1. **(obligatorio) Autoevaluación:** Incluir autoevaluación de cada uno de los requerimientos funcionales (obligatorios y extras) solicitados.
2. **(obligatorio) Lenguaje:** La implementación debe ser en el lenguaje de programación Prolog.
3. **(obligatorio) Versión:** Usar Swi-prolog versión 8.x.x .
4. **(1 pts) Documentación:** Todos los predicados deben estar debidamente comentados. Indicando descripción de la relación, términos de entrada y de salida.
5. **(2.5 pts) Historial:** Historial de trabajo en Github tomando en consideración la evolución en el desarrollo de su proyecto en distintas etapas. Se requieren **al menos 10 commits** distribuidos en un periodo de tiempo **mayor o igual a 1 semanas (no espere a terminar la materia para empezar a trabajar en el laboratorio. Puede ir haciendo pequeños incrementos conforme avance el curso)**. Los criterios que se consideran en la evaluación de este ítem son: fecha primer commit, fecha último commit, total commits y máximo de commits diarios. A modo de ejemplo (y solo como una referencia), si hace todos los commits el día antes de la entrega del proyecto, este ítem tendrá 0 pts. De manera similar, si hace dos commits dos semanas antes de la entrega final y el resto los concentra en los últimos dos días, tendrá una evaluación del 25% para este ítem (0.375 pts). Por el contrario, si demuestra constancia en los commits (con aportes claros entre uno y otro) a lo largo del periodo evaluado, este ítem será evaluado con el total del puntaje.
6. **(obligatorio) Ejemplos:** Al final de su código incluir al menos 3 ejemplos de uso para cada uno de los predicados correspondientes a requerimientos funcionales obligatorios y los extra. **Solo se revisarán aquellas funcionalidades para las que existan los ejemplos provistos.**
7. **(obligatorio) Prerrequisitos:** Para cada predicado se establecen prerrequisitos. Estos deben ser cumplidos para que se proceda con la evaluación de la funcionalidad implementada. Ej: Para evaluar la funcionalidad *login*, debe estar implementada la funcionalidad *register*.

**Requerimientos Funcionales. Para que el requerimiento sea evaluado, DEBE cumplir con el prerequisite de evaluación y requisito de implementación. En caso contrario la funcionalidad no será evaluada.**

1. **(0.5 pts) TDAs.** Implementar abstracciones apropiadas para el problema. Recomendamos leer el enunciado completo a fin de que analice el problema y determine el o los TDAs y representaciones apropiadas para la implementación de cada uno.

Para la implementación debe regirse por la estructura de especificación e implementación de TDA vista en clases: Representación, Constructores, Predicados de Pertenencia, Selectores, Modificadores y Otros predicados. Procurar hacer un uso adecuado de esta estructura a fin de no afectar la eficiencia de sus predicados. En el resto de los predicados se debe hacer un uso adecuado de la implementación del TDA (ej: usar selectores, modificadores, constructores, según sea el caso. No basta con implementar un TDA y luego NO hacer uso del mismo). **Solo implementar los predicados necesarios dentro de esta estructura.**

Aplican los mismos requisitos del laboratorio anterior para su TDA.

<b>Prerrequisitos para evaluación</b>	Ninguno
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"><li>- Usar estructuras basadas en listas</li><li>- Especificar representación de manera clara para cada TDA implementado (en el informe y en el código a través de comentarios). Luego implementar constructores, funciones de pertenencia, y según se requiera selectores, modificadores y otras funciones que pueda requerir para las otras funciones requeridas a continuación.</li></ul>
<b>Ejemplo (Solo de referencia. Cada estudiante define sus propios TDAs)</b>	<i>stack( .....SOut) ; constructor de stack</i>  <i>stackQuestions(SOut, Questions) ;selector preguntas desde el stack</i>  <i>stackAnswers(SOut, QuestionID, Answers) ;selector de respuestas a una pregunta</i>  <i>etc.</i>

2. **(0.5 pts) hechos:** A partir de su implementación de los TDAs, proporcionar una base de hechos que contengan al menos: 2 stacks, 4 usuarios registrados, 5 preguntas y 10 respuestas.

<b>Prerrequisitos para evaluación</b>	Implementación de TDAs
---------------------------------------	------------------------

3. **(0.5 pts) register:** Predicado que permite consultar el valor que toma un Stack al ocurrir el registro de un nuevo usuario. Para esto se ingresa un stack inicial, nombre del usuario (identificador único, se debe verificar que no exista para su correcto registro), contraseña y el Stack resultante luego del registro. El retorno del predicado es *true* en caso que se pudo satisfacer el registro del usuario.

<b>Prerrequisitos para evaluación</b>	Implementación de TDAs y base de hechos
<b>Requisitos de implementación</b>	<p>- Debe funcionar cuando se entrega un Stack2 como variable sin valor como también teniendo un valor.</p> <ul style="list-style-type: none"><li>• En el caso de ser una variable esta se unifica con el Stack resultante luego de registrar al usuario.</li><li>• En caso de ser un valor concreto se verifica si el Stack2 cumple con ser el resultante luego de registrar al usuario.</li></ul>
<b>Dominio</b>	stack X string X string x stack
<b>Encabezado</b>	<i>stackRegister(Stack, Username, Password, Stack2).</i>
<b>Ejemplos</b>	<i>stackRegister(S1, "user", "pass", S2).</i>

4. **(0.5 pts) login:** Predicado que permite autenticar a un usuario registrado. Si la autenticación es válida (i.e., que el usuario existe y su contraseña es correcta). El retorno es *true*. La actualización del stack incorpora al usuario autenticado en sesión activa (fundamental para que los predicados siguientes puedan funcionar).

<b>Prerrequisitos para evaluación</b>	Implementación de TDAs, predicado <i>register</i>
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"><li>- Si "Stack2" es una variable, se debe unificar con el stack resultante luego de agregar una sesión activa con el usuario que realizó login.</li><li>- Si el usuario y/o la contraseña son incorrectas se debe retornar <i>false</i>.</li></ul>
<b>Dominio</b>	stack X string X string X stack
<b>Encabezado</b>	<i>login(Stack, Username, Password, Stack2).</i>
<b>Ejemplos</b>	<i>login(Stack, "user", "pass", Stack2).</i>

5. **(0.5 pts) ask:** Predicado que permite a un usuario con sesión iniciada en la plataforma realizar una nueva pregunta. Cada pregunta registra el autor de la misma (obtenido desde la sesión iniciada con login), fecha de publicación, la pregunta, un listado de etiquetas. El retorno es *true* si se puede satisfacer en “Stack2” el stack con la nueva pregunta incluida y sin la sesión activa del usuario que realizó la pregunta.

<b>Prerrequisitos para evaluación</b>	Implementación de TDAs, predicado login
<b>Requisitos de implementación</b>	<p>-Debe funcionar cuando se entrega un Stack como variable sin valor como también teniendo un valor.</p> <ul style="list-style-type: none"><li>• En el caso de ser una variable esta se unifica con el Stack2 resultante luego de agregar la pregunta y eliminar la sesión del usuario activo.</li><li>• En caso de ser un valor concreto se verifica si el Stack cumple con ser el resultante luego de haber realizado la pregunta y sin la sesión del usuario activo.</li></ul> <p>- En caso de no haber una sesión activa debe retornar <i>false</i>.</p> <p>- La fecha puede estar representada como lo estime conveniente, pero debe indicarlo en los ejemplos de uso.</p>
<b>Dominio</b>	stack X date X string X string list x stack
<b>Encabezado</b>	<i>ask(Stack, Fecha, TextoPregunta, ListaEtiquetas, Stack2).</i>
<b>Ejemplos</b>	<i>ask(Stack, “2020-10-30”, “mi pregunta”, [“e1” “e2” “e3”], Stack2).</i>

6.

7. **(0.5 pts) answer:** Predicado que permite consultar el valor que toma un Stack al realizar la respuesta a una pregunta. Para esto se ingresa un stack inicial con sesión iniciada, la fecha de la respuesta, el id de la pregunta, el texto de la respuesta, un listado de etiquetas y el Stack resultante luego del registro. El retorno del predicado es *true* en caso que se pudo satisfacer la creación de esta respuesta en el Stack resultante.

<b>Prerrequisitos para evaluación</b>	Implementación de TDAs, predicado <b>ask</b>
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"><li>- Debe retornar <i>false</i> si no existe una sesión activa. en el stack.</li><li>- Debe retornar false si la pregunta con ese id no existe.</li><li>- Para los casos anteriormente descritos debe hacer uso de predicado de corte !.</li><li>- La fecha puede estar representada como lo estime conveniente, pero debe indicarlo en los ejemplos de uso.</li></ul>
<b>Dominio</b>	stack X date X number X string X string list X stack
<b>Encabezado</b>	<i>answer(Stack, Fecha, IdPregunta, TextoRespuesta, ListaEtiquetas, Stack2).</i>
<b>Ejemplos</b>	<i>answer(Stack, "2020-10-31", 132, "mi respuesta", ["e1" "e2" "e3"], Stack2).</i>

8. **(1 pts) accept:** Predicado que permite consultar el valor que toma un Stack al aceptar una respuesta a una de sus preguntas. Para esto se ingresa un stack inicial, el id de una pregunta, el id de una respuesta y el Stack resultante luego de aceptar la respuesta. El retorno del predicado es *true* en caso que se pudo satisfacer la aceptación de la pregunta. Un usuario solo puede aceptar respuestas a sus propias preguntas. El stack resultante debe reflejar el nuevo estado de pregunta, respuesta, **actualización de puntajes** según indicaciones señaladas al comienzo de este enunciado, y la sesión activa del usuario en el stack debe desaparecer.

<b>Prerrequisitos para evaluación</b>	Implementación de TDAs, predicado answer
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"><li>- Debe retornar <i>false</i> si no existe una sesión activa. en el stack.</li><li>- Debe retornar <i>false</i> si la sesión iniciada no es del mismo usuario que hizo la pregunta a la que se está aceptando.</li><li>- Debe retornar <i>false</i> si la pregunta con ese id no existe o si la respuesta con ese Id no existe.</li><li>- Para los casos anteriormente descritos debe hacer uso de predicado de corte !.</li></ul>
<b>Dominio</b>	stack X number X number X stack
<b>Encabezado</b>	<i>accept(Stack, IdPregunta, IdRespuesta, Stack2).</i>
<b>Ejemplos</b>	<i>accept(Stack, 10, 5, Stack2).</i>



9. **(0.5 pts) stackToString:** Predicado que permite consultar el valor que toma un String al intentar representar en éste cierto Stack. Para esto se ingresa un stack inicial y el String equivalente al Stack. El retorno del predicado es *true* en caso que se pudiera satisfacer esta equivalencia entre el stack y el string. En el String equivalente debe hacer uso del char ‘\n’ para los saltos de línea. **No utilice los predicados write y/o display en el desarrollo de este predicado.**

<b>Prerrequisitos para evaluación</b>	Implementación de TDAs, predicado login y ask.
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"><li>- El string resultante al ser impreso por el predicado write o display debe ilustrar con claridad los elementos de un stack (i.e., preguntas con sus respuestas, usuarios, sus valoraciones, sesiones activas, etc.).</li><li>- El resultado de aplicar stackToString directamente sobre stack sin usar función login de por medio, imprime todo el stack.</li><li>- El resultado de aplicar stackToString con una sesión activa debe hacer que la unificación de la variable “StackStr” tenga solo las preguntas realizadas por el usuario con la sesión activa junto a los detalles de ese usuario.</li></ul>
<b>Dominio</b>	stack X string
<b>Encabezado</b>	stackToString(Stack, StackStr).
<b>Ejemplos</b>	<i>stackLogin(Stack, “user”, “pass”, Stack2),</i> <i>stackToString(Stack2, Str),</i> <i>write(Str).</i>

De los siguientes predicados, implementar sólo 1 conforme al cumplimiento de los prerequisites. La nota máxima en total será un 7.0 aun cuando se desborde puntaje.

1. **(2 pts) vote:** Predicado que permite consultar el valor que toma un Stack al ocurrir un voto del usuario con sesión activa a una pregunta o respuesta. Para esto se ingresa un stack inicial, un TDA pregunta o respuesta, el voto que puede ser positivo o negativo y el Stack resultante luego de realizado el voto y sin la sesión activa del usuario que realizó el voto. Los puntajes a favor o en contra se deben ajustar a las indicadas presentadas al comienzo de este enunciado. El retorno del predicado es *true* en caso que se pudo satisfacer el voto del usuario.

<b>Prerrequisitos para evaluación</b>	Implementación de TDAs, predicados ask, reward, answer, accept y stackToString
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"><li>- Para este predicado debe implementar dos predicados complementarios (getQuestion y getAnswer). Estos predicados se emplean para determinar el TDA pregunta o respuesta donde se aplica el voto.</li><li>- El predicado complementario getQuestion permite obtener una pregunta en base a su ID. Por otro lado, getAnswer permite seleccionar la respuesta identificada con ID para una pregunta en base a su ID. El predicado vote debe ser capaz de determinar si su segundo argumento corresponde a una pregunta o una respuesta.</li><li>- Debe retornar <i>false</i> si el voto va para una pregunta que no es del usuario que tiene sesión activa en el stack.</li></ul>
<b>Dominio</b>	stack X Pregunta o Respuesta X boolean X stack
<b>Encabezado</b>	vote(Stack, TdaPreguntaORespuesta, Voto, Stack).
<b>Ejemplos</b>	<pre><i>;voto positivo para la pregunta 25</i> getQuestion(Stack, 25, Preg), vote(Stack, Preg, true, Stack2).  <i>;voto negativo para la respuesta 4 de la pregunta 10</i> getAnswer(Stack, 10, 4, Resp), vote(Stack, Resp, false, Stack2).</pre>

**(2 pts) reportOffense:** Predicado que permite consultar el valor que toma un Stack al reportar preguntas o respuestas que contengan contenido ofensivo. Este contenido puede encontrarse en preguntas o respuestas. Para esto se ingresa un stack inicial, el TDA de pregunta o el TDA de respuesta que se desea reportar y el Stack resultante luego de reportar como ofensiva esta pregunta o respuesta. Además, el stack resultante debe tener el ajuste de puntaje asociado a reputación (en caso de que se hayan alcanzado 6 o más votos de ofensa, en cuyo caso se aplica descuento según indicaciones provistas al comienzo del enunciado) y se elimina la sesión activa del usuario en el stack.

<b>Prerrequisitos para evaluación</b>	Implementación de TDAs, predicados ask, reward, answer y stackToString
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"> <li>- Para este predicado debe implementar dos predicados complementarios (getQuestion y getAnswer).</li> <li>- El predicado complementario getQuestion permite obtener una pregunta en base a su ID. Por otro lado, getAnswer permite seleccionar la respuesta identificada con ID para una pregunta en base a su ID.</li> </ul>
<b>Dominio</b>	stack X Pregunta o Respuesta X stack
<b>Encabezado</b>	reportOffense(Stack1, Question, Stack2).  reportOffense(Stack1, Answer, Stack2).
<b>Ejemplos</b>	<pre> ;reporte de ofensa para la pregunta con ID 25 stackLogin(Stack, "user", "pass", Stack2), getQuestion(Stack2, 25, Q), reportOffense(Stack2, Q, Stack3).  ;reporte de ofensa para la respuesta 4 de la pregunta 10 stackLogin(Stack, "user", "pass", Stack2), getAnswer(Stack2, 10, A), reportOffense(Stack2, A, Stack3). </pre>

**(2 pts) ranking:** Predicado que permite consultar el valor que toma una lista con el ranking de las preguntas ordenadas según votación, pudiendo ser en orden ascendente (menos votado al más votado) o descendente (más votado al menos votado). El predicado se puede usar directamente sin tener una sesión activa en el Stack.

<b>Prerrequisitos para evaluación</b>	Implementación de TDAs, predicados ask, reward, answer, accept y stackToString
<b>Requisitos de implementación</b>	- Puede usar funciones de ordenamiento de listas que ya vienen implementadas en Prolog.
<b>Dominio</b>	stack x string x lista preguntas
<b>Encabezado</b>	ranking(Stack, Orden, PreguntasOrdenadas).
<b>Ejemplos</b>	<i>;ranking ascendente de preguntas</i> <i>ranking(Stack, "ASC", L).</i>  <i>;ranking descendente de preguntas</i> <i>ranking(Stack, "DESC", L).</i>

**(2 pts) search:** Predicado que permite consultar el valor que toman dos listas al buscar por cierto texto en un stack. La primera lista es con las preguntas en que se encontró el texto y la segunda lista es con las respuestas en que se encontró el texto. El predicado se puede usar directamente sin tener una sesión activa mediante login.

<b>Prerrequisitos para evaluación</b>	Implementación de TDAs, predicados ask, reward, answer, accept y stack->string
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"> <li>- Debe encontrar cualquier coincidencia en el cuerpo de todas las preguntas o respuestas. Si existe una coincidencia tanto en una pregunta como en una de las respuestas de la misma pregunta, entonces en la lista resultante debe estar tanto el TDA de la pregunta como el TDA de la respuesta encontrada.</li> </ul>

	<ul style="list-style-type: none"> <li>- El string que se busca puede contener cualquier carácter admitido en el texto de las preguntas o respuestas: Letras, números, espacios, puntos, comas, saltos de línea, etc.</li> </ul>
<b>Dominio</b>	stack x string x lista preguntas x lista respuestas
<b>Encabezado</b>	search(Stack, Texto, ListaPreguntas, ListaRespuestas).
<b>Ejemplos</b>	<p><i>;búsqueda del texto “curr” en todo el cuerpo de texto</i></p> <p><i>;retorna preguntas en LP y respuestas en LR que contengan el substring “curr” (ej: currying, currificación)</i></p> <p>search(Stack, LP, LR, “curr”).</p>

# Laboratorio 3

## (Paradigma Orientado a Objetos - Lenguaje Java)

Versión 1.4 - Última Edición: 08/01/2021

(Cambios menores pueden incorporarse en futuras versiones a fin de aclarar o corregir errores)

(Sus dudas las puede expresar en este mismo enunciado, incluso puede responder a preguntas de compañeros en caso de que conozca la respuesta)

**Fecha de Entrega Vespertino:** 13/01/2021

**Fecha de Entrega Diurno:** 11/01/2021

**Objetivo del laboratorio:** Aplicar y demostrar los conocimientos adquiridos en cátedra con respecto al paradigma orientado a objetos usando el lenguaje de programación Java.

**Resultado esperado:** Programa que simula las funcionalidades y comportamientos de un sistema de preguntas y respuestas como StackOverflow.

**Requerimientos No Funcionales obligatorios.** Algunos son ineludibles, esto quiere decir que al no cumplir con dicho requerimiento, su proyecto será evaluado con la nota mínima.

1. **(obligatorio) Autoevaluación:** Incluir archivo de texto plano con autoevaluación de cada uno de los requerimientos funcionales (obligatorios y extras) solicitados, señalando una evaluación para cada uno según la siguiente escala:
  - a. 0: No realizado.
  - b. 0.25: Implementación con problemas mayores (funciona 25% de las veces o no funciona)
  - c. 0.5: Implementación con funcionamiento irregular (funciona 50% de las veces)
  - d. 0.75: Implementación con problemas menores (funciona 75% de las veces)
  - e. 1: Implementación completa sin problemas (funciona 100% de las veces)
2. **(obligatorio) Lenguaje y herramientas de trabajo:** La implementación de este laboratorio debe ser en el lenguaje de programación Java, utilizando OpenJDK versión 11 ([link](#)) con alguno de los siguientes IDEs (a su elección):
  - a. Apache NetBeans 12 ([link](#))
  - b. Eclipse IDE 2020-09 ([link](#))
  - c. IntelliJ IDEA Community 2020.3 ([link](#))

Para el proceso de compilación, se puede elegir entre una de las siguientes herramientas:

1. Un *script de compilación manual* en Batch (Windows) o Bash (GNU/Linux, macOS o Windows Subsystem for Linux (WSL)) que utilice el comando `javac` para compilar sus clases. El entregable debe ser una carpeta con el código fuente y el script de compilación (no debe incluir archivos binarios ni archivos `.class`)
2. Integrar a su proyecto la herramienta *Gradle* ([link a guía para crear proyecto Java usando Gradle](#)). El entregable final debe ser un proyecto creado con el IDE de su elección, integrado con Gradle (y Gradle Wrapper) e incluyendo solo código fuente y archivos de configuración (no debe incluir archivos binarios ni archivos `.class`)

Independientemente de la opción escogida para compilar el proyecto, su Informe debe especificar las instrucciones de compilación, la herramienta de compilación usada (script manual o Gradle) y el ambiente de desarrollo donde se ejecutó el entregable (Sistema Operativo y versión exacta de JDK). En caso de no poder compilar y ejecutar su proyecto con las instrucciones que usted indique, el laboratorio no será revisado.

3. **(obligatorio) Interacciones con el programa:** Todas las interacciones con el programa deben ser mediante consola/terminal. Puede recurrir al uso de `System.in` y `System.out` (en Java). Para sus funcionalidades, solo se permite el uso de la biblioteca estándar de Java (sin tener dependencias externas, como las disponibles en formato JAR).

4. **(obligatorio) Uso del paradigma:** Su solución debe demostrar la aplicación del paradigma orientado a objetos. No basta con que su solución esté implementada en Java. Su diseño y correspondiente implementación debe seguir los lineamientos del paradigma Orientado a Objetos.
5. **(obligatorio) Prerrequisitos:** Para cada funcionalidad se establecen prerrequisitos. Estos deben ser cumplidos para que se proceda con la evaluación de la funcionalidad implementada. Ej: Para evaluar la funcionalidad *ask*, debe estar implementada la funcionalidad *register/login/logout*.
6. **(1 pto) Documentación:** Se debe documentar el código indicando una breve descripción de las clases creadas, sus atributos, métodos públicos y relaciones. Procure utilizar comentarios tipo Javadoc ([link](#)) para esto.
7. **(1 pto) Organización del código:** Se debe cuidar la organización del código (orden y claridad). Procure que su diseño de clases no viole los principios de bajo acoplamiento y alta cohesión.
8. **(1.5 pto) Diagrama de análisis:** Como parte de su Informe de Laboratorio, debe incluir un diagrama de clases UML a nivel de análisis que describa las entidades y relaciones del problema abordado. Este diagrama se debe crear antes del proceso de desarrollo.
9. **(1.5 pto) Diagrama de diseño:** Como parte de su Informe de Laboratorio, debe incluir un diagrama de clases UML tras la implementación de la solución, este diagrama debe ser coherente con la implementación en código de su solución incluyendo todas las clases de su código. Este diagrama se debe crear después del desarrollo de la solución.
10. **(1 pto) Uso de git:** Historial de trabajo en GitHub tomando en consideración la evolución en el desarrollo de su proyecto en distintas etapas. Se requieren **al menos 10 commits** distribuidos en un periodo de tiempo **mayor o igual a 2 semanas**. Los criterios que se consideran en la evaluación de este ítem son: fecha primer commit, fecha último commit, total commits y máximo de commits diarios. A modo de ejemplo (y solo como una referencia), si hace todos los commits el día antes de la entrega del proyecto, este ítem tendrá 0 pts. De manera similar, si hace dos commits dos semanas antes de la entrega final y el resto los concentra en los últimos dos días, tendrá una evaluación del 25% para este ítem (0.25 pts). Por el contrario, si demuestra constancia en los commits (con aportes claros entre uno y otro) a lo largo del periodo evaluado, este ítem será evaluado con el total del puntaje.



**Requerimientos Funcionales. Para que el requerimiento sea evaluado, DEBE cumplir con el prerequisite de evaluación y requisito de implementación. En caso contrario la funcionalidad no será evaluada.**

1. **(1 pto) Clases y estructuras que forman el programa:** Como parte del diseño orientado a objetos de su solución, considere como mínimo modelar las siguientes entidades (y sus respectivas relaciones) dentro de su programa:
  - a. Usuario: Debe tener una representación del usuario dentro del sistema de preguntas, como mínimo debe incluir nombre de usuario, contraseña y reputación.
  - b. Etiqueta: Debe representar las etiquetas (tags) dentro del sistema de preguntas, incluyendo un nombre y una descripción.
  - c. Pregunta: Representa las preguntas dentro de la aplicación. Algunas características a considerar son:
    - i. ID (único y autoincremental)
    - ii. Respuestas
    - iii. Etiquetas
    - iv. Título
    - v. Contenido
    - vi. Fecha de publicación
    - vii. Autor (usuario)
    - viii. Estado (con respuesta aceptada o no)
    - ix. Recompensa: El usuario puede ofrecer una recompensa en puntos de reputación, los cuales se deducen de su propia reputación.
  - d. Respuesta: Representa las respuestas dentro de la aplicación. Considere incluir un ID (único y autoincremental), Autor, Fecha de publicación y Contenido.
  - e. Stack: Agrupa las estructuras de datos necesarias para un sistema de preguntas y respuestas.

<b>Prerrequisitos para evaluación</b>	Ninguno
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"><li>- Usar estructuras basadas en clases</li><li>- Especificar representación de manera clara para cada clase implementado (en el informe y en el código a través de comentarios). Luego implementar constructores, getters, setters y otros métodos según lo requerido en los requisitos a continuación.</li></ul>

2. **(1 pto) Menú interactivo por terminal:** Su programa debe incluir un menú por terminal/consola que permita la interacción del usuario con la solución, implementando las entidades y funcionalidades que permitan utilizar su simulación del sistema de preguntas y respuestas. El sistema debe estar poblado con algunos elementos al iniciar y además debe proveer de retroalimentación al usuario de las acciones realizadas por él (si la acción pudo concretarse con éxito o no). Las instrucciones de funcionamiento de su solución deben estar documentadas en su Informe de Laboratorio.

Un ejemplo del menú se representa a continuación:

```
### SISTEMA DE PREGUNTAS Y RESPUESTAS ###
```

```
## Registrado como: User123 ##
```

```
Escoja su opción:
```

1. Agregar nueva pregunta
2. Responder pregunta
3. Dar recompensa
4. Aceptar respuesta
5. Cerrar sesión
6. Salir del programa

```
INTRODUZCA SU OPCIÓN: _
```

<b>Prerrequisitos para evaluación</b>	Clases y estructuras que forman el programa
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"><li>- Implementar sistema de menú basado en terminal/consola</li><li>- La aplicación debe permitir cargar de forma inicial al menos 1 Stack, 4 usuarios registrados, 5 preguntas, 10 respuestas y 3 etiquetas.</li></ul>

Las siguientes funcionalidades mínimas deben estar implementadas en su solución. Estas funcionalidades pueden ser ejecutadas mediante uno o más métodos dentro de su programa.

3. **(0.5 pto) register/login/logout:** Esta funcionalidad debe permitir el registro de nuevos usuarios en la plataforma, además de su ingreso y salida a la plataforma con sus credenciales.

<b>Prerrequisitos para evaluación</b>	<ul style="list-style-type: none"><li>- Clases y estructuras que forman el programa</li><li>- Menú interactivo por terminal</li></ul>
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"><li>- Se debe almacenar el nuevo usuario en el Stack</li><li>- El programa debe validar las credenciales del usuario en el Stack antes de permitir el ingreso.</li><li>- En caso de <i>logout</i> debe existir una sesión iniciada previamente en el stack.</li></ul>
<b>Parámetros de entrada</b>	Nombre de usuario y contraseña

4. **(0.5 pto) ask:** Funcionalidad que permite a un usuario (con sesión iniciada en el Stack) realizar una nueva pregunta.

<b>Prerrequisitos para evaluación</b>	<ul style="list-style-type: none"><li>- register/login/logout</li></ul>
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"><li>- Se debe almacenar la nueva pregunta en el Stack, rellorando los atributos asociados a esta al momento de crear la pregunta (cómo el ID, Fecha de creación, Autor y Estado).</li><li>- Solo los usuarios con sesión iniciada pueden crear preguntas en el Stack.</li></ul>
<b>Parámetros de entrada</b>	Título de la pregunta, Contenido y Lista de etiquetas

5. **(0.5 pto) answer:** Funcionalidad que permite añadir una respuesta a una pregunta dentro del Stack, dejándola almacenada en la estructura correspondiente.

<b>Prerrequisitos para evaluación</b>	- ask
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"><li>- Esta funcionalidad debe mostrar una lista con todas las preguntas del Stack, permitiendo elegir una de las preguntas para poder escribir una respuesta a ésta.</li><li>- Solo los usuarios con sesión iniciada pueden responder preguntas del Stack.</li></ul>
<b>Parámetros de entrada</b>	Pregunta seleccionada y contenido de la respuesta

6. **(0.5 pto) reward:** Funcionalidad que permite a un usuario (con sesión iniciada) en la plataforma ofrecer una recompensa para una determinada pregunta. La recompensa puede ir dirigida a preguntas propias o de terceros. No se puede ofrecer recompensas mayores a la reputación con la que cuenta el usuario. Una vez ofrecida la recompensa, el puntaje ofrecido queda retenido temporalmente y descontado de la reputación actual. De esta forma se resguarda que el usuario no pueda ofrecer más recompensas de las que podrá eventualmente pagar. Una vez que una respuesta es aceptada, la recompensa se descuenta definitivamente de la reputación del autor y se traspasa al usuario que entregó la respuesta.

<b>Prerrequisitos para evaluación</b>	- ask
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"><li>- Para esta funcionalidad, en el menú de la aplicación debe mostrar una lista con todas las preguntas del Stack, permitiendo elegir una de las preguntas para poder ofrecer la recompensa.</li><li>- El puntaje retenido debe quedar almacenado en el Stack hasta que la pregunta tenga respuesta aceptada.</li></ul>

	<ul style="list-style-type: none"> <li>- Una pregunta puede recibir más de una recompensa por usuarios distintos, una vez aceptada la respuesta, debe distribuirse la reputación entre los usuarios involucrados.</li> <li>- Solo los usuarios con sesión iniciada pueden ofrecer recompensas.</li> </ul>
<b>Parámetros de entrada</b>	Pregunta seleccionada y recompensa (en puntos)

7. **(0.5 pto) accept:** Funcionalidad que permite a un usuario (con sesión iniciada en la plataforma) aceptar una respuesta a una de sus preguntas.

<b>Prerrequisitos para evaluación</b>	- ask
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"> <li>- Para esta funcionalidad, en el menú de la aplicación se debe mostrar una lista solo con las preguntas (y sus respuestas) del propio usuario en Stack, permitir elegir una de las preguntas para aceptar una de las respuestas indicadas en ella.</li> <li>- Si hay recompensas activas en la pregunta, estas deben resolverse otorgando los puntos de reputación al usuario.</li> <li>- Solo los usuarios con sesión iniciada pueden aceptar respuestas.</li> </ul>
<b>Parámetros de entrada</b>	Pregunta seleccionada y respuesta seleccionada

**De las siguientes funcionalidades implementar sólo 1 conforme al cumplimiento de los prerrequisitos. La nota máxima en total será un 7.0 aun cuando se desborde puntaje.**

1. **(2 pto) vote:** Función que permite a un usuario votar por una pregunta o respuesta. El voto puede ser positivo o negativo, ajustando los puntajes a otorgar según las instrucciones descritas al principio de este enunciado.

<b>Prerrequisitos para evaluación</b>	<ul style="list-style-type: none"> <li>- ask</li> <li>- answer</li> </ul>
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"> <li>- Las entidades de Pregunta y Respuesta deben poder registrar la cantidad de votos.</li> <li>- Se debe otorgar el puntaje correspondiente al Autor.</li> <li>- Para esta funcionalidad, en el menú de la aplicación se debe mostrar una lista con todas las preguntas y respuestas del Stack (de forma ordenada) que no sean del mismo usuario para permitir elegir una pregunta o una respuesta para otorgar el voto positivo o negativo (la elección de la pregunta/respuesta a votar puede ser por ID).</li> <li>- Solo los usuarios con sesión iniciada pueden votar.</li> </ul>
<b>Parámetros de entrada</b>	Pregunta o respuesta seleccionada, Tipo de voto (positivo o negativo)

2. **(2 pto) report:** Funcionalidad que permite a un usuario reportar preguntas o respuestas que contengan contenido ofensivo. Este contenido puede encontrarse en preguntas o respuestas.

<b>Prerrequisitos para evaluación</b>	<ul style="list-style-type: none"> <li>- ask</li> <li>- answer</li> </ul>
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"> <li>- Las entidades de Pregunta y Respuesta deben poder registrar la cantidad de reportes.</li> <li>- Para esta funcionalidad, en el menú de la aplicación se debe mostrar una lista con todas las preguntas y respuestas del Stack (de forma ordenada) para permitir elegir una pregunta o una respuesta para ser reportada.</li> <li>- En caso de que se hayan alcanzado 6 o más votos de ofensa, en cuyo caso se aplica descuento según indicaciones provistas al comienzo del enunciado.</li> </ul>

	- Solo los usuarios con sesión iniciada pueden reportar.
<b>Parámetros de entrada</b>	Pregunta o respuesta seleccionada, Tipo de voto (positivo o negativo)

3. **(2 pto) ranking:** Funcionalidad que retorna un ranking de las preguntas o respuestas de una pregunta según votación, pudiendo ser el orden ascendente (menos votado al más votado) o descendente (más votado al menos votado). La función se puede usar directamente sin pasar por login.

<b>Prerrequisitos para evaluación</b>	<ul style="list-style-type: none"> <li>- ask</li> <li>- answer</li> <li>- reward o vote</li> </ul>
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"> <li>- La funcionalidad debe permitir ordenar por ranking de puntaje (ascendente o descendente) todas las preguntas del stack, o bien las respuestas a una pregunta.</li> <li>- Puede utilizar métodos de ordenamiento disponibles en la biblioteca estándar de Java, o bien implementar uno propio.</li> <li>- No es necesario tener sesión iniciada para usar la funcionalidad ranking.</li> </ul>
<b>Parámetros de entrada</b>	Stack o pregunta seleccionada, Criterio de ordenamiento (ascendente o descendente).
<b>Parámetros de salida</b>	Lista con los elementos ordenados según criterio

4. **(2 pto) search:** Funcionalidad que permite buscar preguntas o respuestas en base a una coincidencia parcial de texto. Se debe identificar el tipo de búsqueda a realizar (si se hará por cualquier texto dentro de las preguntas, respuestas o etiquetas). La función se puede usar directamente sin pasar por login.

<b>Prerrequisitos para evaluación</b>	<ul style="list-style-type: none"> <li>- ask</li> <li>- answer</li> </ul>
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"> <li>- La funcionalidad debe permitir ordenar buscar por preguntas, respuestas o etiquetas dentro del Stack.</li> <li>- La funcionalidad debe permitir coincidencias parciales de texto (substring) y debe ser insensible a las mayúsculas.</li> <li>- No es necesario tener sesión iniciada para usar la funcionalidad search.</li> </ul>
<b>Parámetros de entrada</b>	Stack seleccionado, criterio de búsqueda (preguntas, respuestas o etiquetas), <b>string a buscar</b> .
<b>Parámetros de salida</b>	Preguntas o respuestas que coincidan con los criterios de búsqueda.



# Laboratorio 4 Vespertino

## (Multiparadigma - Lenguaje C# o Java)

Versión 1.0 - Enunciado solo para vespertino - Última Edición: 22/01/2021

(Cambios menores pueden incorporarse en futuras versiones a fin de aclarar o corregir errores)

(Sus dudas las puede expresar en este mismo enunciado a través del uso de comentarios, incluso puede responder a preguntas de compañeros en caso de que conozca la respuesta)

**Fecha de Entrega Vespertino:** Lunes 25/01/2020 hasta las 19.00 hrs a través de Campus Virtual.

**Nota:** La revisión se realizará de forma presencial (remotamente) durante la clase posterior al día de la entrega. Usted deberá mostrar desde su computador la ejecución de su entrega.

**Objetivo del presente laboratorio:** Aplicar y demostrar los conocimientos adquiridos en cátedra con respecto al paradigma Orientado a Objetos (OOP) y dirigido por eventos (Event-Driven Programming) usando el lenguaje de programación C# o Java.

**Resultado esperado:** Programa con interfaz gráfica (GUI) implementado en C# o Java que simula las funcionalidades y comportamientos de un sistema de preguntas y respuestas como StackOverflow.

## Requerimientos No Funcionales obligatorios (RNF)

**Algunos son ineludibles, esto quiere decir que al no cumplir con dicho requerimiento, su proyecto será evaluado con la nota mínima.**

1. **(obligatorio) Autoevaluación:** Incluir archivo de texto plano con autoevaluación de cada uno de los requerimientos funcionales (obligatorios y extras) solicitados, señalando una evaluación para cada uno según la siguiente escala:
  - a. 0: No realizado.
  - b. 0.25: Implementación con problemas mayores (funciona 25% de las veces o no funciona)
  - c. 0.5: Implementación con funcionamiento irregular (funciona 50% de las veces)
  - d. 0.75: Implementación con problemas menores (funciona 75% de las veces)
  - e. 1: Implementación completa sin problemas (funciona 100% de las veces)
2. **(obligatorio) Lenguaje y herramientas de trabajo:** En este laboratorio, el lenguaje de programación a ocupar queda a elección del estudiante. No obstante, los lenguajes a seleccionar pueden ser **C#** o **Java**. Si implementa su solución en otro lenguaje, será calificado con la nota mínima.

### a. En el caso de utilizar C#:

- La implementación de este laboratorio debe ser en el lenguaje de programación **C# en su versión 8.0** ([link](#)) con alguno de los siguientes IDEs (a su elección):
  - a. Visual Studio Community (gratis) ([link](#))
  - b. Visual Studio Ultimate (en caso de disponer de esta) ([link](#))
- La implementación debe utilizar **WPF** o **Windows Form** como biblioteca de entorno gráfico. Estas bibliotecas son parte del *core* de C#, por lo que no se debe agregar dependencias externas al proyecto.
- WPF o Windows Form **solo puede ser implementado y ejecutado en sistemas operativos Windows. No es compatible con Linux ni con MacOS. En caso de querer utilizar estas plataformas, optar por una implementación basada en Java.**
- Su entregable final debe ser el proyecto generado por el IDE, sin incluir archivos binarios.

### b. En el caso de utilizar Java:

- La implementación de este laboratorio debe ser en el lenguaje de programación Java, utilizando **OpenJDK versión 11** ([link](#)) con alguno de los siguientes IDEs (a su elección):
  - a. Apache NetBeans 12 ([link](#))
  - b. Eclipse IDE 2020-09 ([link](#))
  - c. IntelliJ IDEA Community 2020.3 ([link](#))

- La implementación debe utilizar **AWT** o **Swing** como biblioteca de entorno gráfico. Estas bibliotecas son parte del *core* de Java, por lo que no se debe agregar dependencias externas al proyecto.

**Al igual que en su entrega del laboratorio 3, para el proceso de compilación, se puede elegir entre una de las siguientes herramientas:**

1. Un *script de compilación manual* en Batch (Windows) o Bash (GNU/Linux, macOS o Windows Subsystem for Linux (WSL)) que utilice el comando `javac` para compilar sus clases. El entregable debe ser una carpeta con el código fuente y el script de compilación (no debe incluir archivos binarios ni archivos `.class`)
2. Integrar a su proyecto la herramienta *Gradle* ([link a guía para crear proyecto Java usando Gradle](#)). El entregable final debe ser un proyecto creado con el IDE de su elección, integrado con Gradle (y Gradle Wrapper) e incluyendo solo código fuente y archivos de configuración (no debe incluir archivos binarios ni archivos `.class`)

Independientemente de la opción escogida para compilar el proyecto, su Informe debe especificar las instrucciones de compilación, la herramienta de compilación usada (script manual o Gradle) y el ambiente de desarrollo donde se ejecutó el entregable (Sistema Operativo y versión exacta de JDK). En caso de no poder compilar y ejecutar su proyecto con las instrucciones que usted indique, el laboratorio no será revisado.

3. **(obligatorio) Interacciones con el programa:** Todas las interacciones con el programa deben ser mediante el **uso de la interfaz gráfica**. Dependiendo de su lenguaje escogido, su implementación debe utilizar las librerías gráficas previamente nombradas. Para sus funcionalidades, solo se permite el uso de la biblioteca estándar de Java (sin tener dependencias externas, como las disponibles en formato JAR).
4. **(obligatorio) Uso del paradigma:** Su solución debe demostrar la aplicación del **Paradigma Orientado a Objetos y Dirigido por Eventos**.
  - a. No basta con que su solución esté implementada en Java o en C#. Su diseño y correspondiente implementación debe seguir los lineamientos de dichos paradigmas.
5. **(obligatorio) Separación responsabilidad Modelo-Vista:** Si bien no es necesario diseñar e implementar su solución en base al patrón Modelo-Vista-Controlador (MVC) ([link](#)), es obligatorio separar las capas de modelo y vista. **Los objetos del modelo no deben tener ningún grado de acoplamiento con respecto a los objetos de la vista.** Solo se permitirá que la vista tenga dependencia con objetos del modelo.

- Los objetos del modelo no pueden realizar modificaciones en la vista o realizar llamados a métodos de las mismas. Por otro lado, los objetos de la vista, si pueden operar con objetos del modelo. Finalmente, los objetos que sean implementados con respecto a la interfaz gráfica de usuario (GUI) solo deben manejar las interacciones del usuario.
6. **(obligatorio) Prerrequisitos:** Al igual en laboratorio pasado, para cada funcionalidad se establecen prerrequisitos. Estos deben ser cumplidos para que se proceda con la evaluación de la funcionalidad implementada. Ej: Para evaluar la funcionalidad *ask*, debe estar implementada la funcionalidad *register/login/logout*.
  7. **(1 pto) Documentación:** Se debe documentar el código indicando una breve descripción de las clases creadas, sus atributos, métodos y relaciones. Procure utilizar comentarios tipo [Javadoc \(link\)](#) en el caso de **Java** o [XML \(link\)](#) si su lenguaje es **C#**. A la hora de entrega, solo dejar la documentación comentada en el código, no entregar reportes en HTML, PDF, Word u otro.
  8. **(1 pto) Organización del código:** Se debe cuidar la organización del código (orden y claridad). Procure que su diseño de clases no viole los principios de bajo acoplamiento y alta cohesión.
  9. **(1.5 pto) Diagrama de análisis:** Como parte de su Informe de Laboratorio, debe incluir un diagrama de clases UML a nivel de análisis que describa las entidades y relaciones del problema abordado. Dicho diagrama solo debe realizarse con respecto a las clases del modelo a implementar, no sobre las vistas a crear. Este diagrama debe ser creado antes del proceso de desarrollo.
  10. **(1.5 pto) Diagrama de diseño:** Como parte de su Informe de Laboratorio, debe incluir un diagrama de clases UML tras la implementación de la solución, este diagrama debe ser coherente con la implementación en código de su solución incluyendo todas las clases de su código. Dicho diagrama debe realizarse con respecto a las clases del modelo y de las vistas implementadas. Este diagrama se debe crear después del desarrollo de la solución.
  11. **(1 pto) Uso de git:** Historial de trabajo en GitHub tomando en consideración la evolución en el desarrollo de su proyecto en distintas etapas.
    - Se requieren **al menos 10 commits** distribuidos en un periodo de tiempo **mayor o igual a 3 días**.
    - Los criterios que se consideran en la evaluación de este ítem son: fecha primer commit, fecha último commit, total commits y máximo de commits diarios.

- A modo de ejemplo (y solo como una referencia), si hace todos los commits el día antes de la entrega del proyecto, este ítem tendrá 0 pts. De manera similar, si hace dos commits dos semanas antes de la entrega final y el resto los concentra en los últimos dos días, tendrá una evaluación del 25% para este ítem (0.25 pts). Por el contrario, si demuestra constancia en los commits (con aportes claros entre uno y otro) a lo largo del periodo evaluado, este ítem será evaluado con el total del puntaje.

## Requerimientos Funcionales (RF)

**Para que el requerimiento sea evaluado, DEBE cumplir con el prerequisite de evaluación y requisito de implementación. En caso contrario la funcionalidad no será evaluada.**

1. **(1 pto) Clases y estructuras que forman el programa:** Como parte del diseño orientado a objetos de su solución, considere como mínimo modelar las siguientes entidades (y sus respectivas relaciones) dentro de su programa:

**Puede tomar como referencia su implementación del laboratorio 3.**

- a. **Usuario:** Debe tener una representación del usuario dentro del sistema de preguntas, como mínimo debe incluir nombre de usuario, contraseña y reputación.
- b. **Etiqueta:** Debe representar las etiquetas (*tags*) dentro del sistema de preguntas, incluyendo un nombre y una descripción.
- c. **Pregunta:** Representa las preguntas dentro de la aplicación. Algunas características a considerar son:
  - i. ID (único y autoincremental)
  - ii. Respuestas
  - iii. Etiquetas
  - iv. Título
  - v. Contenido
  - vi. Fecha de publicación
  - vii. Autor (usuario)
  - viii. Estado (con respuesta aceptada o no)
- d. **Respuesta:** Representa las respuestas dentro de la aplicación. Considere incluir un ID (único y autoincremental), Autor, Fecha de publicación y Contenido.
- e. **Stack:** Agrupa las estructuras de datos necesarias para un sistema de preguntas y respuestas.

<b>Prerrequisitos para evaluación</b>	Ninguno
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"> <li>- Usar estructuras basadas en clases, conforme al paradigma orientado a objetos.</li> <li>- Especificar representación de manera clara para cada clase implementado (en el informe y en el código a través de comentarios). Luego implementar constructores, getters, setters y otros métodos según lo requerido en los requisitos a continuación.</li> </ul>

### Funcionalidades mínimas obligatorias a implementar

**Al igual que el laboratorio 3, las siguientes funcionalidades mínimas deben estar implementadas en su solución. Estas funcionalidades pueden ser ejecutadas mediante uno o más métodos dentro de su programa. Puede utilizar como referencia su implementación del laboratorio pasado.**

1. **(0.7 pto) register/login/logout:** Esta funcionalidad debe permitir el registro de nuevos usuarios en la plataforma, además de su ingreso y salida a la plataforma con sus credenciales.

<b>Prerrequisitos para evaluación</b>	- Clases y estructuras que forman el programa
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"> <li>- Estas funcionalidades deben estar implementadas a través de la interfaz gráfica utilizando el paradigma dirigido por eventos</li> <li>- Se debe almacenar el nuevo usuario en el Stack</li> <li>- El programa debe validar las credenciales del usuario en el Stack antes de permitir el ingreso. En caso de que la contraseña sea incorrecta o el usuario no exista en el <i>stack</i> debe reflejar dicho comportamiento en la interfaz gráfica.</li> <li>- En caso de <del>register</del> <b>logout</b> debe existir una sesión iniciada previamente en el stack.</li> </ul>

<b>Parámetros de entrada</b>	Nombre de usuario y contraseña
------------------------------	--------------------------------

2. **(0.7 pto) ask:** Funcionalidad que permite a un usuario (con sesión iniciada en el Stack) realizar una nueva pregunta.

<b>Prerrequisitos para evaluación</b>	- register/login/logout
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"> <li>- Implementar la vista relacionada con la creación y envío de preguntas. Es decir, debe implementar una vista que le da la posibilidad al usuario de ingresar un texto correspondiente al cuerpo de la pregunta y posteriormente el envío al stack.</li> <li>- Se debe almacenar la nueva pregunta en el Stack, rellenando los atributos asociados a esta al momento de crear la pregunta (cómo el ID, Fecha de creación, Autor y Estado).</li> <li>- Solo los usuarios con sesión iniciada pueden crear preguntas en el Stack.</li> </ul>
<b>Parámetros de entrada</b>	Título de la pregunta, Contenido y Lista de etiquetas

3. **(0.7 pto) answer:** Funcionalidad que permite añadir una respuesta a una pregunta dentro del Stack, dejándola almacenada en la estructura correspondiente.

<b>Prerrequisitos para evaluación</b>	- ask
<b>Requisitos de implementación</b>	- Esta funcionalidad debe mostrar una lista con todas las preguntas del Stack, permitiendo elegir al usuario seleccionar una de ellas desde la interfaz gráfica para poder escribir una respuesta a ésta.

	- Solo los usuarios con sesión iniciada pueden responder preguntas del Stack.
<b>Parámetros de entrada</b>	Pregunta seleccionada y contenido de la respuesta

4. **(0.7 pto) accept:** Funcionalidad que permite a un usuario (con sesión iniciada en la plataforma) aceptar una respuesta a una de sus preguntas.

<b>Prerrequisitos para evaluación</b>	- ask
<b>Requisitos de implementación</b>	<p>- Para esta funcionalidad, implementar una vista que muestre solo la lista con las preguntas del propio usuario en Stack, y que permita seleccionar una de las preguntas para aceptar una de las respuestas indicadas en ella.</p> <p>- Solo los usuarios con sesión iniciada pueden aceptar respuestas.</p>
<b>Parámetros de entrada</b>	Pregunta seleccionada y respuesta seleccionada

5. **(0.7 pto) sort by publication date:** Funcionalidad que permite ordenar cualquier lista de preguntas de acuerdo a la fecha de publicación. La funcionalidad debe poder permitir ordenar de forma ascendente y descendente de acuerdo a la fecha de publicación. Por defecto, las listas se ordenan de las más recientes a las más antiguas. La función se puede usar directamente sin pasar por login.

<b>Prerrequisitos para evaluación</b>	- answer
<b>Requisitos de implementación</b>	- Debe permitir mediante un evento, ordenar todas las listas de preguntas, de las más recientes a las más antiguas y viceversa. La implementación la puede realizar mediante un



	<p>botón u cualquier otro componente que usted estime necesario.</p> <ul style="list-style-type: none"> <li>- Puede utilizar métodos de ordenamiento disponibles en la biblioteca estándar de C# u Java, o bien implementar uno propio.</li> <li>- No es necesario tener sesión iniciada para usar la funcionalidad sort by publication date.</li> </ul>
<b>Parámetros de entrada</b>	- Lista de preguntas, Criterio de ordenamiento (ascendente o descendente).
<b>Parámetros de salida</b>	Lista con las preguntas ordenadas según criterio

**De las siguientes funcionalidades implementar sólo 1 conforme al cumplimiento de los prerequisites. La nota máxima en total será un 7.0 aun cuando se desborde puntaje.**

1. **(2 pto) vote:** Función que permite a un usuario votar por una pregunta o respuesta. El voto puede ser positivo o negativo.

<b>Prerrequisitos para evaluación</b>	<ul style="list-style-type: none"> <li>- ask</li> <li>- answer</li> <li>- sort by publication date</li> </ul>
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"> <li>- Las entidades de Pregunta y Respuesta deben poder registrar la cantidad de votos.</li> <li>- Se debe otorgar el puntaje correspondiente a la Pregunta y Respuesta.</li> <li>- Para esta funcionalidad, la interfaz gráfica debe mostrar una lista con todas las preguntas y respuestas del Stack (de forma ordenada por la fecha de publicación) que no sean del mismo usuario para permitir elegir una pregunta o una respuesta para otorgar el voto positivo o negativo</li> </ul>

	<ul style="list-style-type: none"> <li>- La elección de la pregunta/respuesta a votar puede ser por ID mediante el evento de presionar un botón, haciendo click sobre un ítem de una lista u otro evento, queda a su decisión.</li> <li>- Solo los usuarios con sesión iniciada pueden votar.</li> </ul>
<b>Parámetros de entrada</b>	Pregunta o respuesta seleccionada, Tipo de voto (positivo o negativo)

2. **(2 pto) report:** Funcionalidad que permite a un usuario reportar preguntas o respuestas que contengan contenido ofensivo. Este contenido puede encontrarse en preguntas o respuestas.

<b>Prerrequisitos para evaluación</b>	<ul style="list-style-type: none"> <li>- ask</li> <li>- answer</li> <li>- sort by publication date</li> </ul>
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"> <li>- Las entidades de Pregunta y Respuesta deben poder registrar la cantidad de reportes.</li> <li>- Solo los usuarios con sesión iniciada pueden reportar.</li> </ul>
<b>Parámetros de entrada</b>	Pregunta o respuesta seleccionada, Tipo de voto (positivo o negativo)

3. **(2 pto) ranking:** Funcionalidad que retorna un ranking de las preguntas o respuestas de una pregunta según votación, pudiendo ser el orden ascendente (menos votado al más votado) o descendente (más votado al menos votado). La función se puede usar directamente sin pasar por login.

<b>Prerrequisitos para evaluación</b>	<ul style="list-style-type: none"> <li>- ask</li> <li>- answer</li> <li>- vote</li> </ul>
---------------------------------------	---

<b>Requisitos de implementación</b>	<ul style="list-style-type: none"> <li>- La funcionalidad debe permitir ordenar por ranking de puntaje (ascendente o descendente) todas las preguntas del stack, o bien todas las respuestas a una pregunta.</li> <li>- Puede utilizar métodos de ordenamiento disponibles en la biblioteca estándar de C# u Java, o bien implementar uno propio.</li> <li>- No es necesario tener sesión iniciada para usar la funcionalidad ranking.</li> </ul>
<b>Parámetros de entrada</b>	Stack o pregunta seleccionada, Criterio de ordenamiento (ascendente o descendente).
<b>Parámetros de salida</b>	Lista con los elementos ordenados según criterio

4. **(2 pto) search:** Funcionalidad que permite buscar preguntas o respuestas en base a una coincidencia parcial de texto. Se debe identificar el tipo de búsqueda a realizar (si se hará por cualquier texto dentro de las preguntas, respuestas o etiquetas). La función se puede usar directamente sin pasar por login.

<b>Prerrequisitos para evaluación</b>	<ul style="list-style-type: none"> <li>- ask</li> <li>- answer</li> </ul>
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"> <li>- La funcionalidad debe permitir ordenar buscar por preguntas, respuestas o etiquetas dentro del Stack.</li> <li>- La funcionalidad debe permitir coincidencias parciales de texto (substring) y debe ser insensible a las mayúsculas.</li> <li>- No es necesario tener sesión iniciada para usar la funcionalidad search.</li> </ul>
<b>Parámetros de entrada</b>	Stack seleccionado, criterio de búsqueda (preguntas, respuestas o etiquetas), string a buscar.

<b>Parámetros de salida</b>	Preguntas o respuestas que coincidan con los criterios de búsqueda.
-----------------------------	---

## Referencia para el diseño e implementación de la interfaz gráfica

**Importante:** En este laboratorio, no se evalúa su diseño gráfico o qué tan atractivo es visualmente la interfaz implementada. Se evalúa el cumplimiento de las funcionalidades previamente descritas. Implemente el diseño que más le acomode. La presente sección es solo para orientarlo. La implementación de la interfaz gráfica es libre.

El diseño de las vistas de la interfaz gráfica queda a su elección. En caso de que le oriente, puede utilizar las siguientes maquetas como referencia para la base de su laboratorio. **No es necesario implementar las vistas tal cual cómo se ve en las siguientes maquetas**, es sólo una orientación.

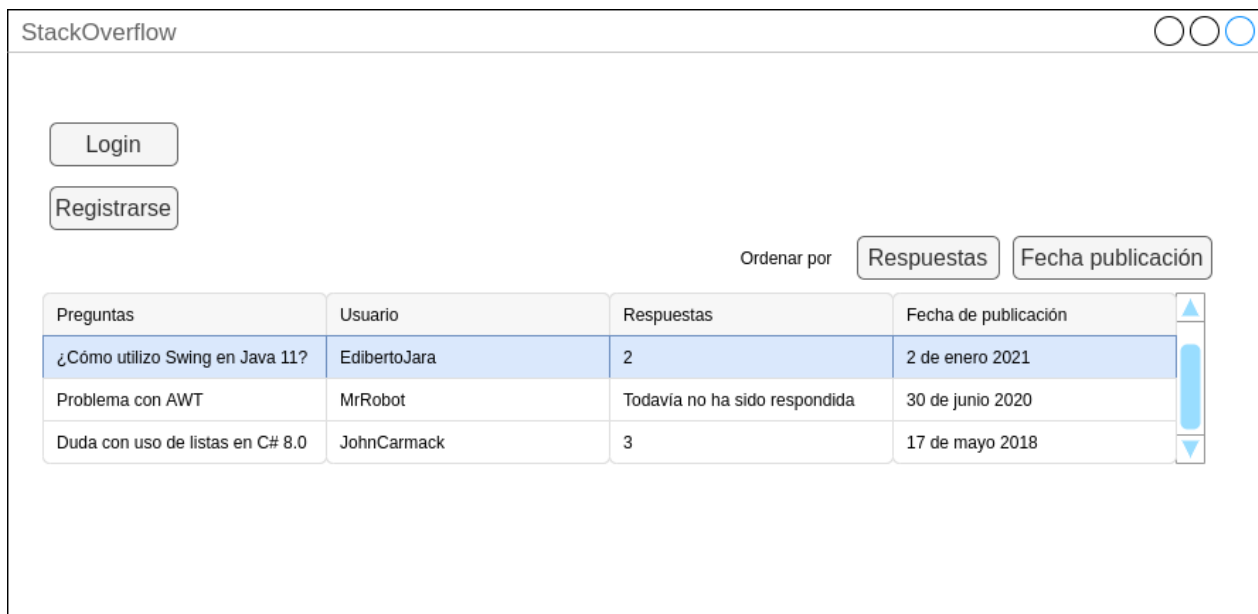


Figura 1. Maqueta de referencia para la vista principal de despliegue de preguntas del *stack* sin tener sesión iniciada.

The image shows two side-by-side wireframe windows for a StackOverflow application. Both windows have a title bar with the text 'StackOverflow' and three window control buttons (minimize, maximize, close). The left window is for login, featuring a 'Username' label, a text input field with placeholder text 'Ingresar username', a 'Password' label, a text input field with placeholder text 'Ingresar password', a blue 'Login' button, and a white 'Volver' button. The right window is for registration, featuring a title 'Registro de usuario', a 'Username' label, a text input field with placeholder text 'Ingresar username', a 'Password' label, a text input field with placeholder text 'Ingresar password', a blue 'Registrarse' button, and a white 'Volver' button.

Figura 2. Maqueta de referencia para el login y registro de usuario

The image shows a single wireframe window for a StackOverflow application. It has a title bar with the text 'StackOverflow' and three window control buttons (minimize, maximize, close). The main content area displays the message 'Usuario no existe!' in the center. Below the message is a white button with the text 'Volver'.

Figura 3. Maqueta de referencia para cualquier diálogo de información al usuario

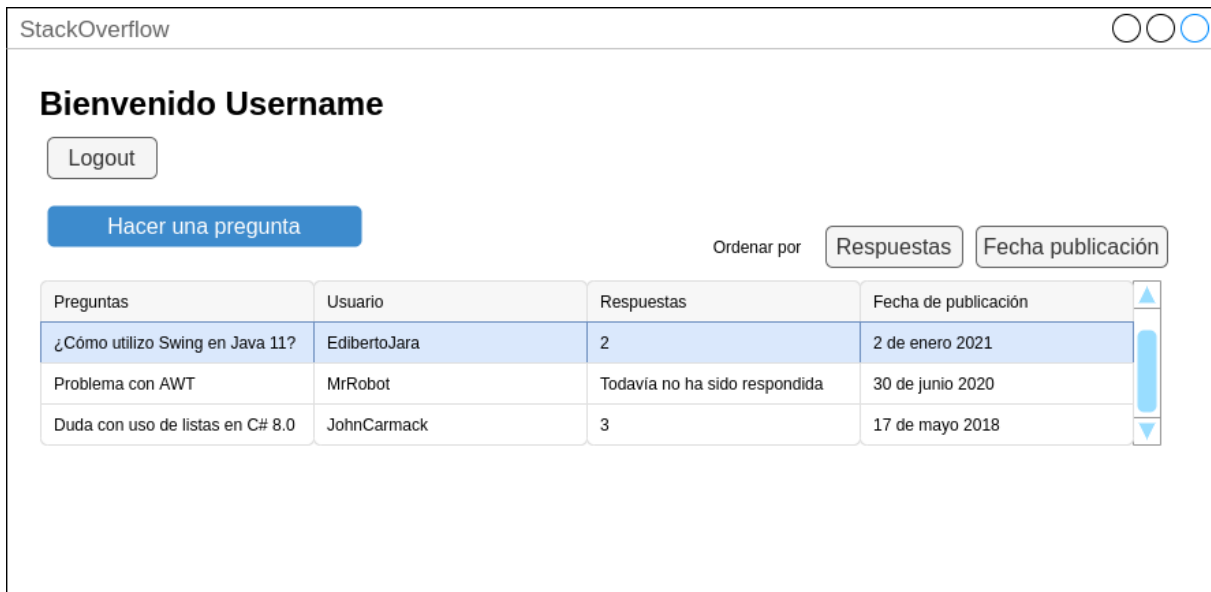


Figura 4. Maqueta de referencia para la vista principal de despliegue de preguntas del *stack* con la sesión iniciada.



Figura 5. Maqueta de referencia para la vista de despliegue de respuestas del *stack* con la sesión iniciada.

# Laboratorio 4 Diurno

## (Multiparadigma - Lenguaje C# o Java)

Versión 1.0 - Enunciado solo para diurno - Última Edición: 22/01/2021

(Cambios menores pueden incorporarse en futuras versiones a fin de aclarar o corregir errores)

(Sus dudas las puede expresar en este mismo enunciado a través del uso de comentarios, incluso puede responder a preguntas de compañeros en caso de que conozca la respuesta)

**Fecha de Entrega Diurno:** Jueves 04/03/2020 hasta las 23.59 hrs a través de Campus Virtual.

**Nota:** La revisión se realizará de forma presencial (remotamente) durante la clase posterior al día de la entrega. Usted deberá mostrar desde su computador la ejecución de su entrega.

**Objetivo del presente laboratorio:** Aplicar y demostrar los conocimientos adquiridos en cátedra con respecto al paradigma Orientado a Objetos (OOP) y dirigido por eventos (Event-Driven Programming) usando el lenguaje de programación C# o Java.

**Resultado esperado:** Programa con interfaz gráfica (GUI) implementado en C# o Java que simula las funcionalidades y comportamientos de un sistema de preguntas y respuestas como StackOverflow.

## Requerimientos No Funcionales obligatorios (RNF)

**Algunos son ineludibles, esto quiere decir que al no cumplir con dicho requerimiento, su proyecto será evaluado con la nota mínima.**

1. **(obligatorio) Autoevaluación:** Incluir archivo de texto plano con autoevaluación de cada uno de los requerimientos funcionales (obligatorios y extras) solicitados, señalando una evaluación para cada uno según la siguiente escala:
  - a. 0: No realizado.
  - b. 0.25: Implementación con problemas mayores (funciona 25% de las veces o no funciona)
  - c. 0.5: Implementación con funcionamiento irregular (funciona 50% de las veces)
  - d. 0.75: Implementación con problemas menores (funciona 75% de las veces)
  - e. 1: Implementación completa sin problemas (funciona 100% de las veces)
2. **(obligatorio) Lenguaje y herramientas de trabajo:** En este laboratorio, el lenguaje de programación a ocupar queda a elección del estudiante. No obstante, los lenguajes a seleccionar pueden ser **C#** o **Java**. Si implementa su solución en otro lenguaje, será calificado con la nota mínima.

### a. En el caso de utilizar C#:

- La implementación de este laboratorio debe ser en el lenguaje de programación **C# en su versión 8.0** ([link](#)) con alguno de los siguientes IDEs (a su elección):
  - a. Visual Studio Community (gratis) ([link](#))
  - b. Visual Studio Ultimate (en caso de disponer de esta) ([link](#))
- La implementación debe utilizar **WPF** o **Windows Form** como biblioteca de entorno gráfico. Estas bibliotecas son parte del *core* de C#, por lo que no se debe agregar dependencias externas al proyecto.
- WPF o Windows Form **solo puede ser implementado y ejecutado en sistemas operativos Windows. No es compatible con Linux ni con MacOS. En caso de querer utilizar estas plataformas, optar por una implementación basada en Java.**
- Su entregable final debe ser el proyecto generado por el IDE, sin incluir archivos binarios.

### b. En el caso de utilizar Java:

- La implementación de este laboratorio debe ser en el lenguaje de programación Java, utilizando **OpenJDK versión 11** ([link](#)) con alguno de los siguientes IDEs (a su elección):
  - a. Apache NetBeans 12 ([link](#))
  - b. Eclipse IDE 2020-09 ([link](#))
  - c. IntelliJ IDEA Community 2020.3 ([link](#))



- La implementación debe utilizar **AWT** o **Swing** como biblioteca de entorno gráfico. Estas bibliotecas son parte del *core* de Java, por lo que no se debe agregar dependencias externas al proyecto.

**Al igual que en su entrega del laboratorio 3, para el proceso de compilación, se puede elegir entre una de las siguientes herramientas:**

1. Un *script de compilación manual* en Batch (Windows) o Bash (GNU/Linux, macOS o Windows Subsystem for Linux (WSL)) que utilice el comando `javac` para compilar sus clases. El entregable debe ser una carpeta con el código fuente y el script de compilación (no debe incluir archivos binarios ni archivos `.class`)
2. Integrar a su proyecto la herramienta *Gradle* ([link a guía para crear proyecto Java usando Gradle](#)). El entregable final debe ser un proyecto creado con el IDE de su elección, integrado con Gradle (y Gradle Wrapper) e incluyendo solo código fuente y archivos de configuración (no debe incluir archivos binarios ni archivos `.class`)

Independientemente de la opción escogida para compilar el proyecto, su Informe debe especificar las instrucciones de compilación, la herramienta de compilación usada (script manual o Gradle) y el ambiente de desarrollo donde se ejecutó el entregable (Sistema Operativo y versión exacta de JDK). En caso de no poder compilar y ejecutar su proyecto con las instrucciones que usted indique, el laboratorio no será revisado.

3. **(obligatorio) Interacciones con el programa:** Todas las interacciones con el programa deben ser mediante el **uso de la interfaz gráfica**. Dependiendo de su lenguaje escogido, su implementación debe utilizar las librerías gráficas previamente nombradas. Para sus funcionalidades, solo se permite el uso de la biblioteca estándar de Java (sin tener dependencias externas, como las disponibles en formato JAR).
4. **(obligatorio) Uso del paradigma:** Su solución debe demostrar la aplicación del **Paradigma Orientado a Objetos y Dirigido por Eventos**.
  - a. No basta con que su solución esté implementada en Java o en C#. Su diseño y correspondiente implementación debe seguir los lineamientos de dichos paradigmas.
5. **(obligatorio) Separación responsabilidad Modelo-Vista:** Si bien no es necesario diseñar e implementar su solución en base al patrón Modelo-Vista-Controlador (MVC) ([link](#)), es obligatorio separar las capas de modelo y vista. **Los objetos del modelo no deben tener ningún grado de acoplamiento con respecto a los objetos de la vista.** Solo se permitirá que la vista tenga dependencia con objetos del modelo.

- Los objetos del modelo no pueden realizar modificaciones en la vista o realizar llamados a métodos de las mismas. Por otro lado, los objetos de la vista, si pueden operar con objetos del modelo. Finalmente, los objetos que sean implementados con respecto a la interfaz gráfica de usuario (GUI) solo deben manejar las interacciones del usuario.
6. **(obligatorio) Prerrequisitos:** Al igual en laboratorio pasado, para cada funcionalidad se establecen prerrequisitos. Estos deben ser cumplidos para que se proceda con la evaluación de la funcionalidad implementada. Ej: Para evaluar la funcionalidad *ask*, debe estar implementada la funcionalidad *register/login/logout*.
  7. **(1 pto) Documentación:** Se debe documentar el código indicando una breve descripción de las clases creadas, sus atributos, métodos y relaciones. Procure utilizar comentarios tipo [Javadoc \(link\)](#) en el caso de **Java** o [XML \(link\)](#) si su lenguaje es **C#**. A la hora de entrega, solo dejar la documentación comentada en el código, no entregar reportes en HTML, PDF, Word u otro.
  8. **(1 pto) Organización del código:** Se debe cuidar la organización del código (orden y claridad). Procure que su diseño de clases no viole los principios de bajo acoplamiento y alta cohesión.
  9. **(1.5 pto) Diagrama de análisis:** Como parte de su Informe de Laboratorio, debe incluir un diagrama de clases UML a nivel de análisis que describa las entidades y relaciones del problema abordado. Dicho diagrama solo debe realizarse con respecto a las clases del modelo a implementar, no sobre las vistas a crear. Este diagrama debe ser creado antes del proceso de desarrollo.
  10. **(1.5 pto) Diagrama de diseño:** Como parte de su Informe de Laboratorio, debe incluir un diagrama de clases UML tras la implementación de la solución, este diagrama debe ser coherente con la implementación en código de su solución incluyendo todas las clases de su código. Dicho diagrama debe realizarse con respecto a las clases del modelo y de las vistas implementadas. Este diagrama se debe crear después del desarrollo de la solución.
  11. **(1 pto) Uso de git:** Historial de trabajo en GitHub tomando en consideración la evolución en el desarrollo de su proyecto en distintas etapas.
    - Se requieren **al menos 10 commits** distribuidos en un periodo de tiempo **mayor o igual a 1 semana**.
    - Los criterios que se consideran en la evaluación de este ítem son: fecha primer commit, fecha último commit, total commits y máximo de commits diarios.

- A modo de ejemplo (y solo como una referencia), si hace todos los commits el día antes de la entrega del proyecto, este ítem tendrá 0 pts. De manera similar, si hace dos commits dos semanas antes de la entrega final y el resto los concentra en los últimos dos días, tendrá una evaluación del 25% para este ítem (0.25 pts). Por el contrario, si demuestra constancia en los commits (con aportes claros entre uno y otro) a lo largo del periodo evaluado, este ítem será evaluado con el total del puntaje.

## Requerimientos Funcionales (RF)

**Para que el requerimiento sea evaluado, DEBE cumplir con el prerequisite de evaluación y requisito de implementación. En caso contrario la funcionalidad no será evaluada.**

1. **(1 pto) Clases y estructuras que forman el programa:** Como parte del diseño orientado a objetos de su solución, considere como mínimo modelar las siguientes entidades (y sus respectivas relaciones) dentro de su programa:

**Puede tomar como referencia su implementación del laboratorio 3.**

- a. **Usuario:** Debe tener una representación del usuario dentro del sistema de preguntas, como mínimo debe incluir nombre de usuario, contraseña y reputación.
- b. **Etiqueta:** Debe representar las etiquetas (*tags*) dentro del sistema de preguntas, incluyendo un nombre y una descripción.
- c. **Pregunta:** Representa las preguntas dentro de la aplicación. Algunas características a considerar son:
  - i. ID (único y autoincremental)
  - ii. Respuestas
  - iii. Etiquetas
  - iv. Título
  - v. Contenido
  - vi. Fecha de publicación
  - vii. Autor (usuario)
  - viii. Estado (con respuesta aceptada o no)
  - ix. Recompensa: El usuario puede ofrecer una recompensa en puntos de reputación, los cuales se deducen de su propia reputación.
- d. **Respuesta:** Representa las respuestas dentro de la aplicación. Considere incluir un ID (único y autoincremental), Autor, Fecha de publicación y Contenido.
- e. **Stack:** Agrupa las estructuras de datos necesarias para un sistema de preguntas y respuestas.

<b>Prerrequisitos para evaluación</b>	Ninguno
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"> <li>- Usar estructuras basadas en clases, conforme al paradigma orientado a objetos.</li> <li>- Especificar representación de manera clara para cada clase implementado (en el informe y en el código a través de comentarios). Luego implementar constructores, getters, setters y otros métodos según lo requerido en los requisitos a continuación.</li> </ul>

### Funcionalidades mínimas obligatorias a implementar

**Al igual que el laboratorio 3, las siguientes funcionalidades mínimas deben estar implementadas en su solución. Estas funcionalidades pueden ser ejecutadas mediante uno o más métodos dentro de su programa. Puede utilizar como referencia su implementación del laboratorio pasado.**

1. **(0.7 pto) register/login/logout:** Esta funcionalidad debe permitir el registro de nuevos usuarios en la plataforma, además de su ingreso y salida a la plataforma con sus credenciales.

<b>Prerrequisitos para evaluación</b>	- Clases y estructuras que forman el programa
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"> <li>- Estas funcionalidades deben estar implementadas a través de la interfaz gráfica utilizando el paradigma dirigido por eventos</li> <li>- Se debe almacenar el nuevo usuario en el Stack</li> <li>- El programa debe validar las credenciales del usuario en el Stack antes de permitir el ingreso. En caso de que la contraseña sea incorrecta o el usuario no exista en el <i>stack</i> debe reflejar dicho comportamiento en la interfaz gráfica.</li> <li>- En caso de <i>logout</i> debe existir una sesión iniciada previamente en el stack.</li> </ul>

<b>Parámetros de entrada</b>	Nombre de usuario y contraseña
------------------------------	--------------------------------

2. **(0.7 pto) ask:** Funcionalidad que permite a un usuario (con sesión iniciada en el Stack) realizar una nueva pregunta.

<b>Prerrequisitos para evaluación</b>	- register/login/logout
<b>Requisitos de implementación</b>	<p>- Implementar la vista relacionada con la creación y envío de preguntas. Es decir, debe implementar una vista que le da la posibilidad al usuario de ingresar un texto correspondiente al cuerpo de la pregunta y posteriormente el envío al stack.</p> <p>- Se debe almacenar la nueva pregunta en el Stack, rellenando los atributos asociados a esta al momento de crear la pregunta (cómo el ID, Fecha de creación, Autor y Estado).</p> <p>- Solo los usuarios con sesión iniciada pueden crear preguntas en el Stack.</p>
<b>Parámetros de entrada</b>	Título de la pregunta, Contenido y Lista de etiquetas

3. **(0.7 pto) answer:** Funcionalidad que permite añadir una respuesta a una pregunta dentro del Stack, dejándola almacenada en la estructura correspondiente.

<b>Prerrequisitos para evaluación</b>	- ask
<b>Requisitos de implementación</b>	- Esta funcionalidad debe mostrar una lista con todas las preguntas del Stack, permitiendo elegir al usuario seleccionar una de ellas desde la interfaz gráfica para poder escribir una respuesta a ésta.

	- Solo los usuarios con sesión iniciada pueden responder preguntas del Stack.
<b>Parámetros de entrada</b>	Pregunta seleccionada y contenido de la respuesta

4. **(0.7 pto) reward:** Funcionalidad que permite a un usuario (con sesión iniciada) en la plataforma ofrecer una recompensa para una determinada pregunta.
- La recompensa puede ir dirigida a preguntas propias o de terceros.
  - No se puede ofrecer recompensas mayores a la reputación con la que cuenta el usuario.
  - Una vez ofrecida la recompensa, el puntaje ofrecido queda retenido temporalmente y descontado de la reputación actual.
  - De esta forma se resguarda que el usuario no pueda ofrecer más recompensas de las que podrá eventualmente pagar.
  - Una vez que una respuesta es aceptada, la recompensa se descuenta definitivamente de la reputación del autor y se traspasa al usuario que entregó la respuesta.

<b>Prerrequisitos para evaluación</b>	- ask
<b>Requisitos de implementación</b>	<p>- Para esta funcionalidad, en la interfaz gráfica se debe mostrar una lista con todas las preguntas del Stack, permitiendo elegir una de las preguntas para poder ofrecer la recompensa.</p> <p>- El puntaje retenido debe quedar almacenado en el Stack hasta que la pregunta tenga respuesta aceptada.</p> <p>- Solo los usuarios con sesión iniciada pueden ofrecer recompensas.</p>
<b>Parámetros de entrada</b>	Pregunta seleccionada y recompensa (en puntos)
<b>Prerrequisitos para evaluación</b>	- ask

5. **(0.7 pto) accept:** Funcionalidad que permite a un usuario (con sesión iniciada en la plataforma) aceptar una respuesta a una de sus preguntas.

<b>Prerrequisitos para evaluación</b>	- ask
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"><li>- Para esta funcionalidad, implementar una vista que muestre solo la lista con las preguntas del propio usuario en Stack, y que permita seleccionar una de las preguntas para aceptar una de las respuestas indicadas en ella.</li><li>- Si hay recompensas activas en la pregunta, estas deben resolverse otorgando los puntos de reputación al usuario.</li><li>- Solo los usuarios con sesión iniciada pueden aceptar respuestas.</li></ul>
<b>Parámetros de entrada</b>	Pregunta seleccionada y respuesta seleccionada

**De las siguientes funcionalidades implementar sólo 1 conforme al cumplimiento de los prerrequisitos. La nota máxima en total será un 7.0 aun cuando se desborde puntaje.**

1. **(2 pto) vote:** Función que permite a un usuario votar por una pregunta o respuesta. El voto puede ser positivo o negativo.

<b>Prerrequisitos para evaluación</b>	<ul style="list-style-type: none"><li>- ask</li><li>- answer</li></ul>
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"><li>- Las entidades de Pregunta y Respuesta deben poder registrar la cantidad de votos.</li><li>- Se debe otorgar el puntaje correspondiente a la Pregunta y Respuesta.</li><li>- Para esta funcionalidad, la interfaz gráfica debe mostrar una lista con todas las preguntas y respuestas del Stack (de</li></ul>

	<p>forma ordenada por la fecha de publicación) que no sean del mismo usuario para permitir elegir una pregunta o una respuesta para otorgar el voto positivo o negativo</p> <ul style="list-style-type: none"> <li>- La elección de la pregunta/respuesta a votar puede ser por ID mediante el evento de presionar un botón, haciendo click sobre un ítem de una lista u otro evento, queda a su decisión.</li> <li>- Solo los usuarios con sesión iniciada pueden votar.</li> </ul>
<b>Parámetros de entrada</b>	Pregunta o respuesta seleccionada, Tipo de voto (positivo o negativo)

2. **(2 pto) report:** Funcionalidad que permite a un usuario reportar preguntas o respuestas que contengan contenido ofensivo. Este contenido puede encontrarse en preguntas o respuestas.

<b>Prerrequisitos para evaluación</b>	<ul style="list-style-type: none"> <li>- ask</li> <li>- answer</li> </ul>
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"> <li>- Las entidades de Pregunta y Respuesta deben poder registrar la cantidad de reportes.</li> <li>- Solo los usuarios con sesión iniciada pueden reportar.</li> </ul>
<b>Parámetros de entrada</b>	Pregunta o respuesta seleccionada, Tipo de voto (positivo o negativo)

3. **(2 pto) ranking:** Funcionalidad que retorna un ranking de las preguntas o respuestas de una pregunta según votación, pudiendo ser el orden ascendente (menos votado al más votado) o descendente (más votado al menos votado). La función se puede usar directamente sin pasar por login.

<b>Prerrequisitos para evaluación</b>	<ul style="list-style-type: none"> <li>- ask</li> <li>- answer</li> </ul>
---------------------------------------	---



	<ul style="list-style-type: none"> <li>- reward o vote</li> </ul>
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"> <li>- La funcionalidad debe permitir ordenar por ranking de puntaje (ascendente o descendente) todas las preguntas del stack, o bien todas las respuestas a una pregunta.</li> <li>- Puede utilizar métodos de ordenamiento disponibles en la biblioteca estándar de C# u Java, o bien implementar uno propio.</li> <li>- No es necesario tener sesión iniciada para usar la funcionalidad ranking.</li> </ul>
<b>Parámetros de entrada</b>	Stack o pregunta seleccionada, Criterio de ordenamiento (ascendente o descendente).
<b>Parámetros de salida</b>	Lista con los elementos ordenados según criterio

4. **(2 pto) search:** Funcionalidad que permite buscar preguntas o respuestas en base a una coincidencia parcial de texto. Se debe identificar el tipo de búsqueda a realizar (si se hará por cualquier texto dentro de las preguntas, respuestas o etiquetas). La función se puede usar directamente sin pasar por login.

<b>Prerrequisitos para evaluación</b>	<ul style="list-style-type: none"><li>- ask</li><li>- answer</li></ul>
<b>Requisitos de implementación</b>	<ul style="list-style-type: none"><li>- La funcionalidad debe permitir ordenar buscar por preguntas, respuestas o etiquetas dentro del Stack.</li><li>- La funcionalidad debe permitir coincidencias parciales de texto (substring) y debe ser insensible a las mayúsculas.</li><li>- No es necesario tener sesión iniciada para usar la funcionalidad search.</li></ul>
<b>Parámetros de entrada</b>	Stack seleccionado, criterio de búsqueda (preguntas, respuestas o etiquetas), string a buscar.
<b>Parámetros de salida</b>	Preguntas o respuestas que coincidan con los criterios de búsqueda.

## Referencia para el diseño e implementación de la interfaz gráfica

**Importante:** En este laboratorio, no se evalúa su diseño gráfico o qué tan atractivo es visualmente la interfaz implementada. Se evalúa el cumplimiento de las funcionalidades previamente descritas. Implemente el diseño que más le acomode. La presente sección es solo para orientarlo. La implementación de la interfaz gráfica es libre.

El diseño de las vistas de la interfaz gráfica queda a su elección. En caso de que le oriente, puede utilizar las siguientes maquetas como referencia para la base de su laboratorio. **No es necesario implementar las vistas tal cual cómo se ve en las siguientes maquetas**, es sólo una orientación.



Figura 1. Maqueta de referencia para la vista principal de despliegue de preguntas del *stack* sin tener sesión iniciada.

The image shows two side-by-side wireframe windows for a StackOverflow application. Both windows have a title bar with the text 'StackOverflow' and three window control buttons (minimize, maximize, close). The left window is for login, featuring a 'Username' label, a text input field with placeholder text 'Ingresar username', a 'Password' label, a text input field with placeholder text 'Ingresar password', a blue 'Login' button, and a white 'Volver' button. The right window is for registration, titled 'Registro de usuario', featuring a 'Username' label, a text input field with placeholder text 'Ingresar username', a 'Password' label, a text input field with placeholder text 'Ingresar password', a blue 'Registrarse' button, and a white 'Volver' button.

Figura 2. Maqueta de referencia para el login y registro de usuario

The image shows a single wireframe window for a StackOverflow application. The title bar contains the text 'StackOverflow' and three window control buttons (minimize, maximize, close). The main content area displays the message 'Usuario no existe!' in a bold font, centered. Below the message is a white button with the text 'Volver'.

Figura 3. Maqueta de referencia para cualquier diálogo de información al usuario

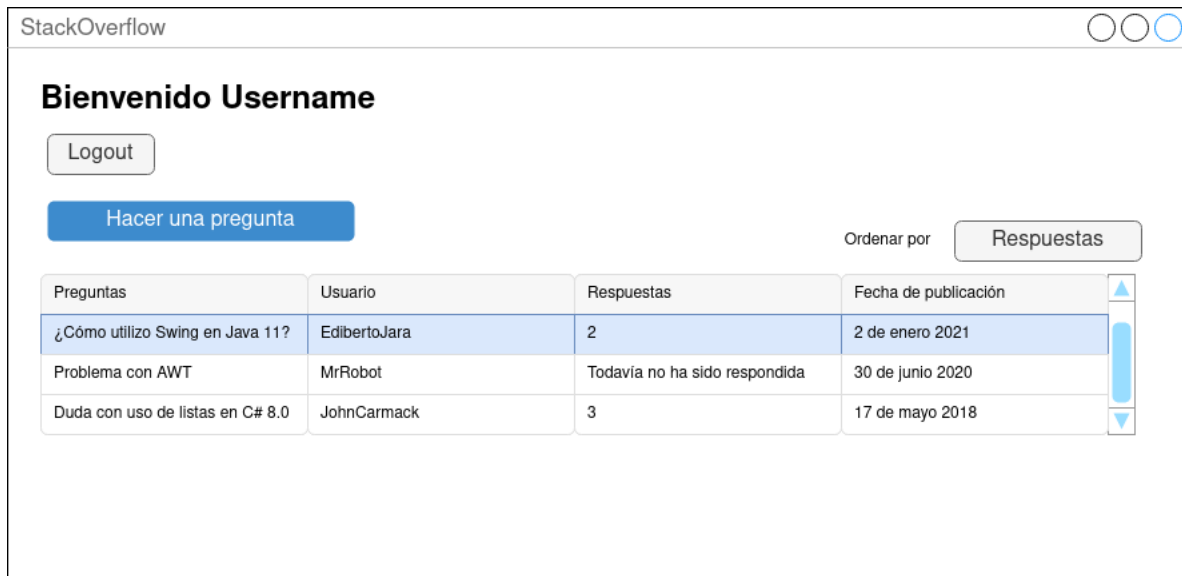


Figura 4. Maqueta de referencia para la vista principal de despliegue de preguntas del *stack* con la sesión iniciada.



Figura 5. Maqueta de referencia para la vista de despliegue de respuestas del *stack* con la sesión iniciada.