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

Laboratorio Paradigmas de la programación
Paradigma Funcional

Catalina Isidora Riquelme Zamora

Profesor: Víctor Flores
Sección: B-2

Julio 2021

Santiago- Chile

Contenido

Introducción

Para un programador que desea desarrollarse de manera óptima es imprescindible conocer y adquirir diversos paradigmas de la programación, siendo este un marco de referencia que indica la forma de programar, enfrentar un problema, analizarlo, diseñar una solución e implementación, estos son constituidos por principios y un grupo acotado de conceptos sobre cómo se percibe el mundo. Los paradigmas de programación a diferencia de los lenguajes de programación no son tan variados siendo esto una ventaja para quien desea aprenderlos, dentro de estos se encuentra el paradigma lógico el cual destaca por centrarse en el qué y no en el cómo, en base a esto el proyecto a trabajar se verá totalmente ligado a este paradigma.

Descripción del problema

El problema por abordar en este laboratorio consta de la simulación e implementación a través del paradigma funcional de una red social compuesta por un grupo de actores interconectadas a través de relaciones ya sean unidireccionales o bien bidireccionales, en donde pueden existir publicaciones y reacciones.

Descripción breve del paradigma

El paradigma lógico se basa fundamentalmente en la lógica matemática en donde las declaraciones de programas se expresan en hechos y reglas sobre problemas dentro de un sistema, estas reglas se escriben como clausulas lógicas con un cuerpo y una cabeza. Además, es importante destacar que este paradigma cuenta con un enfoque declarativo en lugar de uno imperativo. La programación lógica se caracteriza por la posibilidad de expresar el conocimiento de una manera que no dependa de la implementación, logrando que los programas sean más flexibles, comprimidos y compresibles. Por otra parte, este paradigma permite ser utilizado en disciplinas no computacionales que se basan en el razonamiento y medios precisos de expresión.

Prolog se basa en el paradigma lógico, siendo uno de los mayores exponentes de este. Dentro de este existen tres instrucciones básicas; los hechos, los cuales son afirmaciones fundamentales sobre el dominio del problema; las reglas, las cuales son inferencias sobre hechos en el dominio y las preguntas, las cuales son preguntas sobre el dominio.

Análisis del problema

Para el desarrollo de este proyecto es necesario considerar la implementación del paradigma lógico simulando las funcionalidades de una red social. Para poder llevar a cabo el proyecto se deben cubrir requisitos específicos además de cumplir con el lenguaje de programación Prolog.

Para implementar una socialnetwork en primera instancia se realizaron implementaciones de diversos TDAs, los cuales fueron basados en listas, incluyendo constructores, selectores, modificadores, entre otros. En primera instancia se realizó una estructura base, TDA Social, la cual corresponde a un TDA en donde se ven contenidos los usuarios, publicaciones, comentarios y el usuario que tenga la sesión activada. Con el fin de representar a los usuarios de la red social se realizó un TDA User el cual almacena nombre del usuario, contraseña, fecha de registro, una lista que contiene los id de todas las publicaciones que ha realizado este, una lista que contiene los usuarios a los cuales este sigue, una lista que contiene los id de los cuales han sido compartidos a modo de muro y finalmente una lista que contiene todos los id de los comentarios que ha hecho un determinado usuario. Representado de la siguiente manera:

TDA User

Name = string
Password = string
Date_user = list (number)
Id_post_user = list (number)
Follower = list (string)
Id_feed = list (number)
Id_comment_user = list (number)

Para la representación de las publicaciones realizadas en la red social se realizó un TDA Post el cual almacena el id del post, contenido de este (texto), autor de la publicación, fecha en la fue realizada, el número de me gusta que tenga registrada la publicación y una lista que contiene los id de los comentarios que se encuentran relacionadas a esta. Representado de la siguiente manera:

TDA Post

Id_post: number
Content_post: string
Author_post: string
Date_post: date
Like_post: number
[Id_comment_post]: list (number)

Finalmente, para la representación de los comentarios realizados a diversas publicaciones de la red social se realizó un TDA Comment el cual almacena los id del comentario, contenido de este (texto), autor de la publicación, fecha en la fue realizada y el número de me gusta que tenga registrada el comentario. Representado de la siguiente manera:

TDA Comment

> Id_comment: number
> Content_comment: string
> Author_comment: string
> Date_comment: date
> Like_comment: number

Finalmente, una vez definido el TDA User, Post y Comment se puede representar la red social en el TDA Social, en donde se almacenan los usuarios registrados en esta en una lista, las publicaciones en otra lista al igual que los comentarios que se realizan a esta, finalmente se almacena el usuario activo. Representado de la siguiente manera

TDA Social

> Users: list(User)
> Posts: list(Post)
> Comments: list(Comment)
> Active_user: User

Diseño de la solución

Tras la definición de los TDA requeridos para el correcto funcionamiento de todo el programa se logra facilitar de manera considerable la implantación de los demás requisitos funcionales. En el caso de predicado socialNetworkRegister permite consultar el valor que toma una red social tras el registro de un nuevo usuario, donde los parámetros son un SocialNetwork inicial, fecha de registro, nombre del usuario, contraseña y el SocialNetwork resultante luego del registro. En primera instancia se revisa la existencia de un usuario activo, en el caso de existir el predicado retorna automáticamente falso, en el caso contrario que no exista un usuario activo en la red social se verifica que el SocialNetwork inicial efectivamente corresponda a una red social, por otra parte se debe cumplir que la fecha corresponda a un dato del tipo date y finalmente que tanto el nombre como la contraseña ingresada correspondan a strings, posteriormente se revisa que el nombre del usuario a registrar se encuentre disponible en la red social, en el caso que se cumpla se actualiza para finalmente retornar Sn2.

Para el predicado socialNetworkLogin permite autenticar a un usuario registrado, donde los parámetros son un SocialNetwork inicial, nombre del usuario, contraseña y el SocialNetwork resultante luego del ingreso. En donde, en primera instancia se revisa la existencia de un usuario activo, para posteriormente verificar que el SocialNetwork inicial corresponda a una red social, además se verifica que tanto el usuario como la contraseña sean strings, para posteriormente revisar si el usuario ingresado se encuentra registrado dentro de la red social en el caso que este exista se agrega el usuario seleccionado a la lista Active_user, dando como resultado la variable Sn2.

Para el predicado socialNetworkPost permite a un usuario con sesión iniciada en la plataforma realizar una nueva publicación propia o dirigida a otros usuarios, donde los parámetros son un SocialNetwork inicial, fecha de la publicación, contenido de la publicación (texto), lista de usuarios a los cuales va dirigido el post y el SocialNetwork resultante posterior a la publicación. En primera

instancia se revisa la existencia de un usuario activo, en el caso de no existir el predicado retorna automáticamente falso, en el caso contrario que no exista un usuario activo en la red social se verifica que el SocialNetwork inicial efectivamente corresponda a una red social, por otra parte se debe cumplir que la fecha corresponda a un dato del tipo date, además de que el contenido de la publicación corresponda a un string y finalmente que la lista de usuarios corresponda a una lista conformada por strings. Una vez revisado los aspectos anteriormente nombrados sea crea un nuevo Post, actualizando la lista de ID de post realizadas por el usuario que se encuentra activo, retornando Sn2.

Para el predicado socialNetworkFollow permite a un usuario con sesión iniciada seguir a otro usuario dentro de la red social, donde los parámetros son un SocialNetwork inicial, usuario al que se desea seguir y el SocialNetwork resultante. En primera instancia se revisa la existencia de un usuario activo, en el caso de no existir el predicado retorna automáticamente falso, en el caso contrario que no exista un usuario activo en la red social se verifica que el SocialNetwork inicial efectivamente corresponda a una red social, además de verificar si el usuario que se desea seguir es un string y se encuentra registrado en la red social, también es importante verificar que el usuario activo sea diferente al que se desea seguir, con la finalidad de que ningún usuario pueda seguirse a sí mismo. Una vez revisados estos aspectos anteriormente nombrados se crea un nuevo usuario con la finalidad de actualizar lista de seguidores del usuario activo, retornando Sn2.

Para el predicado socialNetworkShare permite un usuario con sesión iniciada en la plataforma compartir contenido de otro usuario en su propio espacio o dirigido a otros usuarios más, esto se verá representado en una lista feed que contendrá los id compartidos, donde los parámetros son un SocialNetwork inicial, fecha, id del post, usuario y el SocialNetwork resultante. En primera instancia se revisa la existencia de un usuario activo, en el caso de no existir el predicado retorna automáticamente falso, en el caso contrario que no exista un usuario activo en la red social se verifica que el SocialNetwork inicial efectivamente corresponda a una red social, además de que la fecha ingresada corresponda un valor Date, por otra parte, que el id se encuentre registrado y corresponda a un número y finalmente que el usuario que se busca compartir sea un string. Una vez revisados estos aspectos anteriormente nombrados se crea un nuevo usuario con la finalidad de actualizar lista de seguidores del usuario activo, retornando Sn2.

Para el predicado socialNetworkToString permite obtener una representación de un TDA socialNetwork como un string posible de visualizar de forma comprensible al usuario, donde los parámetros son un un SocialNetwork inicial y un SocialNetwork resultante. En el caso que exista un usuario activo el SocialNetwork resultante contendrá los datos del usuario activo además de las publicaciones y comentarios creados por él. A diferencia de cuando no existe un usuario activo en donde se mostrará todo los usuarios, publicaciones y comentarios realizados en la red social.

Para el predicado comment permite a un usuario con sesión iniciada en la plataforma comentar publicaciones, donde los parámetros son un SocialNetwork inicial, fecha en donde fue creado el comentario, id del post que se desea comentar, contenido del comentario y un socialNetwork resultante. En primera instancia se revisa la existencia de un usuario activo, en el caso de no existir el predicado retorna automáticamente falso, en el caso contrario que no exista un usuario activo en

la red social se verifica que el SocialNetwork inicial efectivamente corresponda a una red social, por otra parte, se debe cumplir que la fecha corresponda a un dato del tipo date, también se verifica que el id del post que se desea comentar exista dentro de la red social y por último se verifica que el contenido del comentario sea un string. Una vez revisado los aspectos anteriormente nombrados se crea un nuevo comentario, actualizando la lista de ID de comentarios que ha realizado el usuario que se encuentra activo, retornando Sn2.

Para el predicado socialNetworkLike permite a un usuario con sesión iniciada en la plataforma reaccionar a publicaciones a través de "me gusta", donde los parámetros son un SocialNetwork inicial, fecha cuando se realizo la interacción, id del post que se desea reaccionar y un SocialNetwork resultante. En primera instancia se revisa la existencia de un usuario activo, en el caso de no existir el predicado retorna automáticamente falso, en el caso contrario que no exista un usuario activo en la red social se verifica que el SocialNetwork, además se verifica que la fecha corresponda a una variable tipo date y que la variable Id_post sea un número.

Aspectos de implementación

Para lograr una correcta implementación de cada una de las funciones a través del proyecto se utilizó la implementación en código abierto de Prolog llamada SWI-Prolog, versión 8.2.4 además del programa Visual Studio Code como editor de la base de conocimiento. Todos los TDA utilizados a lo largo de programa se encuentras localizados en un solo archivo incluyendo además los Dominios, Predicados, Metas y Clausulas requeridas.

Instrucciones de uso

Para registrarse en la red social se utiliza la consulta socialNetworkRegister, la cual requiere un socialNetwork para trabajar. Si todo fue ingresado correctamente debería retornar el valor de la variable socialNetwork actualizado con el nuevo usuario registrado. Si al llamar esta consulta de nuevo se le ingresara el valor de la variable, esta retornará true.

Para iniciar sesión en el foro sirve la consulta socialNetworLogin, la que requiere un socialNetwork. Si todo fue ingresado correctamente, entonces se debería retornar el valor de la variable como el socialNetwork actualizado con el usuario activo. Si al llamar esta consulta de nuevo se le ingresara el valor de la variable, esta retornará true.

Para realizar un post en la red social sirve la consulta socialNetworkPost, la cual requiere un socialNetwork para trabajar. Si todo fue ingresado correctamente, entonces se debería retornar el valor de la variable como el socialNetwork actualizado con el nuevo post realizado y la lista de IDs de post actualizada del usuario activo. Si al llamar esta consulta de nuevo se le ingresara el valor de la variable, esta retornará true.

socialNetworkFollow, la cual requiere un socialNetwork para trabajar. Si todo fue ingresado correctamente, entonces se debería retornar el valor de la variable como el socialNetwork actualizado con la lista de Followers del usuario ingresado. Si al llamar esta consulta de nuevo se le ingresara el valor de la variable, esta retornará true.

socialNetworkShare, la cual requiere un socialNetwork para trabajar. Si todo fue ingresado correctamente, entonces se debería retornar el valor de la variable como el socialNetwork actualizado con la lista Feed del usuario ingresado. Si al llamar esta consulta de nuevo se le ingresara el valor de la variable, esta retornará true.

Para responder a un post sirve la consulta socialNetworkComment la cual requiere un socialNetwork para trabajar. Si todo fue ingresado correctamente, entonces se debería retornar el valor de la variable valor de la variable como el socialNetwork actualizado con el nuevo comentario realizado y la lista de IDs de comentarios actualizada del usuario activo. Si al llamar esta consulta de nuevo se le ingresara el valor de la variable, esta retornará true.

socialNetworkLike, la cual requiere un socialNetwork para trabajar. Si todo fue ingresado correctamente, entonces se debería retornar el valor de la variable como el socialNetwork actualizado con la lista post actualizada. Si al llamar esta consulta de nuevo se le ingresara el valor de la variable, esta retornará true.

Finalmente, si se desea visualizar la red social sirve la consulta socialNetworkToString , esta requiere un socialNetwork. Dependiendo de si hay un usuario activo o no es si muestra todo lo relacionado ala red social o solo los datos del usuario activo junto a sus posts y comment. Si al llamar esta consulta de nuevo se le ingresara el valor de la variable, esta retornará true.

Resultados y autoevaluación

Para la realización de este proyecto hubo varios requerimientos que debían cumplirse para la posterior evaluación, entre estos requerimientos se pueden apreciar los funcionales y los NO funcionales. A continuación, se presentarán dos tablas indicando los requerimientos y el grado de logro de cada uno, es decir, la autoevaluación.

Estado	Puntaje
No realizado	0
Funciona el 25% de las veces o no funciona	0.25
Funciona el 50% de las veces	0.50
Funciona el 75% de las veces	0.75
Funciona el 100% de las veces	0.1

Requerimientos funcionales	Evaluación
TDAs	1
socialNetworkRegister	1
socialNetworkLogin	1
socialNetworkPost	1
socialNetworkFollow	1
socialNetworkShare	1
socialNetworkToString	1
Comment	1
socialNetworkLike	1
socialNetworkViral	0
socialNetworkSearch	0

Requerimientos NO funcionales	Evaluación
Autoevaluación	1
Lenguaje	1
Versión	1
Historial	1
Ejemplos	1
Prerrequisitos	1

Se realizaron un total de 24 pruebas, realizándose 3 pruebas por cada requerimiento funcional (las pruebas realizadas se pueden ver en el anexo adjunto), estas funcionaron en su totalidad, logrando un 100% en su funcionalidad. No se lograron realizar dos requerimientos funcionales, socialNetworkViral y socialNetworkSearch, correspondientes a los predicados a implementar de manera libre, estos no fueron implantados ya que no se logro llegar a un correcta implantación de estos por lo que se decidió dar preferencia a los requerimientos anteriores, para que funcionaran de manera óptima.

Conclusión

Al realizar el proyecto, se presentaron diversas complicaciones en su elaboración debido principalmente a la sintaxis del lenguaje ocasionado por la costumbre de trabajar constantemente con el paradigma imperativo. Un aspecto positivo al momento de realizar este proyecto fue contar con información desarrollada durante el laboratorio de lenguaje funcional el cual ayudo en la definir las bases del actual proyecto.

Como aspecto negativo se puede encontrar que este paradigma no es nada amigable con el usuario que interactúa con el, llevando como consecuencia un trabajo arduo registrarse, entrar a la red social, entre las otras funciones, en cada operación hay que copiar y pegar, llamar constructores, entre otros temas, resultado complejo para una persona ajena a la que construyo el código.

Los resultados alcanzados en la elaboración de este proyecto utilizando el paradigma lógico fue superior respecto al paradigma funcional, debido a que se alcanzó a realizar de manera correcta un mayor número de requerimientos funcionales, debido principalmente al aumento de tiempo otorgado y al estudio superior respecto al laboratorio anterior.

Referencias

SWI Prolog (s.f.) *"The SWI-Prolog syntax".* Recuperado de https://www.swi-prolog.org/pldoc/man?section=syntax

Anexo

Register

socialNetworkRegister([[["user","pass",[30, 11, 2020],[],[],[],[]]],[],[],[[]]], [12, 3, 2020], "Elizabeth","123", Sn2).

socialNetworkRegister([[["Elizabeth","123",[12, 3, 2020],[],[],[],[]],["user","pass",[30, 11, 2020],[],[],[],[]]],[],[],[[]]], [14, 3, 2020], "Maria","pass", Sn2).

(Ejemplo usuario ya registrado, no se puede registrar)

socialNetworkRegister([[["Elizabeth","123",[12, 3, 2020],[],[],[],[]], ["user","pass",[30, 11, 2020],[],[],[],[]]], [],[],[[]]], [14, 3, 2020], "Elizabeth","123", Sn2).

Login

socialNetworkLogin([[["Elizabeth","123",[12, 3, 2020],[],[],[],[]], ["user","pass",[30, 11, 2020],[],[],[],[]]], [],[],[[]]],"user", "pass", Sn2).

socialNetworkLogin([[["Elizabeth","123",[12, 3, 2020],[],[],[],[]], ["user","pass",[30, 11, 2020],[],[],[],[]]], [],[],[[]]],"Elizabeth","123", Sn2).

(Ejemplo usuario no registrado)

socialNetworkLogin([[["Elizabeth","123",[12, 3, 2020],[],[],[],[]],["user","pass",[30, 11, 2020],[],[],[],[]]], [],[],[[]]],"Lizzie","123", Sn2).

Post

socialNetworkPost([[["Elizabeth","pass",[30, 11, 2020],[],[],[],[]], ["user","pass",[30, 11, 2020],[],[],[],[]]],[],[],[["user","pass",[30, 11, 2020],[],[],[],[]]]], [30, 11, 2020],"Hola",[], Sn2).

socialNetworkPost([[["Elizabeth", "pass", [30, 11, 2020],[],[],[],[]], ["user", "pass", [30, 11, 2020], [1],[],[],[]]], [[1, "Hola", "user", [30, 11, 2020], 0, []]], [], [["Elizabeth","pass",[30, 11, 2020],[],[],[],[]]]], [30, 11, 2020],"Nuevo post",[], Sn2).

(Ejemplo no hay usuario activo, no se puede hacer post)

socialNetworkPost([[["Elizabeth", "pass", [30, 11, 2020],[],[],[],[]], ["user", "pass", [30, 11, 2020], [1],[],[],[]]], [[1, "Hola", "user", [30, 11, 2020], 0, []]], [], [[]]], [30, 11, 2020],"Nuevo post",[], Sn2).

Follow

socialNetworkFollow([[["Elizabeth","123",[12, 3, 2020],[],[],[],[]], ["user","pass",[30, 11, 2020],[],[],[],[]]],[],[],["user","pass",[30, 11, 2020],[],[],[],[]]],"Elizabeth", Sn2).

(Ejemplo seguirse a si mismo)

socialNetworkFollow([[["Elizabeth","123",[12, 3, 2020],[],[],[],[]], ["user","pass",[30, 11, 2020],[],[],[],[]]], [],[],["user","pass",[30, 11, 2020],[],[],[],[]]],"user", Sn2).

(Ejemplo sin usuario activo)

socialNetworkFollow([[["Elizabeth","123",[12, 3, 2020],[],[],[],[]], ["user","pass",[30, 11, 2020],[],[],[],[]]],[],[],[[]]],"user", Sn2)

Share

socialNetworkShare([[["Elizabeth", "pass", [30, 11, 2020], [], ["user"], [],[]], ["user", "pass", [30, 11, 2020], [1],["Elizabeth"],[],[]]], [[1, "Hola", "user", [30, 11, 2020], 0, []]], [], [["user", "pass", [30, 11, 2020], [1], ["Elizabeth"],[],[]]]], [30, 11, 2020],1,"Elizabeth", Sn2).

(Ejemplo sin usuario activo)

socialNetworkShare([[["Elizabeth", "pass", [30, 11, 2020], [], ["user"], [],[]], ["user", "pass", [30, 11, 2020], [1],["Elizabeth"],[],[]]], [[1, "Hola", "user", [30, 11, 2020], 0, []]], [], [[]]], [30, 11, 2020],1,"Elizabeth", Sn2).

(Ejemplo id no válido)

socialNetworkShare([[["Elizabeth", "pass", [30, 11, 2020], [], ["user"], [],[]], ["user", "pass", [30, 11, 2020], [1],["Elizabeth"],[],[]]], [[1, "Hola", "user", [30, 11, 2020], 0, []]], [], [["user", "pass", [30, 11, 2020], [1], ["Elizabeth"],[],[]]]], [30, 11, 2020],13,"Elizabeth", Sn2).

String

(Ejemplo sin usuario activo)

socialNetworkToString([[["Elizabeth","pass",[30, 11, 2020],[1],[],[],[]], ["user","pass",[30, 11, 2020],[],[],[],[]]], [[1,"Hola mundo","Elizabeth",[30, 11, 2020],3,[]], [2,"Mi primer post","user",[30, 11, 2020],1,[]]], [],[[]]],Sn3),

write(Sn3).

(Ejemplo con usuario activo)

socialNetworkToString([[["Elizabeth","pass",[30, 11, 2020],[1],[],[],[]], ["user","pass",[30, 11, 2020],[],[],[],[]]], [[1,"Hola mundo","Elizabeth",[30, 11, 2020],3,[]], [2,"Mi primer post","user",[30, 11, 2020],1,[]]],[],[["Elizabeth","pass",[30, 11, 2020],[1],[],[],[]]]],Sn3),

write(Sn3).

(Ejemplo con usuario activo)

socialNetworkToString([[["Elizabeth","pass",[30, 11, 2020],[1],[],[],[1]], ["user","pass",[30, 11, 2020],[],[],[],[]]], [[1,"Hola mundo","Elizabeth",[30, 11, 2020],3,[]], [2,"Mi primer post","user",[30, 11, 2020],1,[]]],[[1,"Primer comentario","Elizabeth",[30, 11, 2020],2]], [["Elizabeth","pass",[30, 11, 2020],[1],[],[],[1]]]],Sn3),

write(Sn3).

Comment

comment([[["Elizabeth","pass",[30, 11, 2020],[],[],[],[]], ["user","pass",[30, 11, 2020],[],[],[],[]]], [[1,"Hola mundo","user",[30, 11, 2020],0,[]]],[],["user","pass",[30, 11, 2020],[],[],[],[]]], [30, 11, 2020],1,"uwu", Sn2).

(Ejemplo con id inexistentente dentro de la red social)

comment([[["Elizabeth","pass",[30, 11, 2020],[],[],[],[]],["user","pass",[30, 11, 2020],[],[],[],[]]], [[1,"Hola mundo","user",[30, 11, 2020],0,[]]], [],[["user","pass",[30, 11, 2020],[],[],[],[]]]], [30, 11, 2020],13,"uwu", Sn2).

(Ejemplo sin usuario activo)

comment([[["Elizabeth","pass",[30, 11, 2020],[],[],[],[]], ["user","pass",[30, 11, 2020],[],[],[],[]]],

[[1,"Hola mundo","user",[30, 11, 2020],0,[]]], [],[[]]], [30, 11, 2020],1,"uwu", Sn2).

Like

socialNetworkLike([[["Elizabeth","pass",[30, 11, 2020],[],[],[],[]], ["user","pass",[30, 11, 2020],[],[],[],[]]], [[1,"Hola mundo","user",[30, 11, 2020],0,[]]], [],["user","pass",[30, 11, 2020],[],[],[],[]]], [30, 11, 2020],1, Sn2).

socialNetworkLike([[["user", "pass", [30, 11, 2020],[1],[],[],[]],["Elizabeth", "pass", [30, 11, 2020], [2],[],[],[]]],[[1, "Hola", "user", [30, 11, 2020], 0, []], [2, "Nuevo post", "Elizabeth", [30, 11, 2020], 0, []]], [], [["user","pass",[30, 11, 2020],[],[],[],[]]], [30, 11, 2020],2, Sn2).

(Ejemplo sin usuario activo)

socialNetworkLike([[["user", "pass", [30, 11, 2020],[1],[],[],[]], ["Elizabeth", "pass", [30, 11, 2020],[2],[],[],[]]],[[1, "Hola", "user", [30, 11, 2020], 0, []], [2, "Nuevo post", "Elizabeth", [30, 11, 2020], 0, []]], [], [[]]], [30, 11, 2020],2, Sn2).