Facebook

Final Report
MFES 2018/2019

Mestrado Integrado em Engenharia Informática e Computação

4MIEIC03 - Group 11:
António Almeida - up201505836, up201505836@fe.up.pt
Diogo Torres - up201506428, up201506428@fe.up.pt
João Damas - up201504088, up201504088@fe.up.pt

7th January 2019

1. Informal system description and list of requirements

Informal system description

The system developed aims at partially representing and managing a social network, more specifically Facebook.

Facebook is a platform for people to get to know each other and share experiences, proposing to “bring the world together” as it suggests. In Facebook, there are several users registered, each with a unique name.

A user can have connections with other users, i.e. a friends list, and users that he does not wish to have contact with, i.e. blocked. Each user can also make publications to his timeline. Each publication has a string content associated, a timestamp to mark when it was posted and permission settings to eventually assure that only a subset of the platform’s users can access its content. Permissions vary from public (anyone can see) to friends only, friends and friends of friends or transitive connections (people that can be reached from publication author’s connections). Publications can be liked by users with the necessary permissions. A user can consult his or other people’s timelines, showing the content he has permission to view and check his main feed where the system generates ordered content that is considered most relevant to the user. He can also get suggestions on potential friends.

Private group chats can be set up in order to allow a specific subset of users to exchange messages between each other. A user can search messages in the chats he’s in via a query text or filter messages between specific dates. Finally, a user can also search other users and publications.

List of requirements

ID	Mandatory?	Description
R1	Yes	As a visitor, I want to enter the platform as a new user
R2	Yes	As a user, I want to look up users in the platform
R3	Yes	As a user, I want to manage my friends list (add or remove connections)
R4	Yes	As a user, I want to manage my blocked list (add or remove users)
R5	Yes	As a user, I want to manage my publications (i.e. make new, update or remove existing)
R6	Yes	As a user, I want to like (or unlike) publications I have access to
R7	Yes	As a user, I want to look up publications (that I have access to) in the platform
R8	Yes	As a user, I want to communicate with a specific set of friends in a private group chat
R9	Yes	As a user, I want to lookup messages in a group chat and filter results to a specific period
R10	Yes	As a user, I want to get potential friend suggestions
R11	Yes	As a user, I want to check my or other people's timelines to see what everyone is up to
R12	Yes	As a user, I want to check my main feed to check the most relevant content to me

Table 1 - System requirements

The requirements are translated into use cases which are shown and major ones are described more thoroughly next.

2. Visual UML Model

Use case model

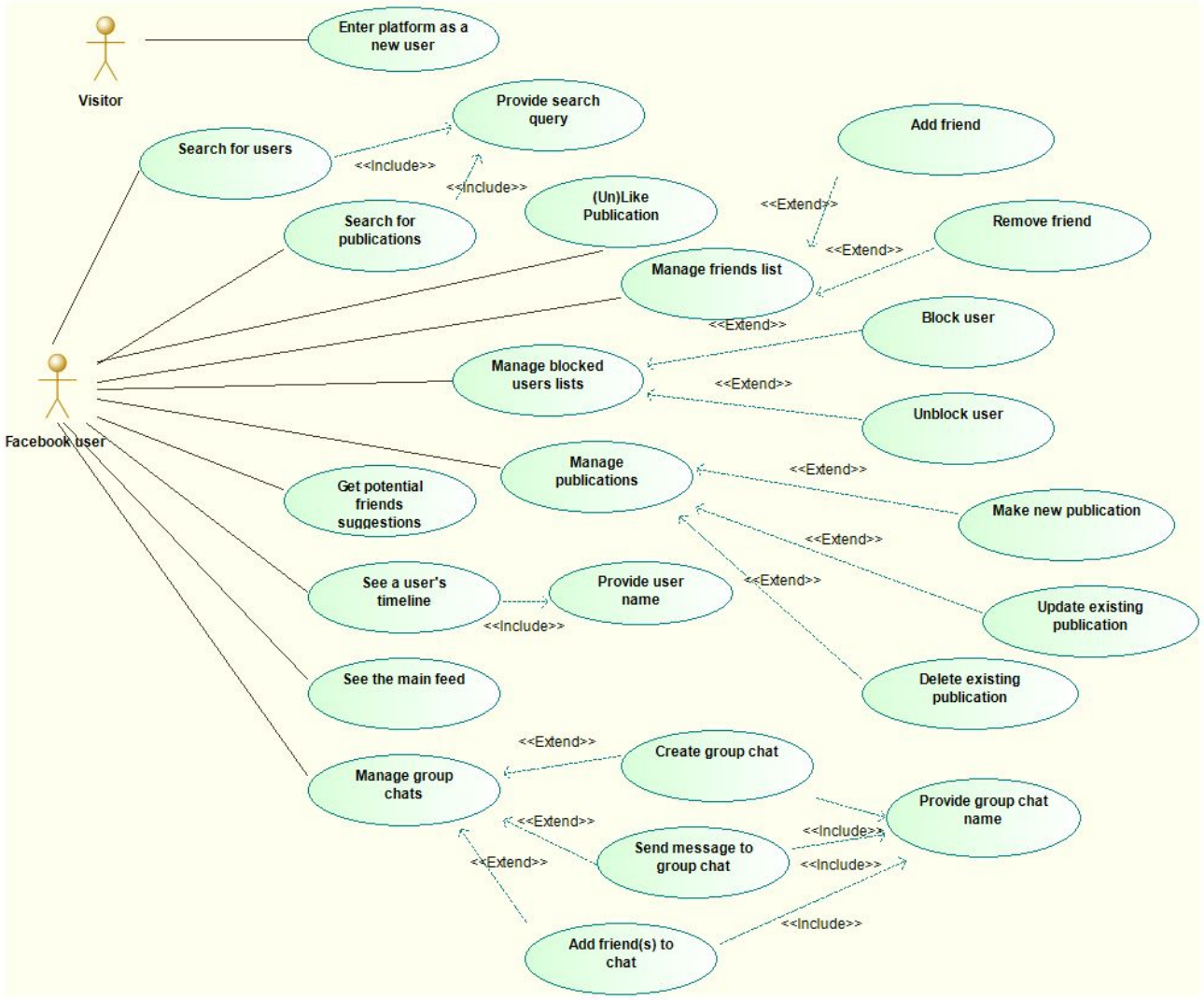


Figure 1 - UML Use case model

Scenario	**Registration**
Description	Scenario where a visitor registers as a new user into the platform
Preconditions	- User name must not exist in the platform (*input*)
Postconditions	- User is part of the platform (*final system state*)
Steps	1. Provide user name
Exceptions	- User name already taken

Scenario	**Add a friend**
Description	Scenario where a user adds another user to his friends list
Preconditions	- Adding user must exist in the platform (*initial system state*) - Adding user must not be blocked by added user (*initial system state*) - Added user must exist in the platform (*input*)
Postconditions	- Both users are in each others' friends list (*final system state*)
Steps	1. Choose option "Add friend" 2. Provide end user name
Exceptions	- Added user does not exist in the platform - Adding user is blocked by added user

Scenario	**Block a user**
Description	Scenario where a user blocks another user to prevent him from seeing all his publications no matter what
Preconditions	- Blocking user must exist in the platform (*initial system state*) - Blocked user must not be in blocking user's friends list (*initial system state*) - Blocked user must exist in the platform (*input*)
Postconditions	- Blocked user is in blocking user's blocked list (*final system state*)
Steps	1. Choose option "Block user" 2. Provide end user name
Exceptions	- Blocked user does not exist in platform - Blocked user is in blocking user's friends list

Scenario	**Make a publication**
Description	Scenario where a user adds a publication to his timeline (sequence of publications)
Preconditions	- User must exist in the platform (*initial system state*) - New publication timestamp must be equal or further in time than current latest publication (*input*)

Postconditions	- Publication is added to the user's timeline (*final system state*)
Steps	2. Choose option "Make publication" 3. Provide publication content, timestamp and access permissions
Exceptions	- User does not exist in platform

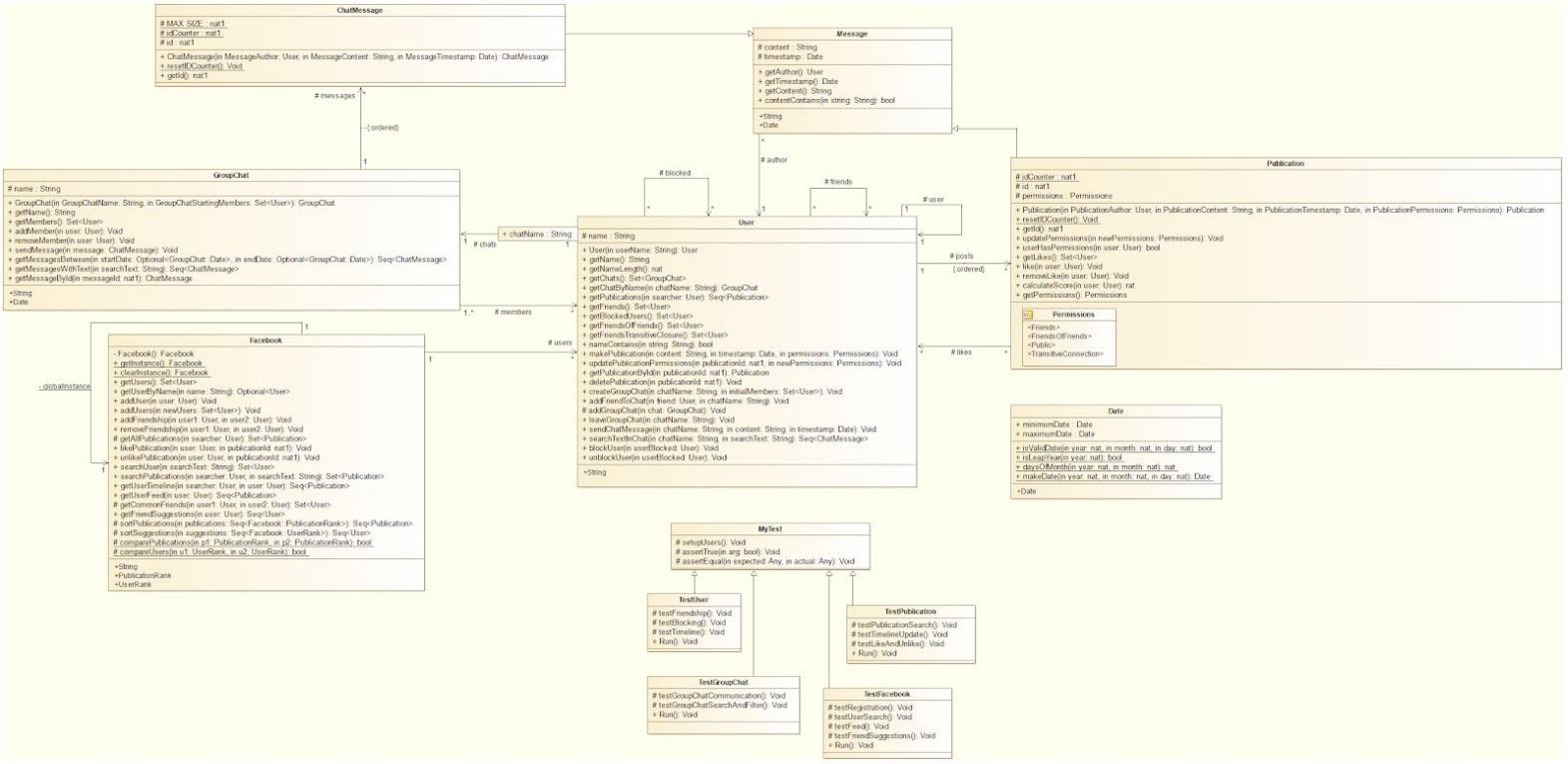
Scenario	**Like a publication**
Description	Scenario where a user joins the set of users that have liked a publication
Preconditions	- User must exist in the platform (*input*) - Liking user must have permissions to access publication (*initial system state*)
Postconditions	- User is part of set of publication's likes (*final system state*)
Steps	4. Choose option "Like publication" 5. Provide publication unique identifier
Exceptions	- User does not exist in platform - User does not have permissions to access publication

Scenario	**Private group chat communication**
Description	Scenario where a user communicates via messages with friends on a private chat
Preconditions	- User must exist in the platform (*initial system state*) - Initial members (in addition to user, as stated above) must exist in the platform (*input*) - Initial members (aside from user) must be user's friends (*input*) - No user from the initial member set can be in another chat with a similar name to the new one (*input*)
Postconditions	- Group chat with initial members and user exists and all can send messages there (*final system state*)
Steps	1. Choose option "Create group chat" 2. Provide chat name 3. (Optional) Provide chat's initial members 4. Send a message to that chat

Exceptions	- One or more users not in the platform - One or more of the initial members (aside from user) not in user's friend list - One or more of the initial members already in a group chat with a similar name

Scenario	**See main feed**
Description	Scenario where a user checks his main feed to assess content that is considered most relevant to him
Preconditions	- User must exist in the platform (*initial system state*)
Postconditions	- Sequence of publications, visible to the user and sorted by relevance, are presented to him (*output*)
Steps	1. Choose option "Get main feed"
Exceptions	- User does not exist in platform

Class model



Class name	Description
Message	Class representing a generic message (with a content and timestamp) authored by a user
ChatMessage	Class representing a message sent to a private group chat
Publication	Class representing a publication (simplified to a message) made for other users in the platform to see
GroupChat	Class representing a private group conversation between users
Date	Class containing the definition of the date type and respective helper functions
User	Class representing a user in the platform
Facebook	Global system manager (singleton)
MyTest	Generic test class containing assertTrue and assertEqual operations
TestFacebok	Test class containing tests for more generic platform requirements
TestUser	Test class containing tests for more user related requirements
TestGroupChat	Test class containing tests for group chat related requirements
TestPublication	Test class containing tests for publication related requirements

3. Formal VDM++ Model

ChatMessage

```
-- Class representing a message sent in a GroupChat
class ChatMessage is subclass of Message
types
values
	protected MAX_SIZE = 250; --Private conversation, so smaller messages
instance variables
	-- Unique ID to identify message
	protected static idCounter: nat1 := 1;
	protected id: nat1 := idCounter;

	-- Guarantee content is smaller than the maximum allowed size
	inv len content <= MAX_SIZE;
operations
	-- Constructor
  public ChatMessage: User * String * Date ==> ChatMessage
    ChatMessage(MessageAuthor, MessageContent, MessageTimestamp) == (
	    idCounter := idCounter + 1;
      author := MessageAuthor;
```

```
        content := MessageContent;
        timestamp := MessageTimestamp;
        return self;
      )
      pre MessageContent <> ""
      post author = MessageAuthor and content = MessageContent and timestamp = MessageTimestamp and
idCounter = idCounter~ + 1;

  -- Unique ID reset
  public static resetIDCounter: () ==> ()
                resetIDCounter() == (
                        idCounter := 1;
                );

  -- ID getter
  pure public getId: () ==> nat1
                        getId() == return id
                        post RESULT = id;
functions
traces
end ChatMessage
```

Function or operation	Line	Coverage	Calls
ChatMessage	15	100.0%	19
getId	33	100.0%	868
resetIDCounter	27	100.0%	12
ChatMessage.vdmpp		100.0%	899

Date

```
-- Utility class to represent a date as an integer in the format YYYYMMDD
class Date
  types
        -- Type definition
    public Date = nat
      inv d == isValidDate(d div 10000, (d div 100) mod 100, d mod 100);

  values
        -- Minimum date possible
                public minimumDate = makeDate(1,1,1);
                -- Maximum date possible
                public maximumDate = makeDate(9999,12,31);
  functions

    -- Checks if a date is valid
    -- OUT true if date is valid, false otherwise
    public static isValidDate: nat * nat * nat -> bool
      isValidDate(year, month, day) ==
        year >= 1 and month >= 1 and month <= 12 and day >= 1 and day <= daysOfMonth(year, month);

    -- Checks if a given year is a leap year. Used for verifying date correctness
    -- OUT true if leap year, false otherwise
    public static isLeapYear: nat -> bool
```

```
    isLeapYear(year) ==
      year mod 4 = 0 and year mod 100 <> 0 or year mod 400 = 0;

                -- Returns the number of days in a month for a given year
                -- OUT number of days for pair month-year
  public static daysOfMonth: nat * nat -> nat
    daysOfMonth(year, month) == (
      cases month :
        1, 3, 5, 7, 8, 10, 12 -> 31,
        4, 6, 9, 11 -> 30,
        2 -> if isLeapYear(year) then 29 else 28
      end
     )
    pre month >= 1 and month <= 12;

                -- "Constructor"
  public static makeDate: nat * nat * nat -> Date
    makeDate(year, month, day) ==
      year * 10000 + month * 100 + day
    pre isValidDate(year, month, day);
end Date
```

Function or operation	Line	Coverage	Calls
daysOfMonth	29	100.0%	10
isLeapYear	23	100.0%	19
isValidDate	17	100.0%	216
makeDate	40	100.0%	49
Date.vdmpp		100.0%	294

Facebook

```
class Facebook
types
        public String = seq1 of char;
        -- Class private type to use in publication sorting for main feed generation
        protected PublicationRank :: publication: Publication

                score: nat;

        protected UserRank :: user: User

                score: nat;

values
instance variables
  public users: set of User := {};
  -- Singleton
  private static globalInstance: Facebook := new Facebook();

  -- User name uniqueness
  inv not exists user1, user2 in set users & user1 <> user2 and user1.getName() = user2.getName();
operations
```

```
        -- Constructor
private Facebook: () ==> Facebook
  Facebook() == (
        users := {};
        return self;
  )
  post users = {};

-- Singleton getter
public pure static getInstance: () ==> Facebook
  getInstance() == return globalInstance;

-- Resets the singleton
public static clearInstance: () ==> Facebook
        clearInstance() == (
                globalInstance := new Facebook();
                Publication`resetIDCounter();
                ChatMessage`resetIDCounter();
                return getInstance();
        );

-- Users getter
public pure getUsers: () ==> set of User
                getUsers() == return users
                post RESULT = users;

-- Retrieve a specific user by his (unique) name, if he exists
-- IN name - user name
-- OUT user object if there is a user with given name, nil otherwise
public pure getUserByName: String ==> [User]
                getUserByName(name) == (
                        dcl result: [User] := nil;

                                for all user in set users do
                                        if user.getName() = name then
                                                result := user
                                        else
                                                skip;

                                return result
                )
                post (exists user in set users & user.getName() = name) => RESULT <> nil;

-- Add a user to the platform
-- IN user - user to be added
public addUser: User ==> ()
  addUser(user) == (
    users := users union {user}
  )
  pre not exists member in set users & member.getName() = user.getName()
  post users = users~ union {user};

-- Add a set of user to the platform simultaneously
-- IN users - set of users to be added
public addUsers: set of User ==> ()
  addUsers(newUsers) == (
    users := users union newUsers
  )
```

```
    pre newUsers inter users = {}
    post users = users~ union newUsers;

  -- Add a friendship between two users
  public addFriendship: User * User ==> ()
    addFriendship(user1, user2) == (
      dcl u1: User := user1;
      dcl u2: User := user2;
      atomic(
        u1.friends := user1.friends union {user2};
        u2.friends := user2.friends union {user1}
      )
    )
    pre user1 in set users and user2 in set users and user1 not in set user2.getBlockedUsers() and user2 not
in set user1.getBlockedUsers()
    post user1 in set user2.friends and user2 in set user1.friends;

  -- Remove a previously existing friendship between users
  public removeFriendship: User * User ==> ()
          removeFriendship(user1, user2) == (
                  dcl u1: User := user1;
      dcl u2: User := user2;
      atomic(
        u1.friends := user1.friends \ {user2};
        u2.friends := user2.friends \ {user1}
      )
          )
          pre user1 in set users and user2 in set users
          post user1 not in set user2.friends and user2 not in set user1.friends;

  -- Retrieve all publications user can see. For internal usage, since post conditions do not allow set
comprehensions for code generation
  -- IN searcher - user to assume perspective from
  -- OUT publications he has access to
  pure protected getAllPublications: User ==> set of Publication
                  getAllPublications(searcher) == return dunion {elems getUserTimeline(searcher,member) |
member in set users};

  -- User likes a publication
  -- IN user - user that likes publication
  -- IN publicationId - unique publication identifier
  public likePublication: User * nat1 ==> ()
                  likePublication(user, publicationId) == (
                          let publication = iota publication in set getAllPublications(user) & publication.getId() =
publicationId in publication.like(user);
                  )
                  pre user in set users and exists1 publication in set getAllPublications(user) &
publication.getId() = publicationId;

          -- User unlikes a publication
  -- IN user - user that currently likes publication
  -- IN publicationId - unique publication identifier
  public unlikePublication: User * nat1 ==> ()
                  unlikePublication(user, publicationId) == (
                          let publication = iota publication in set getAllPublications(user) & publication.getId() =
publicationId in publication.removeLike(user);
                  )
```

```
                pre user in set users and exists1 publication in set getAllPublications(user) &
publication.getId() = publicationId and user in set publication.getLikes();

  -- Look up users whose name matches a search pattern
  -- IN searchText - string to look up
  -- OUT set of users whose name contains search text
  public searchUser: String ==> set of User
                searchUser(searchText) == (
                        return {user | user in set users & user.nameContains(searchText)}
                )
                post forall user in set RESULT & exists i,j in set inds user.getName() &
user.getName()(i,...,j) = searchText;

  -- Look up publications that user has access to and whose content matches a search pattern
  -- IN user - user that is searching
  -- IN searchText - string to look up
  -- OUT set of publications that user has access to and whose content contains search text
  public searchPublications: User * String ==> set of Publication
                searchPublications(searcher, searchText) == (
                        return {publication |
                                publication in set dunion {elems user.getPublications(searcher) | user in set
users & user <> searcher}
                                & publication.contentContains(searchText)}
                )
                pre searcher in set users
                post forall publication in set RESULT & exists i,j in set inds publication.getContent() &
publication.getContent()(i,...,j) = searchText;

        -- Get a user's timeline from another user's perspective
        -- IN searcher - user that is looking up timeline
        -- IN user - user whose timeline is being checked
        -- OUT user timeline (i.e. all publications, sorted by timestamp, that searcher user has access to)
        pure public getUserTimeline: User * User ==> seq of Publication
                        getUserTimeline(searcher, user) == return user.getPublications(searcher)
                        pre {user, searcher} subset users;

        -- Generated main feed for a user.
        -- This involves retrieving all publications that are considered to be relevant for the user (and, of
course, he has access to)
        -- (i.e. publications from friends or that friends have liked)
        -- Then the publications are sorted according to a score that privileges likes by friends and author
friendship
        -- IN user - user to generate feed for
        -- OUT sequence of publications, sorted by score - the feed
        public getUserFeed: User ==> seq of Publication
                        getUserFeed(user) == (
                                -- Retrieve all (visible) publications that either 1) are from a friend or 2) one or
more friends have liked
                                dcl allPublications: set of Publication := {publication |


                publication in set dunion {elems getUserTimeline(user, member) | member in set users &
user <> member}


                & publication.getAuthor() in set user.friends or card (publication.getLikes() inter user.friends)
> 0
```

```
};

                -- Calculate each publication's score
                dcl seqOfPubs: seq of PublicationRank := [];
                for all publication in set allPublications do
                        seqOfPubs := seqOfPubs ^ [mk_PublicationRank(publication,
publication.calculateScore(user))];

                -- Sort and present result
                return sortPublications(seqOfPubs);
        )
        pre user in set users
        post forall i in set inds RESULT & RESULT(i).userHasPermissions(user); -- User has
access to all publications recommended

    -- Retrieves common friends between two users
    protected getCommonFriends: User * User ==> set of User
        getCommonFriends(user1, user2) == return user1.friends inter user2.friends
        pre {user1, user2} subset users and user1 <> user2
        post RESULT subset user1.friends and RESULT subset user2.friends;

    -- Retrieves friend suggestions for a user
    -- This involves calculating the user's friends of friends,
    -- and sorting them based on who has the most friends in common
    -- with the target user. Returns the top 5
    -- IN user - user to generate friend suggestions for
    -- OUT sequence of user, sorted by # of friends in common - the suggestions
    public getFriendSuggestions: User ==> seq of User
        getFriendSuggestions(user) == (
                dcl unsortedSuggestions : set of User := user.getFriendsOfFriends() \
user.friends;

                dcl scoredSuggestions : seq of UserRank := [];
                dcl sortedSuggestions : seq of User := [];

                for all suggestion in set unsortedSuggestions do
                        scoredSuggestions := [mk_UserRank(suggestion, card
getCommonFriends(user, suggestion))] ^ scoredSuggestions;

                sortedSuggestions := sortSuggestions(scoredSuggestions);

                return sortedSuggestions(1,...,5)
        )
        pre user in set users
        post elems RESULT inter user.friends = {} and elems RESULT inter
user.getBlockedUsers() = {};

    -- Sorts publications using bubble sort
    protected sortPublications: seq of PublicationRank ==> seq of Publication
        sortPublications(publications) == (
                dcl sorted_list: seq of PublicationRank := publications;
                for i = len publications to 1 by -1 do
                        for j = 1 to i-1 do
                                if comparePublications(sorted_list(j), sorted_list(j+1)) then
                                        (dcl temp: PublicationRank := sorted_list(j);
                                        sorted_list(j) := sorted_list(j+1);
                                        sorted_list(j+1) := temp
```

```
                                        );
                                return [sorted_list(i).publication | i in set inds sorted_list];
                                );

        -- Sorts user suggestions using bubble sort
        protected sortSuggestions: seq of UserRank ==> seq of User
                sortSuggestions(suggestions) == (
                        dcl sorted_list: seq of UserRank := suggestions;
                        for i = len suggestions to 1 by -1 do
                                for j = 1 to i-1 do
                                        if compareUsers(sorted_list(j), sorted_list(j+1)) then
                                                (dcl temp: UserRank := sorted_list(j);
                                                sorted_list(j) := sorted_list(j+1);
                                                sorted_list(j+1) := temp
                                                );
                        return [sorted_list(i).user | i in set inds sorted_list];
                        );
functions
        -- Compare 2 publications based on their score and timestamp for tie break. Used in sorting
publications for main feed
        -- IN p1 first publication
        -- IN p2 second publication
        -- OUT true if first publication is to come after, false otherwise
        protected comparePublications: PublicationRank * PublicationRank +> bool
                comparePublications(p1, p2) == (
                        p1.score < p2.score or p1.score = p2.score and p1.publication.getTimestamp() <
p2.publication.getTimestamp()
                );

        -- Compare 2 users based on their score and name length for tie break. Used in sorting people for
friend suggestions
        -- IN u1 first user
        -- IN u2 second user
        -- OUT true if first user comes first, false otherwise
        protected compareUsers: UserRank * UserRank +> bool
                compareUsers(u1, u2) == (
                        u1.score < u2.score or u1.score = u2.score and u1.user.getNameLength() <
u2.user.getNameLength()
                );
traces
end Facebook
```

Function or operation	Line	Coverage	Calls
Facebook	21	100.0%	13
addFriendship	82	100.0%	43
addUser	65	100.0%	3
addUsers	74	100.0%	11
clearInstance	33	100.0%	12
comparePublications	247	100.0%	18
compareUsers	256	100.0%	3
getAllPublications	110	100.0%	26
getCommonFriends	187	100.0%	5
getFriendSuggestions	198	100.0%	3
getInstance	29	100.0%	16
getUserByName	49	100.0%	2
getUserFeed	167	100.0%	2
getUserTimeline	157	100.0%	203
getUsers	42	100.0%	3
likePublication	116	100.0%	11
removeFriendship	95	100.0%	3
searchPublications	144	100.0%	6
searchUser	134	100.0%	6
sortPublications	216	100.0%	3
sortSuggestions	230	100.0%	3
unlikePublication	125	100.0%	9
Facebook.vdmpp		100.0%	404

GroupChat

```
-- Class representing a private group conversation between users
class GroupChat
types
        -- String type for chat name
  public String = seq1 of char;
  public Date = Date`Date;
values
instance variables
        -- Chat name to uniquely identify conversations user is in (check User class)
  protected name: String;
  protected members: set of User;
  protected messages: seq of ChatMessage;

  -- Messages are ordered by timestamp
  inv forall i1, i2 in set inds messages & i1 < i2 => messages(i1).getTimestamp() <=
messages(i2).getTimestamp();
```

```
operations
        -- Constructor
  public GroupChat: String * set1 of User ==> GroupChat
    GroupChat(GroupChatName, GroupChatStartingMembers) == (
      name := GroupChatName;
      members := GroupChatStartingMembers;
      messages := [];
      return self;
    )
    pre GroupChatName <> ""
    post name = GroupChatName and members = GroupChatStartingMembers and messages = [];

        -- Getter for chat name
        pure public getName: () ==> String
                        getName() == return name
                        post RESULT = name;

        -- Getter for chat members
        pure public getMembers: () ==> set of User
                        getMembers() == return members;

        -- Add a user to the group chat
        -- IN user - user to be added
  public addMember: User ==> ()
    addMember(user) == (
      members := members union {user}
    )
    pre user not in set members
    post members = members~ union {user};

        -- Remove a user that is currently in the chat (guaranteed by precondition)
        -- IN user - user to be removed
  public removeMember: User ==> ()
    removeMember(user) == (
      members := members \ {user}
    )
    pre user in set members
    post members = members~ \ {user};

        -- Send a message to the chat. Precondition guarantees message's author is a chat member at that
time
        -- IN message - new message
  public sendMessage: ChatMessage ==> ()
    sendMessage(message) == (
      messages := messages ^ [message];
    )
    pre message.getAuthor() in set members
    post messages = messages~ ^ [message];

  -- Get chat messages. Optionally, filter by start and/or end date
  -- IN [startdate] - Optional filter start date, defaults to 1-1-1 if not given
  -- IN [enddate] - Optional filter end date, defaults to 31-12-9999 if not given
  -- OUT chat messages in given period
  pure public getMessagesBetween: [Date] * [Date] ==> seq of ChatMessage
    getMessagesBetween(startDate, endDate) == (
        return [messages(i) |
                                                i in set inds messages &
```

```
                                        messages(i).getTimestamp() >= (if startDate <> nil then
startDate else Date`minimumDate) and
                                        messages(i).getTimestamp() <= (if endDate <> nil then
endDate else Date`maximumDate)
                                    ]
    )
    pre startDate <> nil and endDate <> nil => startDate <= endDate and startDate >= Date`minimumDate
and endDate <= Date`maximumDate;

        -- Get chat messages that contain a search query text
        -- IN searchText - search query
        -- OUT chat messages that contain search text
        pure public getMessagesWithText: String ==> seq of ChatMessage
                getMessagesWithText(searchText) == (
                        return [messages(i) |
                                i in set inds messages &
                                messages(i).contentContains(searchText)
                        ]
                );

        -- Retrieve a specific chat message by its unique ID
        -- IN messageId - unique identifier (guaranteed valid by precondition)
        -- OUT respective chat message
        pure public getMessageById: nat1 ==> ChatMessage
                getMessageById(messageId) == (
                        return iota message in set elems messages & message.getId() = messageId;
                )
                pre exists1 index in set inds messages & messages(index).getId() = messageId;
functions
traces
end GroupChat
```

Function or operation	Line	Coverage	Calls
GroupChat	19	100.0%	3
addMember	40	100.0%	2
getMembers	35	100.0%	359
getMessageById	93	100.0%	1736
getMessagesBetween	69	100.0%	6
getMessagesWithText	82	100.0%	2
getName	30	100.0%	29
removeMember	49	100.0%	2
sendMessage	58	100.0%	19
GroupChat.vdmpp		100.0%	2158

Message

```
--Base Message class. Chat messages and publications derive from it
class Message
types
```

```
        -- String abstraction for the message's content
  public String = seq1 of char;
  public Date = Date`Date;
values
instance variables
  protected content: String;
  protected author: User;
  protected timestamp: Date;

operations
        -- Author getter
  pure public getAuthor: () ==> User
     getAuthor() == return author
     post RESULT = author;

  -- Timestamp (date) getter
  pure public getTimestamp: () ==> Date
     getTimestamp() == return timestamp
     post RESULT = timestamp;

  -- Content getter
  pure public getContent: () ==> String
                getContent() == return content
                post RESULT = content;

        -- Checks if the message's content contains a given string
        -- IN string - string to search in content
        -- OUT true if content contains string, false otherwise
        pure public contentContains: String ==> bool
                contentContains(string) == (
                        dcl msgContent: seq of char := content;

                        while len msgContent >= len string do (
                                if msgContent(1,...,len string) = string
                                then return true
                                else msgContent := tl msgContent
                        );

                        return false
                )

functions
traces
end Message
```

Function or operation	Line	Coverage	Calls
contentContains	32	100.0%	1038
getAuthor	15	100.0%	536
getContent	25	100.0%	3914
getTimestamp	20	100.0%	4153
Message.vdmpp		100.0%	9641

Publication

```
-- Class representing a user publication to his timeline
class Publication is subclass of Message
types
  -- Publication visibility permissions (note that, in any case, author's blocked users do not have access):
  -- Public - Anyone can see
  -- Friends - Only author's friends can see
  -- FriendsOfFriends - Only author's friends and their friends can see
  -- TransitiveConnection - Anyone with some eventual connection to the author (i.e. friends set transitive
closure) can see
  public Permissions = <Public> | <Friends> | <FriendsOfFriends> | <TransitiveConnection>
values
instance variables
        -- Unique ID to identify publication
        protected static idCounter: nat1 := 1;
        protected id: nat1 := idCounter;
        -- Users that have liked publication
  protected likes: set of User;
  protected permissions: Permissions;

operations
        -- Constructor
  public Publication: User * String * Date * Permissions ==> Publication
    Publication(PublicationAuthor, PublicationContent, PublicationTimestamp, PublicationPermissions) == (
        idCounter := idCounter + 1;
        author := PublicationAuthor;
      content := PublicationContent;
      timestamp := PublicationTimestamp;
        permissions := PublicationPermissions;
        likes := {};
        return self;
    )
    pre PublicationContent <> ""
    post idCounter = idCounter~ + 1 and author = PublicationAuthor and content = PublicationContent and
timestamp = PublicationTimestamp and likes = {} and permissions = PublicationPermissions;

  -- Unique ID counter reset
  public static resetIDCounter: () ==> ()
                resetIDCounter() == (
                        idCounter := 1;
                );

  -- ID getter
  pure public getId: () ==> nat1
                getId() == return id
                post RESULT = id;

  -- Permissions getter
  pure public getPermissions: () ==> Permissions
                getPermissions() == return permissions
                post RESULT = permissions;

  -- Update publication's access permitions
  -- IN newPermissions - new publication permission settings
  public updatePermissions: Permissions ==> ()
```

```
                updatePermissions(newPermissions) == (
                        permissions := newPermissions;
                )

                post permissions = newPermissions;

  -- Checks if a user has access to this publication
  -- IN user - user to be checked
  -- OUT true if user has access to publication, false otherwise
  pure public userHasPermissions: User ==> bool
                userHasPermissions(user) == (
                        cases permissions:
                        <Public> -> return user not in set author.getBlockedUsers(),
                        <Friends> -> return user in set author.getFriends() union {author} \
author.getBlockedUsers(),
                        <FriendsOfFriends> -> return user in set author.getFriendsOfFriends() union {author}
\ author.getBlockedUsers(),
                        others -> return user in set author.getFriendsTransitiveClosure() union {author} \
author.getBlockedUsers() -- permissions = <TransitiveConnection>
                        end
                )
                pre user in set Facebook`getInstance().getUsers();

  -- Getter for users that liked publication
  pure public getLikes: () ==> set of User
                getLikes() == return likes
                post RESULT = likes;

  -- Register that a user (that has access to the publication - guaranteed by precondition) liked this publication
  -- IN user - user that liked publication
  public like: User ==> ()
                like(user) == (
                        likes := likes union {user}
                )
                pre userHasPermissions(user)
                post user in set likes and card likes = card likes~ + 1;

        -- Register that a user that previously liked this publication (guaranteed by precondition) no longer
does so
        public removeLike: User ==> ()
                        removeLike(user) == (
                                likes := likes \ {user}
                        )
                        pre user in set likes
                        post user not in set likes and card likes = card likes~ - 1;

        -- Calculate a publication's score in regards to a user to assess its relevance for a main feed position
        -- IN user - user to generate feed for
        -- OUT publication score
        public calculateScore: User ==> rat
                        calculateScore(user) == (
                                -- Criteria: Number of likes, boosted by number of likes from friends and
possibly fact that author is a friend
                                return (1 + card likes) * (1 + card (likes inter user.friends)) * (if user in set
author.friends then 2 else 1);
                        )
                        pre user <> author
                        post RESULT >= 0;
```

```
functions
traces
end Publication
```

Function or operation	Line	Coverage	Calls
Publication	21	100.0%	24
calculateScore	96	100.0%	7
getId	41	100.0%	342
getLikes	72	100.0%	11
getPermissions	46	100.0%	3
like	78	100.0%	11
removeLike	86	100.0%	2
resetIDCounter	35	100.0%	12
updatePermissions	52	100.0%	3
userHasPermissions	61	100.0%	44
Publication.vdmpp		100.0%	459

User

```
class User
types
        -- String type used for names
  public String = seq1 of char;
values
instance variables
  protected user: User; --Keep a self reference for invariant usage (self is not recognised in that scope)
  protected name: String;
  public friends: set of User;
  protected blocked: set of User;
  protected posts: seq of Publication;
  protected chats: inmap String to GroupChat; -- chat_name |-> chat object

  -- No self friendship nor self blocking
  inv user not in set friends and user not in set blocked;

  -- Friendship is bidirectional
  inv forall friend in set friends & user in set friend.friends;

  -- No friends can on the blocked list
  inv blocked inter friends = {};

  -- All of user's posts must be of his authorship
  -- Same as forall ... getAuthor=user
  inv not exists publication in seq posts & publication.getAuthor() <> user;

  -- User is a member in all of the group chats he has stored
  inv forall chat in set rng chats & user in set chat.getMembers();
operations
        -- Constructor
```

```
  public User: String ==> User
     User(userName) == (
       name := userName;
       friends := {};
       blocked := {};
       posts := [];
       chats := {|->};
       user := self;
       return self;
     )
     pre userName <> ""
     post name = userName and friends = {} and posts = [];

  -- Name getter
  pure public getName: () ==> String
     getName() == return name
     post RESULT = name;

    pure public getNameLength: () ==> nat
     getNameLength() == return len name
     post RESULT = len name;

  -- Group chats getter
  pure public getChats: () ==> set of GroupChat
                  getChats() == return rng chats
                  post forall chat in set RESULT & chat.getName() in set dom chats;

  -- Specific group chat getter
  -- IN chat name
  pure public getChatByName: String ==> GroupChat
                  getChatByName(chatName) == return chats(chatName)
                  pre chatName in set dom chats;

  -- Publications getter from another user's perspective (i.e. retrieves publications that given user has access
to)
  -- IN searcher - user to assume perspective from
  -- OUT publications visible to given user
  pure public getPublications: User ==> seq of Publication
                  getPublications(searcher) == (
                           return [posts(i) | i in set inds posts & posts(i).userHasPermissions(searcher)]
                  );

          -- Friends getter
  pure public getFriends: () ==> set of User
     getFriends() == return friends
     post RESULT = friends;

  -- Blocked users getter
  pure public getBlockedUsers: () ==> set of User
                  getBlockedUsers() == return blocked
                  post RESULT = blocked;

  -- Friends of friends getter (user's friends and all of their friends)
  pure public getFriendsOfFriends: () ==> set of User
                  getFriendsOfFriends() == (
                           -- Union of friends set with every other friends set from user's friends. Then, exclude
itself
```

```
                        return {people | people in set (friends union dunion {friend.friends | friend in set
friends} \ {user}) }
                    )
                    post friends subset RESULT and card RESULT >= card friends;

    -- Transitive connection (all people that user can reach through friend connections)
    pure public getFriendsTransitiveClosure: () ==> set of User
                        getFriendsTransitiveClosure() ==  (
                            dcl closure : set of User := friends;
                            dcl visited : set of User := {};
                            while visited <> closure do
                                let t in set (closure \ visited) in (
                                closure := closure union t.friends;
                                visited := visited union {t}
                                );
                            return closure \ {user}
                        )
                        post friends subset RESULT and card RESULT >= card friends;

        -- Checks if user's name contains a search query text
        -- IN string - search query
        -- OUT true if name contains search text, false otherwise
        public nameContains: String ==> bool
                        nameContains(string) == (
                            --seq of char instead of string because, if not found, can eventually become
empty string
                            --and String is defined as seq1 of char
                            dcl userName: seq of char := name;

                            while userName <> "" do (
                                if userName(1,...,len string) = string
                                then return true
                                else userName := tl userName
                            );

                            return false
                        );

        -- Make a new publication to user's timeline
        -- IN content - publication message
        -- IN timestamp - publication timestamp
        -- IN permissions - access permissions
        public makePublication: Publication`String * Publication`Date * Publication`Permissions ==> ()
                        makePublication(content, timestamp, permissions) == (
                            posts := [new Publication(self, content, timestamp, permissions)] ^ posts
                        )
                        pre len posts > 0 => timestamp >= posts(1).getTimestamp()
                        post len posts = len posts~ + 1;

        -- Update a publication's access permissions
        -- IN publicationId - publication unique identifier (guaranteed valid by precondition)
        -- IN newPermissions - new access permissions
        public updatePublicationPermissions: nat1 * Publication`Permissions ==> ()
                        updatePublicationPermissions(publicationId, newPermissions) == (
                            let index = iota i in set inds posts & posts(i).getId() = publicationId in
posts(index).updatePermissions(newPermissions)
                        )
                        pre exists1 index in set inds posts & posts(index).getId() = publicationId;
```

```
	-- Retrieve a publication through its unique ID
	-- IN publicationId - publication unique identifier (guaranteed valid by precondition)
	-- OUT publication
	public getPublicationById: nat1 ==> Publication
			getPublicationById(publicationId) == (
				return iota publication in set elems posts & publication.getId() = publicationId;
			)
			pre exists1 index in set inds posts & posts(index).getId() = publicationId;

	-- Delete a publication
	-- IN publicationID - publication unique identifier (guaranteed valid by precondition)
	public deletePublication: nat1 ==> ()
			deletePublication(publicationId) == (
				posts := [posts(i) | i in set inds posts & posts(i).getId() <> publicationId]
			)
			pre exists1 index in set inds posts & posts(index).getId() = publicationId
			post not exists index in set inds posts & posts(index).getId() = publicationId;

	-- Create a new group chat
	-- IN chatName - new group chat's name
	-- IN initialMembers - friends that user intends to have a conversation with
	public createGroupChat: String * set of User ==> ()
			createGroupChat(chatName, initialMembers) == (
				--let expression so group chat reference is the same for all users
				let newChat = new GroupChat(chatName, initialMembers union {self}) in (
					chats := chats munion {chatName |-> newChat};
					for all member in set initialMembers do
						member.addGroupChat(newChat);
					)
			)
			pre chatName not in set dom chats and initialMembers subset friends
			post chatName in set dom chats;

	-- Add a friend to the chat
	-- IN friend - friend to add
	-- IN chatName - chat to add friend to
	public addFriendToChat: User * String ==> ()
			addFriendToChat(friend, chatName) == (
				friend.addGroupChat(chats(chatName))
			)
			pre friend in set friends and chatName in set dom chats and not exists cName in set
{chat.getName() | chat in set friend.getChats()} & cName = chatName
			post exists1 cName in set {chat.getName() | chat in set friend.getChats()} & cName
= chatName;

	-- When user is added to an existing chat (hence, protected) - used in two previous operations
	-- IN chat - group chat he was added to
	protected addGroupChat: GroupChat ==> ()
			addGroupChat(chat) == (
				if self not in set chat.getMembers() then
					chat.addMember(self);
				chats := chats munion {chat.getName() |-> chat};
			)
			pre chat.getName() not in set dom chats
			post chat.getName() in set dom chats;

	-- Leave a group chat
```

```
	-- IN chatName - group chat's name
	public leaveGroupChat: String ==> ()
			leaveGroupChat(chatName) == (
				dcl g: GroupChat := chats(chatName);
				chats := {chatName} <-: chats;
				g.removeMember(self);
			)
			pre chatName in set dom chats
			post chatName not in set dom chats;

	-- Send a message to a group chat
	-- IN chatName - chat to send message to
	-- IN content - message content
	-- IN timestamp - message timestamp
	public sendChatMessage: String * ChatMessage`String * ChatMessage`Date ==> ()
			sendChatMessage(chatName, content, timestamp) == (
				chats(chatName).sendMessage(new ChatMessage(user, content,
timestamp));
			)
			pre chatName in set dom chats;

	-- Search for a message in a group chat
	-- IN chatName - chat to search in
	-- IN searchText - text pattern to search
	-- OUT messages from chat that include search pattern
	public searchTextInChat: String * String ==> seq of ChatMessage
			searchTextInChat(chatName, searchText) == (
				return chats(chatName).getMessagesWithText(searchText);
			)
			pre chatName in set dom chats;

	-- Block a user (that cannot be a friend - guaranteed by invariant)
	-- IN userBlocked - user to be blocked
	public blockUser: User ==> ()
			blockUser(userBlocked) == (
				blocked := blocked union {userBlocked};
			)
			pre userBlocked in set Facebook`getInstance().getUsers()
			post card blocked = card blocked~ + 1;

	-- Unblock a previously blocked user (guaranteed by precondition)
	-- IN userBlocked - user to be unblocked
	public unblockUser: User ==> ()
			unblockUser(userBlocked) == (
				blocked := blocked \ {userBlocked};
			)
			pre userBlocked in set blocked;

functions
traces
end User
```

Function or operation	Line	Coverage	Calls
User	31	100.0%	114
addFriendToChat	178	100.0%	2
addGroupChat	187	100.0%	7
blockUser	229	100.0%	8
createGroupChat	163	100.0%	3
deletePublication	153	100.0%	1
getBlockedUsers	78	100.0%	332
getChatByName	60	100.0%	12
getChats	54	100.0%	5
getFriends	73	100.0%	84
getFriendsOfFriends	83	100.0%	49
getFriendsTransitiveClosure	91	100.0%	49
getName	45	100.0%	23322
getNameLength	49	100.0%	2
getPublicationById	145	100.0%	98
getPublications	67	100.0%	239
leaveGroupChat	198	100.0%	2
makePublication	126	100.0%	24
nameContains	107	100.0%	48
searchTextInChat	221	100.0%	4
sendChatMessage	211	100.0%	19
unblockUser	237	100.0%	3
updatePublicationPermissions	136	100.0%	3
User.vdmpp		100.0%	24430

4. Model Validation

MyTest

```
class MyTest
types
values
instance variables
	public facebook: Facebook := Facebook`getInstance();
	public u1 : User;
	public u2 : User;
	public u3 : User;
	public u4 : User;
	public u5 : User;
```

```
        public u6 : User;
        public u7 : User;
        public u8 : User;
        public u9 : User;
operations
                protected setupUsers: () ==> ()
                setupUsers() == (
                        u1 := new User("user1");
                        u2 := new User("user2");
                        u3 := new User("user3");
                        u4 := new User("user4");
                        u5 := new User("user5");
                        u6 := new User("user6");
                        u7 := new User("user7");
                        u8 := new User("user8");
                        u9 := new User("user9");
                );

                protected assertTrue: bool ==> ()
                assertTrue(arg) ==
                        return
                pre arg;

                protected assertEqual: ? * ? ==> ()
                assertEqual(expected, actual) ==
                        if expected <> actual then (
                                IO`print("Actual value (");
                                IO`print(actual);
                                IO`print(") different from expected (");
                                IO`print(expected);
                                IO`println(")\n")
                        )
        post expected = actual
traces
end MyTest
```

TestFacebook

```
class TestFacebook is subclass of MyTest

instance variables
operations
        -- Test scenario where users are added into the platform (registration)
        -- Covers the respective scenario described in section 2.1 and requirement R1
        protected testRegistration: () ==> ()
                testRegistration() == (
                        facebook := Facebook`clearInstance();
                        setupUsers();

                        -- add 1 user
                        facebook.addUser(u1);
                        facebook.addUser(u2);
                        assertTrue(card facebook.getUsers() = 2);

                        -- add multiple users
                        facebook.addUsers({u3,u4,u5});
                        assertTrue(card facebook.getUsers() = 5);

                        assertTrue({u1,u2,u3,u4,u5} = facebook.getUsers());

                        -- check existing and non existing users
                        assertEqual(facebook.getUserByName("user1"), u1);
                        assertEqual(facebook.getUserByName("user6"), nil);

                        -- failing code (trying to register a new user with a name that's already taken)
                        --facebook.addUser(new User("user1"));
                );

        -- Test scenario where users are looked up by their names
        -- Covers requirement R2
        protected testUserSearch: () ==> ()
                testUserSearch() == (
                        -- create some users with more sophisticated names
                        dcl ua: User := new User("joseph");
                        dcl ub: User := new User("john");
                        dcl uc: User := new User("amanda");
                        dcl ud: User := new User("anna");
                        dcl ue: User := new User("username6");
                        facebook := Facebook`clearInstance();
                        setupUsers();

                        -- add some users to the platform
                        facebook.addUsers({u1,u2,u3,ua,ub,uc,ud,ue});

                        assertTrue(card facebook.searchUser("user") = 4);
                        assertEqual(facebook.searchUser("user"), {u1,u2,u3,ue});

                        assertTrue(card facebook.searchUser("jo") = 2);
                        assertEqual(facebook.searchUser("jo"), {ua,ub});

                        assertTrue(ud in set facebook.searchUser("ann"));
                        assertTrue(uc not in set facebook.searchUser("ann"));
```

```
		);

	-- Test scenario where a user checks his main feed to assess the most relevant content for
him
	-- Covers the respective scenario described in section 2.1 and requirement R12
	protected testFeed: () ==> ()
		testFeed() == (
			facebook := Facebook`clearInstance();
			setupUsers();


			facebook.addUsers({u1,u2,u3,u4,u5,u6,u7});

			-- set up some friendships
			facebook.addFriendship(u1,u2);
			facebook.addFriendship(u1,u3);
			facebook.addFriendship(u2,u4);
			facebook.addFriendship(u4,u5);

			-- create publications
			u1.makePublication("greetings, eager young minds",
Date`makeDate(2016,2,29), <TransitiveConnection>); --ID 1 | Will show up to u2,u3 (friends), u4 (friend liked)
			u1.makePublication("greetings, friends", Date`makeDate(2016,12,13),
<Friends>); --ID 2 | Will show up to u2,u3 (friends); u2 liked but will not show up to u4 because it is friends
only
			u2.makePublication("fear me, friends, for i am back",
Date`makeDate(2016,2,29), <Public>); --ID 3 | Will show up to u1, u4 (friends), u3 (friend liked)
			u2.makePublication("back, looking for new friendships",
Date`makeDate(2017,01,01), <FriendsOfFriends>); --ID 4 | Will show up to u4 (friend), u5 (friend liked)
			u3.makePublication("what do i need friends for? popularity, of course",
Date`makeDate(2018,01,13), <Friends>); --ID 5 | Will show up to u1 (friend)
			u3.makePublication("what's up my dudes it's 2019",
Date`makeDate(2019,1,1), <Public>); -- ID 6 | Will show up to u1 (friend), u4 (friend liked)

			-- add some likes
			facebook.likePublication(u2, 1);
			facebook.likePublication(u3, 1);
			facebook.likePublication(u2, 2);
			facebook.likePublication(u1, 3);
			facebook.likePublication(u4, 4);
			facebook.likePublication(u5, 6);

			-- Recall score formula: (1+#likes)*(1+#likesFromFriends)*(if authorIsFriend 2
else 1)

			-- Feed from u3's perspective
			-- Publication 1's score: (1+2)*(1+1)*(2) = 12
			-- Publication 2's score: (1+1)*(1+0)*(2) = 4
			-- Publication 3's score: (1+1)*(1+1)*(1) = 4
			-- Order then is 1, 2, 3 (2 has a more recent timestamp than 3)
			assertEqual(facebook.getUserFeed(u3), [u1.getPublicationById(1),
u1.getPublicationById(2), u2.getPublicationById(3)]);

			-- Feed from u4's perspective
			-- Publication 1's score: (1+2)*(1+1)*(1) = 6
			-- Publication 3's score: (1+1)*(1+0)*(2) = 4
			-- Publication 4's score: (1+1)*(1+0)*(2) = 4
			-- Publication 6's score: (1+1)*(1+1)*(0) = 4
```

```
                -- Order is then 1, 6, 4, 3 (regarding timestamps, 6 > 4 > 3)
                assertEqual(facebook.getUserFeed(u4), [u1.getPublicationById(1),
u3.getPublicationById(6), u2.getPublicationById(4), u2.getPublicationById(3)]);
            );

        -- Test scenario where users receive potential friend suggestions
        -- Covers requirement R10
        protected testFriendSuggestions: () ==> ()
            testFriendSuggestions() == (
                dcl userSeq: seq of User;
                dcl userWithLongerName: User := new User("looooongname");
                facebook := Facebook`clearInstance();
                setupUsers();

                facebook.addUsers({u1, u2, u3, u4, u5, u6, u7, u8, u9,
userWithLongerName});

                -- Testing general case
                -- u1 has 3 friends: u2, u3, and u4
                facebook.addFriendship(u1, u2);
                facebook.addFriendship(u1, u3);
                facebook.addFriendship(u1, u4);

                -- u5 has 2 friends: u3, and u4
                facebook.addFriendship(u5, u3);
                facebook.addFriendship(u5, u4);

                facebook.addFriendship(userWithLongerName, u3);
                facebook.addFriendship(u7, u2);
                facebook.addFriendship(u7, u8);

                userSeq := facebook.getFriendSuggestions(u1);
                assertEqual(3, len userSeq);
                assertEqual([u5,userWithLongerName,u7], userSeq); --u5 has 2 common
friends (u3, u4), others have 1

                userSeq := facebook.getFriendSuggestions(u5);
                assertEqual(2, len userSeq);
                assertEqual([u1, userWithLongerName], userSeq); --u1 has 2 common
friends (u3, u4), userWithLongerName has 1

                -- Testing case where target user has no friends
                userSeq := facebook.getFriendSuggestions(u9);
                assertEqual(0, len userSeq);
        );


    public Run: () ==> ()
            Run() == (
                IO`println("Running Facebook Tests\n");
                testRegistration();
                testUserSearch();
                testFeed();
                testFriendSuggestions();
                IO`println("Facebook Tests ran successfully\n");
            );

end TestFacebook
```

TestGroupChat

```
class TestGroupChat is subclass of MyTest

instance variables
operations
	-- Test scenario where users create chats with other users and communicate in it
	-- Covers the respective scenario described in section 2.1 and requirement R8
	protected testGroupChatCommunication: () ==> ()
		testGroupChatCommunication() == (
			facebook := Facebook`clearInstance();
			setupUsers();

			facebook.addUsers({u1,u2,u3,u4});
			-- setup some friendships
			facebook.addFriendship(u1,u2);
			facebook.addFriendship(u1,u3);
			facebook.addFriendship(u1,u4);
			facebook.addFriendship(u3,u4);

			u1.createGroupChat("i hate eclipse", {u2,u3});
			u4.createGroupChat("by myself", {u3});
			assertEqual(u1.getChatByName("i hate eclipse").getMembers(), {u1,u2,u3}); --3
starting members
			assertTrue(u1.getChatByName("i hate eclipse") = u2.getChatByName("i hate
eclipse") and u2.getChatByName("i hate eclipse") = u3.getChatByName("i hate eclipse")); --same chat for
every member

			u1.sendChatMessage("i hate eclipse", "hey guys, we have to do MFES",
Date`makeDate(2018,12,22));
			u3.sendChatMessage("i hate eclipse", "well user4 is missing, so let me add him",
Date`makeDate(2018,12,22));
			assertEqual(len u1.getChatByName("i hate eclipse").getMessagesBetween(nil, nil),
2); --2 messages in chat

			u3.addFriendToChat(u4, "i hate eclipse");
			assertTrue(u4 in set u3.getChatByName("i hate eclipse").getMembers());
			assertEqual(card u3.getChatByName("i hate eclipse").getMembers(), 4); --4 members
now

			u3.sendChatMessage("i hate eclipse", "hey user4, MFES man, gotta do it",
Date`makeDate(2018,12,23));
			u4.sendChatMessage("i hate eclipse", "nope, GL", Date`makeDate(2018,12,25));
			u2.sendChatMessage("i hate eclipse", "fine, then i won't do anything too",
Date`makeDate(2018,12,26));
			u2.leaveGroupChat("i hate eclipse");

			assertEqual(len u1.getChatByName("i hate eclipse").getMessagesBetween(nil, nil),
5); --5 messages in chat now
			assertEqual(card u1.getChatByName("i hate eclipse").getMembers(), 3);
			assertTrue(u2 not in set u1.getChatByName("i hate eclipse").getMembers()); --u2 left
the chat
			assertEqual(card u2.getChats(), 0); --u2 is no longer in any group chat

			-- failing code (user tries to add a non-friend to a group chat)
```

```
                --u1.addFriendToChat(u5, "i hate eclipse");

                -- failing code (user tries to send message to a chat he's not in)
                --u1.sendChatMessage("by myself", "hello", Date`makeDate(2019,12,31));
        );

    -- Test scenario where chat messages are looked up by their content or filtered by their date
    -- Covers requirement R9
    protected testGroupChatSearchAndFilter: () ==> ()
        testGroupChatSearchAndFilter() == (
            dcl chat: GroupChat;
            dcl expected: seq of ChatMessage;
            dcl actual: seq of ChatMessage;
            facebook := Facebook`clearInstance();
            setupUsers();

            facebook.addUsers({u1,u2,u3,u4});
            -- setup some friendships
            facebook.addFriendship(u1,u2);
            facebook.addFriendship(u1,u3);
            facebook.addFriendship(u1,u4);
            facebook.addFriendship(u3,u4);

            u1.createGroupChat("superior users", {u2,u3});
            chat := u1.getChatByName("superior users");

            -- Small conversation
            u1.sendChatMessage("superior users", "hey guys, let's play something",
Date`makeDate(2018,12,22)); --ID 1
            u3.sendChatMessage("superior users", "user4 loves to game, let me add him",
Date`makeDate(2018,12,22)); --ID 2
            u3.addFriendToChat(u4, "superior users");
            u3.sendChatMessage("superior users", "hey 4 let's game",
Date`makeDate(2018,12,22)); --ID 3
            u4.sendChatMessage("superior users", "meh i don't feel like gaming rn",
Date`makeDate(2018,12,23)); --ID 4
            u1.sendChatMessage("superior users", "stop being so negative, you love games no
matter what", Date`makeDate(2018,12,24)); --ID 5
            u4.sendChatMessage("superior users", "well sure but i have to study",
Date`makeDate(2018,12,25)); --ID 6
            u3.sendChatMessage("superior users", "pff come on you always get straight As",
Date`makeDate(2018,12,26)); --ID 7
            u2.sendChatMessage("superior users", "yeah, besides exams are like a month away,
studying now makes no sense!", Date`makeDate(2018,12,28)); --ID 8
            u4.sendChatMessage("superior users", "look guys i just don't want to game",
Date`makeDate(2019,1,1)); --ID 9
            u2.sendChatMessage("superior users", "ok, 1 and 3 let's game ourselves then",
Date`makeDate(2019,2,15)); --ID 10
            u1.sendChatMessage("superior users", "sure thing, just give me a second",
Date`makeDate(2019,5,5));-- ID 11
            u3.sendChatMessage("superior users", "absolutely, brb too",
Date`makeDate(2019,6,1)); --ID 12
            u4.sendChatMessage("superior users", "so what did you guys end up doing for a
game?", Date`makeDate(2019,6,2)); --ID 13
            u1.sendChatMessage("superior users", "lol nothing... let's talk later, l8r bro",
Date`makeDate(2019,6,3)); --ID 14
            u1.leaveGroupChat("superior users");
```

```
                -- Filter messages through different periods
                assertEqual(len chat.getMessagesBetween(nil, nil), 14); --providing no dates means
no filtering which simply returns all chat messages

                actual := chat.getMessagesBetween(Date`makeDate(2018,12,26), nil); --filtering all
messages from a certain date onwards
                expected :=
[chat.getMessageById(7),chat.getMessageById(8),chat.getMessageById(9),chat.getMessageById(10),chat.ge
tMessageById(11),chat.getMessageById(12),chat.getMessageById(13),chat.getMessageById(14)]; -- all
messages since 26th December 2018
                assertTrue(len actual = len expected and forall i in set inds actual & actual(i) =
expected(i));

                actual := chat.getMessagesBetween(nil, Date`makeDate(2018,12,26)); --filtering all
messages until a certain date
                expected :=
[chat.getMessageById(1),chat.getMessageById(2),chat.getMessageById(3),chat.getMessageById(4),chat.get
MessageById(5),chat.getMessageById(6),chat.getMessageById(7)]; -- all messages until 26th December
2018
                assertTrue(len actual = len expected and forall i in set inds actual & actual(i) =
expected(i));

                actual := chat.getMessagesBetween(Date`makeDate(2018,12,26),
Date`makeDate(2019,6,1)); --filtering all messages between two well defined dates
                expected :=
[chat.getMessageById(7),chat.getMessageById(8),chat.getMessageById(9),chat.getMessageById(10),chat.ge
tMessageById(11),chat.getMessageById(12)]; -- all messages since 26th December 2018 and 1st June 2019
                assertTrue(len actual = len expected and forall i in set inds actual & actual(i) =
expected(i));

                -- Filter messages through different search patterns
                actual := u2.searchTextInChat("superior users", "game");
                expected :=
[chat.getMessageById(2),chat.getMessageById(3),chat.getMessageById(5),chat.getMessageById(9),chat.get
MessageById(10),chat.getMessageById(13)];
                assertTrue(len actual = len expected and forall i in set inds actual & actual(i) =
expected(i));

                actual := u2.searchTextInChat("superior users", "let's");
                expected :=
[chat.getMessageById(1),chat.getMessageById(3),chat.getMessageById(10),chat.getMessageById(14)];
                assertTrue(len actual = len expected and forall i in set inds actual & actual(i) =
expected(i));

                -- failing code (user tries to search in chat he's not in)
                --u1.searchTextInChat("superior users", "game");
        );

    public Run: () ==> ()
                Run() == (
                        IO`println("Running GroupChat Tests\n");
                        testGroupChatCommunication();
                        testGroupChatSearchAndFilter();
                        IO`println("GroupChat Tests ran successfully\n");
                );

end TestGroupChat
```

TestPublication

```
class TestPublication is subclass of MyTest

instance variables

operations
	-- Test scenario where publications are looked up by their content
	-- Covers requirement R7
	protected testPublicationSearch: () ==> ()
		testPublicationSearch() == (
			facebook := Facebook`clearInstance();
			setupUsers();


			facebook.addUsers({u1,u2,u3,u4,u5,u6,u7});

			-- set up some friendships
			facebook.addFriendship(u1,u2);
			facebook.addFriendship(u1,u3);
			facebook.addFriendship(u2,u4);
			facebook.addFriendship(u4,u5);
			facebook.addFriendship(u5,u6);

			-- create publications
			u1.makePublication("greetings, eager young minds", Date`makeDate(2016,12,12),
<TransitiveConnection>); --ID 1
			u1.makePublication("greetings, friends", Date`makeDate(2016,12,13), <Friends>);
--ID 2
			u1.makePublication("fear me, friends, for i am back", Date`makeDate(2017,12,17),
<Public>); --ID 3
			u2.makePublication("back, looking for new friendships", Date`makeDate(2017,01,01),
<FriendsOfFriends>); --ID 4
			u3.makePublication("what do i need friends for? popularity, of course",
Date`makeDate(2018,01,13), <Friends>); --ID 5

			-- search from different user's perspectives
			assertEqual(facebook.searchPublications(u6, "greetings"),
{u1.getPublicationById(1)}); -- Publications ID 1 and 2 have the search text. 1 was found by friends' closure,
but 2 is restricted to u2's friends
			u1.blockUser(u6);
			assertEqual(facebook.searchPublications(u6, "greetings"), {}); -- Blocked by u1, so
now can't see the publication found earlier

			assertEqual(facebook.searchPublications(u5, "friends"), {u1.getPublicationById(3),
u2.getPublicationById(4)});
			u1.updatePublicationPermissions(2, <Public>);
			assertEqual(facebook.searchPublications(u5, "friends"), {u1.getPublicationById(2),
u1.getPublicationById(3), u2.getPublicationById(4)}); -- Now has ID 2 as well, since it was made public

			assertEqual(facebook.searchPublications(u3, "greetings"), {u1.getPublicationById(1),
u1.getPublicationById(2)});
			facebook.removeFriendship(u1,u3);
			assertEqual(facebook.searchPublications(u3, "greetings"),
{u1.getPublicationById(2)});
```

```
		);

	-- Test scenario where a user manages his timeline (make, update, delete publications)
	-- Covers the respective scenario described in section 2.1 and requirement R5
	protected testTimelineUpdate: () ==> ()
		testTimelineUpdate() == (
			facebook := Facebook`clearInstance();
			setupUsers();

			facebook.addUser(u1);
			-- make new publications
			u1.makePublication("my first publication", Date`makeDate(2016,12,12), <Public>);
--ID: 1
			u1.makePublication("so, how is everybody doing", Date`makeDate(2017,01,01),
<Public>); --ID: 2
			u1.makePublication("hello, but only to my friends!", Date`makeDate(2017,02,14),
<Friends>); --ID: 3

			assertEqual(u1.getPublicationById(1).getContent(), "my first publication");
			assertEqual(len facebook.getUserTimeline(u1,u1), 3); --3 posts made
			assertEqual(u1.getPublicationById(3).getPermissions(), <Friends>);

			-- update existing publications' permissions
			u1.updatePublicationPermissions(1, <FriendsOfFriends>);
			u1.updatePublicationPermissions(3, <Public>);

			assertEqual(u1.getPublicationById(1).getPermissions(), <FriendsOfFriends>);
			assertEqual(u1.getPublicationById(3).getPermissions(), <Public>);
			assertEqual(len facebook.getUserTimeline(u1,u1), 3); -- no addition or deletion, just
modification, so still 3 posts

			-- delete existing publications
			u1.deletePublication(2);
			assertEqual(u1.getPublicationById(1).getContent(), "my first publication"); -- ID 1 still
exists
			assertEqual(u1.getPublicationById(3).getContent(), "hello, but only to my friends!"); --
ID 3 still exists
			assertEqual(len facebook.getUserTimeline(u1,u1), 2); -- Only 2 posts now (the
previous ones)
		);

	-- Test scenario where a user manages likes (or removes like from) publications he has access to
	-- Covers the respective scenario described in section 2.1 and requirement R6
	protected testLikeAndUnlike: () ==> ()
		testLikeAndUnlike() == (
			facebook := Facebook`clearInstance();
			setupUsers();

			facebook.addUsers({u1,u2,u3,u4,u5,u6,u7});

			-- set up some friendships
			facebook.addFriendship(u1,u2);
			facebook.addFriendship(u1,u3);
			facebook.addFriendship(u2,u4);
			facebook.addFriendship(u4,u5);
			facebook.addFriendship(u5,u6);

			-- create publications
```

```
                u1.makePublication("greetings, eager young minds", Date`makeDate(2016,12,12),
<TransitiveConnection>); --ID 1
                u1.makePublication("greetings, friends", Date`makeDate(2016,12,13), <Friends>);
--ID 2
                u1.makePublication("fear me, friends, for i am back", Date`makeDate(2017,12,17),
<Public>); --ID 3
                u2.makePublication("back, looking for new friendships", Date`makeDate(2017,01,01),
<FriendsOfFriends>); --ID 4
                u3.makePublication("what do i need friends for? popularity, of course",
Date`makeDate(2018,01,13), <Friends>); --ID 5

                -- add some likes
                facebook.likePublication(u1, 1); --Bunch of self likes
                facebook.likePublication(u1, 2);
                facebook.likePublication(u2, 3);
                facebook.likePublication(u3, 1);
                facebook.likePublication(u6, 1); --Possible through transitive connection

                assertEqual(u1.getPublicationById(1).getLikes(), {u1,u3,u6});
                assertEqual(u1.getPublicationById(2).getLikes(), {u1});
                assertEqual(u1.getPublicationById(3).getLikes(), {u2});

                -- remove some likes
                facebook.unlikePublication(u1, 1);
                facebook.unlikePublication(u2, 3);

                assertEqual(u1.getPublicationById(1).getLikes(), {u3,u6});
                assertEqual(card u1.getPublicationById(3).getLikes(), 0);

                -- failing code (user tries to like a publication he does not have access to)
                --facebook.likePublication(u6, 2);
        );

    public Run: () ==> ()
                Run() == (
                        IO`println("Running Publication Tests\n");
                        testPublicationSearch();
                        testTimelineUpdate();
                        testLikeAndUnlike();
                        IO`println("Publication Tests ran successfully\n");
                );

end TestPublication
```

TestUser

```
class TestUser is subclass of MyTest

instance variables

operations
		-- Test scenario where users add each other to their friends list
		-- Covers the respective scenario described in section 2.1 and requirement R3
		protected testFriendship: () ==> ()
			testFriendship() == (
				facebook := Facebook`clearInstance();
				setupUsers();

				facebook.addUsers({u1,u2,u3,u4,u5,u6,u7,u8});

				-- setup friendships
				facebook.addFriendship(u1,u2);
				facebook.addFriendship(u1,u3);
				facebook.addFriendship(u1,u4);
				facebook.addFriendship(u2,u5);
				facebook.addFriendship(u3,u6);
				facebook.addFriendship(u3,u4);
				facebook.addFriendship(u4,u7);
				facebook.addFriendship(u7,u8);

				assertEqual(u1.getFriends(), {u2,u3,u4});
				assertEqual(u2.getFriends(), {u1,u5});
				assertEqual(u5.getFriends(), {u2}); --test friendship biconnection property
				assertEqual(u1.getFriendsOfFriends(), {u2,u3,u4,u5,u6,u7});
				assertEqual(u1.getFriendsTransitiveClosure(), {u2,u3,u4,u5,u6,u7,u8});

				-- remove some friendships
				facebook.removeFriendship(u1,u2);
				facebook.removeFriendship(u1,u4);

				assertEqual(u1.getFriends(), {u3});
				assertTrue(u1 not in set u2.getFriends());
				assertEqual(u2.getFriends(), {u5});
				assertEqual(u1.getFriendsOfFriends(), {u3,u4,u6});
				assertEqual(u1.getFriendsTransitiveClosure(), {u3,u4,u6,u7,u8});

				-- failing code (friendship between a user in the platform and a user not in the
platform)
				--facebook.addFriendship(u1,u9);
			);

		-- Test scenario where users add other users to their blocked users list
		-- Covers the respective scenario described in section 2.1 and requirement R4
		protected testBlocking: () ==> ()
			testBlocking() == (
				facebook := Facebook`clearInstance();
				setupUsers();

				facebook.addUsers({u1,u2,u3,u4,u5,u6,u7,u8});
```

```
		-- setup friendships
		u1.blockUser(u2);
		u1.blockUser(u3);
		u1.blockUser(u4);
		u2.blockUser(u5);
		u3.blockUser(u6);
		u3.blockUser(u4);

		assertEqual(u1.getBlockedUsers(), {u2,u3,u4});
		assertEqual(u2.getBlockedUsers(), {u5});
		assertEqual(u5.getBlockedUsers(), {}); --blocking is not necessarily
bidirectional, unlike friendship
		assertEqual(card u3.getBlockedUsers(), 2);

		-- remove some friendships
		u1.unblockUser(u2);
		u1.unblockUser(u4);
		u3.unblockUser(u4);

		assertEqual(card u1.getBlockedUsers(), 1);
		assertTrue(u2 not in set u1.getBlockedUsers());
		assertTrue(u4 not in set u1.getBlockedUsers());

		-- failing code (blocking a user that is currently a friend)
		--facebook.addFriendship(u1,u2);
		--u1.blockUser(u2);
	);

-- Test scenario where a user checks other users' timeline
-- Covers requirement R11
protected testTimeline: () ==> ()
	testTimeline() == (
		facebook := Facebook`clearInstance();
		setupUsers();


		facebook.addUsers({u1,u2,u3,u4,u5,u6,u7,u8});

		-- set up some friendships
		facebook.addFriendship(u1,u2);
		facebook.addFriendship(u1,u3);
		facebook.addFriendship(u2,u4);
		facebook.addFriendship(u4,u5);
		facebook.addFriendship(u5,u6);

		-- block a user
		u1.blockUser(u7);

		-- create publications
		u1.makePublication("greetings, eager young minds",
Date`makeDate(2016,12,12), <TransitiveConnection>); --ID 1
		u1.makePublication("greetings, friends", Date`makeDate(2016,12,13),
<Friends>); --ID 2
		u1.makePublication("fear me, friends, for i am back",
Date`makeDate(2017,12,17), <Public>); --ID 3
		u1.makePublication("back, looking for new friendships",
Date`makeDate(2018,01,01), <FriendsOfFriends>); --ID 4
```

```
                u1.makePublication("what do i need friends for? popularity, of course",
Date`makeDate(2018,01,13), <Friends>); --ID 5

                assertEqual(len facebook.getUserTimeline(u1, u1), 5); -- Author always has
full access to his timeline
                assertEqual(len facebook.getUserTimeline(u2, u1), 5); -- u2 is u1's friend so
has full access to his timeline too
                assertEqual(facebook.getUserTimeline(u4, u1),
[u1.getPublicationById(4),u1.getPublicationById(3),u1.getPublicationById(1)]); -- u4 only a friend of a friend,
so does not have access to publications 2 and 5
                assertEqual(facebook.getUserTimeline(u5, u1),
[u1.getPublicationById(3),u1.getPublicationById(1)]); -- u5 is only reachable through friends transitive closure
(u1->u2->u4->u5), so does not have access to posts 5, 4 and 2
                assertEqual(facebook.getUserTimeline(u8, u1), [u1.getPublicationById(3)]); --
u8 has no connection whatsoever to u1, so only sees public post
                assertEqual(len facebook.getUserTimeline(u7, u1), 0); -- u7 is blocked by u1,
so can't see any of his publications
            );
    public Run: () ==> ()
            Run() == (
                IO`println("Running User Tests\n");
                testFriendship();
                testBlocking();
                testTimeline();
                IO`println("User Tests ran successfully\n");
            );

end TestUser
```

MyTestRunner

```
class MyTestRunner

operations
    public static main: () ==> ()
        main() == (
            IO`println("Running tests\n");
        new TestGroupChat().Run();
        new TestPublication().Run();
        new TestUser().Run();
        new TestFacebook().Run();
        IO`println("Tests ran successfully\n");
        );

end MyTestRunner
```

5. Model Verification

Domain verification

One example of a domain verification proof obligation generated by the tool was:

5	User`getChatByName(User`String)	legal map application

(forall chatName:User`String & ((chatName in set (dom chats)) => (chatName in set (dom chats))))

The code under analysis is presented below:

```
-- Specific group chat getter
-- IN chat name
pure public getChatByName: String ==> GroupChat
                getChatByName(chatName) == return chats(chatName)
                pre chatName in set dom chats;
```

Through the precondition we can assure that the map *chats* is only accessed inside its domain and, therefore, its usage is always valid.

Invariant verification

One example of a state invariant verification proof obligation generated by the tool was:

7	Facebook`addUser(User)	state invariant holds

(forall name:Facebook`String & ((not (exists user1, user2 in set users & ((user1 <> user2) and ((user1.getName)() = (user2.getName)())))) => (not (exists user1, user2 in set users & ((user1 <> user2) and ((user1.getName)() = (user2.getName)()))))))

The code under analysis is presented below:

```
-- Add a user to the platform
-- IN user - user to be added
 public addUser: User ==> ()
  addUser(user) == (
   users := users union {user}
  )
  pre not exists member in set users & member.getName() = user.getName()
  post users = users~ union {user};
```

And the invariant under analysis is:

```
-- User name uniqueness
inv not exists user1, user2 in set users & user1 <> user2 and user1.getName() = user2.getName();
```

Which states that two different users cannot have the same name. The operation addUser inserts a new user into the system, therefore it must be assured that this addition does not break the invariant. Through the operation's precondition we ensure that a user can only effectively be added if there isn't already another user with the same name, therefore guaranteeing that the invariant still holds after the operation is complete.

6. Code Generation

Overall, the generated Java code worked as expected from the VDM++ source, except for a couple of easily solvable errors: on the Facebook class, we faced some variable size issues, namely on comparisons between *long* and *int* types, which we fixed by manually casting said variables to *long* before usage; secondly, the function *likePublication* had a call to what was perceived as an inherited method, namely *super.getAllPublications*, which was invalid as both operations belonged to the same class, and therefore replaced with the correct call, *getAllPublications*.

It should also be noted that the original pre and post conditions were not generated, which deprives the Java version from most input validation and overall consistency of the model logic.

A CLI was developed to showcase the model's features and usage in a real-life scenario. It's divided into five menus: Search, Timeline, Posts, Friends, and Chat, each which multiple operations. It's worth noting that all functionalities list and described in Section 2 can be executed through this interface.

7. Conclusions

The final result was positive. A system that covered most of Facebook's main functionalities (users, friendships, publications with permissions, search, relevant feed generation) was successfully defined in a formal model. All proposed requirements were implemented. 100% coverage in testing and the successful code generation and integration into a small CLI allows to infer that the project was completed successfully.

In spite of this, there can always be room for improvement. In this case, a more thorough set of permissions (e.g. add a specific set of users to have access or be excluded per publication) or more secondary features such as the full reaction system (instead of just liking) and Facebook pages could have possibly been explored given more time.

All the members of the group worked equally throughout the project's development and, therefore, have an equal amount of contribution.

8. References

[1] Facebook, https://www.facebook.com

[2] VDM slides (pt. 1), https://moodle.up.pt/pluginfile.php/183550/mod_resource/content/0/VDM%2B%2B1.pdf

[3] VDM slides (pt. 2), https://moodle.up.pt/pluginfile.php/185263/mod_resource/content/0/VDM%2B%2B2.pdf

[4] Overture Quick Start, https://moodle.up.pt/pluginfile.php/25618/mod_resource/content/0/OverturQuickStartExercise.pdf

[5] Overture User Guide, http://lausdahl.github.io/overturetool.github.io/files/OvertureIDEUserGuide.pdf