LetsGo

Architecture Description

Version 1.0

March 2, 2020

–

Suchi Kapur

Nick Garrett

Khaled Khalil

Table of Contents

1. Introduction

LetsGo is a social media mobile application that operates based on user interaction and user events. LetsGo will be designed using the Model-View-Controller (MVC) architecture pattern. This architecture pattern utilizes three components: a model, a view, and a controller. The model contains information used by the application. The view is the screen presentation of the information. Lastly, the controller defines the way that the user interface reacts to user input and maintains the view-model consistency.

In this application, the user interacts with their LetsGo mobile application. The user's events are correspondingly handled by the controller. The controller communicates with the model and updates any necessary information, and the corresponding view if changed. The model for this application is the Firebase database, where user account and application-specific information are available for storage and retrieval. The controller essentially is the input-handler and client-side code, while the view(s) are.xaml and .xaml.cs files/pages. The high-level architecture design is shown in Figure 1.

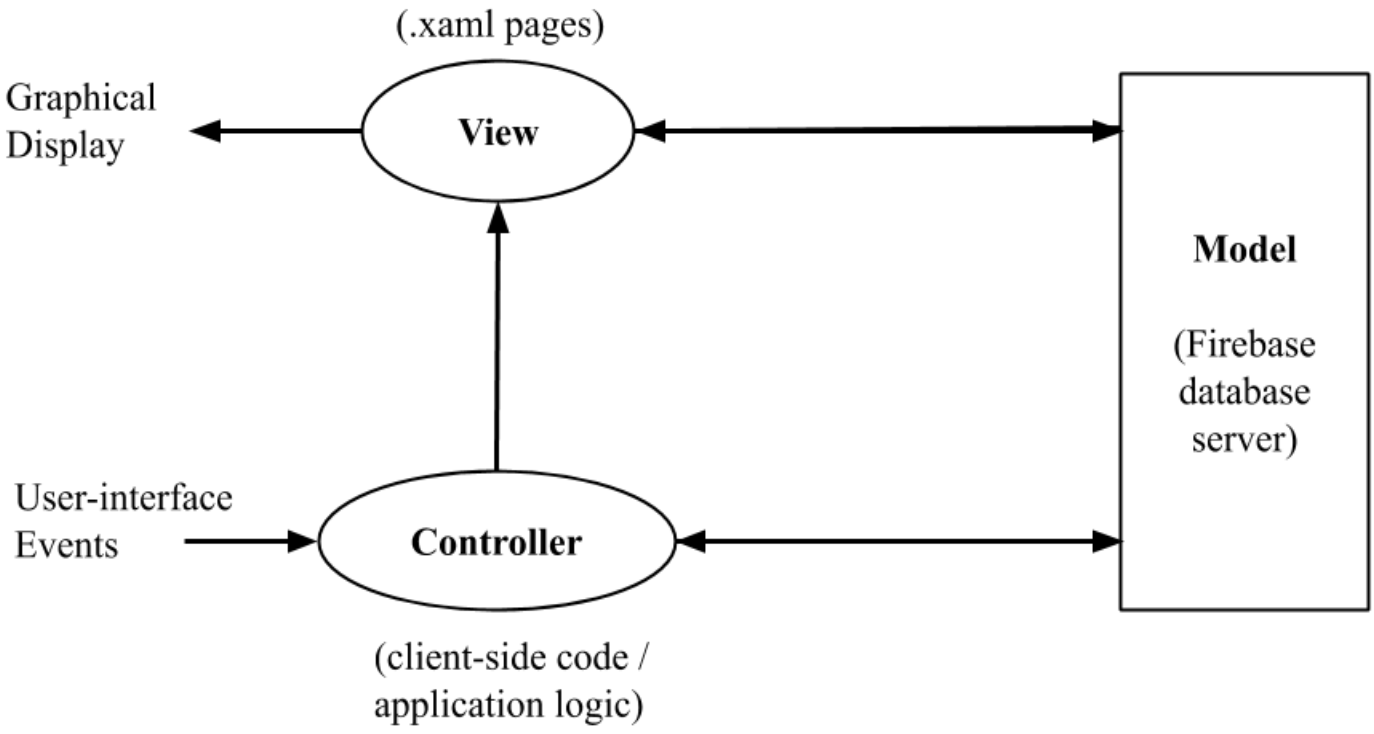

Figure 1. High-Level Architecture of LetsGo MVC Pattern.

2. Model Design

2.1 Model Module

Purpose

The purpose of this module is to encapsulate application-specific data. More specifically, the model module acts as a central area for LetsGo application data to be stored and retrieved from.

Rationale

This module allows users to access and retrieve information relative to their interactions with the application that are handled by the controller and accessed by the model. The model will store information related to user accounts, communities, events, user friends, and friend requests. Additionally, all user authentication and profile information will be stored in the model.

High-Level Module Design

The model module is broken down further into two lower-level modules, as shown in Figure 2 below.

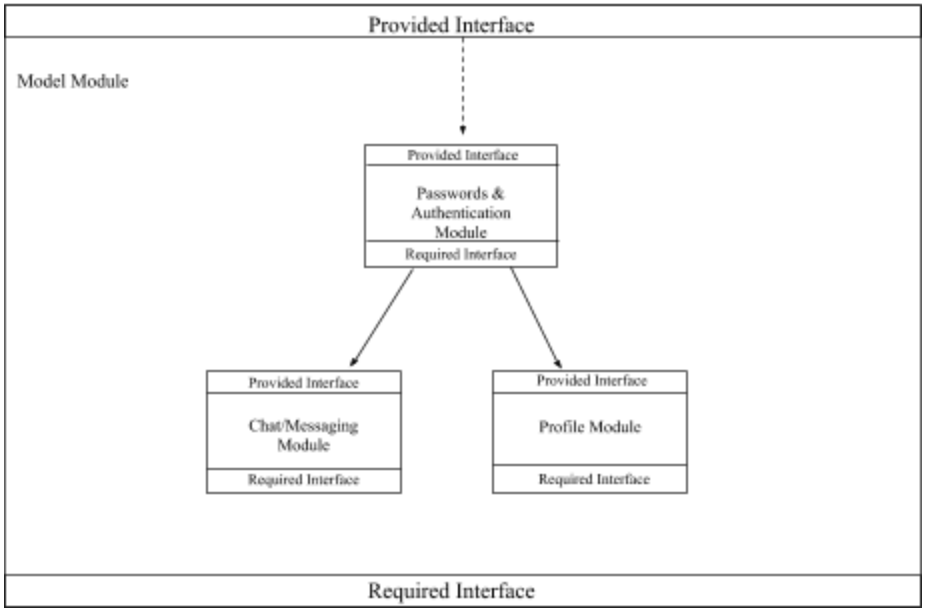

Figure 2. High-Level Model Module Design

Required Interface

This module has no required interface.

Provided Interface

The provided interface of this module are the passwords and authentication module, the profiles module and the chat/messaging module.

2.2 Profiles Module

Purpose
The purpose of this module is to provide access to user profile information, and the ability to create, join, or leave communities, and community leaders' abilities to add or remove community members. Users can also create and rsvp to events, while event owners' are able to invite users, update or delete events. Users can add or remove friends. Additionally, this module provides access to the user's feed and search capabilities within the application. Essentially, the profiles module provides the details and functionalities available to users.

Rationale
This module allows users to access and retrieve any relevant application information from their accounts after successful login to the application. This module is created to consolidate user data.

Required Interfaces
```
Boolean login(Email email, Password password);
```

Provided Interfaces

```
Boolean userExists(Email email);
```

Description
Determines if a user exists in the database.

Parameters
`email:` the email address associated with the account to verify

Returns
`Boolean`
The userExists method returns true if the user exists, false if otherwise.

```
Boolean updateProfile(UserProfile user, Name name, InterestTags
interests, Location city, Status status, ProfilePicture
picture);
```

Description

When updating profiles, users can update their full names, hobbies, location (city), account status to public or private, update their profile pictures, and their interests. Users will not be able to change their date of birth. Any information that is not updated by the user will be prepopulated using their previous profile information.

Parameters
`user:` user profile to update
`name:` user's name
`interests:` user's interests, based on interest tag
`city:` user's location
`status:` user account status (public or private)
`picture:` file path for user profile picture

Returns
`Boolean`
If the profile update is successful, the method will return true, otherwise it will return false.

```
Boolean deleteAccount(UserProfile user);
```

Description
Allows a user to delete their account.

Parameters
`user:` the user's account to be deleted

Returns
`Boolean`
Returns true if account deletion was successful, false if not.

```
UserProfile getUserProfile(UserProfile user) throws
NoUserException;
```

Description
Retrieve specified user profile.

Parameters

`user:` user profile to retrieve

Returns
`UserProfile`
The getUserProfile returns the requested user's profile.

Exceptions
`NoUserException`
If the user does not exist, the method will throw a `NoUserException`.

```
Boolean logout() throws UserNotLoggedIn;
```

Description
Allows a user to sign out/log out of account.

Parameters
The logout method does not require any parameters.

Return
If logout is successful, the logout method returns true. If not, it returns false.

Exceptions
`UserNotLoggedIn`
If the user is not logged in, this exception is thrown.

```
Community createCommunity(CommunityName name, CommunityIdentity
identity, InterestTags tags, User leader);
```

Description
Users will be able to create communities with specific interest tags.

Parameters
`name:` the name of the community
`identity:` the identity of the community
`tags:` the different tags associated with the community
`leader:` the community leader

Returns
`Community`
Upon successful creation, the createCommunity method returns the new community.

```
Boolean joinCommunity(Community communityToJoin);
```

Description
Users will be able to request to join communities

Parameters
`communityToJoin:` the community the user wishes to join.

Returns
`Boolean`
If the request is accepted then the user accepted into the community and returns true else false

Constraints
If the community that the user wishes to join has a “private” status, the community leader will receive a request for the user to join.

```
Boolean leaveCommunity(Community communityToLeave);
```

Description
This will allow a user to leave a community.

Parameters
`communityToLeave:` community that the user wishes to leave (remove themselves as a member from).

Returns
`Boolean`
If the user is successfully removed then return true otherwise false.

```
Community[] getCommunities();
```

Description
Retrieve all communities that the user is a member of.

Parameters
The getCommunity does not require any parameters.

Returns
`Community[]`
The getCommunity method returns an array of communities.

```
Boolean updateCommunityMember(UserProfile user, Community
community, User leader);
```

Description
The updateCommunityMember method allows community leaders to add or remove members from their community.

Parameters
`user:` user to update in community
`community:` community to update member for
`leader:` community leader of community

Returns
`Boolean`
Returns true for successful addition or removal of user to a community, false if unsuccessful.

```
Event[] getEvents();
```

Description
The getEvents method allows users to retrieve events that they have accepted an invitation to.

Parameters
The getEvents method does not require any parameters.

Returns
`Event[]`
The getEvents method returns an array of events that the user plans on attending (has accepted an invitation to).

```
Event createEvent(EventName name, Date date, Time time, Location loc, EventDescription description, EventImage image, EventStatus status, UserProfile owner);
```

Description
The createEvent method allows users to create an event.

Parameters
`name:` event name
`date:` event date
`time:` event time
`loc:` event location
`description:` description of event
`image:` file path to image relating to event
`status:` event status of public or private
`owner:` owner of event

Returns
`Event`
The createEvent method will return the newly created event.

```
Boolean addEventToCommunity(Event event, UserProfile eventOwner, Community community, UserProfile communityLeader, UserProfile currentUser);
```

Description
Allows a community leader to add an event to a community if they are the event owner.

Parameters
`event:` event to add to community events
`eventOwner:` owner of event

`community:` community to add event to
`communityLeader:` leader of the community to add the event to
`currentUser:` the user attempting to add the event to the community

Returns
`Boolean`
If the event has been successfully added to the community, the method returns true, otherwise false. If the current user is not the event owner and community leader, the method will return false.

```
Boolean inviteUsers(Event event, UserProfile owner,
UserProfile[] usersToInvite) throws NotEventOwnerException;
```

Description
Allows event owners to invite users to an event they have created.

Parameters
`event:` event to invite users to
`owner:` event owner
`usersToInvite:` array of users to invite to event

Returns
`Boolean`
The inviteUsers method returns true if users have successfully been invited to an event, or false if not.

Exception
`NotEventOwnerException`
Users who are not the owner of the specified event do not have privileges to invite users.

```
Event updateEvent(Event event, EventName name, EventDate date,
Time time, EventLocation loc, EventDescription description,
EventImage image, EventStatus status, UserProfile owner) throws
NotEventOwnerException;
```

Description

Allows event owners to update events they have created.

Parameters
`event:` event to update
`name:` event name
`date:` event date
`time:` event time
`loc:` event location
`description:` description of event
`image:` file path to image relating to event
`status:` event status of public or private
`owner:` owner of event

Returns
`Event`
The updateEvent method returns the updated Event.

Exceptions
`NotEventOwnerException`
Users who are not the owner of the specified event do not have privileges to update an event.

```
Boolean deleteEvent(Event event, UserProfile owner) throws
NotEventOwnerException;
```

Description
The deleteEvent method allows an event owner to delete an event that they have created.

Parameters
`event:` event to delete
`owner:` event owner

Returns
`Boolean`
The deleteEvent method will return true if the event has been deleted and false if not.

Exceptions
`NotEventOwnerException`
This exception is thrown if a user is not the owner of the event that is specified to delete.

```
Boolean rsvpEvent(Event event, Rsvp rsvp, UserProfile user);
```

Description
The rsvpEvent method allows users to rsvp to an event that they have been invited to.

Parameters
`event:` event to rsvp to
`rsvp:` user's choice to accept or decline an event invitation
`user:` user that is responding to invitation

Returns
`Boolean`
The rsvpEvent method returns true for a successful response to an invitation, and false if not.

```
UserProfile[] getFriends();
```

Description
The getFriends method will display all friends of the user in a list.

Parameters
The getFriends method does not require any parameters.

Returns
`UserProfile[]`
An array of users that are the current user's friends will be returned.

```
Boolean addFriend(UserProfile currentUser, UserProfile
friendToAdd);
```

Description
This method allows users to add a friend.

Parameters
`currentUser:` current user

`friendToAdd:` user to add as friend

Returns
`Boolean`
Returns true if friend request was successfully sent, false if not.

```
Boolean removeFriend(UserProfile currentUser, UserProfile
friendToRemove);
```

Description
This interface allows users to remove a friend.

Parameters
`currentUser:` current user
`friendToRemove:` user to remove as friend

Returns
`Boolean`
Returns true if the friend was removed successfully, false if not.

```
SearchResults search(InterestTag tag);
```

Description
Retrieves all matching results given the specified interest tag keyword.

Parameters
`tag:` interest tag user searches for

Returns
`SearchResults`
A collection of users, communities, and events matched via the keyword from the model.

```
Feed getFeed();
```

Description
Retrieves the user's feed, consisting of public events within their location.

Parameters
The getFeed method does not require any parameters.

Returns
`Feed`
The getFeed method returns a list of public events from public communities based on the user's area and interest tags set in their profile.

Module ADTs

```
typedef Email String;
typedef Password String;
typedef Name String;
typedef DateOfBirth String;
typedef Password String;
typedef InterestTag String;
typedef Location String;
typedef Status String;
typedef ProfilePicture Image;

UserProfile
{
     Email email;
     Name name;
     DateOfBirth dob;
     Password pass;
     InterestTag[] interests;
     Location city;
     Status status;
     ProfilePicture profilePicture;
}

typedef CommunityName String;
typedef CommunityIdentity String;
```

```
Community
{
     CommunityName name;
     CommunityIdentity identity;
     InterestTag[] interestTags;
     UserProfile communityLeader;
     UserProfile[] members;
     Event[] events;
}

typedef EventName String;
typedef EventDate String;
typedef Time String;
typedef EventLocation String;
typedef EventDescription String;
typedef EventImage Image;
typedef EventStatus String;
typedef Rsvp String;

Event
{
     EventName name;
     EventDate date;
     Time time;
     EventLocation place;
     EventDescription description;
     EventImage picture;
     EventStatus status;
     UserProfile[] members;
     UserProfile owner;
     InterestTag[] interestTags;
}

SearchResults
{
     Event[] publicEvents;
     UserProfile[] publicUsers;
     Community[] publicCommunity;
}
```

```
Feed
{
     Event[] publicEvents;
}
```

2.3 Passwords and Authentication Module

Purpose
The purpose of this module is to provide authentication services for login of the LetsGo application. This module validates user login credentials against user login information stored within the Firebase database (the model).

Rationale
This module is created to provide a login service to users and promotes the security of user login information with login authentication.

Required Interface
```
Boolean userExists(Email email);
```

Provided Interface

```
Boolean createAccount(Name name, DateofBirth dob, Email email,
Password pass);
```

Description
The createAccount interface allows users to create an account.

Parameters
`name`: user's first name
`dob`: user's date of birth
`email`: user's email address
`pass`: user's password

Returns
`Boolean`
If the user's account is created successfully, the createAccount method will return true, otherwise it will return false.

```
void forgotPassword();
```

Description
The forgotPassword method allows users to access the password recovery service within the profile module.

Parameters
The forgotPassword method does not have any parameters.

Returns
`void`
The method does not return any values.

```
Boolean recoverPassword(Email email);
```

Description
The recoverPassword method allows users to retrieve their password in the event they have forgotten it. When the forgotPassword() method is called, the user will be prompted to input their email which will be verified against the emails stored in the Firebase database. If found, the password recovery service will be triggered, where the user will be sent an email with their password.

Parameters
`email:` user's email address associated with their LetsGo account

Returns
`Boolean`
If the password recovery service was triggered/called, then the method will return true, otherwise it will return false.

```
Boolean login(Email email, Password password);
```

Description
The login method allows users to login to access their account. The login method also determines whether the user's email and password combination is valid or not.

Parameters
`email:` user's email address associated to their LetsGo account
`pass:` user's password

Returns
`Boolean`
The login method returns true if the user has logged in successfully, and false if otherwise.

Module ADTs
There are no module ADTs for this module.
See Section 2.2 Profiles Module for UserProfile, where the newly created account with user information is stored.

2.4 Chat/Messaging Module

Purpose
The purpose of the chat module is to provide access to chats within events or communities, or between users. This module acts as a central storage for all messages within the LetsGo application. Users will be able to send messages to users, communities that they are members of, and events that they are attending. Users will also be able to retrieve all messages with a user, community, or event that they are attending.

Rationale
This module is created to centralize and encapsulate chat services provided by the LetsGo application.

Required Interface
```
Boolean login(Email email, Password password);

Boolean userExists(Email email);
```

Provided Interface

```
Boolean sendMessage<T>(UserProfile user, Message msg, T
recipient);
```

Description
Sends a user's message to the recipient.

Parameters
`user:` user who is sending a message
`msg:` message to be sent
`recipient:` user, community, or event-goers to receive message (using template for flexibility)

Returns
`Boolean`
The sendMessage method returns true if the message was sent successfully, and false if not.

```
Message[] getAllMessages<T>(UserProfile user, T chatGroup);
```

Description
The getAllMessages method retrieves all messages for the provided event, community, or user specified.

Parameters
`user:` user who wishes to retrieve messages
`chatGroup:` chat to retrieve messages from (another user, an event, or community). Using template for flexibility.

Returns
`Message[]`
The list of messages is returned.

Module ADTs

`User:` See Section 2.3 Passwords and Authentication Module.

```
typedef DateSent String;

Message
{
```

```
    UserProfile sender;
    DateSent dateSent;
    String msg;
}
```

3. View Design

3.1 View Module

Purpose

The purpose of this module is to provide page layouts for information to be displayed on.

Rationale

This module is created to encapsulate display choices and information to the graphical display. The view module allows users to interact with the system through the application, and allows users to view their account, profile, and application-related information.

High-Level Module Design

The view module is broken down into lower-level modules, shown in Figure 3. All user interactions with the LetsGo application are done through the User Interface module (Section 3.2), where events are then handled by the controller module.

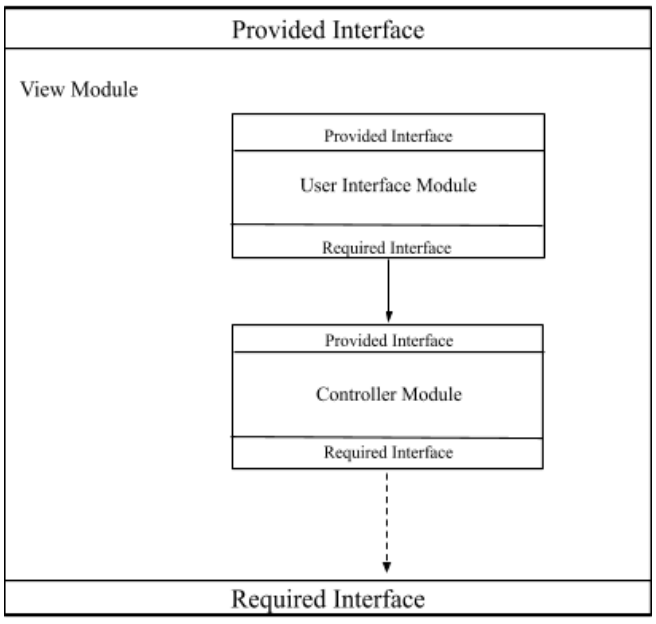

Figure 3. View Module High-Level Architecture

Required Interface

The view module has no required interface.

Provided Interface
The provided interfaces given by the view module is the union of the user interface and controller module.

3.2 User Interface (UI) Module

Purpose
The purpose of the UI module is to describe the user interface on the user's device.

Rationale
This module allows users to make changes to the model through the controller module, and further through the communications module within the controller.

Required Interfaces
The required interface of the UI module is the communications module (See Section 4.2).

Provided Interfaces

```
Feed getFeed();

Event[] getEvents();

UserProfile getUserProfile(UserProfile user) throws
NoUserException;

UserProfile[] getFriends();

void forgotPassword();

Community[] getCommunities();
```

4. Controller Design

4.1 Controller Module

Purpose

The purpose of this module allows for communication between user events and data which are then updated within the view or model accordingly. Given user interaction, this module communicates with the model and loads updated and correct views of any information to the view module.

Rationale

This module is created to encapsulate interaction semantics of user-interface events, and strengthen interactions between user events and view or model changes.

High-Level Module Design

This module intercepts user interactions and completes its role accordingly. The user events trigger the controller to interact with the model and update, or change, any necessary data.This module is broken down into two lower-level modules, as shown in Figure 4. All user interactions are sent to the controller are sent to the communications module. The high-level controller component and the communications module handle the same requests: user interaction, and are therefore the same.

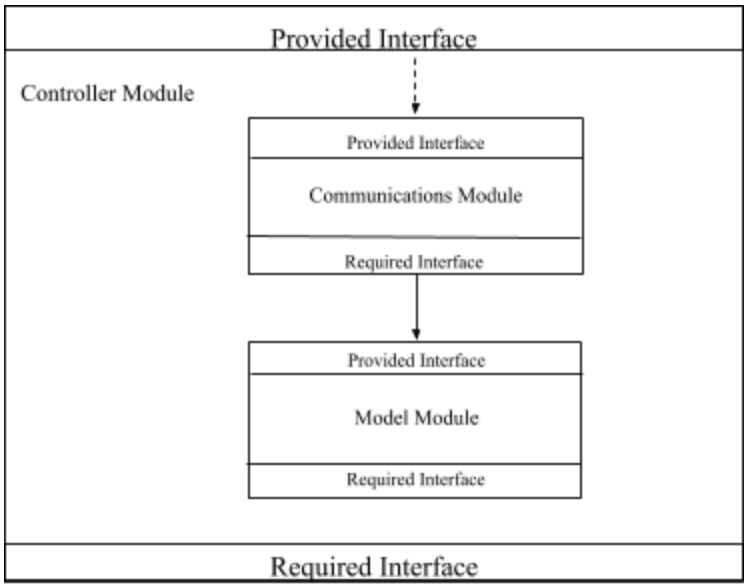

Figure 4. Controller Module High-Level Architecture

Required Interface

The controller module has no required interfaces.

Provided Interface

The provided interfaces of the controller module is its union with the communications module.

4.2 Communications Module

Purpose

The purpose of this module is to provide communication services between the user interactions and the model or view. This module represents the link between the controller, model and view modules based on user interaction. The communications module details how the client retrieves and stores data for the LetsGo application in the Firebase database (the model).

Rationale

This module is created to encapsulate data from user interactions to update the model, and to receive data from the model to be sent to the view.

Required Interfaces

```
Boolean login(Email email, Password password);

Boolean logout() throws UserNotLoggedIn;

Boolean userExists(Email email);

Boolean recoverPassword(Email email);

Boolean createAccount(Name name, DateofBirth dob, Email email,
Password pass);

Boolean updateProfile(UserProfile user, Name name, InterestTag
interests, Location city, Status status, ProfilePicture
picture);

Boolean deleteAccount(UserProfile user);

UserProfile getUserProfile(UserProfile user) throws
NoUserException;

Community createCommunity(CommunityName name, CommunityIdentity
identity, InterestTags tags, User leader);

Boolean joinCommunity(Community communityToJoin);
```

```
Boolean leaveCommunity(Community communityToLeave);

Boolean updateCommunityMember(UserProfile user, Community
community, UserProfile leader);

Community[] getCommunities();

Event createEvent(EventName name, Date date, Time time, Location
loc, EventDescription description, EventImage image, Status
status, UserProfile owner);

Boolean addEventToCommunity(Event event, UserProfile eventOwner,
Community community, UserProfile communityLeader, UserProfile
currentUser);

Boolean inviteUsers(Event event, UserProfile owner,
UserProfile[] usersToInvite) throws NotEventOwnerException;

Event updateEvent(Event event, EventName name, EventDate date,
Time time, EventLocation loc, EventDescription description,
EventImage image, EventStatus status, UserProfile owner) throws
NotEventOwnerException;

Boolean deleteEvent(Event event, UserProfile owner) throws
NotEventOwnerException;

Boolean rsvpEvent(Event event, Rsvp rsvp, UserProfile user);

Event[] getEvents();

Boolean addFriend(UserProfile currentUser, UserProfile
friendToAdd);

Boolean removeFriend(UserProfile currentUser, UserProfile
friendToRemove);

UserProfile[] getFriends();

Boolean sendMessage<T>(UserProfile user, Message msg, T
recipient);
```

```
Message[] getAllMessages<T>(UserProfile user, T chatGroup);
```

Provided Interface

The provided interface of this module is the same as the required interface of the UI module.