NativeToMe

Architecture Design

Date: March 3, 2020

Prepared by: David Miller, Ralph Lira, Kevyn Higbee

Table of Contents

1. Introduction

The purpose of this document is to provide a high level explanation of the architecture styles used in the creation of NativeToMe. The document will provide a mapping of the software-to-be and provide insight into the project without having to dig into the code. This will lay a foundation for a quality, cost effective architecture that lends itself to easy maintainability, and scalability.

The high-level architecture for the system is shown in Figure 1-1.

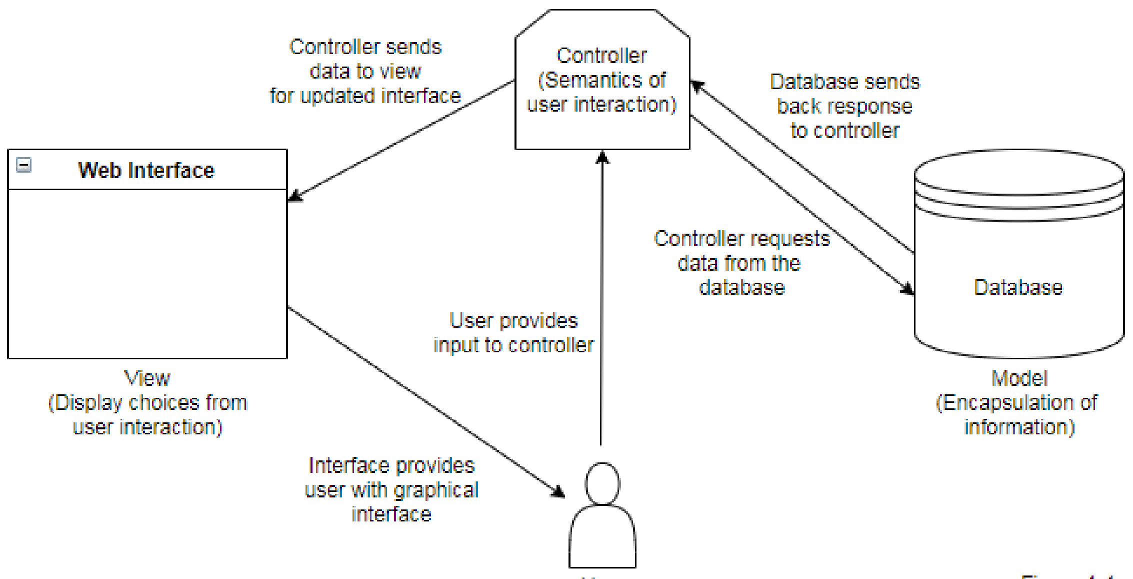

Controller sends data to view for updated interface
Controller (Semantics of user interaction)
Database sends back response to controller
Web Interface
Controller requests data from the database
Database
User provides input to controller
View (Display choices from user interaction)
Model (Encapsulation of information)
Interface provides user with graphical interface
User

Figure 1-1

Controller Module

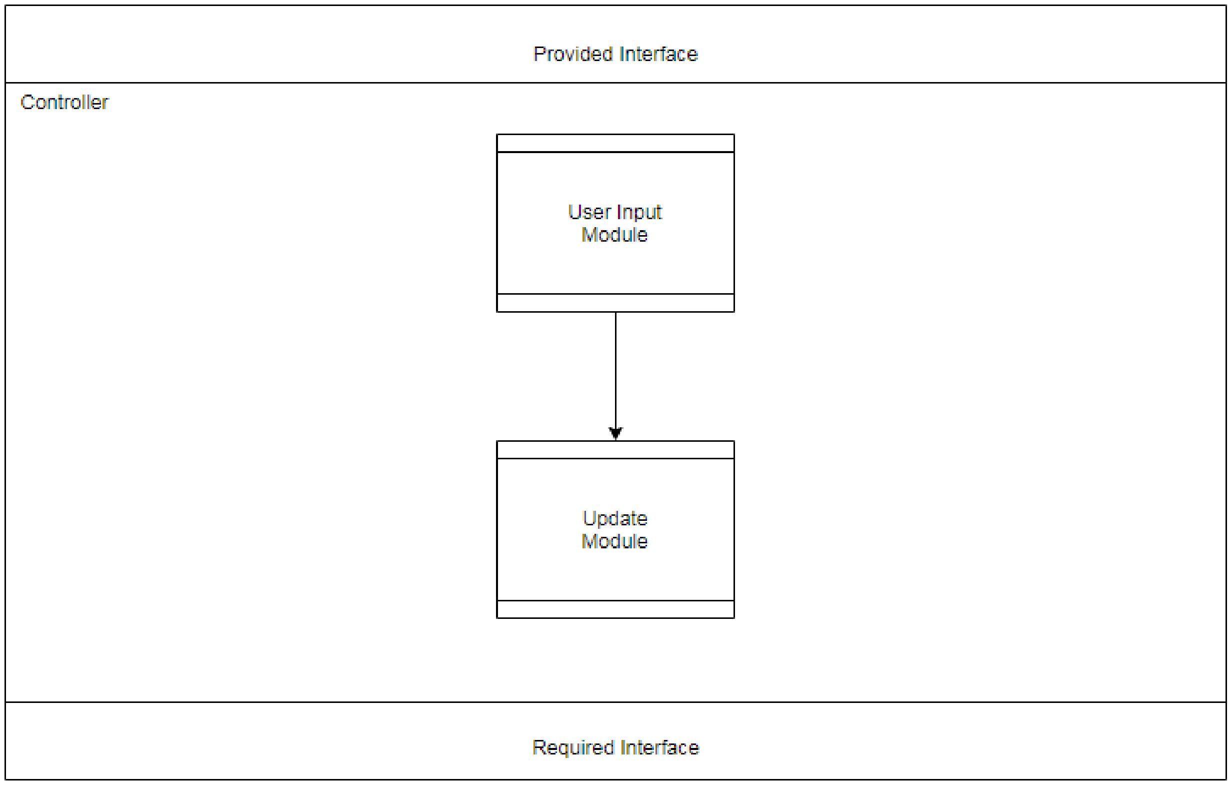
The architectural style used by the system is the Model-View-Controller(MVC). This architectural style was chosen because MVC is used for web-based applications and NativeToMe is a web application. Specifically, the controller module will handle user input and relay it to the model. The model will take this input, and update the model accordingly. It will then proceed to draw/redraw the view that the user ultimately is viewing. The main concept is that the controller is the middle man between the model and the view. The application waits for further input to complete other actions and repeat the process.

Purpose

The purpose of this module is to control the way the user interacts with the web application. The controller is responsible for sending user commands to the model, which in turn, updates the models, and updates the view.

Rationale
This module is created to allow the user to make requests to the system for specified actions and receive a response back from the system.

High-Level Controller Design
The client's communication to the system comes from the user input module. There are two more components within the controller module which is the update module which handles user updates that go through the model. The controller also handles user input through the view of the user interface.



Required Interface
This module has no required interface.

Provided Interface
The provided interface of this module contains the following submodule

- User Input

1.1 User Input Module

Purpose
The purpose of this module is to provide communication from the user to the controller. The user inputs an action and which is sent to the controller which then results in the controller communicating with the model and view to return information to the user.

Rationale
This module is created to provide communication with the user's input and the controller.

Required Interface
This module's required interface is:

- Update Module

Provided Interface
The provided interface is the same as the required interface.

1.2 Update Module

Purpose
The purpose of this module is to allow for the user input to update information within the database.

Rationale
This module is created to provide communication with the controller and the model to update information which will be sent back to the controller to send to the view for an update in the graphical interface.

Required Interface
This module has no required interface.

Provided Interface
The provided interface is the same as the required interface.

Model
Please see Chapter 2 for a detailed description of this module.

View
Please see Chapter 3 for a detailed description of this module.

2. Model Design

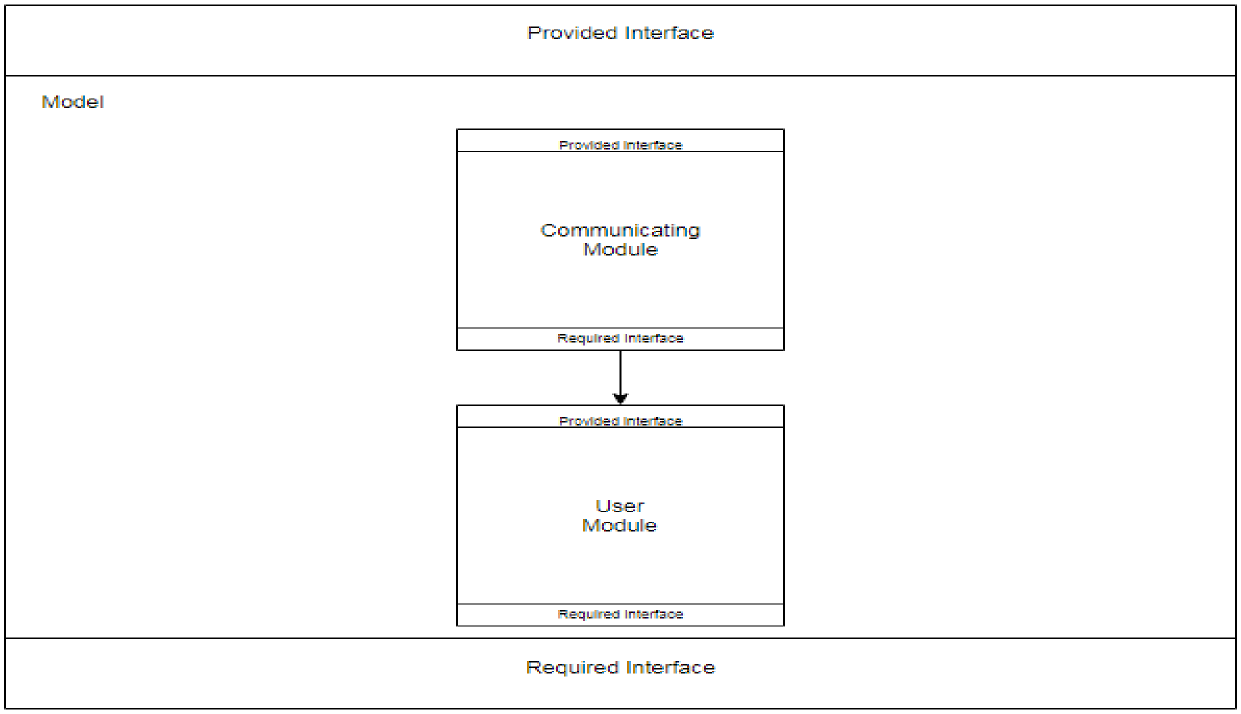
Model Module

Purpose
The purpose of this module is to provide a centralized place where information for the system can be stored, manipulated, and accessed by the controller or view module.

Rationale
This module is created to centralize and encapsulate all data storage and retrieval duties on the system. This includes user profiles, user tribes, user banners, user pictures, and information to describe profiles/tribes. It also provides some services, such as authentication, network communication and search.

High-Level Model Design
The model module is broken down into lower-level modules, as shown in the design below.

The communications from the user to the other components of the design come through the communications module. The user module handles all the user's actions within the system. The communicating module and the user module will be further clarified in the following sections of the document.



Required Interface
This module has no required interface.

Provided Interface
The provided interface of this module is the union of the provided interfaces of the following submodules:

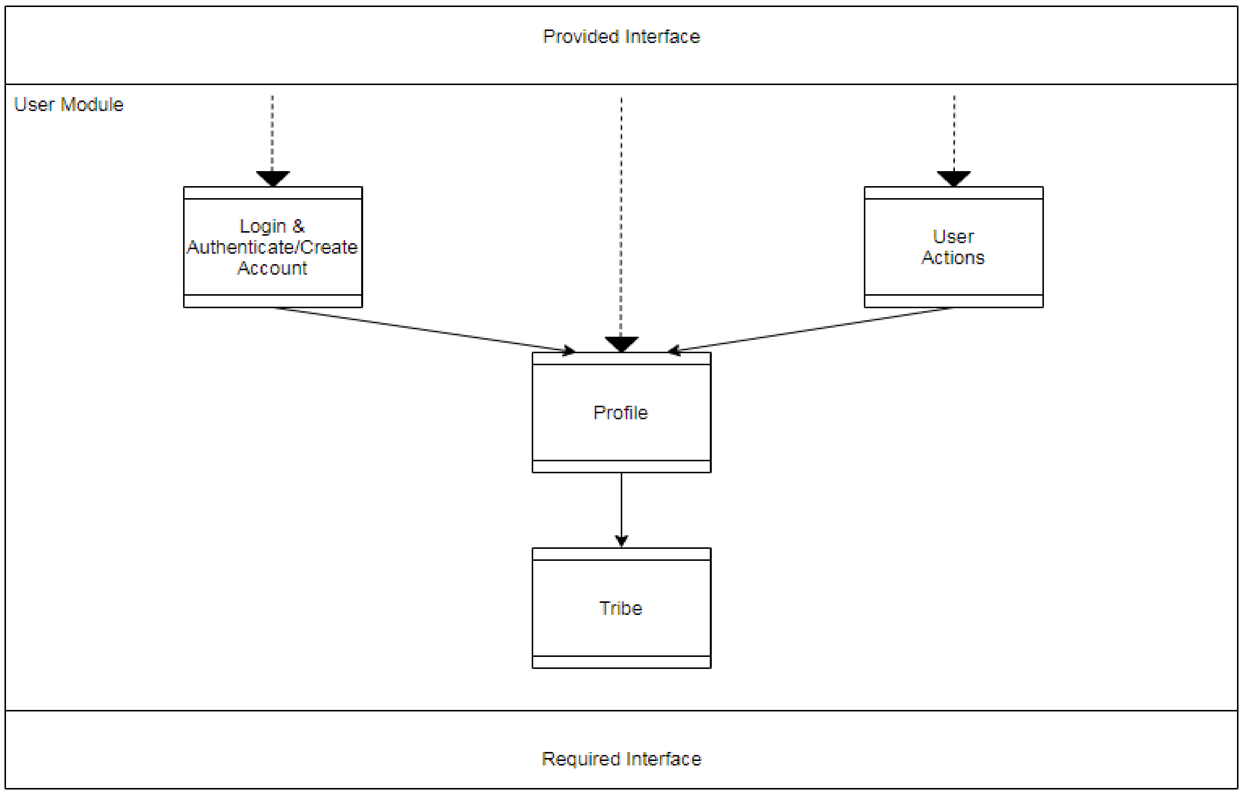
- Communicating

For more detail, please see these modules' descriptions.

2.1 Communicating Module

For more information refer to section.3.2 Communicating Module

2.2 User Module



Purpose
The purpose of this module is to authenticate and create users, keep track of user actions, and contain user and tribe information.

Rationale
This module is created to contain all the information related to any user action.

Required Interface
This module does not have a required interface

Provided Interface
The provided interface of the user module is the combination of the submodules:
- Login & Authenticate / Create Account
- User Actions
- Profile
- Tribe

2.2.1 Login & Authenticate / Create Account Module

Purpose
The purpose of this module is to authenticate login information to determine if the user currently exists in the system and create an account if the user does not exist in the system already.

Rationale
This module is created to contain all user login information and authenticate it or have a new user created.

Required Interface
```
Boolean userExists(UserID userID);
```

Provided Interface
```
Void logIn(UserID userID, Password pass) throws
NoUserException;
```

Description:
Stores user's username and password for login.
Parameters:
`userID`: the username of the user trying to login.
Pass: the password of the username for the user.

Exceptions:
`NoUserException`: The user does not exist; the username and password do not match any existing user in the system.

```
Boolean createUser(UserID userID, Password pass) throws
UserExistsException;
```

Description:
If username and password associated with the username does not already exist in the system, allow the user to create the username.
Parameters:
`userID`: the new username for the user
`Pass`: the password for the new username of the user.
Exceptions:
`UserExistsException`: The username exists within the system already.

2.2.2 User Actions Module

Purpose
The purpose of this module is to provide a persistent way for users to interact with other users and content in their tribe.

Rationale
This module is created to encapsulate the data storage of actions that are made from either the user or the tribe.

Required Interface
```
Boolean userExists(UserID userID);
```

Provided Interface
```
void interactWithPost(UserID userID,PostID postid, ActionType action)
throws NoUserException, NoPostException;
```

Description:
Stores a user like on a user profile or the tribe wall.
Parameters:
`userID`: the name of the user who is doing the action
`postid`: the post that the user is interacting with
`action:` the type of action that the user is performing
Exceptions:
`NoUserException`: the user does not exist to be able to make an action.
`NoPostException`: there is no post for the user to like.
`UnsupportedActionException`: the specified action is not a recognized type

```
void commentOnPost(UserID userID, PostID postID, ActionType action,
String msg) throws NoSuchUserException, NoPostException,
UnsupportedActionException, InvalidMessageException;
```

Description:
Adds a comment to an existing post.
Parameters:
`userID`: the name of the user who is doing the action
`postid`: the post that the user is interacting with
`action:` the type of action that the user is performing
`msg`: the text comment that will be appended to the post
Exceptions:
`NoUserException`: the user does not exist to be able to make an action.
`NoPostException`: there is no post for the user to like.
`UnsupportedActionException`: the specified action is not a recognized type
`InvalidMessageException:` the message is not valid because of its content, or lack of content.
Module ADTs

```
typedef PostID String;

Enum ActionType {
      LIKE, COMMENT, SHARE, REPORT
}
```

2.2.3 Profile Module

Purpose
The purpose of this module is to store all user information (per profile) and provide functionality to access user data.

Rationale
This module is created to provide uniform storage for all data in a user profile as well as methods for retrieving user data.

Required Interface
This module has no required interfaces.

Provided Interface

```
void saveProfile(Profile profile);
```

Description:
Saves the passed profile. This will overwrite any existing profile matching the passed profile's UserID.

Parameters:
`Profile profile:` The profile to be saved.

```
Profile getProfile(UserID userId) throws NoSuchUserException;
```

Description:
Returns the profile matching the userID argument. If there is no matching profile, throws an exception.

Parameters:
`UserID userId:` The UserID of the profile being requested.

Returns:
The requested user profile.

Exceptions:
`NoSuchProfileException:` If there is no stored profile data matching the requested user, throws an exception.

```
void deleteProfile(UserID userId) throws NoSuchUserException;
```

Description:
Deletes the profile matching the userID argument. If there is no matching profile, throws an exception.

Parameters:
`UserID userId:` The UserID of the profile being requested for deletion.

Exceptions:
`NoSuchProfileException:` If there is no stored profile data matching the requested user, exception may be thrown.

```
int getNumProfiles();
```

Description:
Returns the number of profiles stored in the system..

Returns:
An integer value of the number of profiles stored in the system.

Module ADT’s

```
typedef UserID String;
typedef Password String;
typedef Date String;
typedef TribeID String;

Profile {
      UserID name;
      Password password;
      Image profilePicture;
      int age;
      Enum sex {male, female, other);
      TribeID tribes[];
      TribeID isAdminOf[];
      Date dateJoined;
      String likes[];
      String dislikes[];
      String interests[];
      String hobbies[];
      String countryOfResidence;
      PostID archivedPosts[];
}
```

2.2.4 Tribe Module

Purpose
The purpose of this module is to provide storage for tribe data as well as functionality to access a tribe's data.

Rationale
This module was created to provide a data structure for each tribe. This module provides interfaces for interacting with tribe data.

Required Interface
This module requires the following interface(s):

- Profile

Provided Interface

```
Tribe getTribeMembers(Tribe tribeID) throws NoAdditionalTribeMembers;
```

Description:
Gets a list of tribe members.

Parameters:
`tribeID`: The tribeID to retrieve the list of members from.

Returns:
A list of all current members in the tribe.

Exceptions:
`NoAdditionalTribeMembers`: If the tribe has only 1 member, the owner.

```
void setTribeHouse(Tribe tribeID, House house) throws
UnableToSetHouse;
```

Description:
Gets a tribe's house/houses.

Parameters:
`tribeID`: The TribeID where a new house is being created
`house`: The house to be set.

Exceptions:
`NoTribeHouseException`: If unable to set house for tribe.

```
void setTribeName(Tribe tribeID, String name) throws nameTaken;
```

Description:
Assign tribe name.

Parameters:
`tribeID`: The TribeID to retrieve tribe name.
`Name:` The string to set as the tribes name.

Exceptions:
`nameTaken`: If the requested tribe name has already been assigned.

```
Post getMemberPosts(Tribe tribeID, Post memberPosts) throws
NoMemberPosts;
```
Description:
Given a tribeID and memberPosts retrieve the list of member posts for the tribe.

Parameters:
`tribeID:` Tribe ID to retrieve member posts for
`memberPosts`: A list of member posts.

Returns:
A list of all current member posts.

Exceptions:
`NoMemberPosts`: If unable to set quality for tribe.

Module ADTs

`UserID`: See Profiles Module

```
typedef Date String;

Tribe {
      UserID members[];
      UserID leaders[];
      UserID bannedUsers[];
      Image banner;
      int numMembers;
      Date dateCreated;
      String tribeID;
      String values[];
      String traditions[];
      String qualities[];
      House houses[];
      PostID memberPosts[];
}
```

2.2.5 Post Module

Purpose

The purpose of this module is to store user posts and allow interactions with the post.

Rationale

todo

Required Interface

This module requires the following interface(s):

- Profile
- Tribe

Provided Interface

```
Post getPost(PostID postID) throws NoMatchingPostException,
PostLoadException;
```

Description:

Locates the post matching the associated postID.

Parameters:

`PostID:` The postID for the requested post

Returns:

The requested post’s data.

Exceptions:
`NoMatchingPostException`: If there is no post matching the specified PostID.
`PostLoadException`: if there is an error when trying to load the post

```
Void removePost(PostID postid) throws NoMatchingPostException;
```

Description:
Deletes the specified post.

Parameters:
`PostID`: The postID for the requested post for deletion

Exceptions:
`NoMatchingPostException`: If there is no post matching the specified PostID.

Module ADTs

```
typedef Timestamp String;

enum visibility {
      PUBLIC,PRIVATE,HIDDEN
}
Post {
      Image picture[];
      UserID postOwner;
      PostID postID;
      String caption;
      String comments[];
      int likes;
      int shares;
      Timestamp timePosted;
      Visibility status;
}
```

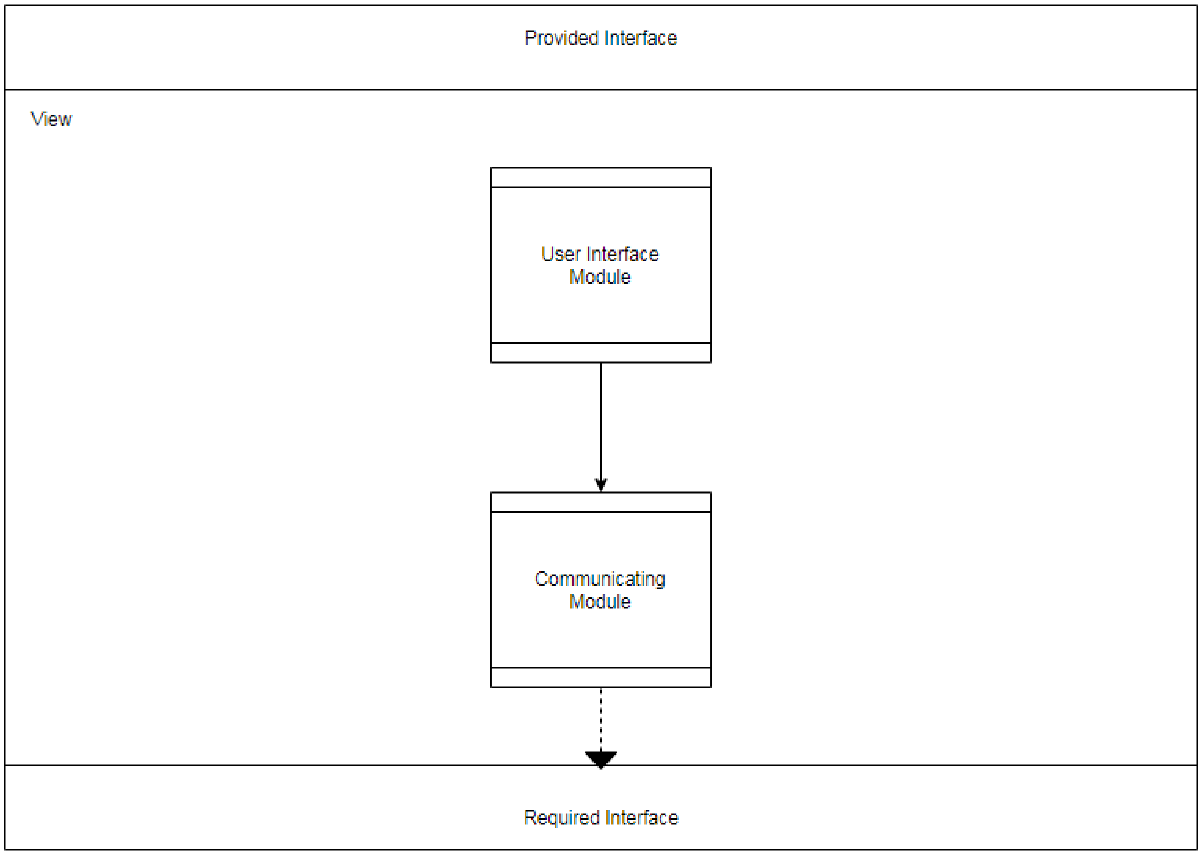
3. View Design

View Module

Purpose
The purpose of this module is to provide an interface to the user for the system. The user interacts with the interface directly and this allows for communication to receive data and also update information within the system.

Rationale
This module is created to provide the user interface to the user.

High-Level Module Design
The view is broken down into lower-level modules, as shown in the figure below:



Provided Interface
This module has no provided interfaces.

Required Interface
The required interface of this module is the same as the required interface of the Communicating module in section 3.2.

3.1 User Interface Module

Purpose
The purpose of this module is to provide a graphical user interface to the user.

Rationale

This module is created to give the user a way to view the possible interaction choices it can make with the system.

Required Interface
The required interface of this module consists of the controller and model of the design for the system.

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

3.2 Communicating Module

Purpose
The purpose of this module is to provide communication services between the user client of the system and the server. This module represents the part of the communications link that resides on the user client.

Rationale
This module is created to centralize and encapsulate network communication duties between the user client and the server.

Required Interface
This module's required interface is the same as its provided interface, but it makes calls to the server's communication module via an implementation-dependent network protocol.

```
Void logIn(UserID userID, Password pass) throws NoUserException;
```

```
Boolean createUser(UserID userID, Password pass) throws
UserExistsException;
```

```
void interactWithPost(UserID userID,PostID postid, ActionType action)
throws NoUserException, NoPostException;
```

```
void commentOnPost(UserID userID, PostID postID, ActionType action,
String msg) throws NoSuchUserException, NoPostException,
UnsupportedActionException, InvalidMessageException;
```

```
void saveProfile(Profile profile);
```

```
Profile getProfile(UserID userId) throws NoSuchUserException;
```

```
void deleteProfile(UserID userId) throws NoSuchUserException;
```

```
int getNumProfiles();
```

```
void setTribeHouse(Tribe tribeID, House house) throws
UnableToSetHouse;
```

```
void setTribeName(Tribe tribeID, String name) throws nameTaken;
```

```
Post getMemberPosts(Tribe tribeID, Post memberPosts) throws
NoMemberPosts;
```

```
Post getPost(PostID postID) throws NoMatchingPostException,
PostLoadException;
```

```
Void removePost(PostID postid) throws NoMatchingPostException;
```

Provided Interface

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

Revision History

Meeting Date	Attendees	Comments
02/29/2020	Ralph Lira, David Miller, Kevyn Higbee	Initial meeting to assign tasks and define overall design elements.
03/04/2020	Ralph Lira, David Miller, Kevyn Higbee	Follow up meeting to verify progression and finalize details
03/05/2020	Ralph Lira, David Miller, Kevyn Higbee	Completed final review