DRAWIUM

By Kırmızı

Mustafa Ozan Çelik 1746544

Ozan Tahiroğlu 1502681

Özgür Saygın Bican 1752450

Seçkin Can Şahin 1631365

Table of Contents

Figure List

1. Introduction

This document is an initial design description for Drawium project. To introduce you the document, we will first give the purpose and scope of this document, then follow with an overall description of the system. After completing a general introduction to the system, we will address specific design for it.

In this document, we used “users” and “web-developers” interchangeably to refer to the users of our systems(who are presumably web-developers). We refer to the users of the web-developer’s websites as “users” but this should not cause ambiguity as it is easy to determine which users we are referring to from context.

1.1 Problem Definition

When a new user lands on a new website that he has not visited before, there is a steep learning curve. The fact that it is –at least kind of- hard to learn how to use the website costs the website owner a lot in terms of the number of users that they lose – users that just close their page or press on the back button.

In fact, this problem is so important that big corporations like Facebook and Google. They dedicate teams to solve these problems by developing custom solutions. For instance when a new user registers for a gmail account, gmail clearly indicates the button that they should use to compose an email by a box that explains it.

So what is the problem we are trying to solve if Google and Facebook already solved this aforementioned problem? The problem is that there is no toolkit for this purpose in the market for general use. The solutions those big corporations have developed over time are not open sourced and smaller websites are either forced to develop their solution in-house by devoting considerable resources or just continue losing those frustrated customers.

We aim to solve this problem by developing a very easy to integrate Javascript library and an accompanying website to customize that library.

1.2 Purpose

The purpose of this document is to give a complete description of the behavior of the Drawium project to be developed. This document is intended to establish the basis for agreement between customers and the suppliers on what the software product is to do, decrease the effort needed for development, provide a basis for validation and verification. What we are going to address will basically constitute a basis for functionality, external interfaces, performance, attributes and the design constraints of the system. This document is better suited for the customers, users, and developers .

1.3 Scope

The software we are going to introduce is named Drawium. It is a tool for web developers to help users understand their web site, and to share their designs with other developers.

1.4 Definitions and Abbreviations

Drawium: It is a drawing --and not just the output of the drawing process but the process itself-- with additional properties. Additional properties include but are not limited by the timing of the drawing(when it starts being drawn and stops) and whether it is animated or a still image.

Drawia: The plural form of Drawium.

JS Library: Javascript Library

SDD: Software Design Description Specification

1.5 References

[1] Canvas Element. Retrieved 15.12.2011 from http://en.wikipedia.org/wiki/Canvas_element

[2] Copyright. Retrieved 15.12.2011 from http://en.wikipedia.org/wiki/Copyright

[3] HTML5. Retrieved 15.12.2011 from http://en.wikipedia.org/wiki/Html5

[4] JavaScript. Retrieved 15.12.2011 from http://en.wikipedia.org/wiki/Javascript

[5] Software Requirements Specifications for Drawium Project, Team Kirmizi.

[6] Model-View-Controller (MVC) architectural pattern. Retrieved 15.12.2011 from http://msdn.microsoft.com/en-us/library/ff649643.aspx

[7] Sequence Diagrams. Retrieved 15.12.2011 from

http://www.ibm.com/developerworks/rational/library/3101.html

[8] Component Diagrams. Retrieved 15.12.2011 from http://www.agilemodeling.com/artifacts/componentDiagram.htm

[9] MVC. Retrieved 15.12.2011 from http://www.devx.com/enterprise/Article/28296/0/page/2

[10] MVC Component Diagrams in UML. Retrieved 15.12.2011 from http://aviadezra.blogspot.com/2009/08/uml-20-component-diagrams.html

1.6 Overview

Following sections of this document include the data design, system architecture design and the interface design of the drawium project.

2. System Overview

The product will be a Javascript library and an accompanying web site for the use of web developers. The product's main purpose is to provide a JS library to the users. The JS library will include the functions to help create Drawia. These functions shall take the ID of an HTML element along with other possible configuration parameters (like the time to start drawing etc.) as a parameter in order to attach a Drawium to the HTML element. For instance, there will be a function called draw_circle(id) which takes the id of the html element to cover with a circle and animates the drawing of a circle to grab the attention of the users.

Figure 1: An example image of what we are envisioning

Moreover, the system shall provide a drawing environment to the web-designers which includes basic drawing and text box creation tools.
As an example use of the text-box that aims to teach something to the visitors, have a look at this screenshot from Google Docs web interface:

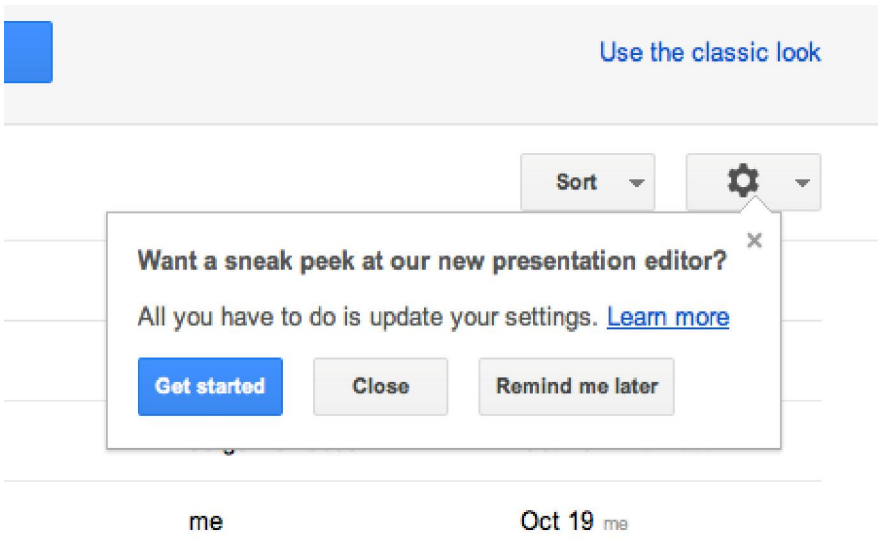

Figure 2: Screenshot from Google Docs web interface

The system will be a website with a database. It will be independent and totally self contained. Users reach to the functions that they can include in their custom JS library (the Drawia Library) via the web-interface which shall store and serve the drawia in and from the database. After each operation as input, the changes shall reflect the database in a way that will insert, remove and change the related fields. The system shall deal with the database issues, i.e searching and retrieving data from the database.

Please check the overall diagram of the system below.

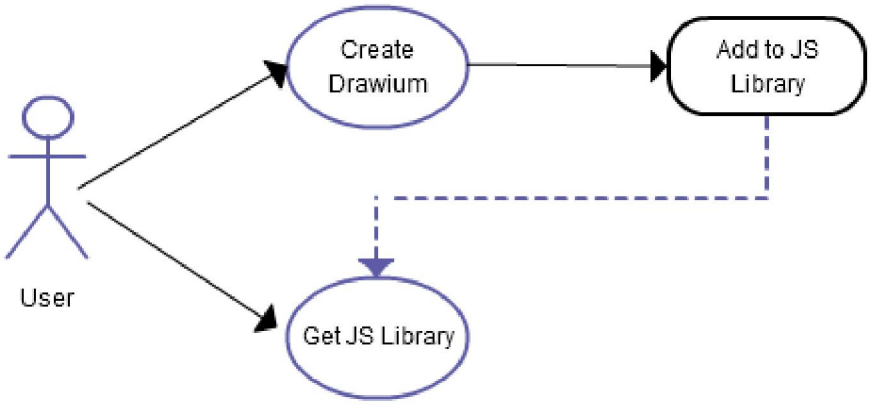

Figure 3: Overall Diagram of the system

3. Design Considerations

3.1. Design Assumptions, Dependencies and Constraints

We are planning to use an MVC framework to help us develop the system according to the MVC model. We haven't decided on a framework yet (since this is not going to change the final system, it is not urgent) and currently we are considering alternatives.

From the end-user's perspective, we are planning to make system very easy to use and we are not going to include "nice-to-have" features as those features should come later with user feedback, according to the "lean startup model" that we are following.

For the first version of our system, we are going to use a Virtual Private Server instance from Linode.com and it will have limited load capacity for that reason. We are expecting it to be able to handle up to 50 connections at the same time and we are expecting this number to be enough for long enough(since having 50 active users at a given moment is not going to be achieved very soon).

We are planning to use sophisticated methods to protect our intellectual property. In particular, we don't want people to share the JS Libraries that we created for them with other people. To be able to solve this problem, we have found solutions that will help us determine whether someone who is using our library is our registered and paying user or not, by loading invisible images from our own domain and checking where the http request is coming from. Also, we will be using code obfuscation tools to obfuscate our JS libraries. Finally, we will have our own commercial license that we distribute with the JS Libraries which will make sure that our users understand we are serious about protecting our intellectual property.

3.2. Design Goals and Guidelines

As we have mentioned above, one of the primary goals we have while designing and developing the website is that we want it to be very easy to use. But this comes with a cost. Making a website easy to use almost always come at the cost of reduced functionalities. We will try our best not to do this mistake and we will provide our users with a viable product that is easy to use.

We are not going to try to develop a highly and prematurely performance-optimized website. The product that we are envisioning is the primary goal that we are aiming to achieve and in terms of performance, we think that since our system is not going to be an infrastructure-heavy or infrastructure-oriented system, we will be able to just add new servers to the existing ones to improve the performance and increase the number of users we can handle.

4. Data Design

4.1. Data Description

This section contains a description of the classes for Drawium.com. We used the MVC model to structure our architecture, hence, we have Model, View and Controller classes associated with each major component of the website.
A description of the structure of defined data entities is provided below.

4.1.1 Description of data entities

4.1.1.1 User Component

The User component is the one that handles all the user related activities on the Drawium website. It is composed of UserController, UserModel and UserView sub-components. This component is also responsible for displaying the profiles of users.

4.1.1.1.1 UserView

Field Name	Data Type	Description
userDataView	UserDataView	Holds the basic data about the user and displays it on the page.
followView	FollowView	This is the view component that displays and holds everything related to the following mechanism on the profile page. For instance, it has “Array<UserData> followees” which is the array consisting of the basic data of the users that follow the profile owner. It also provides functionalities about following/unfollowing the profile owner user.

drawiaView	DrawiaView	This is a component to hold the drawia that is displayed on the profile page. It also provides basic functionalities about drawia, for example, making certain drawia visible/invisible on the profile page.

The methods below are provided by UserView's sub-components:

Method Name	Arguments	Return Type	Description
DisplayFollowers	Array <UserData>	Void	Returns the html to display the followers of the user
DisplayFollowees	Array <UserData>	Void	Returns the html to display the followees of the user
DisplayDrawia	Array <Drawium>	Void	Returns the html to display the drawia on the profile page that is being viewed.
DisplayUserData	UserData	Void	Translates the data of the profile owner user into html format.
FollowUser	userId	Void	Informs the UserModel about the "Follow" request made by the viewing user for the profile owner user.
UnfollowUser	userId	Void	Informs the UserModel about the "Unfollow" request made by the viewing user for the profile owner user.

4.1.1.1.2 UserController

Field Name	Data Type	Description
userData	UserData	Id and name of the user is in UserData.

Method Name	Arguments	Return Type	Description
GetBestDrawiaForUser	userId	Array <Drawium>	Given a userId, this function fetches all the related drawia that could be displayed on the profile page of the user with id equal to userId. Then, it sorts these Drawia and returns a sorted subset of those Drawia.
GetFollowers	userId	Array <UserData>	Fetches the userData of the users who follow the user whose profile is being displayed. Sorts these users if there are more than the allowed number of users that can be displayed on the profile page.

GetFollowees	userId	Array <UserData>	Fetches the userData of the users who are followed by the user whose profile is being displayed. Sorts these users if there are more than the allowed number of users that can be displayed on the profile page's followee part.
getUserData	userId	UserData	Given the id of a user, it returns the basic data about the user(email, name, id)

4.1.1.1.3 UserModel

Keeps the user-related data and returns it to UserController and UserView as needed. UserData is composed of username, password FollowingData is the DB table that holds the user follower/followee relationship data. It has "followerUserId" and "followeeUserId". DrawiumData is the table with all the drawium related data in it. It holds the creation time, creator id, drawiumId and drawium customization data(for now, this is only replay speed) in it. UserDrawiumAssocData holds the "Use This" and "Upvote" data in tables of the structure: (userId, drawiumId).

Field Name	Data Type	Description
displayDrawium	Display Drawium	This is one of the major components of the user page which wraps all the drawium related part of the page.

Method Name	Arguments	Return Type	Description
getUserData	userId	UserData	Connects to the UserData table and fetches UserData of the user which includes the data related to the user, like the name and id of the user.
GetFollowersUserData	Array<userId>	Array <UserData>	Connects to the FollowingData database table which holds the followers and followees in a table. Gets the followers' userIds from there and users getUserData to get their UserData.
getFolloweesUserData	Array<userId>	Array <UserData>	Connects to the FollowingData database table which holds the followers and followees in a table. Gets the followees' userIds from there and users getUserData to get their UserData.
getDrawiumData	drawiumId	Drawium	Fetches all the data about the given drawium from the DrawiumData table.

getDrawiumsCreatedBy User	userId	Array <drawiumId>	Connects with the UserDrawiumAssocData table and fetches the ids of all the drawiums created by a user.
getDrawiumsUpvoted ByUser	userId	Array <drawiumId>	Connects with the UserDrawiumAssocData table and fetches the ids of all the drawiums upvoted by a user.
getDrawiumsUsedBy User	userId	Array <drawiumId>	Connects with the UserDrawiumAssocData table and fetches the ids of all the drawiums used by a user.
FollowUser	followerUserId, followeeUserId	Void	Connects to the FollowingData table and inserts a new row with followerUserId, followeeUserId.
UnfollowUser	followerUserId, followeeUserId	Void	Connects to the FollowingData table and removes the row with followerUserId, followeeUserId.

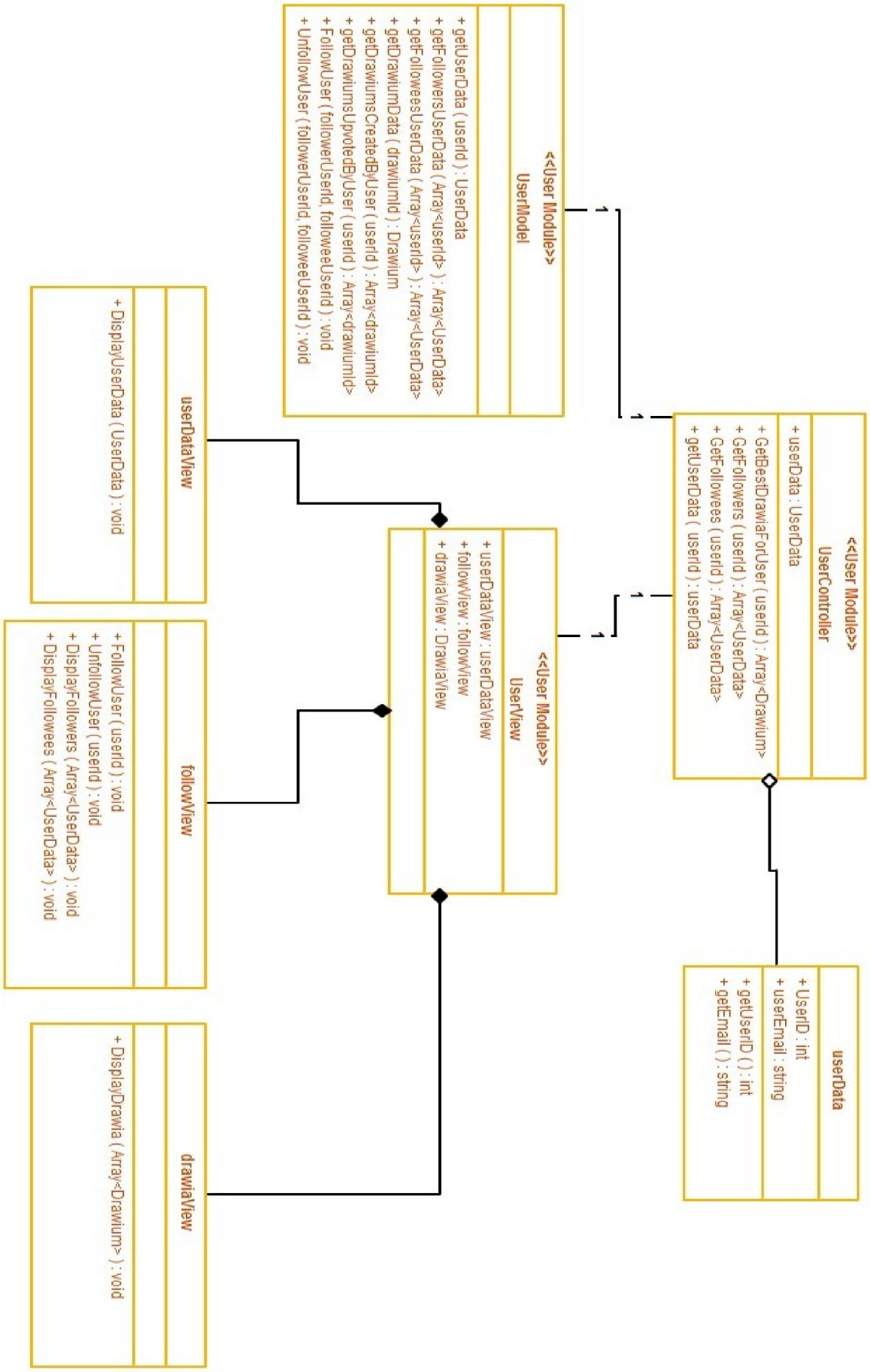

Figure 4: Class Diagram of User(Profile)

4.1.1.2 CreateDrawium Component

This is the component that handles functionalities like displaying the drawium creation page, capturing the drawing made by the user and sending it to the database once it is finished to make it permanent.

4.1.1.2.1 CreateDrawiumView

Field Name	Data Type	Description
capturedEvents View	CapturedEvents View	We capture the events that the user does on the canvas on the CreateDrawium Page. For example, when a user clicks on the canvas to draw a lines, that click event is captured to help build the drawium user is trying to create. This will be sent to the controller later to be converted into the final drawium which will then send it to the model to save.
setAttributes View	SetAttributesView	This holds the drawium configuration data like the replay speed of the drawing.
contructDrawiu mView	Construct DrawiumView	This is the part of the page that wraps everything related to creating and configuring the drawium.
drawingTools View	DrawingTools View	This is the component that is responsible for displaying the drawing tools.

The functions below are provided by the subcomponents of the CreateDrawiumView(which are given as the fields above.)

Method Name	Arguments	Return Type	Description
captureMouseDown	Struct{x,y}	Void	This is called when the user clicks on any point on the canvas.
captureMouseUp	Struct{x,y}	Void	This is called when the user releases the mouse.
captureMouseMoved	Struct{x,y}	Void	This function is called when the user moves the mouse on the canvas.
saveDrawium	Array<Event>	Boolean	This is called to send the events to the controller for the purpose of saving the current state of the drawium(which could or could not be the final state.)
setAttribute	name, value	Void	This is called to set the attribute “name” to “value”.
displayAttributeView	Void	Void	Displays the part of the page where the user can set the attributes.
displayDrawingTools View	Void	Void	Displays the part of the page where the user can see the drawing tools.

4.1.1.2.2 CreateDrawiumController

Field Name	Data Type	Description
session	Session	We attach a unique session to all the drawium creation pages that are viewed.
capturedEvents	Drawing	This holds all the events(mouse clicks and moves) happening in the session, on the canvas of the page.
attributes	Array<Attribute>	This holds the configuration data for the drawium. For instance its replay speed is held in this component.

Method Name	Arguments	Return Type	Description
saveDrawium	Array<Event>	Boolean	This is called with the array of the events created by the view. It transforms the events to a valid drawium object and sends it to the model to save.
convertToDrawium	Array<Event>, Array<Attribute>	Drawium	Converts the event array and attribute array to a drawium object. This is called by the saveDrawium function.

4.1.1.2.3 CreateDrawiumModel

Method Name	Arguments	Return Type	Description
saveDrawium	Draiwum	Void	This function saves the drawium to the database. This also connects to the UserDrawiumAssocData table and adds a row to put the information that the user created the given drawium.

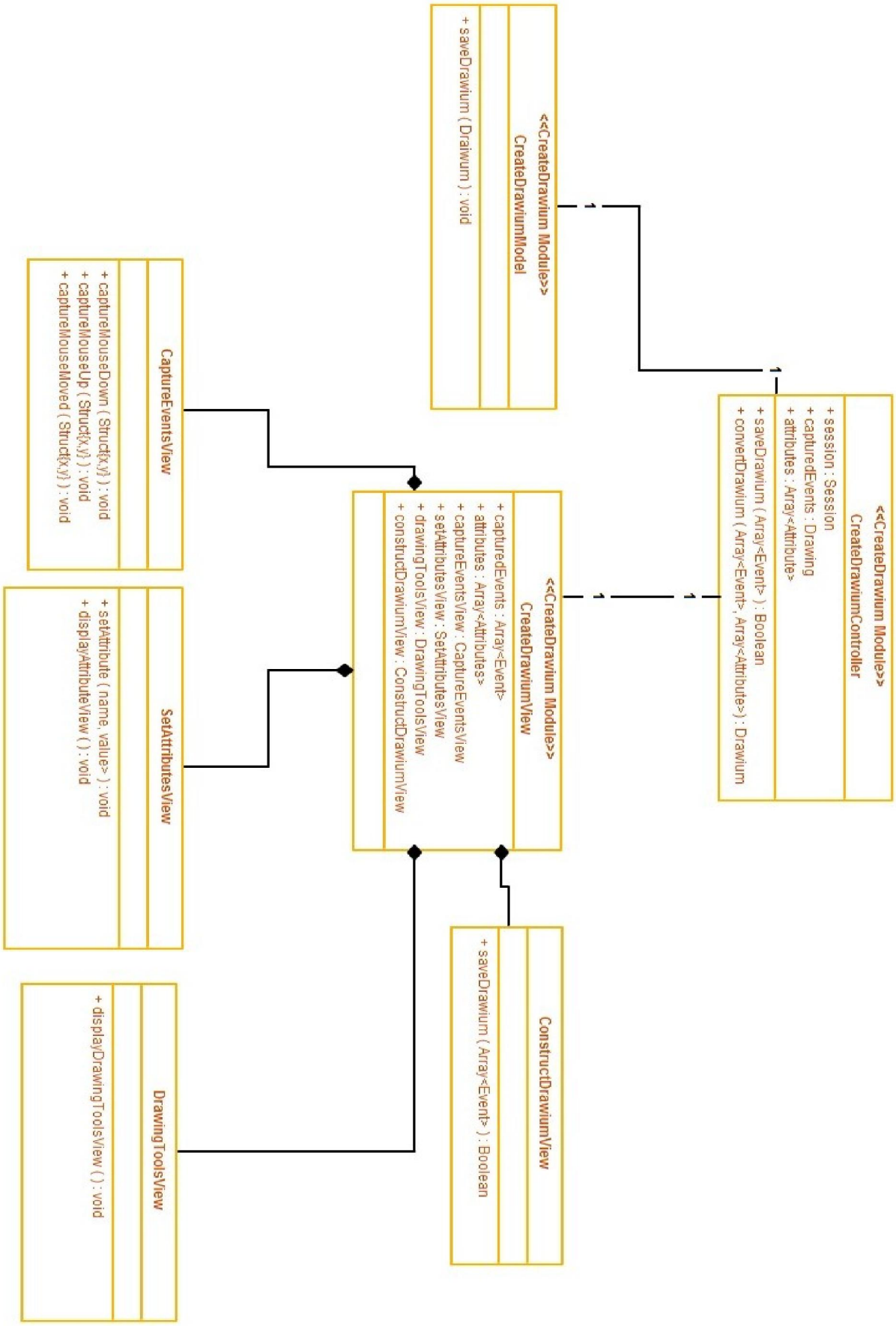

Figure 5: Class Diagram of CreateDrawium

4.1.1.3 CreateJSLibrary Component

This component displays the page which is used by the user to choose the Drawia to include in the Js library that he/she wants to create. Once this is done, the user is presented with the full Drawium library.

4.1.1.3.1 CreateJSLibraryView

Field Name	Data Type	Description
availableDrawia	Array<Drawium>	This holds the drawia available to the user for the purposes of creating his own custom JS Library(Drawia that the user upvoted, created or clicked “Use This” for are available here)
chosenDrawia	Array<Drawium>	The drawia that are chosen by the user out of all the available Drawia to be included in the Library.
selectView	SelectView	Displays the checkboxes that the user will use to select/unselect the drawiums to include in the JS Library.
libContructView	LibConstructView	This displays the current JS Library that the user is working on and is automatically updated when the user selects/unselects drawia to use.

The functions below are provided by the subcomponents of the createJSLibraryView(which are given as the fields above.)

Method Name	Arguments	Return Type	Description
convertDrawiumToJS	Drawium	String	Converts the given drawium object to javascript code to be included in the library.
displayJSLibrary	Array<Drawium>	Void	Calls the convertDrawiumToJS function on all the elements of the array argument. Then wraps them into a proper JS Library and displays it to the user on the page.
displayDrawiaView	Array<Drawium>	Void	Displays the draiwums that are available to the user to be used in the Library.

4.1.1.3.2 CreateJSLibraryController

Field Name	Data Type	Description
userData	UserData	Basic user data.
availableDrawium	Array<Drawium>	This array holds all the drawium that the user can use in his library.

Method Name	Arguments	Return Type	Description
fetchAvailable Drawium	userId	Array <Drawium>	Fetches all the available drawium that the user can use from the model.

4.1.1.3.3 CreateJSLibraryModel

Method Name	Arguments	Return Type	Description
fetchAvailable Drawium	userId	Array <Drawium>	Fetches all the available drawium that the user can use from the model. This includes drawia created, and “use this”’d by the user.
saveLibrary	JSLibrary	Void	This function saves the given JSLibrary into the JSLibraryData table.

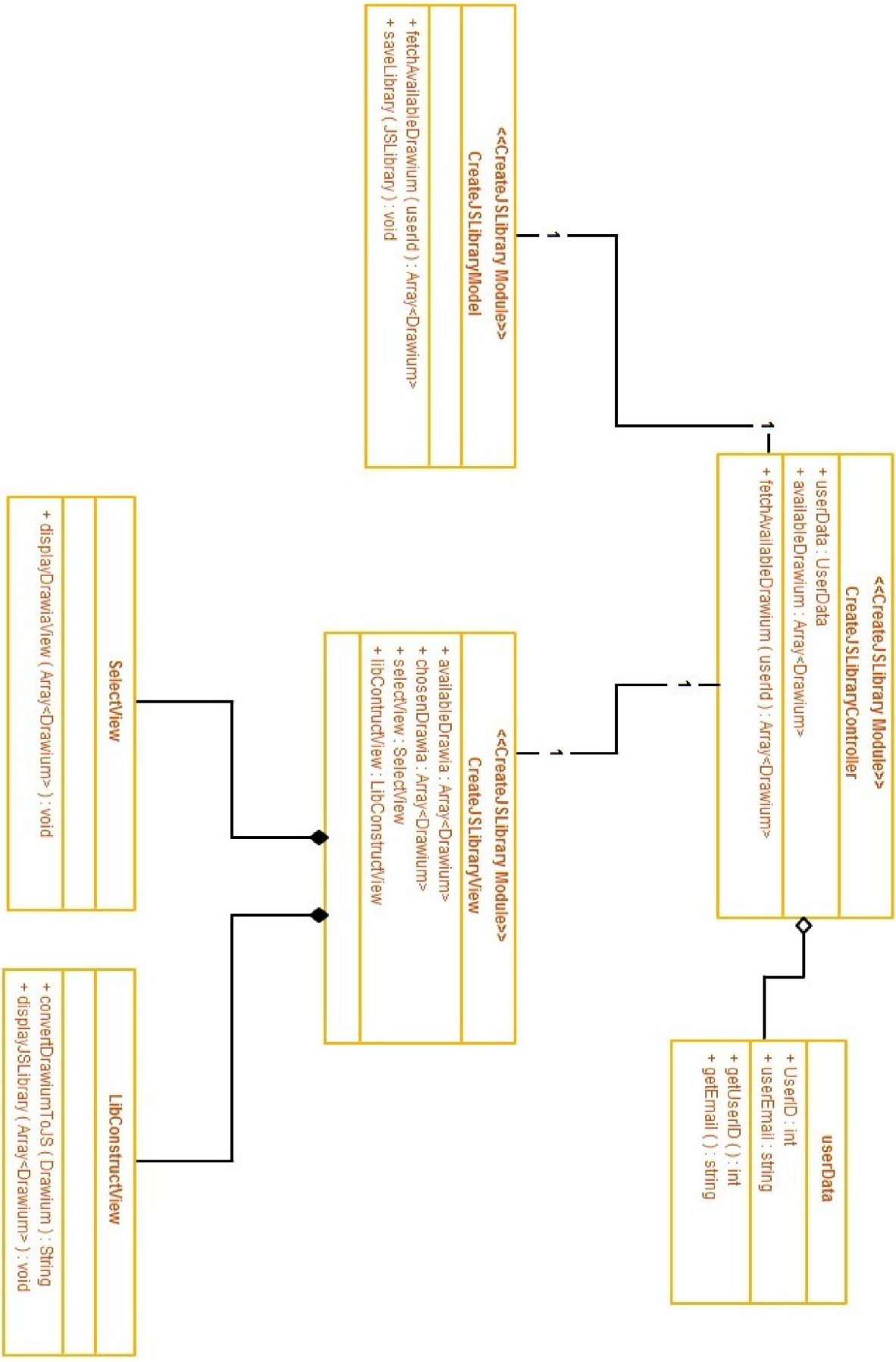

Figure 6: Class Diagram of CreateJSLibrary

4.1.1.4 DisplayDrawium Component

This component is being used by the profile page viewing functionality and news feed displaying functionality. It is used when an individual Drawium or multiple Drawia needs to be displayed. It provides all the Drawia-related functionalities, for instance, upvoting, clicking “use this”, clicking on the drawium to see the animation etc.

4.1.1.4.1 DisplayDrawiumView

Field Name	Data Type	Description
drawium	Drawium	The drawium object to be displayed.

The functions below are provided DisplayDrawiumView and its subcomponents(which are given as the fields above.)

Method Name	Arguments	Return Type	Description
displayDrawium	Drawium	Void	Drawium object with all its additional functionalities(like the upvote and use this functions) is made visible on the page by calling this function, which is also responsible for creating the html of the drawium object.
captureUpvote	drawiumId	Void	Captures the event when user clicks on the upvote button for a drawium.
captureUseThis	drawiumId	Void	Captures the event when a user clicks on Use This Button.
playDrawium	Drawium	Void	Starts the animation of the drawium to be seen by the user.

4.1.1.4.2 DisplayDrawiumController

This is essentially an empty class as of now, since the view only does basic data fetches and data saves directly from and to the model.

4.1.1.4.3 DisplayDrawiumModel

Method Name	Arguments	Return Type	Description
fetchDrawium	drawiumId	Drawium	Returns the drawium object it fetches from the drawium database with id = drawiumId.

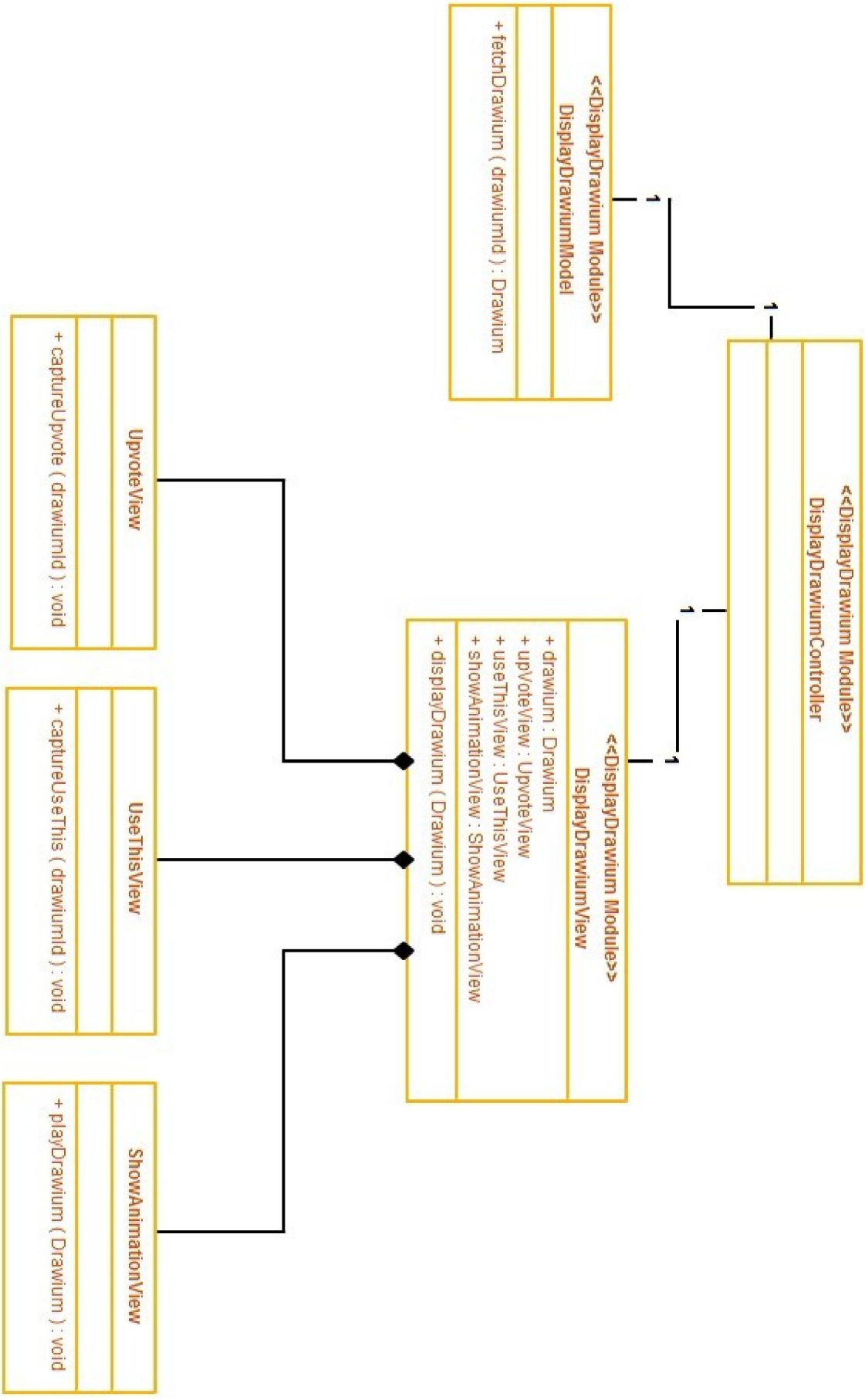

Figure 7: Class Diagram of DisplayDrawium

4.1.1.5 NewsFeed Component

This is the component which is responsible for displaying the Newsfeed, capturing all the data a user can put into the Newsfeed and sending it to the database.

4.1.1.5.1 NewsFeedView

Field Name	Data Type	Description
drawiaView	DrawiaView	This is the wrapper for all the drawium to be displayed on the newsfeed page. DrawiaView is a wrapper for DraiwumView we have mentioned above.
userData	UserData	Keeps the id, email and name of the user.

Method Name	Arguments	Return Type	Description
displayDrawia	Array <Drawium>	Void	This function is provided by drawiaView. It displays all the drawiums on the page.
displayDrawium	Draiwum	Void	Displays a single Drawium given, on the page.

4.1.1.5.2 NewsFeedController

Method Name	Arguments	Return Type	Description
getNewsfeedDrawia	userId	Array <Drawium>	Connects to the model to fetch the drawia that can be displayed on the user's news feed. Then, it sorts them according to time and returns the array.

4.1.1.5.3 NewsFeedModel

Method Name	Arguments	Return Type	Description
getNewsfeedDrawia	userId	Array <Drawium>	This is called by the getNewsfeedDrawia function of the controller. It gets all the drawiums created, upvoted and "use this"d by the followees of the user and returns them in an array.
getFollowees	userId	Array <userId>	Gets all the followees of the user denoted by the userId argument and returns their ids in an array.
multigetDrawia	Array<userId>	Array <Drawium>	Gets all the drawium created, upvoted and "use this"d by the users in the argument array.

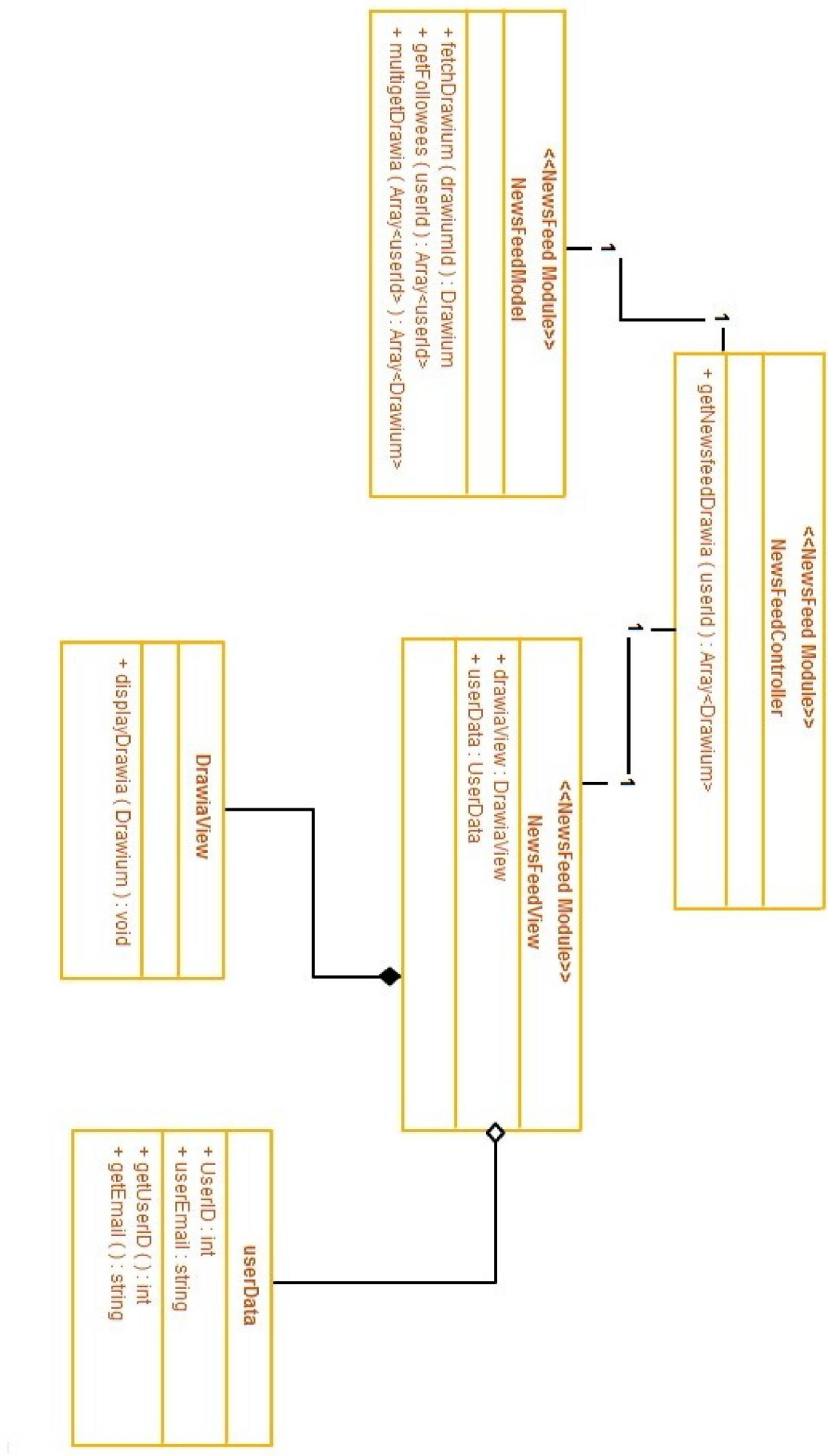

Figure 8: Class Diagram of NewsFeed

4.2. Data Dictionary

Name	Type	Refer to Section
UserDataView	Component	4.1.1.1.1
FollowView	Component	4.1.1.1.1
DrawiaView	Component	4.1.1.1.1
UserData	Data Entity	4.1.1.1.2
CapturedEventsView	Component	4.1.1.2.1
SetAttributesView	Component	4.1.1.2.1
ConstructDrawiumView	Component	4.1.1.2.1
DrawingToolsView	Component	4.1.1.2.1
Session	Component	4.1.1.2.2
Drawing	Data Entity	4.1.1.2.2
Attribute	Data Entity	4.1.1.2.2
SelectView	Component	4.1.1.3.1
LibConstructView	Component	4.1.1.3.1
UpvoteView	Component	4.1.1.4.1
UseThisView	Component	4.1.1.4.1
ShowAnimationView	Component	4.1.1.4.1
DrawiaView	Component	4.1.1.5.1
User	Component	4.1.1.1
UserView	Component	4.1.1.1.1
UserController	Component	4.1.1.1.2
UserModel	Component	4.1.1.1.3
CreateDrawium	Component	4.1.1.2
CreateDrawiumView	Component	4.1.1.2.1
CreateDrawiumController	Component	4.1.1.2.2
CreateDrawiumModel	Component	4.1.1.2.3
CreateJSLibrary	Component	4.1.1.3
CreateJSLibraryView	Component	4.1.1.3.1
CreateJSLibraryController	Component	4.1.1.3.2
CreateJSLibraryModel	Component	4.1.1.3.3
DisplayDrawium	Component	4.1.1.4
DisplayDrawiumView	Component	4.1.1.4.1
DisplayDrawiumController	Component	4.1.1.4.2
DisplayDrawiumModel	Component	4.1.1.4.3
NewsFeed	Component	4.1.1.5
NewsFeedView	Component	4.1.1.5.1
NewsFeedController	Component	4.1.1.5.2
NewsFeedModel	Component	4.1.1.5.3

5. System Architecture

A general description of the Drawium software system architecture is presented in the following sections.

5.1. Architectural Design

In our architectural design, Model View Controller architecture is used. This architecture is currently considered an architectural pattern used in software engineering. Model View Controller pattern provides the separation of applications --input logic, UI logic and business logic respectively-- and it provides loose coupling between these elements.

In our architectural design, the subcomponents of model of our modules are abstract and logical components, which means these subcomponents do not physically exits. The view component of our modules are to capture inputs from user interface and to displaying pages of our system.

5.1.1 Overall Description

The Model View Controller architecture of our system is composed of five components, each formed of component grouped according to their functional similarity. In this section, the brief overview of the responsibilities of each of these components are provided. The UserView and NewsFeed component is dependent on **DisplayDrawiumController** component, because the created drawium of a user or the shared drawium of a user is displayed to the user by the help of the **DisplayDrawiumController** component. The controller basically retrieves the data from the model component by its own interface and sets this data to view component by the interface of view. On the user profile page, a user may want to create a drawium or create a JS library. In order to respond to these requests, the User(Profile) component triggers the **CreateJSLibraryController** component and the **CreateDrawiumController** component through its own controller. In this way, the controllers starts to interact with its subsystems and displays the CreateJSlibrary page and CreateDrawium page to the user which is also explained in detail in the Description of Components section.

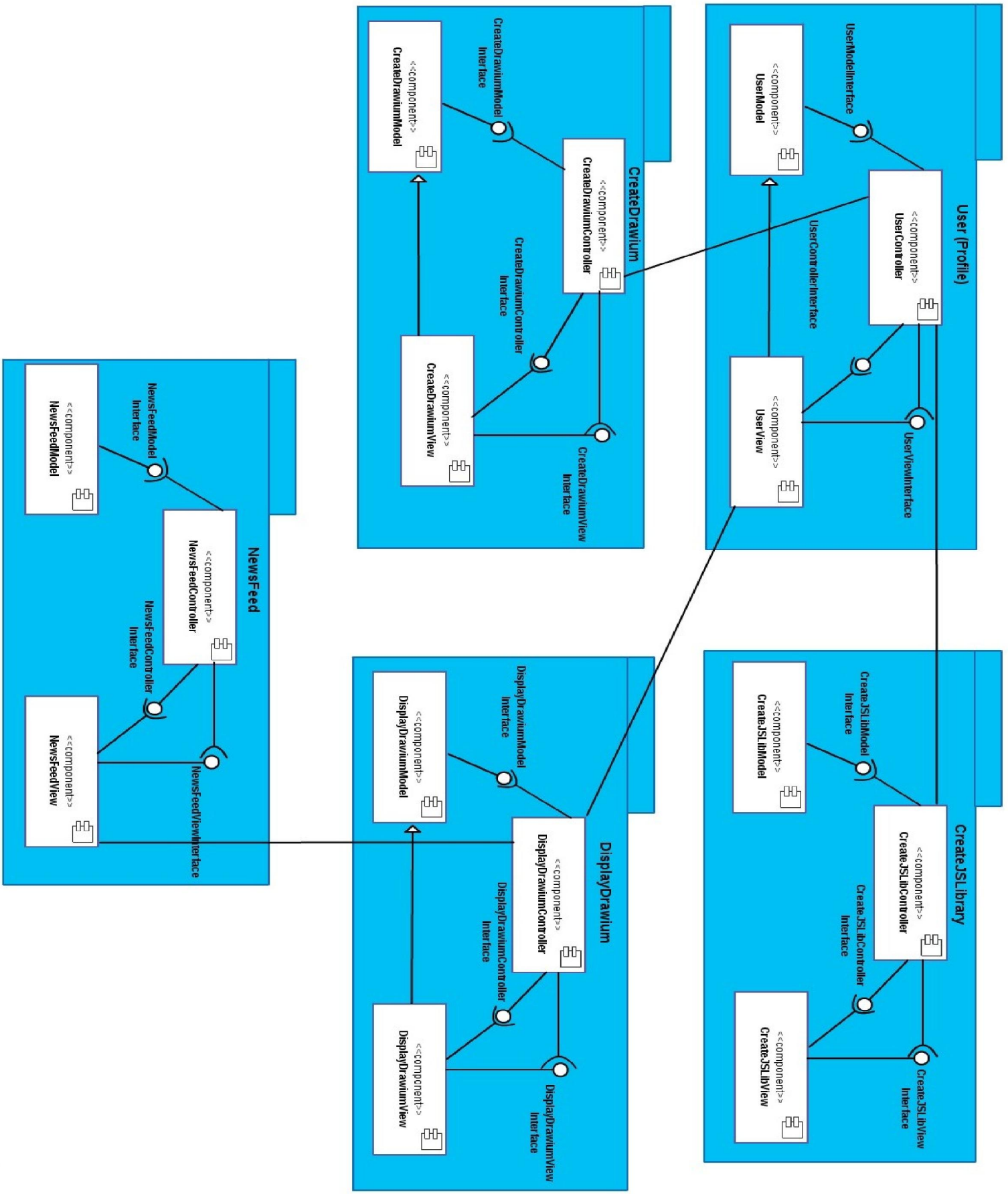

Figure 9: Overall Component Diagram

5.2. Description of Components

5.2.1. Component User(Profile)

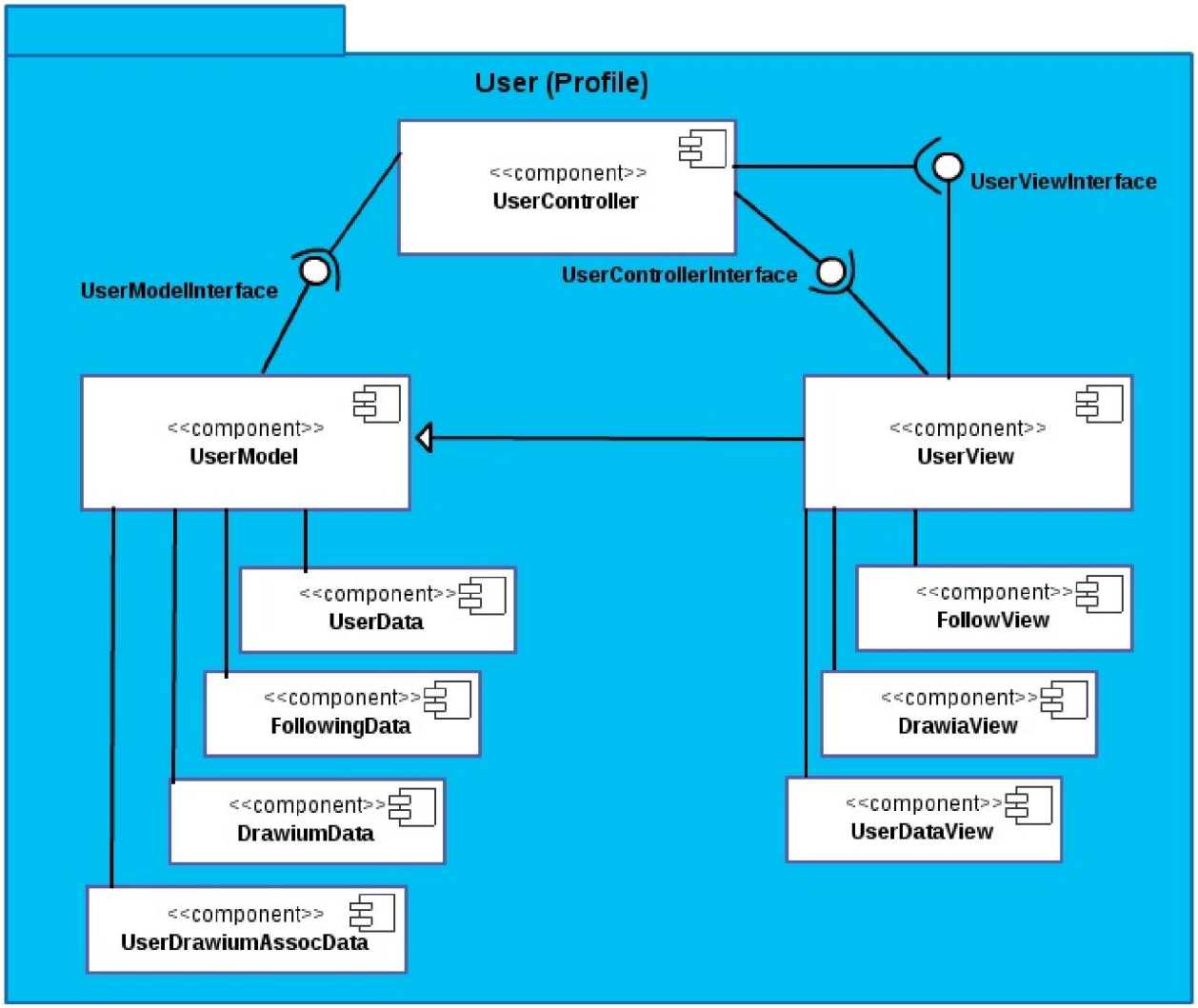

Figure 10: User(Profile) Component Diagram

5.2.1.1. Processing Narrative for Component User(Profile)

The User(Profile) component provides the necessary functionalities for user profile, such as following/unfollowing, seeing drawia. These are **FollowView**(to show followers/followees and provide follow/unfollow functionalities) component, **ProfileDrawiumView**(to show the drawia on the profile) component and **UserDataView**(viewing the user's name, e-mail) component. These view-based components retrieves necessary data from model via controller or directly from model and sets the **UserDataView** , **FollowingDataView** and **DrawiumDataView** for displaying user's drawia at his/her profile and providing the functionalities related with following mechanism. **UserController** component provides necessary information to **UserView** component by taking these information from the components of **UserModel** component.

5.2.1.2. Component User(Profile) Interface Description

UserViewInterface: allows **UserController** component to set the fields of **UserView** component.

UserControllerInterface: allows the **UserView** component to send request to **UserController**.

UserModelInterface: provides information which is needed by **UserController** component to the **UserController** component. In addition, **UserController** component may set the fields of **UserModel** component through this interface.

5.2.1.3. Component User(Profile) Processing Detail

To show the User Profile, **UserView** component is supported by **UserController**. When a request to displaying user profile is sent, **UserController** prepares the follower/followee information, the list of drawia information and user's data information. **UserController** component uses the **UserModel** component directly to update the current information of user profile. In addition, **UserView** component can access the **UserModel** component directly and make changes on it. Finally, **UserView** component shows the user profile based on the information received.

5.2.1.4. Dynamic Behaviour Component User(Profile)

5.2.1.4.1 Displaying Profile Page

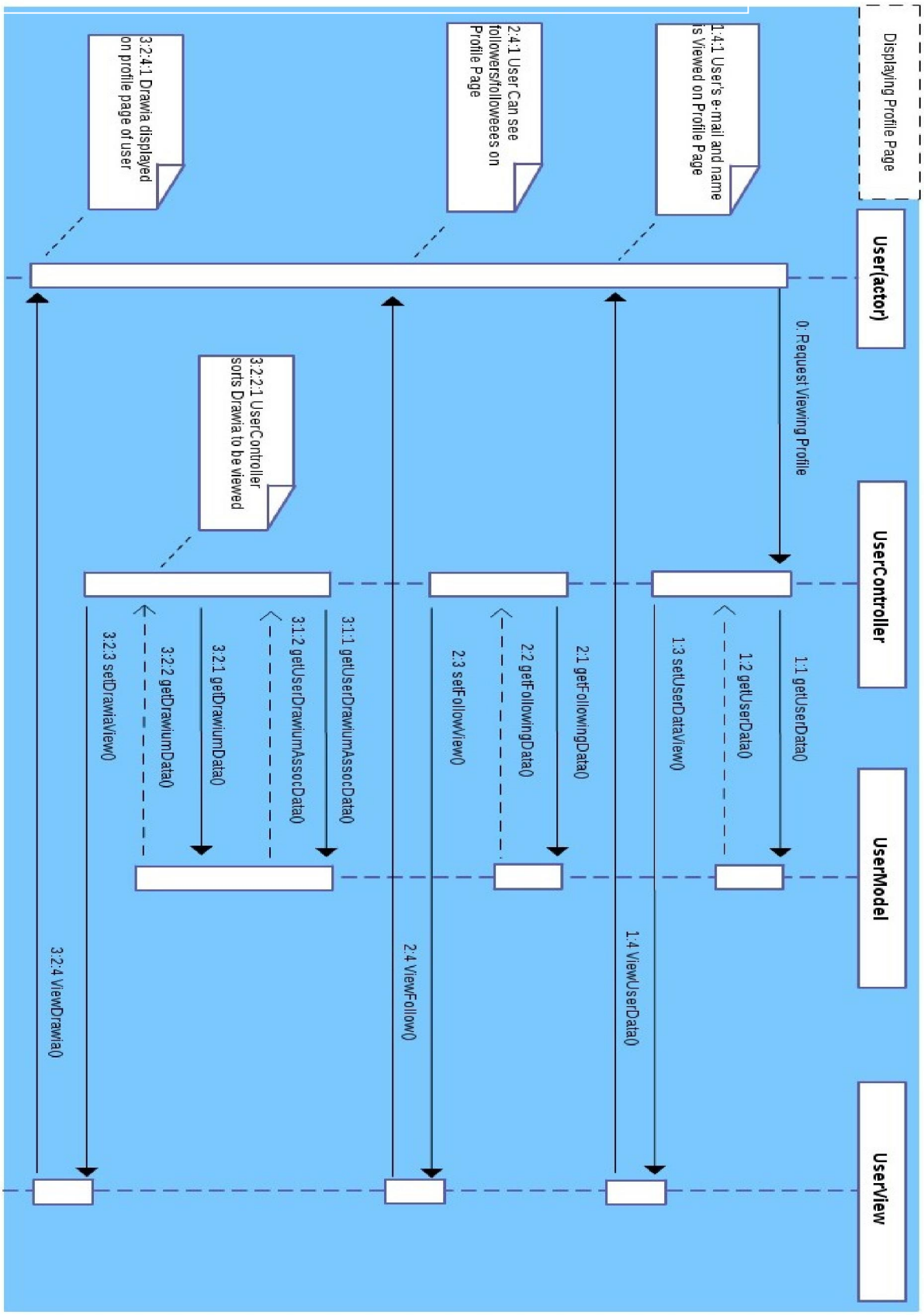

Figure 11: Displaying Profile Page Sequence Diagram

The **UserModel** component first passes user data to the **UserController** component(as a response to **UserController**'s request) when a user wants to display his/her profile page. Controller sets this information to the **UserView** component and user's data is displayed on the profile page. Then **UserController** component sends request to the **UserModel** component to retrieve following data and the **UserModel** component passes following data to the **UserController** component and controller sets this information to the **UserView** component. The following data is displayed on the profile page. The controller retrieves user drawium association data, **UserModel** component supplies it, and also **UserController** component retrieves the drawium data of user. Then, the **UserController** component sorts the drawia to be viewed and sets this information to **UserView** component. Hence, drawia displayed on the profile page of user.

5.2.1.4.2 Follow / Unfollow

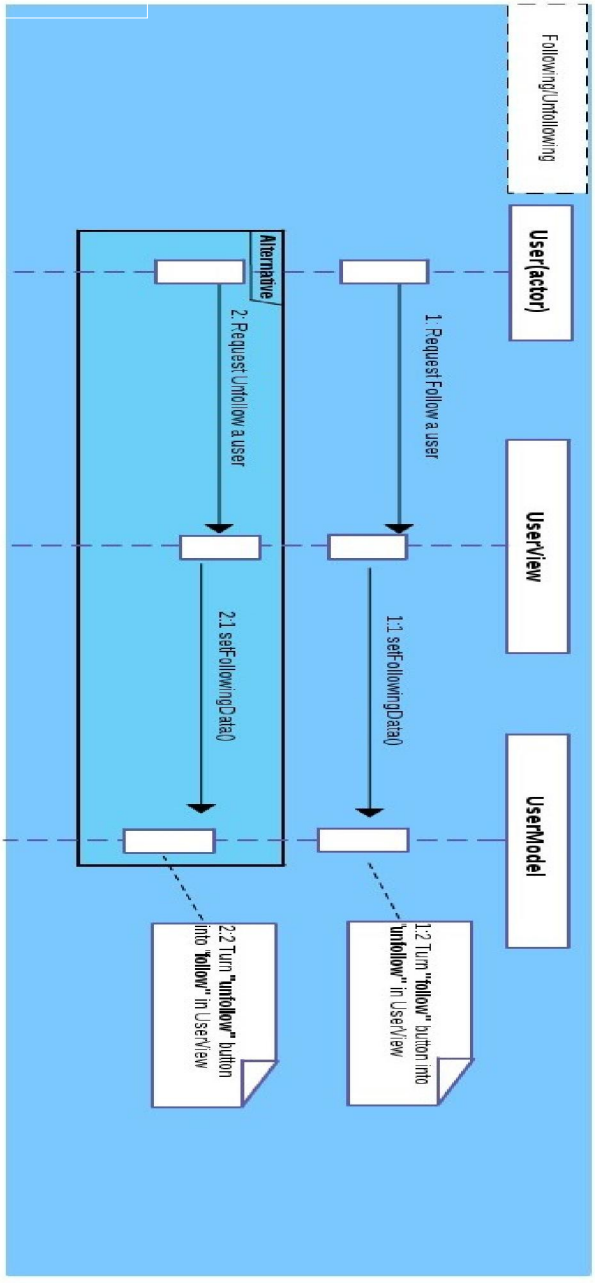

Figure 12: Following/Unfollowing Sequence Diagram

When a user wants to follow another user, it means that user sends a user follow request to the **UserView** component. The **UserView** component takes this input and passes the following data to the **UserModel** component and the follow button is turned into unfollow button. Alternatively, the user may want to unfollow another user, then it means that user sends a user unfollow request to the **UserView** component. The **UserView** component takes this input and passes the unfollowing data to the **UserModel** component.

5.2.2. Component CreateDrawium

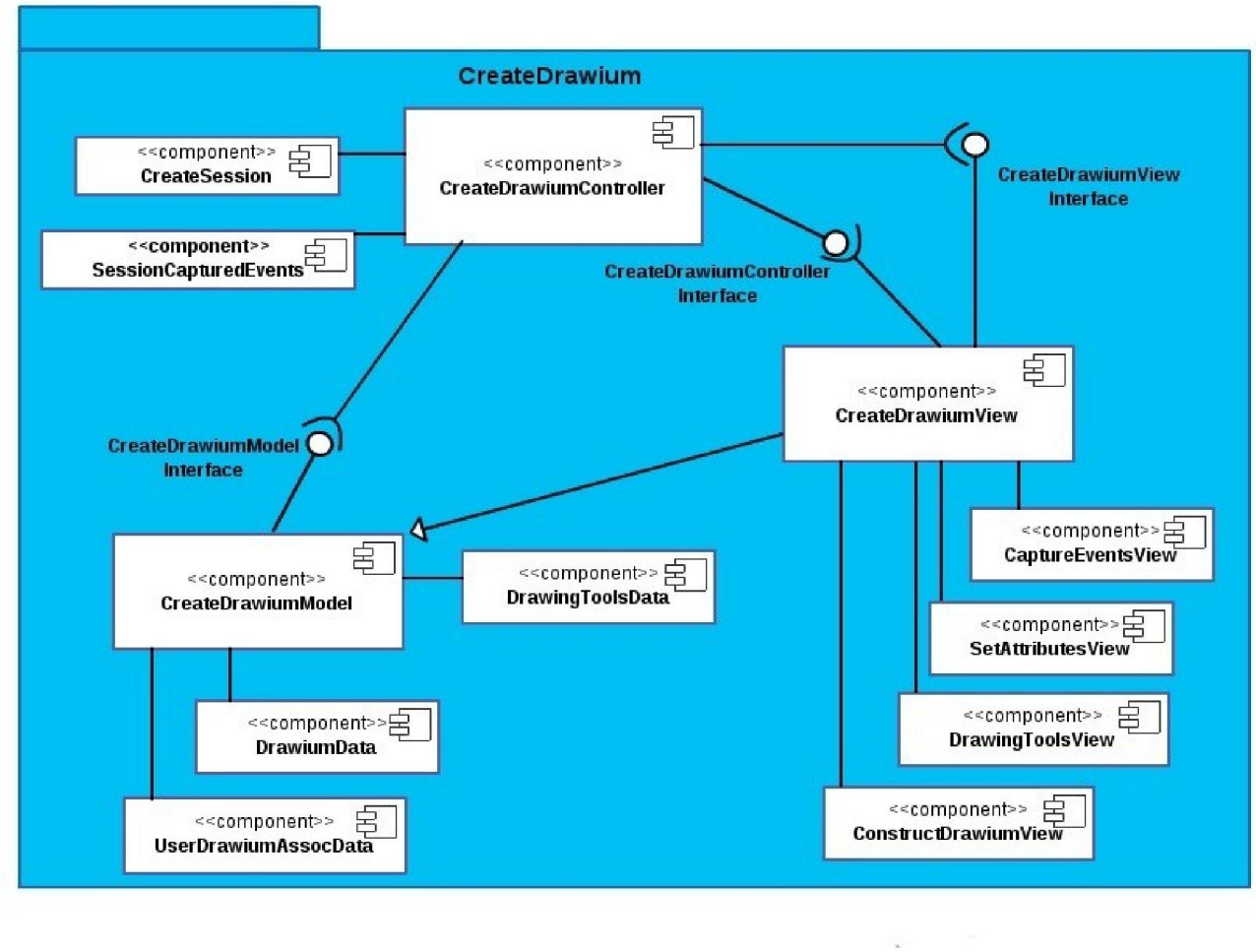

Figure 13: CreateDrawium Component Diagram

5.2.2.1. Processing Narrative for Component CreateDrawium

The **CreateDrawium** component contains the components responsible for creating drawium. The **DrawingToolsData** component offers tools to draw a drawium. The **CaptureEventView** component is responsible for capturing the informations of current drawing of drawium, such as coordinates of drawium with time information. The **SessionCaptureEvent** component contains these information until the drawing ends. The **SetAttributesView** component offers to edit attributes of the drawium. The **ConstructDrawiumView** component is responsible for saving drawium. The **CreateDrawiumView** component offers user-directed view capabilities.

5.2.2.2. Component CreateDrawium Interface Description

CreateDrawiumViewInterface: allows current info of corresponding components to be set to the **CreateDrawiumView** component.

CreateDrawiumControllerInterface: allows the **CreateDrawiumView** component to send request to **CreateDrawiumController**.

CreateDrawiumModelInterface: provides information which is needed by **CreateDrawiumController** component to the **CreateDrawiumController** component. In addition, **CreateDrawiumController** component may set the fields of **CreateDrawiumModel** component through this interface.

5.2.2.3. Component CreateDrawium Processing Detail

When a user displays the create drawium page, **CreateDrawiumController** component retrieves the **DrawingToolData** from **CreateDrawiumModel** and supplies it to **DrawingToolView** component and drawing screen is displayed. The **CreateDrawiumController** component opens up a session through **CreateSession** component. While the user is drawing, the **CaptureEventView** component iteratively captures the coordinates (with time information) of drawium and these informations are transfered to **CreateDrawiumController** component. This event continues until the drawing process ends. The **SetAttributesView** component allows to modify the attributes of the drawium. To save the drawium with selected attributes, the **ConstructDrawiumView** component supplies data of drawium through the **CreateDrawiumControllerInterface** and **CreateDrawiumController** component saves the drawium into the database with selected attributes. Finally, ID of drawium returns to the **CreateDrawiumView** component and the drawium is added into the **UserDrawiumAssocData** component.

5.2.2.4. Dynamic Behaviour Component CreateDrawium

5.2.2.4.1 Displaying Create Drawium Page

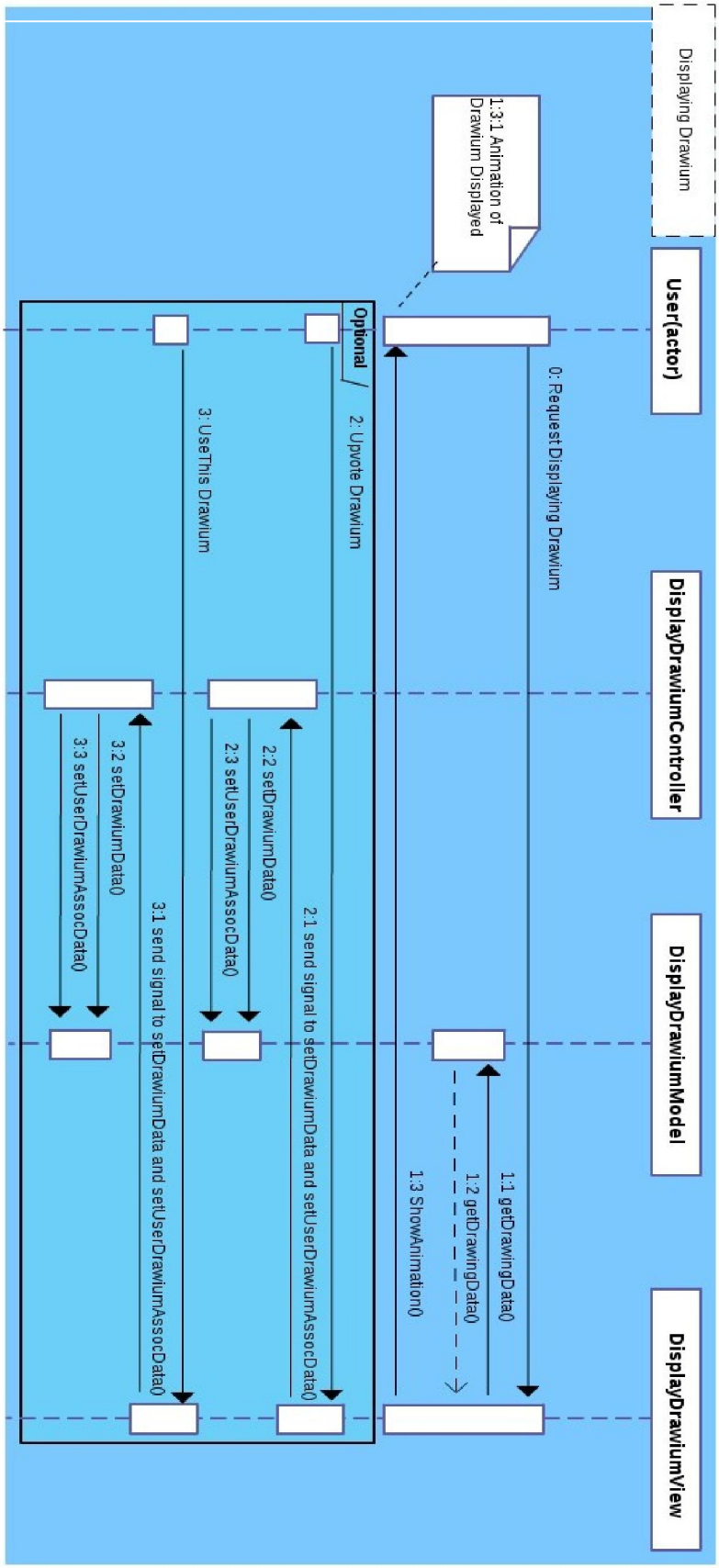

Figure 14: Displaying Create Drawium Page Sequence Diagram

When a user opens up the create drawium page, the signal is transmitted to the **CreateDrawiumController** component and the controller retrieves the drawing tools data from the **CreateDrawiumModel** component. The **CreateDrawiumController** component sets drawing tools to the **CreateDrawiumView** component and drawing tools are displayed on create drawium page. The controller can create more than one session if it is required by the user.

5.2.2.4.2. Creating Drawium

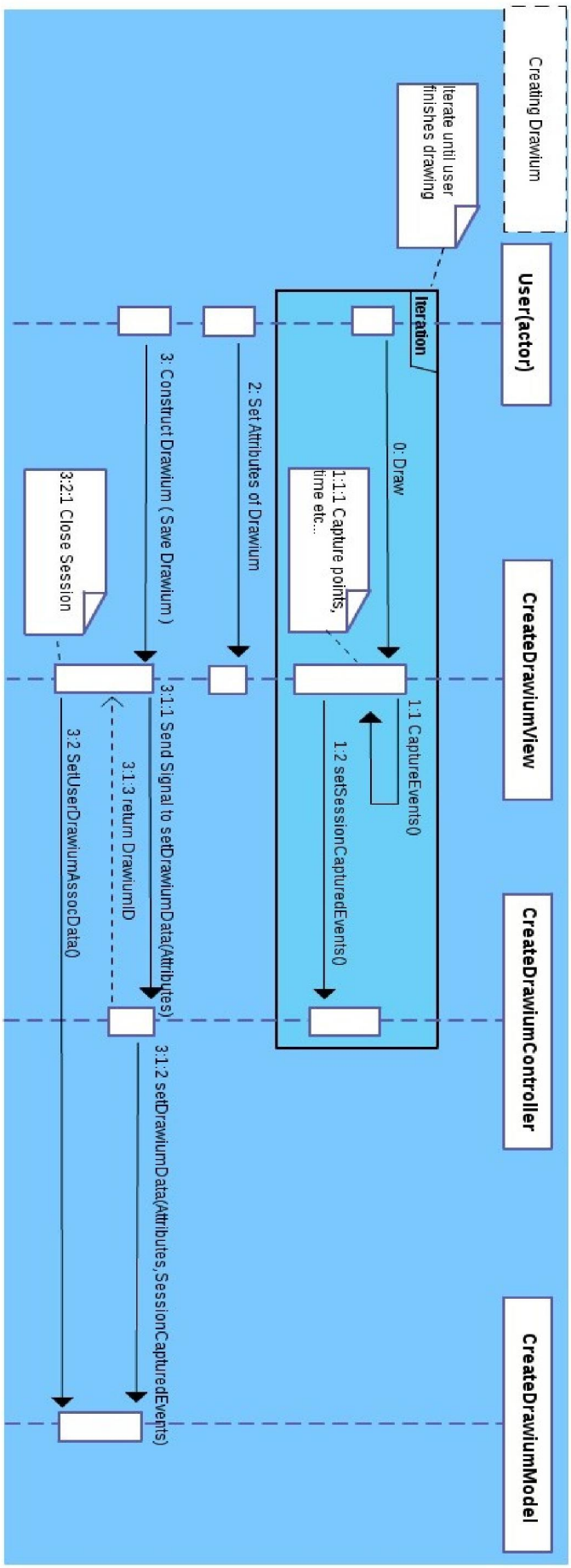

Figure 15: CreatingDrawium Sequence Diagram

When a user starts drawing a drawium, **CreateDrawiumView** component iteratively captures the events and transfers to the **CreateDrawiumController** component. The events are stored in **SessionCapturedEvents** component. Until user finishes drawing, this iteration continues. After being set attributes of the drawium by the user, in order to save the drawium, the **CreateDrawiumView** component sends a signal to the **CreateDrawiumController** component. The controller saves the drawium to database and returns the ID of new drawium to **CreateDrawiumView.** Finally user drawium association data is updated and the session is closed.

5.2.3. Component CreateJSLibrary

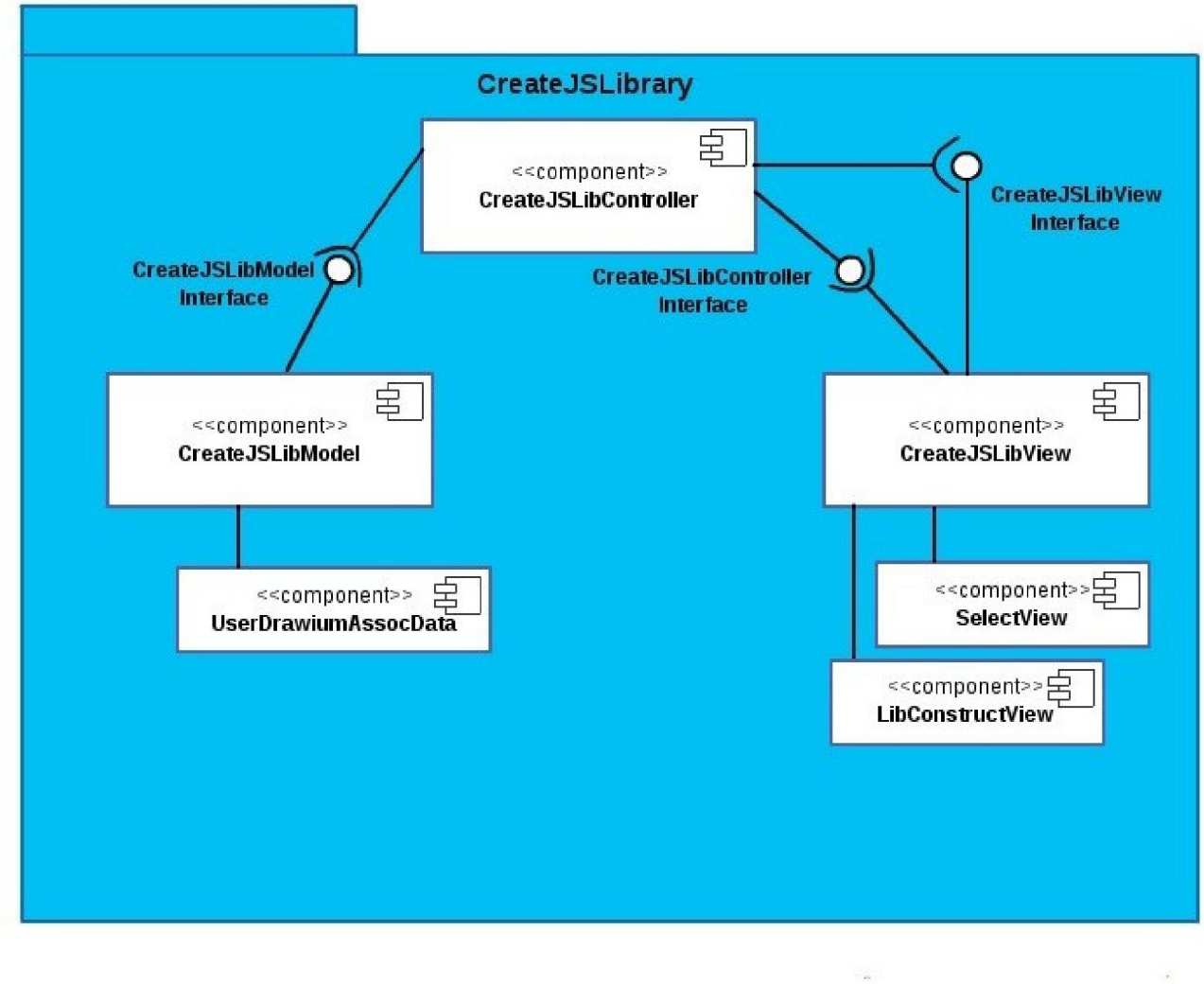

Figure 16: CreatingJSLibrary Component Diagram

5.2.3.1. Processing Narrative for Component CreateJSLibrary

CreateJSLibrary component contains the components responsible for selecting drawia and constructing JS Library. **CreateJSLibController** component allows the **CreateJSLibView** component to have available drawia by the help of **CreateJSLibModel** component. The **SelectView** component captures the selected drawia. Finally, construction of library handled by **LibConstructView** component.

5.2.3.2. Component CreateJSLibrary Interface Description

CreateJSLibInterface: allows current info of corresponding components to be set to the **CreateJSLibView** component.

CreateJSLibControllerInterface: allows the **CreateJSLibView** component to send request to **CreateJSLibController**.

CreateJSLibModelInterface: provides information which is needed by **CreateJSLibController** component to the **CreateJSLibController** component. In addition, **CreateJSLibController** component may set the fields of **CreateJSLibModel** component through this interface.

5.2.3.3. Component CreateJSLibrary Processig Detail

Initially, the **CreateJSLibController** component retrieves the available drawia (available means, to be usable of that drawium for that user) from **CreateJSLibModel** component through **CreateJSLibModelInterface** and supplies them to **CreateJSLibView** component. The selections are captured by **SelectView.** When the selection process is finished, the JS library may be constructed by **LibConstructView** component.

5.2.3.4. Dynamic Behaviour Component CreateJSLibrary

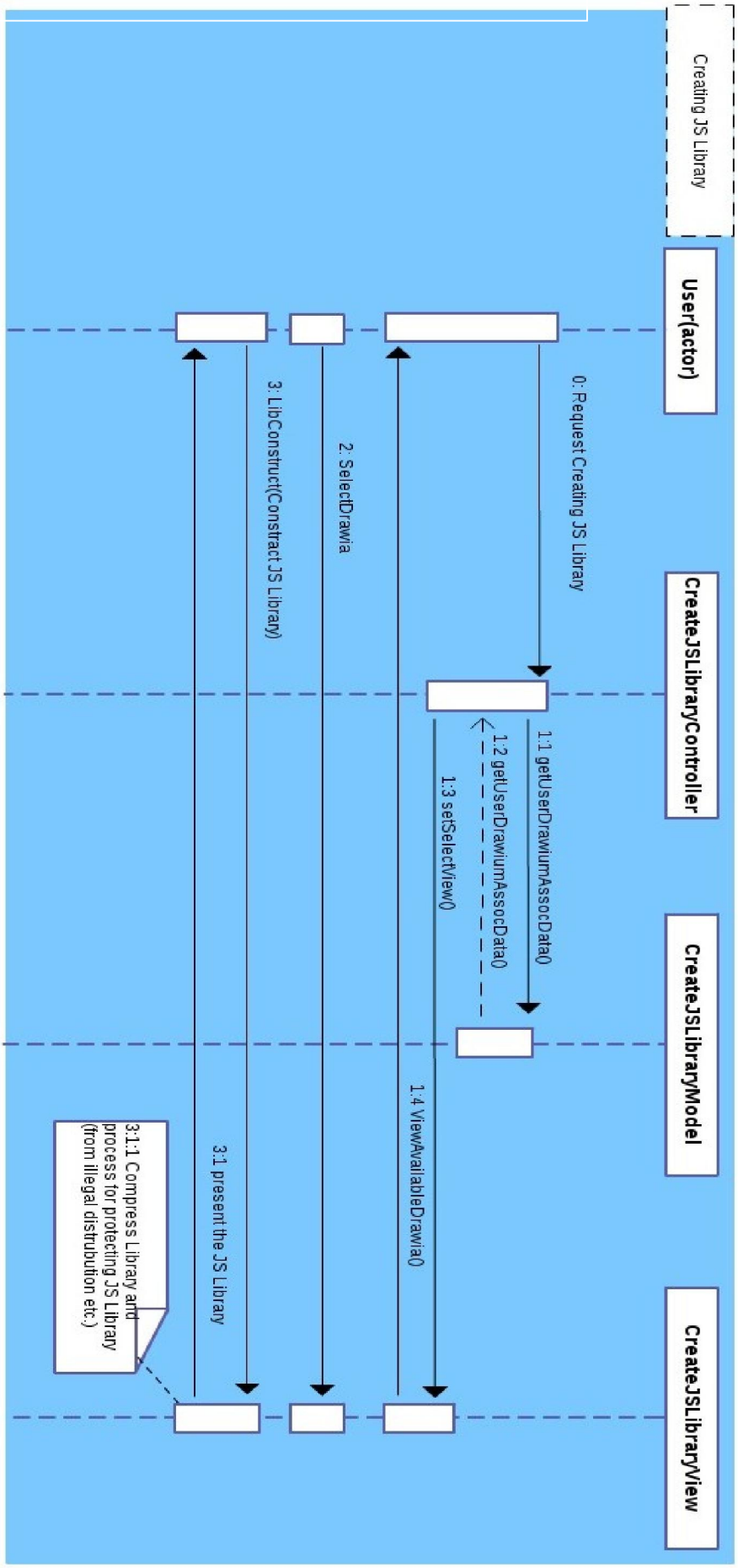

Figure 17: CreatingJSLibrary Sequence Diagram

The **CreateJSLibController** component retrieves the user drawium association data from the **CreateJSLibModel** component when a user wants to create JS Library. Then the user's all drawia are set to the **CreateJSLibView** component. The available drawia are displayed to user. The **SelectView** component allows the user to select the drawia that he/she wants to add his/her JS library. When the user requests to get the JS library, **LibConstructView** component compress the library and make some process to protect JS library from illegal distribution etc.. Then the JS library is presented to the user.

5.2.4. Component DisplayDrawium

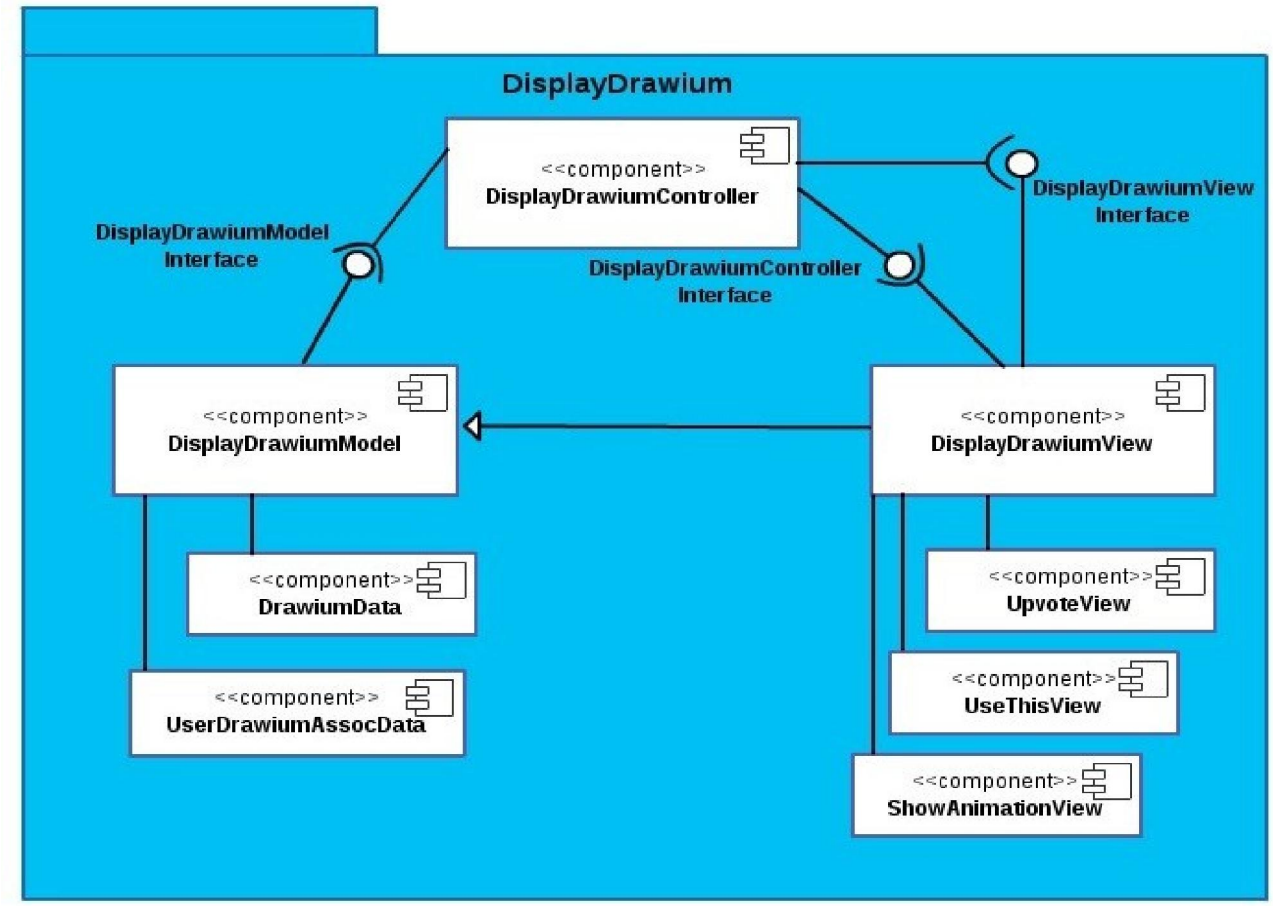

Figure 18: DisplayDrawium Component Diagram

5.2.4.1. Processing Narrative for Component DisplayDrawium

DisplayDrawium component contains the components responsible for the operations on the drawium, such as upvote, use this or show animation of drawium. **UpvoteView**, **UseThisView** and **ShowAnimationView** components offer upvote, use this and show animation capabilities respectively. **DisplayDrawiumModel** component is responsible for storing information of drawium and information of which users make use this or upvote the drawium. **DisplayDrawiumController** component provides essential information of **DisplayDrawiumModel** to **DisplayDrawiumView** component and has ability to modify on them.

5.2.4.2. Component DisplayDrawium Interface Description

DisplayDrawiumViewInterface: allows current info of corresponding components to be set to the **DisplayDrawiumView** component.

DisplayDrawiumControllerInterface: allows the **DisplayDrawiumView** component to send request to **DisplayDrawiumController**.

DisplayDrawiumModelInterface: provides information which is needed by **DisplayDrawiumController** component to the **DisplayDrawiumController** component. In addition, **DisplayDrawiumController** component may set the fields of **DisplayDrawiumModel** component through this interface.

5.2.4.3. Component DisplayDrawium Processing Detail

The **DisplayDrawiumController** component retrieves the drawium data from the **DisplayDrawiumModel** and sends to **DisplayDrawiumView**. Then **DisplayDrawiumView** displays the drawia on the page. **ShowAnimationView** captures the related request via user interface. In order to respond to this request, **DrawiumData** is retrieved from the model, then the animation of drawia is played on the page. **UpvoteView** and **UseThisView** allows the user upvote or “use this” a drawium. **DisplayDrawiumView** sends signal to controller in order to update the model accordingly.

5.2.4.4. Dynamic Behavior Component DisplayDrawium

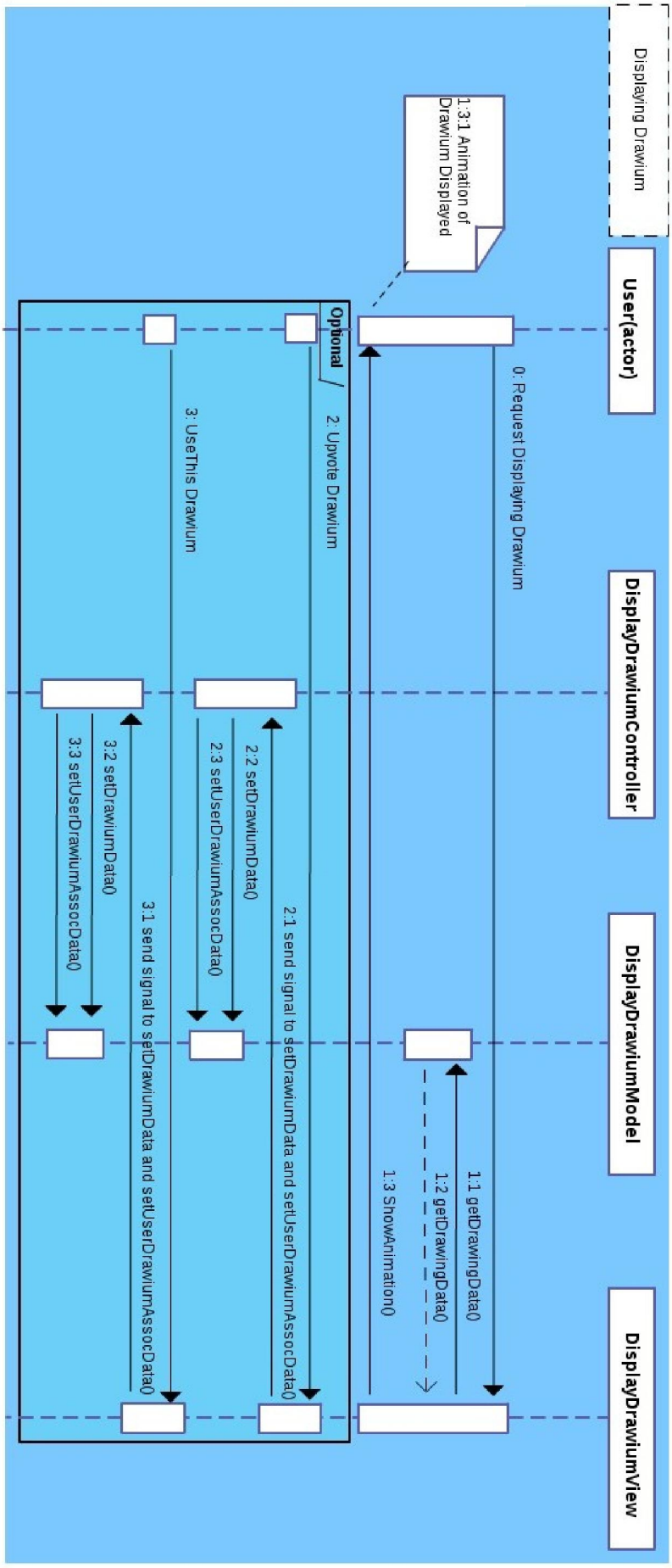

Figure 19: DisplayDrawium Sequence Diagram

The **DisplayDrawiumView** component prepares the drawium for the user once a user wants to display a drawium. The **DisplayDrawiumView** component retrieves the drawium data from the **DisplayDrawiumModel** component and shows the animation to the user. Optionally, when a user wants to upvote a drawium, the **UpvoteView** component sends a signal to the **DisplayDrawiumController** component. Then it updates the drawium data and **UserDrawiumAssocData**. When a user wants to “use this” a drawium, the **UseThisView** component sends a signal to the **DisplayDrawiumController** component. Then it updates the drawium data and **UserDrawiumAssocData**.

5.2.5. Component NewsFeed

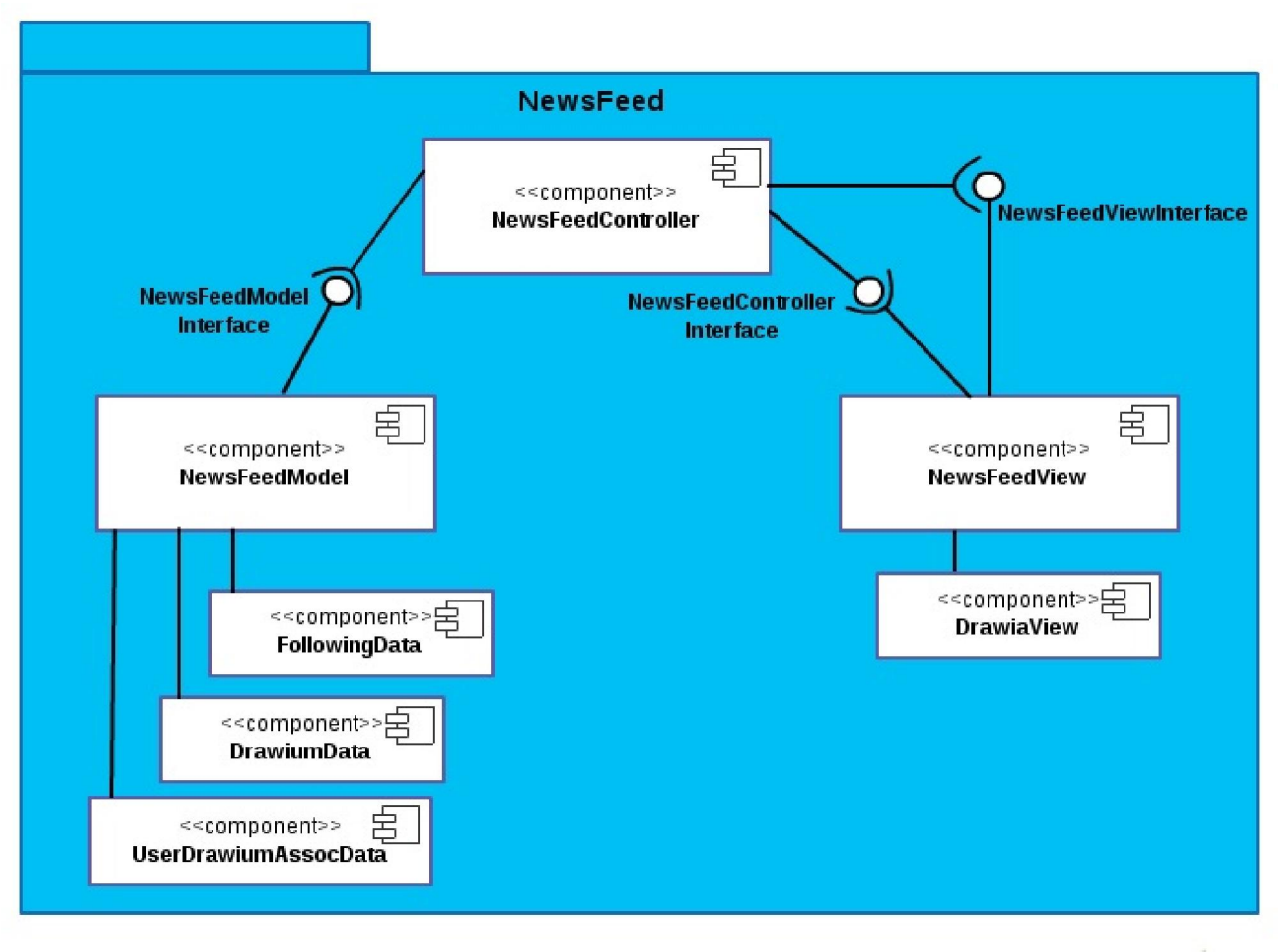

Figure 20: NewsFeed Component Diagram

5.2.5.1.Processing Narrative for Component NewsFeed

NewsFeed component includes the components related to showing newsfeed. **NewsFeedController** allows **NewsFeedView** to have necessary information of drawium and can make changes on data of **NewsFeedModel**. **FollowingData**, **DrawiumData** and **UserDrawiumAssocData** components hold the data of the components of the **NewsFeed** and they provide the essential data to the **NewsFeedView** component.

5.2.5.2. Component NewsFeed interface description

NewsFeedViewInterface: allows current info of corresponding components to be set to the **NewsFeedView** component.

NewsFeedControllerInterface: allows the **NewsFeedView** component to send request to **NewsFeedController**.

NewsFeedModelInterface: provides information which is needed by **NewsFeedController** component to the **NewsFeedController** component. In addition, **NewsFeedController** component may set the fields of **NewsFeedModel** component through this interface.

5.2.5.3. Component NewsFeed Processing Detail

When new users log in to the system, the **NewsFeedController** component is notified. The **NewsFeedController** component then retrieves the corresponding components from **NewsFeedModel** component and supplies them to view-related components through its own view interface. The **NewsFeedView** component then renders and display the newsfeed, enabling the user to take action on the content, such as upvoting drawium. When the content of newsfeed changes in the view by the user, the **NewsFeedController** component is triggered to save this information, through its interface and supplies this information to model through model's interface and the newsfeed content in the database is updated.

5.2.5.4. Dynamic Behaviour Component NewsFeed

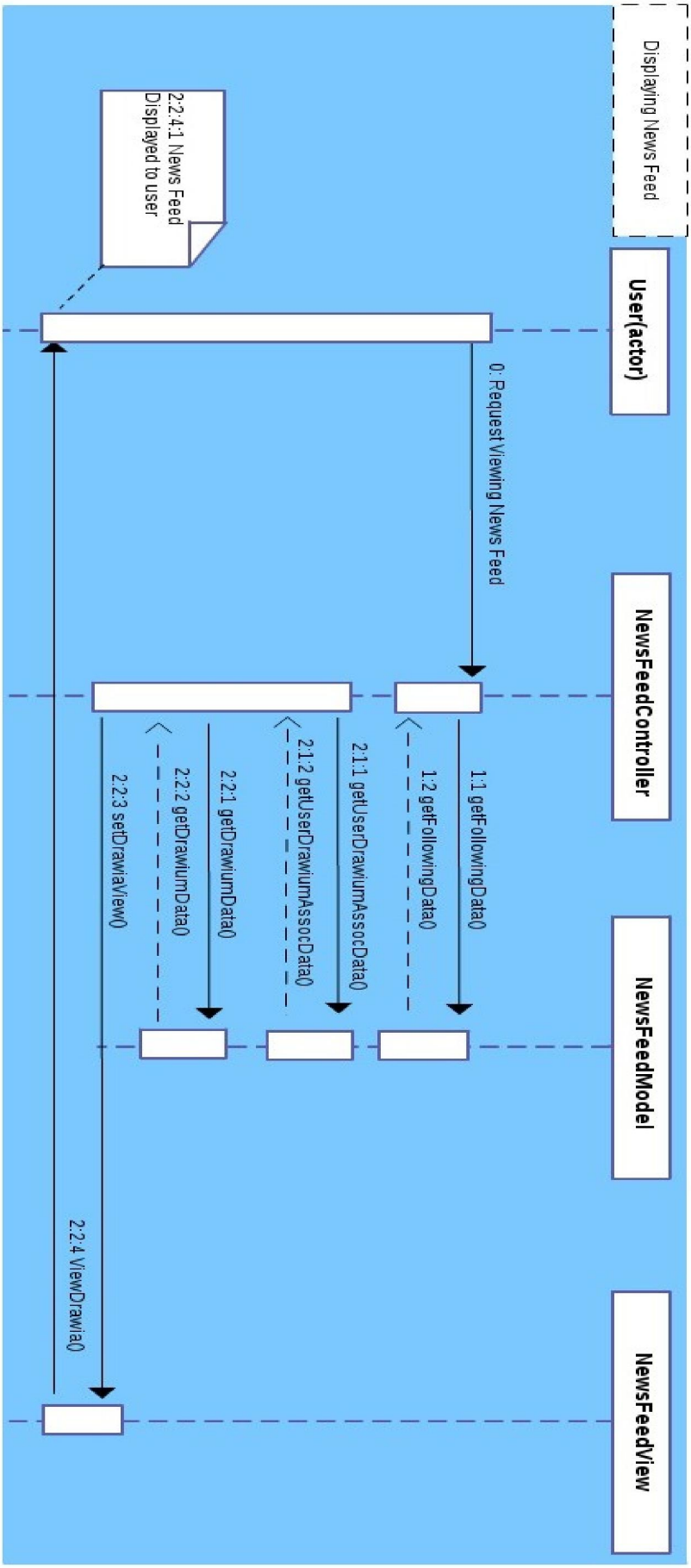

Figure 21: NewsFeed Sequence Diagram

The **NewsFeedController** component retrieves the following data from the **NewsFeedModel** component when a user want to see his/her newsfeed. The **NewsFeedController** component first takes the user drawium association data from the **NewsFeedController** component and then retrieves the drawium data. The **NewsFeedController** component sends the drawium related and user related data to the **NewsFeedView** component and the newsfeed is displayed to the user by this view component.

6. User Interface

6.1 Overview of the User interface

All the system will be on the web and will be accessible with a browser.

Login Screen: This will be a simple login screen for the users to enter their registered email addresses and passwords.

Registration Form: A new user should fill this form to register to the system. Registration screen is very simple, in order not to scare users.

Other Components: Other components, such as the profile tab, will be accessible with a menu bar on the site, after logging in to the system. More information can be found in the screenshots attached below, which are made to give an understanding of the user interface. These screenshots may not be the final user interface, however.
Note that this document does not cover the user interface design of the “Display Drawium” component. Its design will be explained in the Software Design Description (detailed design) document.

6.2 Screenshots

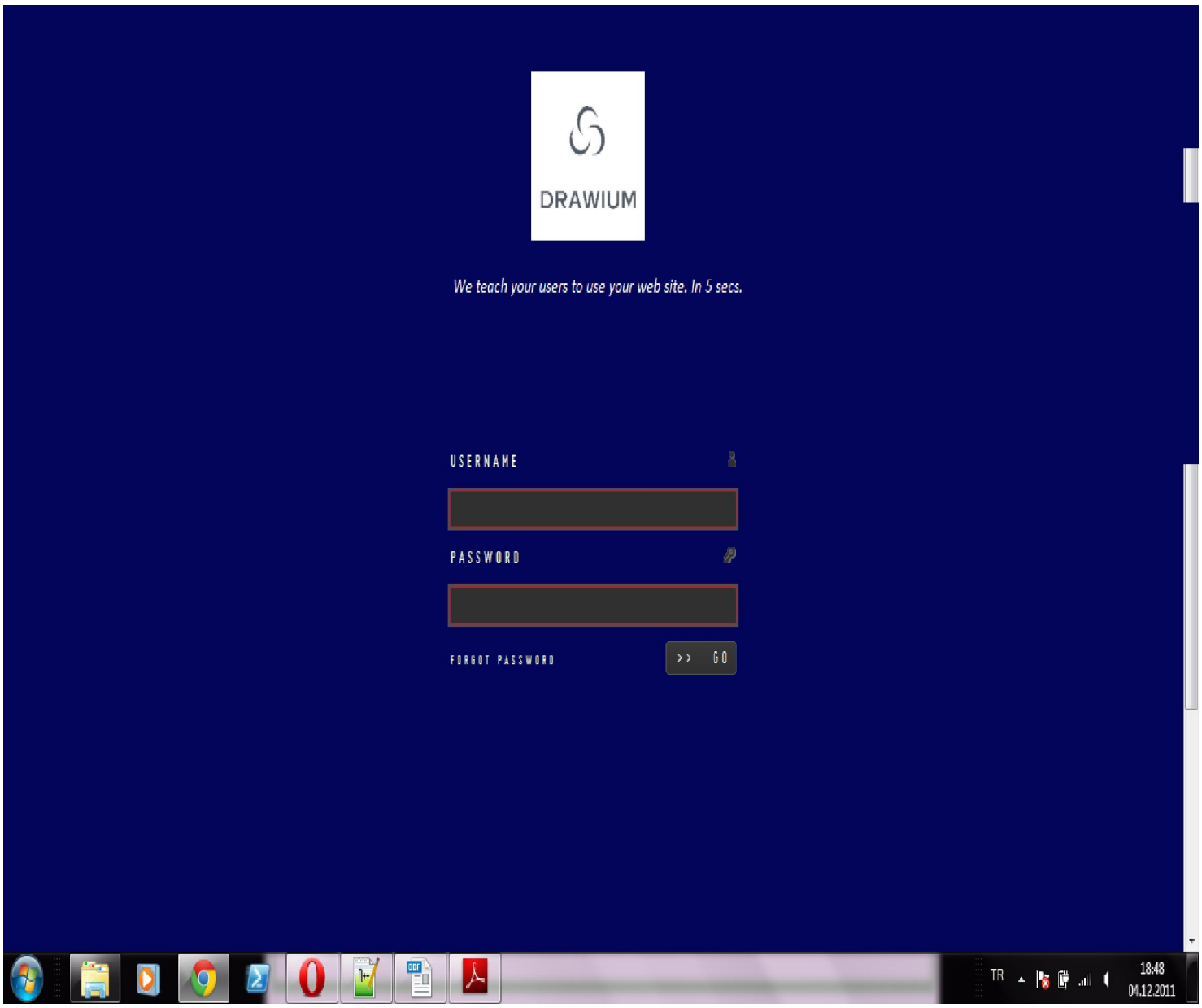

Figure 22: Login Page Screenshot

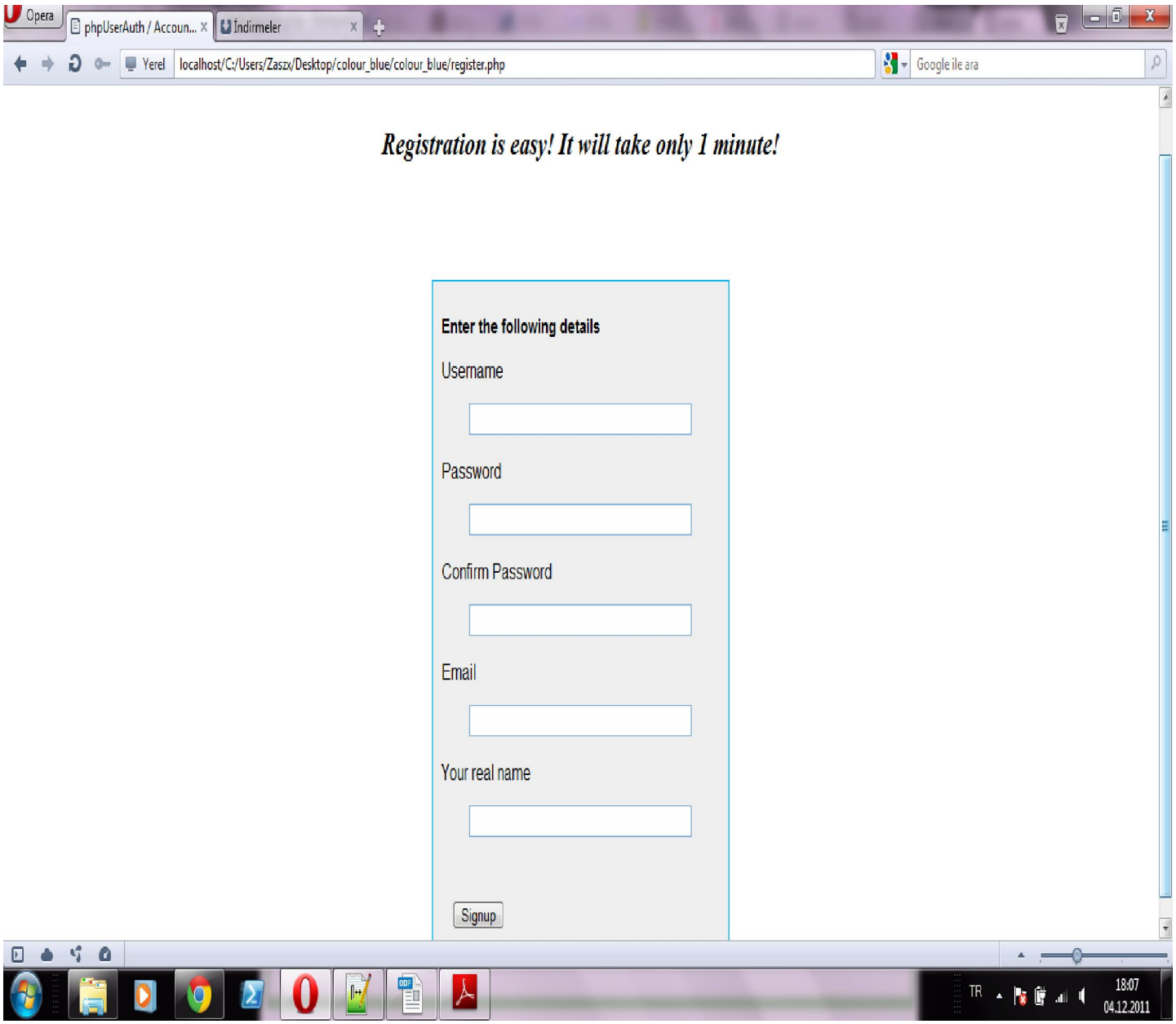

Figure 23: Registration Page Screenshot

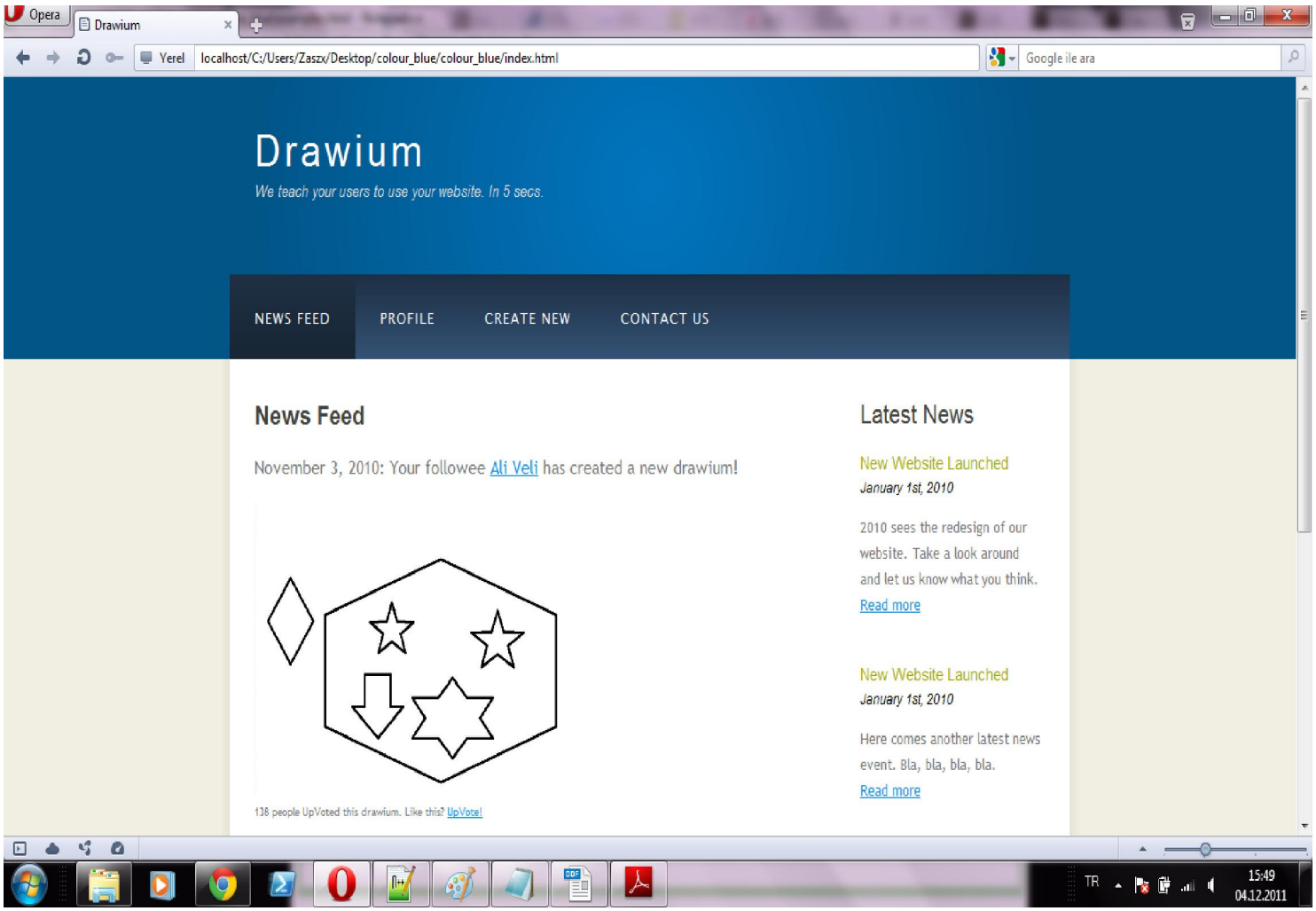

Figure 24: NewsFeed Page Screenshot

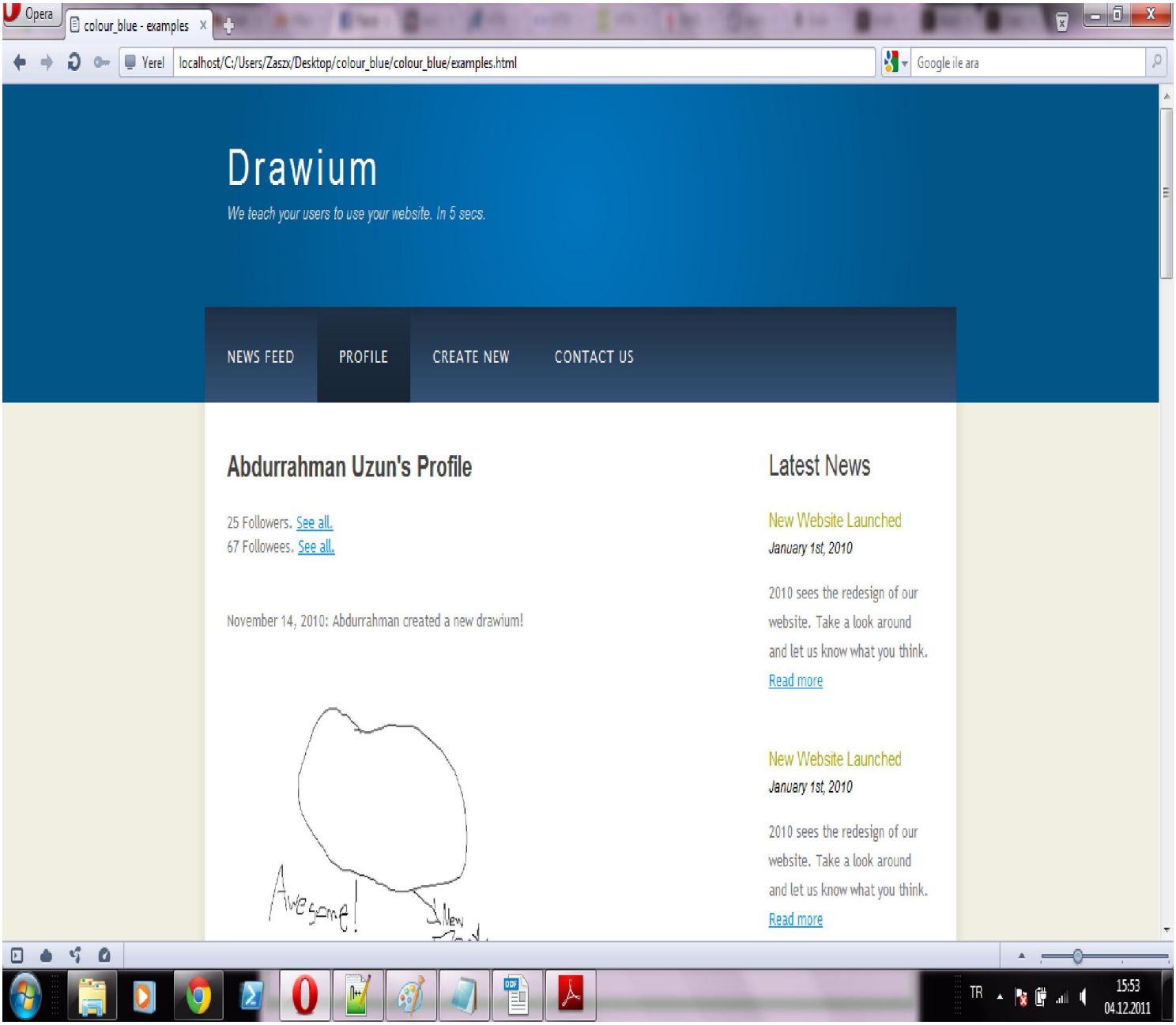

Figure 25: Profile Page Screenshot

6.3 Screen objects and Actions

On all pages, there will be a log out button to return to the login page, restoring the cookies generated.
On the main page (the login screen), there will be a button for registering a new account. It is not shown in the screenshots.
To pass from one component to another (from news feed to profile, for example) the table like links will be used (shown with dark blue color on the screenshots)
For seeing followers and followees of a user, some better looking buttons will be used. This is shown using simple links on the screenshots, but will be improved when we are implementing the user interface further.

7. Time Planning

7.1. Term 1 Gantt Chart

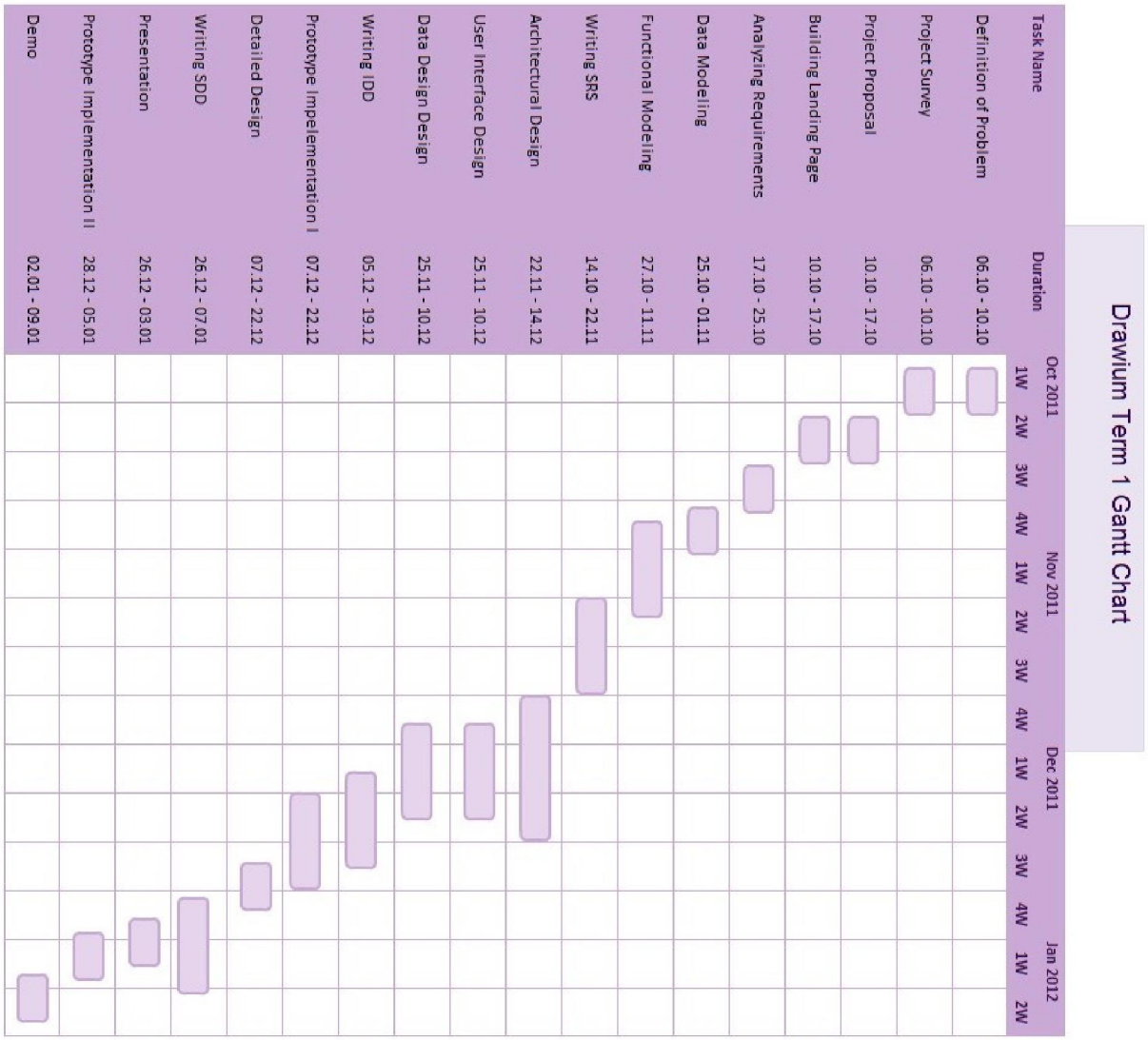

Figure 26: Term 1 Gantt Chart

7.2. Term 2 Gantt Chart

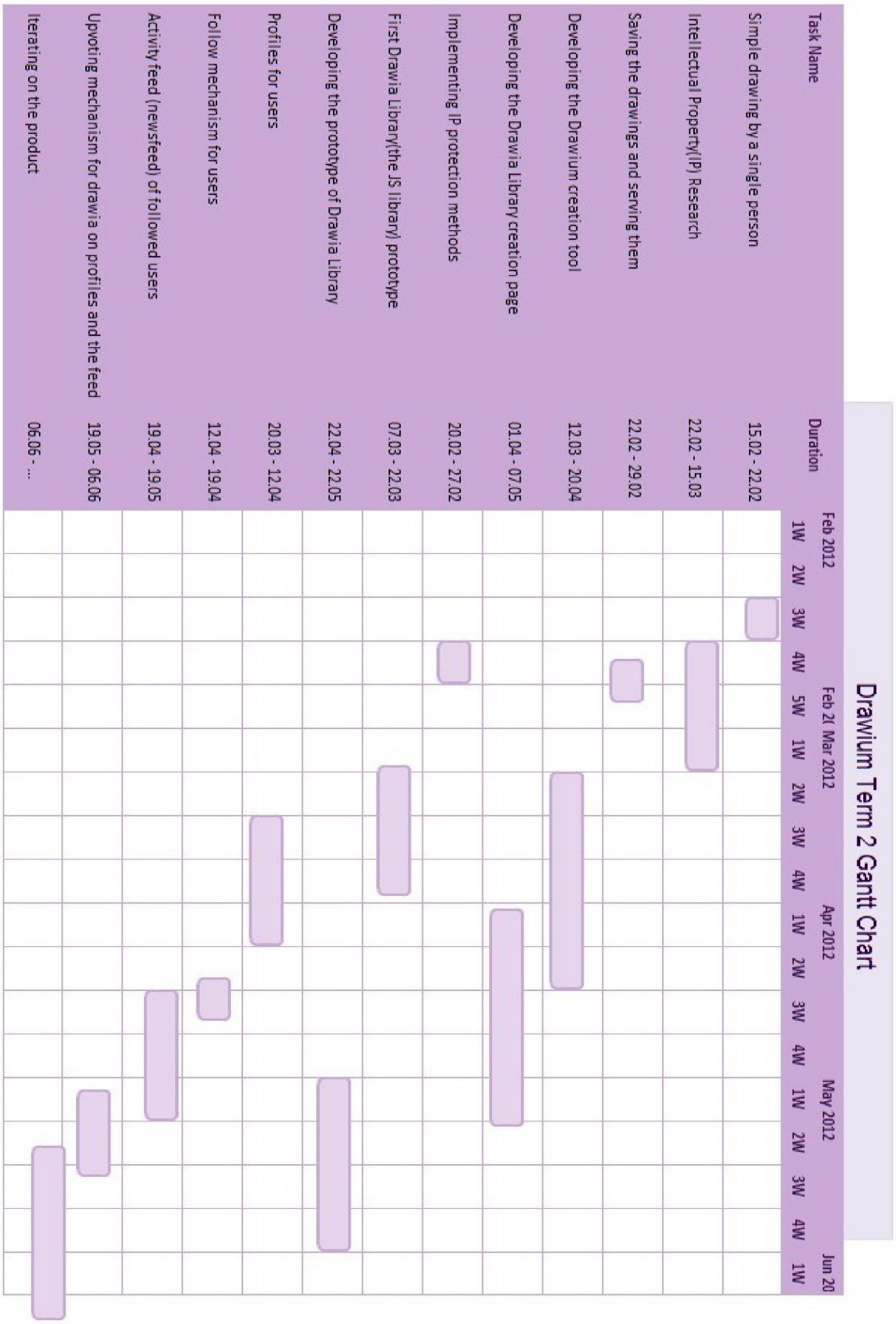

Figure 27: Term 2 Gantt Chart

8. Libraries and Tools

CakePhp: CakePhp is an open source tool for developing, maintaining and deploying web applications. We will use this tool when developing our web site scripts. Using model view controller and object relational mapping over configuration paradigm, this tool helps developers with making them write less code.

JQuery: This is a javascript library that simplifies client side scripting of HTML. It is also open source (actually no other choice, one can not hide any javascript). It is designed to make it easier to navigate a document, create animations, and handle events. It also allows other developers to create plugins on top of the library. A research done in 2011 say that JQuery is used in 49% of web sites that use javascript, making it the most popular javascript library on the web.

HTML5: This is the fifth version of the HTML software. It is used for structuring and presenting content for web. It is a popular choice among web developers, a report released on September 2011 says that 34 of the world's top 100 web sites use this software.

JavaScript: This one is a prototype based scripting language. It is multi paradigm, in other words it supports functional, imperative and object oriented programming. It is mainly used for web development, but it can be used for anything else too.

Canvas: It is a tool used for drawing using scripting, mainly used with javascript. It can be used to draw images, graphs, make photograph compositions and make animations.

9. Conclusion

This Initial Design Report has been intended to explain how the system will be structured to satisfy the requirements. System components, interfaces and data required for the implementation phase have been briefly described.

Again, note that this document does NOT cover all the design of drawium project, for a more detailed design you will have to wait for the Detailed Design Description document.