Even Semester (2019)



**BINUS UNIVERSITY**



**BINUS INTERNATIONAL**



**Assignment Cover Letter**

**(Individual Work)**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | | |  |  |  |  |
| **Student Information**: | **Surname** | | |  |  | **Given Names**  **David** | **Student ID Number**  **2201797304** |
| 1. | **Amadeo** | |
|  |  |  |  |  |  |  |  |
| **Course Code** | **: COMP6510** |  |  |  |  | **Course Name** | **: Programming Language** |
| **Class** | **: L2AC** |  |  |  |  | **Name of Lecturer(s)** | : **Jude Martinez** |
|  |  |  |  |  |  |  |  |
| **Major** | **: CS** |  |  |  |  |  |  |
| **Title of Assignment**  (if any) | : **Binus Activity** |  |  |  |  |  |  |
| **Type of Assignment**    **Submission Pattern** | **: Final Project** | |  |  |  |  |  |
| **Due Date** | **: 02 - 07 - 2019** | |  |  |  | **Submission Date** | **: 02 – 07 -2019** |

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer’s instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

# Plagiarism/Cheating

BiNus International seriously regards all forms of plagiarism, cheating and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

# Declaration of Originality

By signing this assignment, I understand, accept and consent to BiNus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student: (Name of Student)

**David Amadeo**

**II. Table of Contents**

1. **Cover**
2. **Table of Content**
3. **Project Specifications**
4. **Solution Design**
5. **Code Implementation Discussion**
   1. **Implementation**
   2. **How it works**
6. **Evidence of Working Programme**
   * 1. **Home page**
     2. **Filter with tag**
     3. **Login**
     4. **Admin home page**
     5. **Adding post**
     6. **Deleting post**
     7. **Editing post**

**III. Project Specifications**

First of all, let us identify the problem at hand that we would like to solve using a simple yet useful java programme.

As a student in Binus University, we are expected to stay up to date on the campus current events and to know all the necessary information a binusian should know. Moreover, a binusian is required to join seminars, get social hours, join campus committee events and much more however it is very difficult to obtain all these information because they are being delivered in various medium. For example, information about upcoming seminars are being sent to the students through their email while the information about events to get social hours are being posted through the student committee Instagram account. It gets clustered and confusing for both the receiver (the students) and the sender. Another example is when people want to recruit committee members for an upcoming campus event, it is hard to deliver that announcement because not everyone uses these mediums.

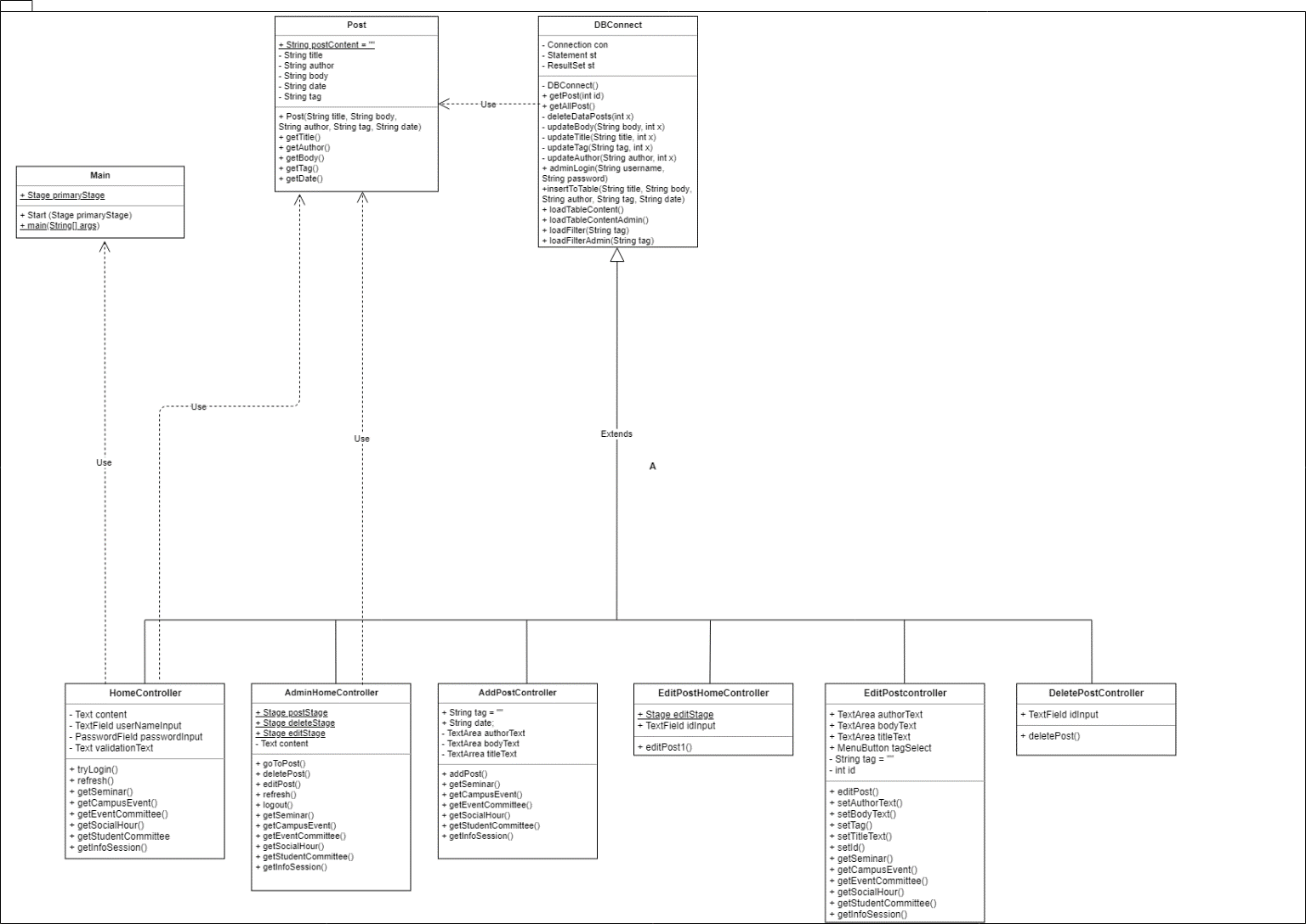
All in all, it is not an efficient way to exchange in information both for the sender and receiver.

So, for my final project, I have made an application called Binus Activity where all these information can be delivered through one medium. Can be categorized with tags of the different kind of information or posts.

For this project, I use mySQL as my database, Java FX with sceneBuilder for my GUI, intelliJ for my IDE and a server.

**IV. Solution Design**

Below is the UML for the final project.

****

**V. Code Implementation Discussion**

**i. Implementation**

For this project, I use Java FX with sceneBuilder to build the user interface and I use mySQL for the database that stores all the users and the posts from Binus Activity with using XAMPP to connect from the local host or server to mySQL and apache.

This project imports many different packages from javafx for the GUI, java.time to collect the local current data and format the date, java.sql to connect with a database and an array list.

import javafx.fxml.FXML; //this import is used to access the fx:id

import javafx.fxml.FXMLLoader; // this import is used to load a new fxml file into the window  
import javafx.scene.Parent;  
import javafx.scene.Scene; // this import is used to set the dimensions of the window

import javafx.stage.Stage; // this import is used to import the stage, so every page will create a new stage

import java.time.format.DateTimeFormatter; // this import is used to format the data dd/mm/yyyy  
import java.time.LocalDate; // this import is used to collect the local current date

import java.sql.\*; // this import is used to connect with the mySQL database in the class DBConnect  
import java.util.ArrayList;

import javafx.scene.control.TextField; // this import is used to get the text from the text areas and text fields

import javafx.scene.control.TextArea;

import javafx.application.Application // this import is used to connect with the javafx application in main, to override their start function

**ii. How it works**

To understand how the code works, the explanation will be divided into different sections.

1. Home page
2. Login
3. Admin home page
4. Adding post
5. Deleting post
6. Editing post
7. Server

Let us begin with the first,

**i. Home page**

When the programme starts, the home page will launch the on the screen ready with all the posts from the database.

In the home page the user can view all the posts currently available from the database from the most recent to the oldest post.

public void loadTableContent()

This is the function that loads all the data (posts) from the database then prints out all the post details with print format. So, they print out all the post from an initialize empty string from the class Post.

public void loadFilter(String tag)

The user can also filter the posts they see with tags that corresponds to a certain post that has a pre-determined tag when they were added.

public void refresh(){ //create the function to refresh the page  
 Post.*postContent* = ""; //initialize that the post is an empty string  
 loadTableContent(); //load the table content  
 content.setText(Post.*postContent*); //insert each post into the empty string  
}

Then the user is also able to refresh the page where first they delete everything from the post text in the home page then reloading the posts from the database again. The initialized empty string from the class Post is set as the text in the home page.

**ii. Login**

Everyone can access Binus Activity without having to register or create a new account, so they can view all the posts however only admins can add a post, edit a post or delete a post. So, on the top right-hand corner of the home page there will be a place to login for admins. To make the programme simpler, there is only one account for the admin.

public int adminLogin(String username, String password)

This is the function from the class DBConnect that is responsible for the login. So, what it does is it checks whether the username and password from the text field in the home\_page.fxml file is the same as that of the one in the database. Since there is only one account in the database (that has an auto-incremented id of 1), the programme only must check through one username and one password from the database.

public void tryLogin()

This is the function form the home controller. tryLogin() function sets a validation text for the admin if they input the wrong username and/or password.

**iii. Admin home page**

The admin home page is similar to the home page however for the admin home page, the auto-incremented id of each post from the database is printed out as well for the admin to see.

public void loadTableContentAdmin()

public void loadFilterAdmin(String tag)

So, the function to load the posts content with and without the tag filter is separated. The admin will have the auto-incremented id of each post from the database printed out.

Moreover, the admin home page also has a drop-down menu button on the top right-hand corner. When the admin presses the drop-down menu button, they will have the option to logout, add a post, delete a post or edit a post.

public void goToPost() throws IOException

public void deletePost() throws IOException

public void editPost() throws IOException

public void logout() throws IOException

If the admin wants to add, delete or edit a post, a new stage will be created a long with a new window. If the admin logs out, the window will change and go back to the home page from the home\_page.fxml.

The admin can also refresh the page and filter the posts with tags like the home page.

public void refresh(){   
 Post.*postContent* = "";   
 loadTableContentAdmin   
 content.setText(Post.*postContent*);

}

public void getCampusEvent(){  
 Post.*postContent* = "";  
 loadFilterAdmin("Campus Event");  
 content.setText(Post.*postContent*);  
}

**iv. Adding post**

The admin will first be directed into a new window and the admin must fill the different text areas, the title, body, author and tag.

DateTimeFormatter dtf = DateTimeFormatter.*ofPattern*("yyyy/MM/dd");   
LocalDate localDate = LocalDate.*now*(); //specify that it is the current date  
  
String tag = "";  
String date = dtf.format(localDate); //create a variable to assign the local date  
  
@FXML  
TextArea authorText; //access the the fx:id from the FXML  
@FXML  
TextArea titleText;  
@FXML  
TextArea bodyText;

First, the programme collects the local date in a form of string. Then, the programme will get the text from the different text areas.

public void getSeminar(){  
 tag = "Seminar";  
} //create function to get each and every tag from the posts  
  
public void getCampusEvent(){  
 tag = "Campus Event";  
}  
  
public void getEventCommittee(){  
 tag = "Event Committee";  
}  
  
public void getSocialHour(){  
 tag = "Social Hour";  
}  
  
public void getStudentCommittee(){  
 tag = "Student Committee";  
}  
  
public void getInfoSession(){  
 tag = "Info Session";  
}

Then, the programme collects the tag that the admin chooses.

public void addPost(){ //create the add post function  
 insertToTable(titleText.getText() , bodyText.getText() , authorText.getText() , tag, date); //insert the details of the post  
 //into the database  
 AdminHomeController.*postStage*.close(); //close the stage  
}

After that the title, author, body, tag, and the date will be inserted into the database.

public void insertToTable(String title , String body , String author , String tag, String date)

**v. Deleting post**

The admin will first be directed to a new window where the admin will be asked to input the id of the post they would like to delete. When the admin presses done, the post from the database will be deleted.

void deleteDataPosts(int x)

This is the function from the class DBConnect. So, this function can be directly called from the delete post controller and use the id as the parameter.

**vi. Editing post**

When the admin wants to edit a post, they will be first directed to a new window that will ask the admin for the id of the post that they would like to delete. Then, when they press done, they window will change similar to the add post page however all the text areas will be filled with the details of the post they chose. The programme

Parent root = loader.load(); //load the page  
EditPostController editPostController = loader.getController();  
ArrayList postData = getPost(Integer.*parseInt*(idInput.getText())); //get the post from the id  
Post post = (Post) postData.get(1);

In this line of code, the programme gets the specific post with the getPost() function from DBConnect.

editPostController.setAuthorText(post.getAuthor()); //set all the post details into the post  
editPostController.setTitleText(post.getTitle());  
editPostController.setBodyText(post.getBody());  
editPostController.setTag(post.getTag());  
editPostController.setId((int) postData.get(0));

In this line of code, the programme gets all the details of the post before it was edited by calling all the getter method from the Post class and use the setter methods from the edit post controller class.

Then, with the new texts and tags updated, we call the update functions from the class DBConnect and after that close the stage.

public void editPost() { //create the function to edit the post  
 updateTitle(titleText.getText(), id); //update each of the post details with the specific id  
 updateTag(tag, id);  
 updateAuthor(authorText.getText(), id);  
 updateBody(bodyText.getText(), id);  
 EditPostHomeController.*editStage*.close(); //close the stage  
}

**vii. Server**

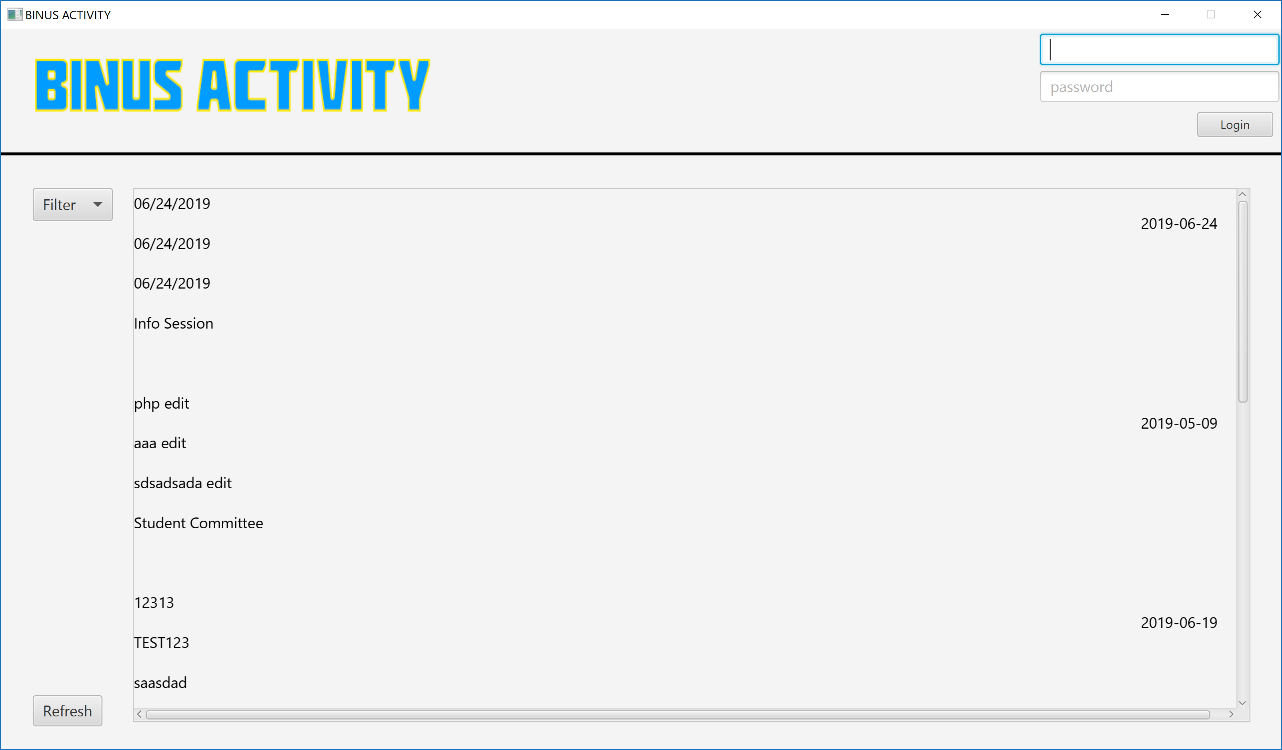
The mySQL database uses a software called XAMPP that uses apache and its own mySQL to open localhost/phpMyAdmin to view the mySQL database. Change the server address of the localhost mySQL database and make it public, connecting it to an online server so that as long as the user is connected to the internet, they can access the database.

The server I use is Zefanya’s server, zefryuuko.

**VI. Evidence of Working Programme**

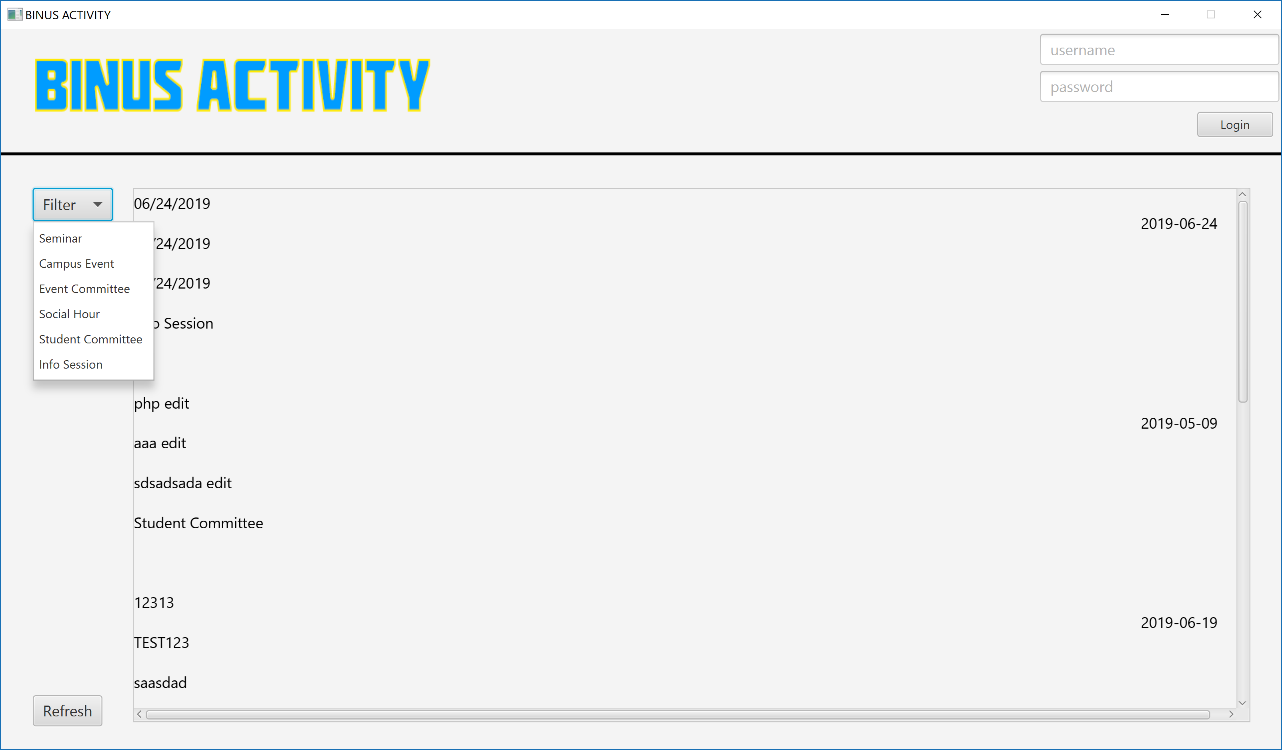
**i. Home page**

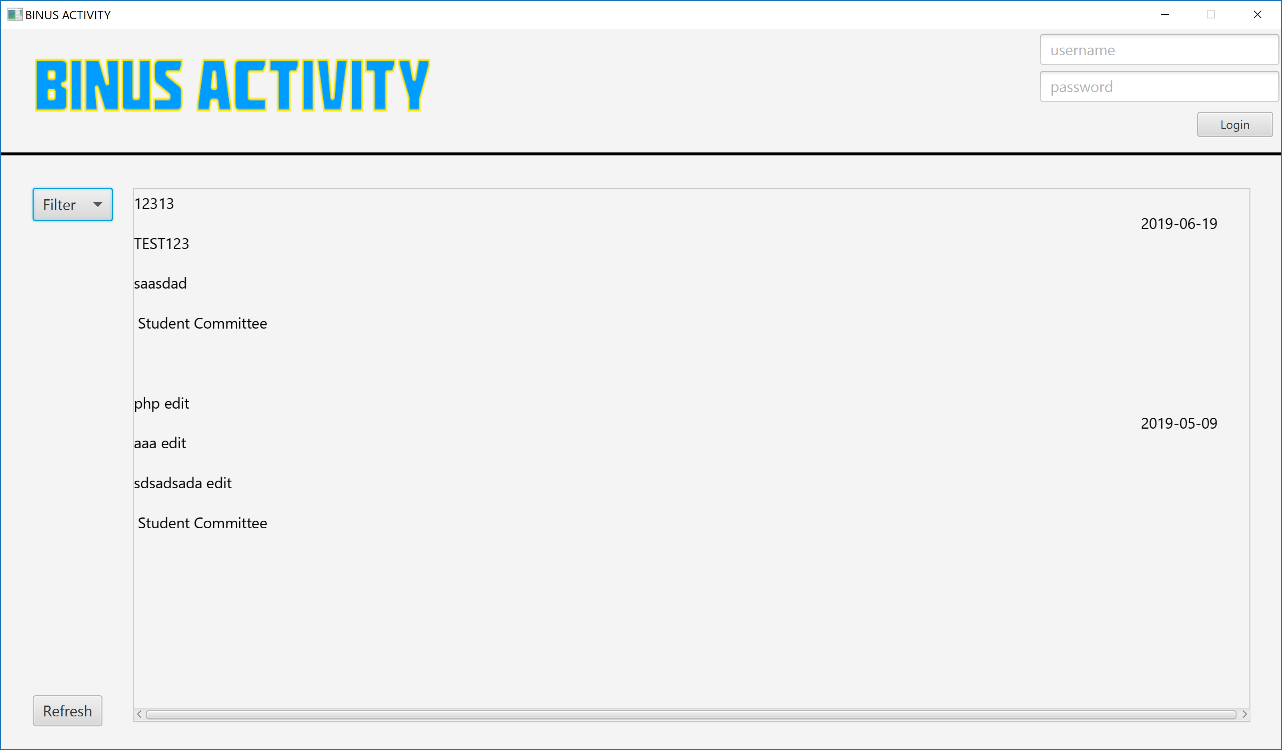
This window is the home page. When the programme launches, the post from the most recent to the oldest will be displayed on the screen.

  
1.1 Home page with the post. The post consisting of the title, body, author, tag, and the date it was uploaded to the server.

**ii. Filter with tag**

This window is an example of when the user filters the posts with a tag.

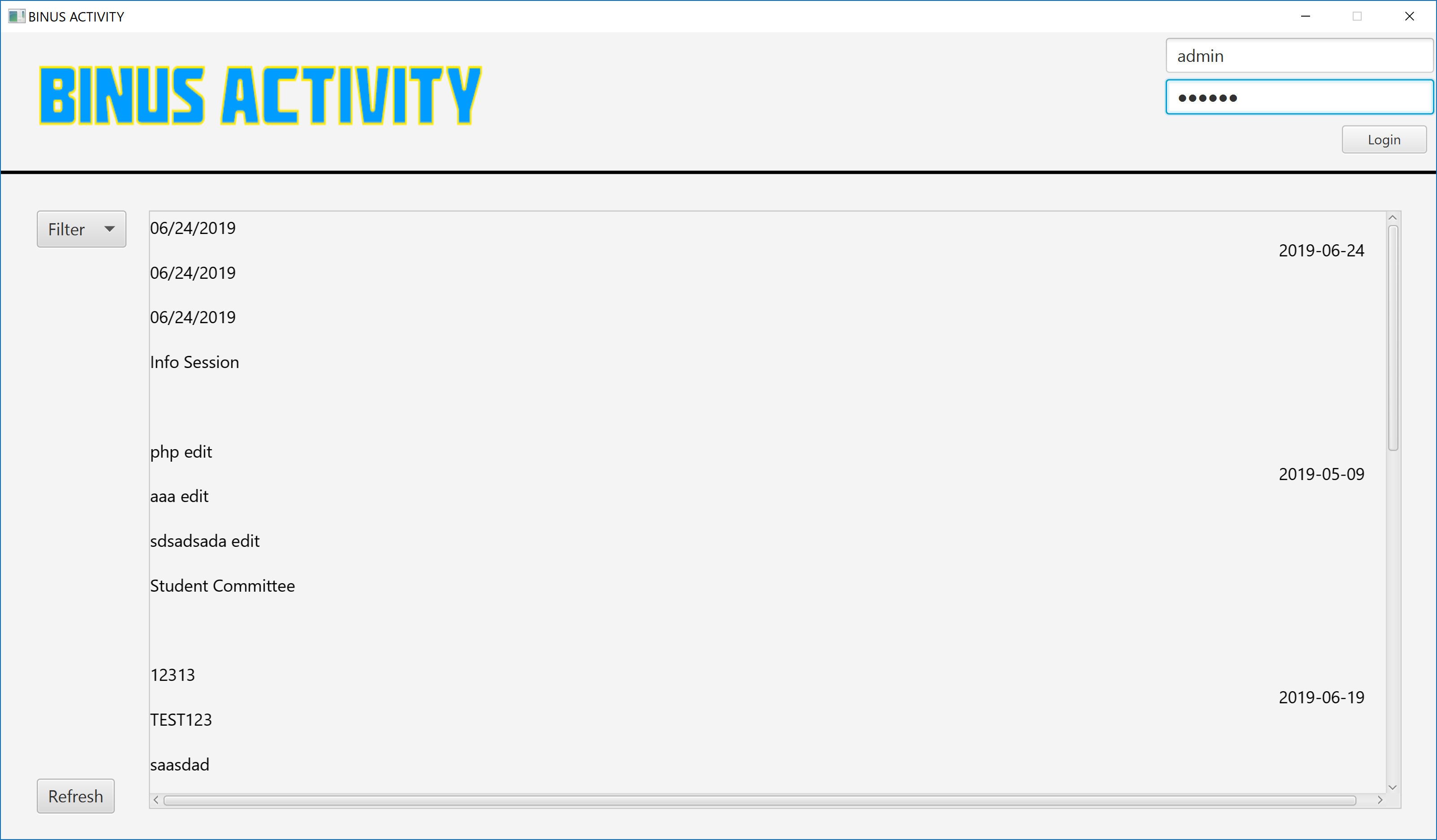
  
1.2 Selecting the filter for the posts

  
1.3 After the user clicks the filter, the posts with the tag and only with that tag is shown. Picture shown filer with Student Committee

  
1.4 Filter with Seminar

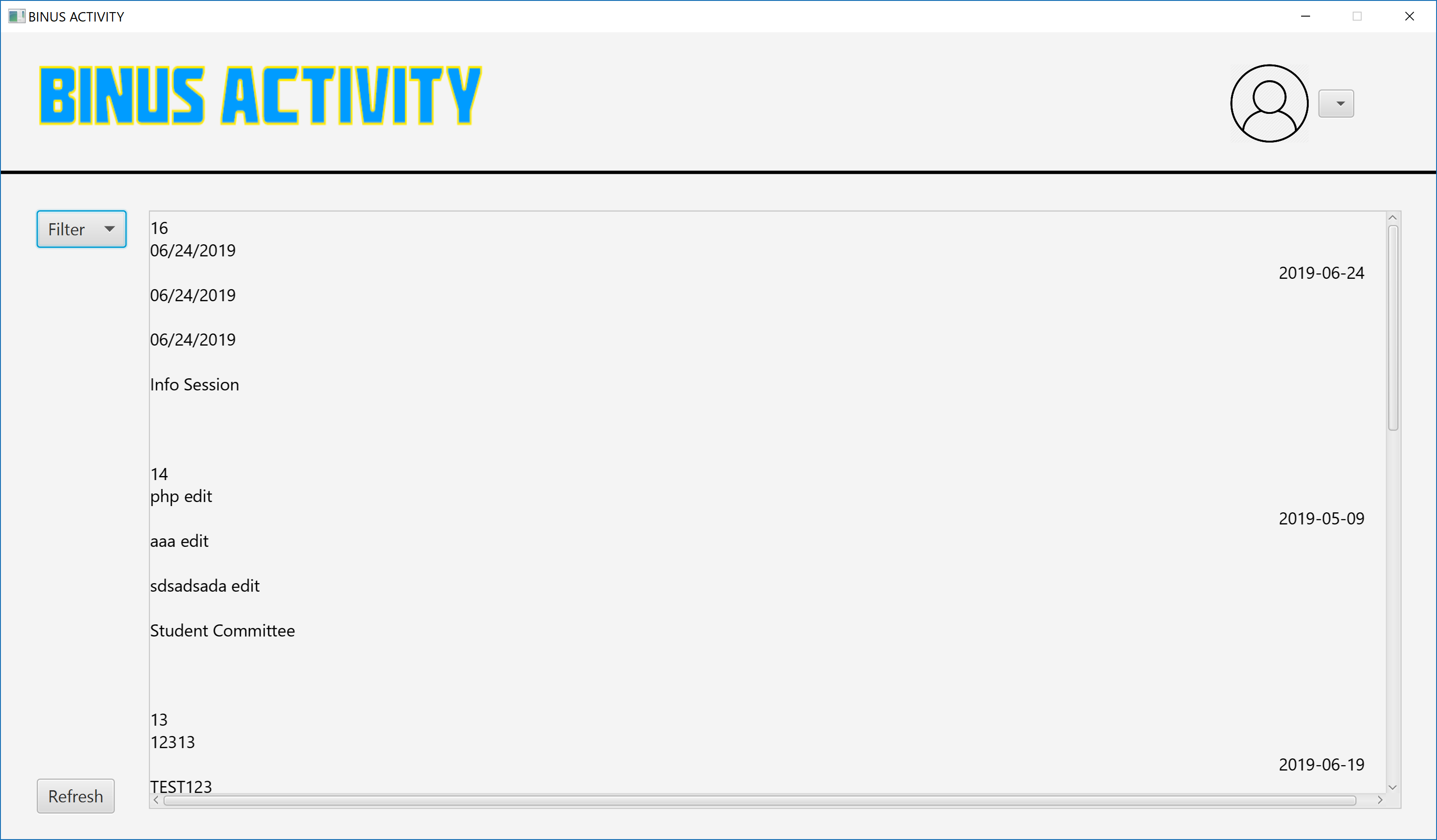
**iii. Login**

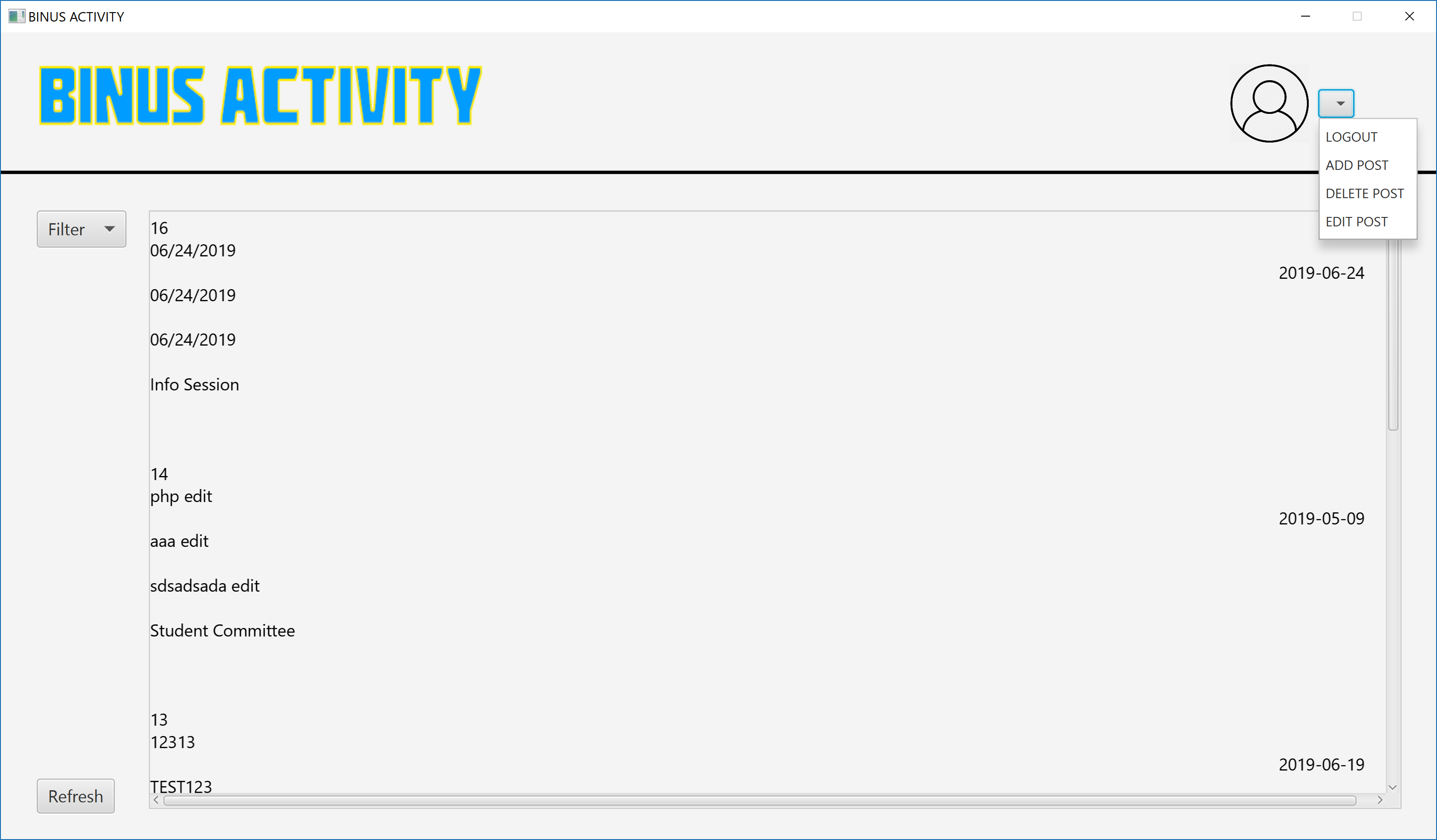
The admin can login through the home page by entering the username and password on the top right of the page.

  
1.5 Input the username and password to the login area then press login to enter admin home page

**iv. Admin Home Page**

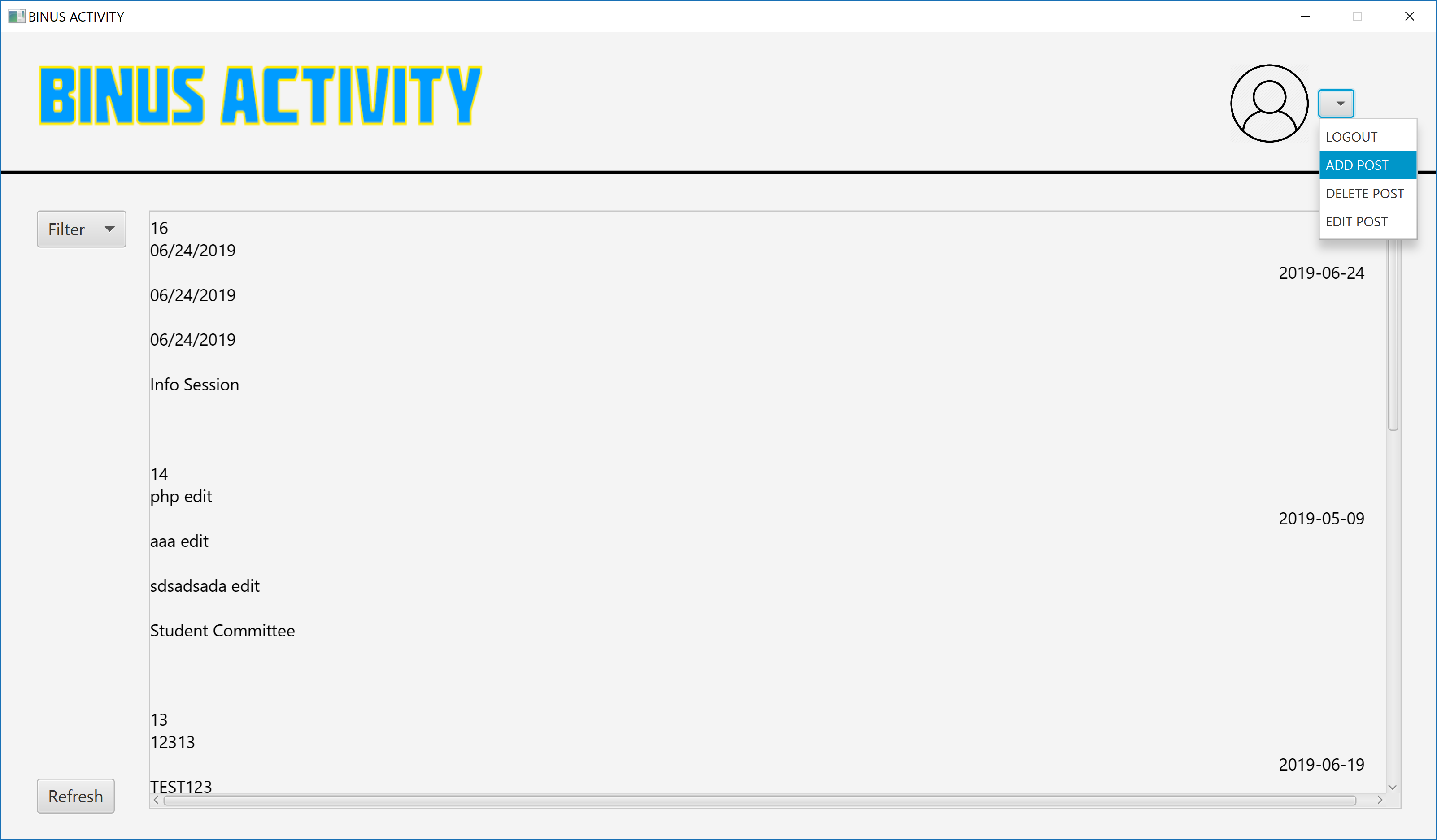
The admin home page is not much different than the home page however there a few key differences between them. First, for every post, they have a number (id) that differentiates between them. So, when they would like to edit or delete a post, they can know which ones they’d like to edit or delete. Second, on the right-hand side where there was a login area is a drop-down menu where they can logout, add, delete, or edit a post.

  
1.6 The admin home page. Like the home page but the post has the auto incremented id printed on top of each post

  
1.7 The admin can press the drop-down menu beside the user icon on the top right to do the only-admin-accessible actions

**v. Adding post**

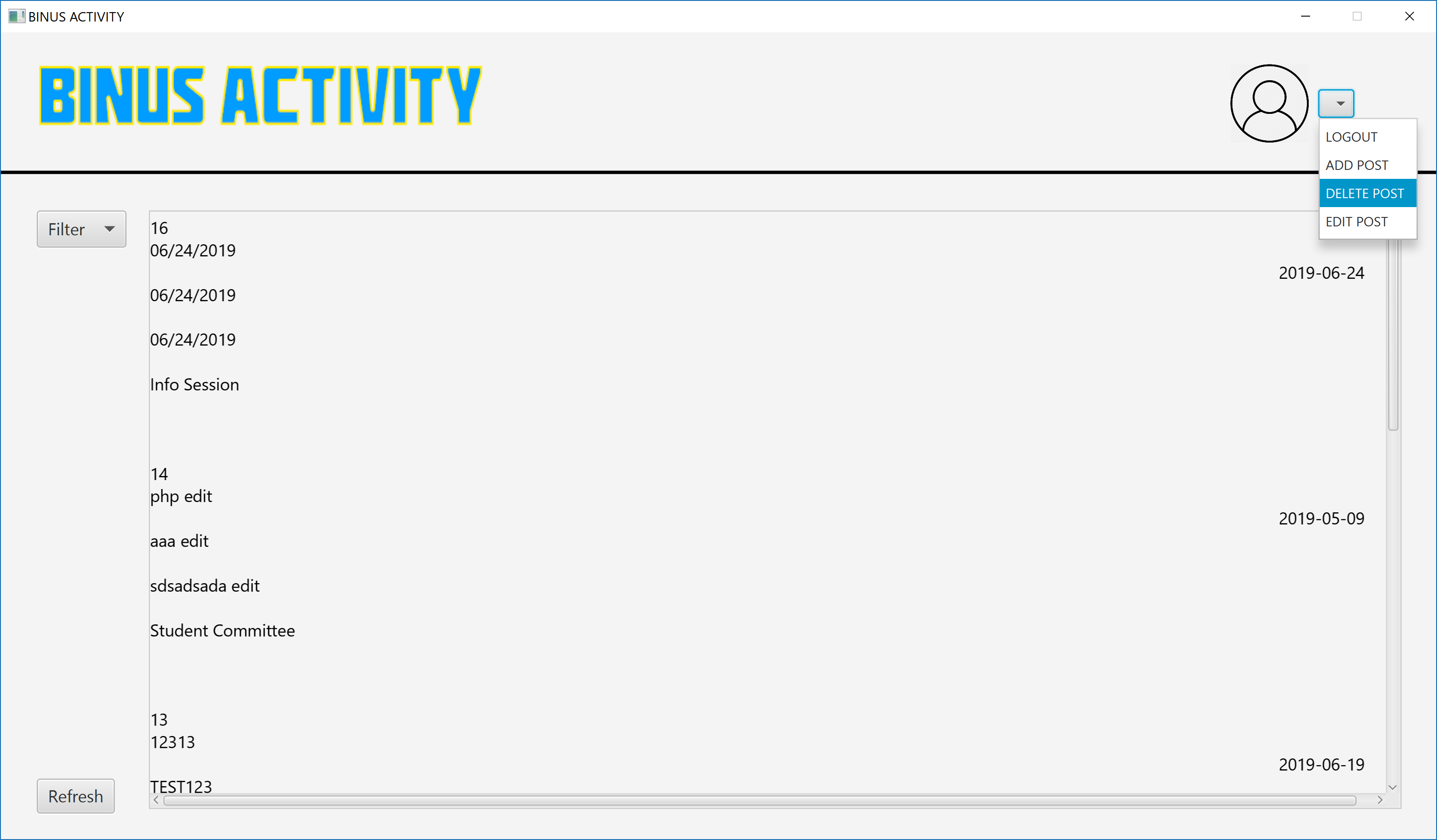
The admin can directly add the details of the post and press done when they finish with the details. Then in the admin home page, they can press refresh to see their latest post. If they want to cancel, they can just close the adding post window.

  
1.8 Press the add post menu

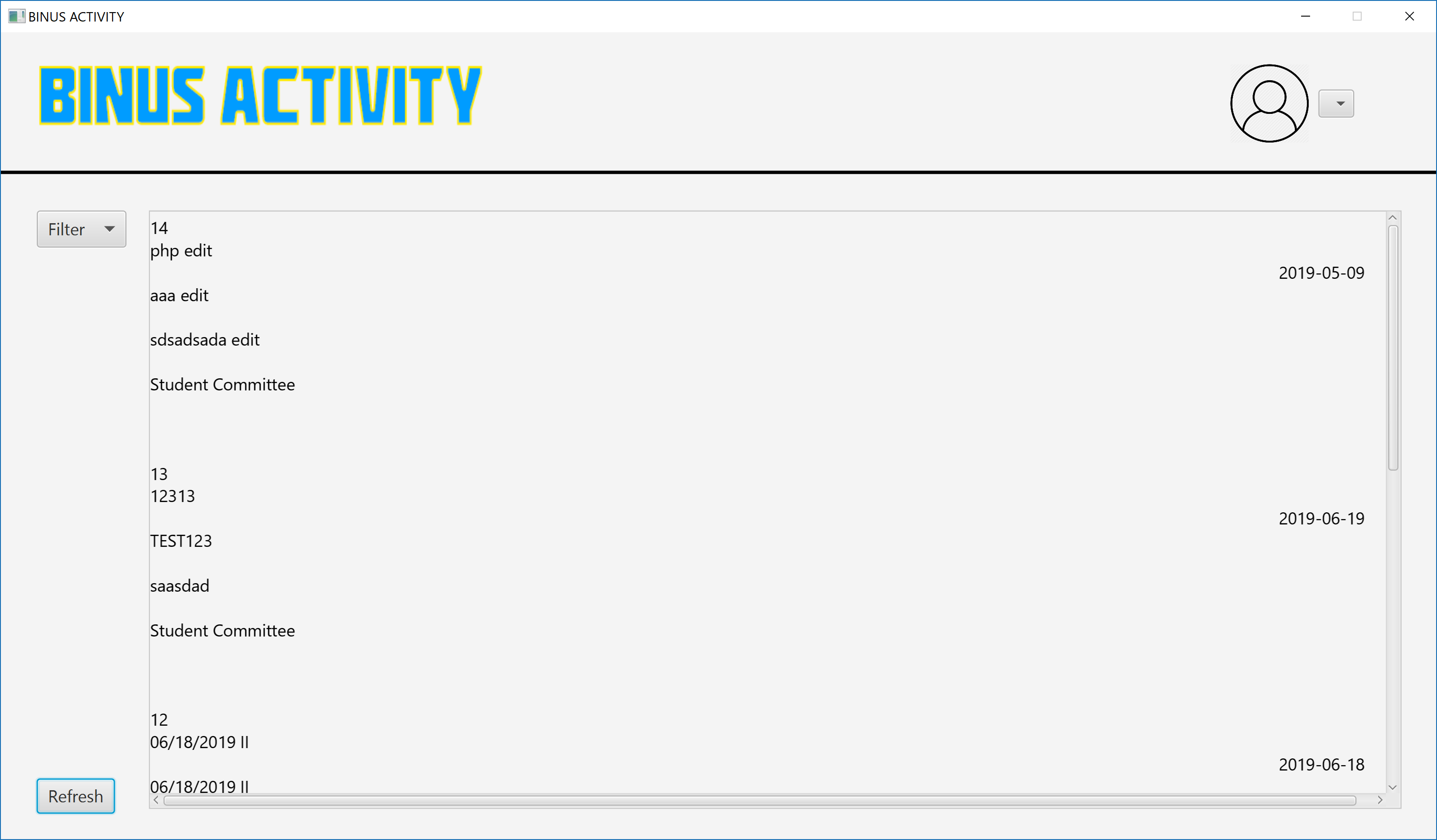
  
1.9 Fill in all the text box and select the tag. Press done when finished then the database will be updated, the post added

**vi. Delete Post**

First, the admin is asked to input the post id. Printed on top of every post in the home page. Then the admin can press done then post will be deleted. They can press refresh on the home page then the post will be deleted.

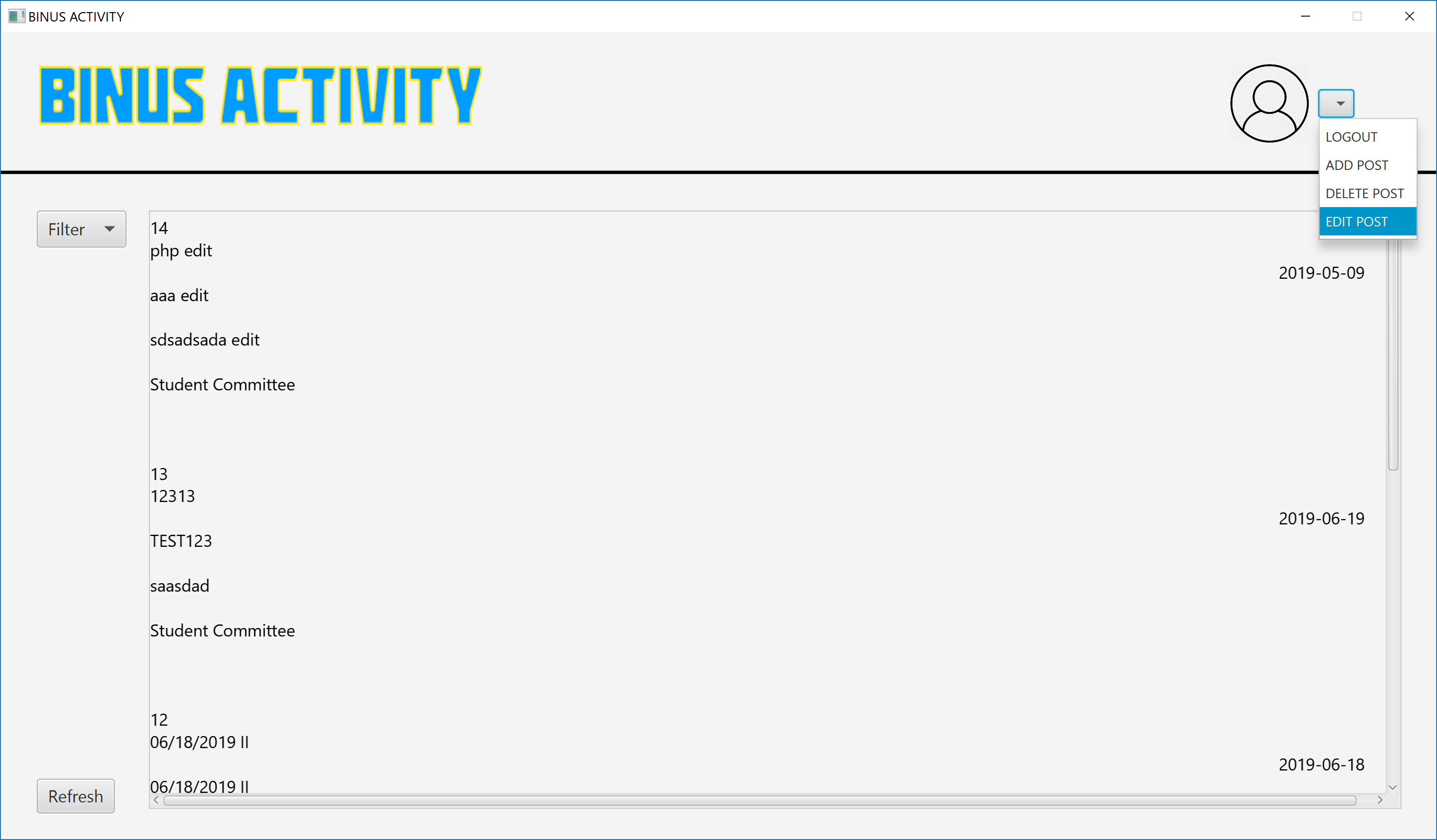
  
1.10 Press the delete post button to delete a post

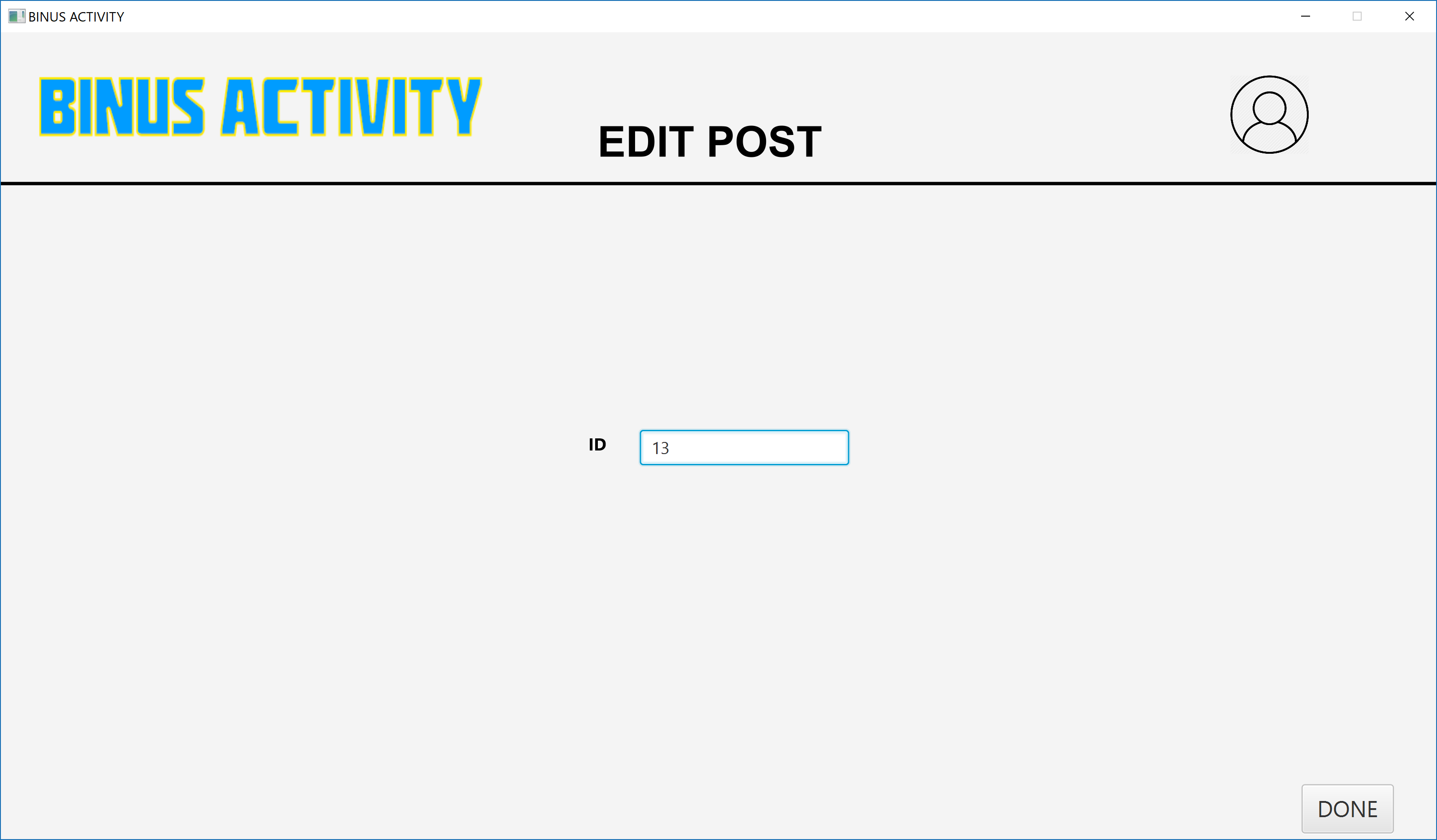
  
1.11 Input the auto incremented id of the post that the admin wants to delete, printed on the admin home page. Press done when the admin is finished, and the database will be updated, the post deleted

  
1.12 Post of id 16 is deleted

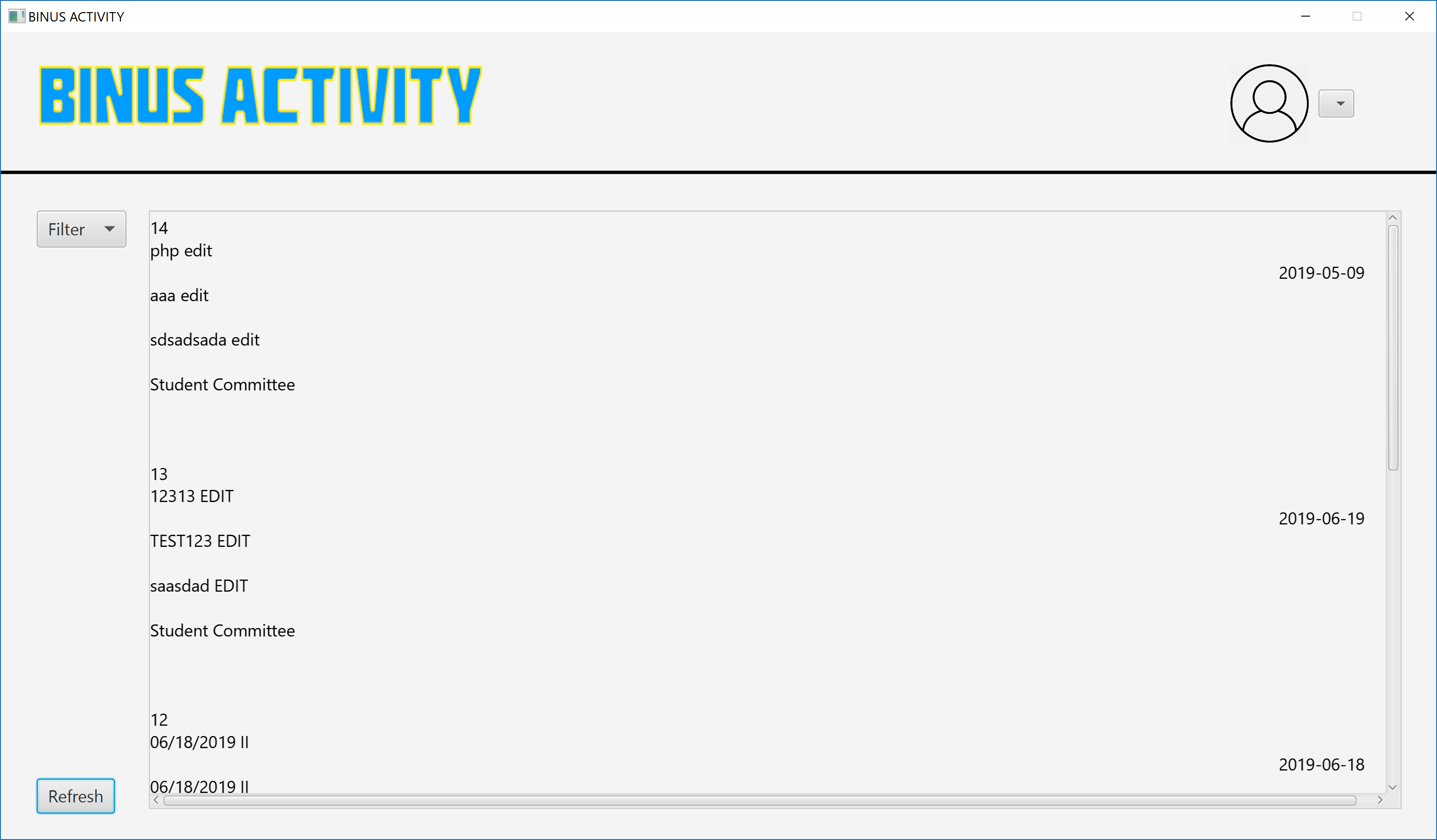
**vii. Edit post**

First, the admin is asked to input the post id. Printed on top of every post in the home page. Then the admin can press done then they will be transported into another window where they can edit every detail of the post. They can press done to complete the action.

  
1.13 Press the edit post button on the drop-down menu

  
1.14 Input the auto incremented id of the post that the admin wants to edit, printed on the admin home page. Press done when the admin is finished, and the admin will be directed to a new age where they can edit the post

  
1.15 The text box will already be filled with the details of the post that the admin would like to edit. The admin should just change any one of them or all of them then press done and the database will be updated, the post updated

  
1.16 Notice that the post of id 13 is already edited and live on the home page