Fpt University

TRƯỜNG ĐẠI HỌC FPT

ACADEMIC BLOG AT FPTU

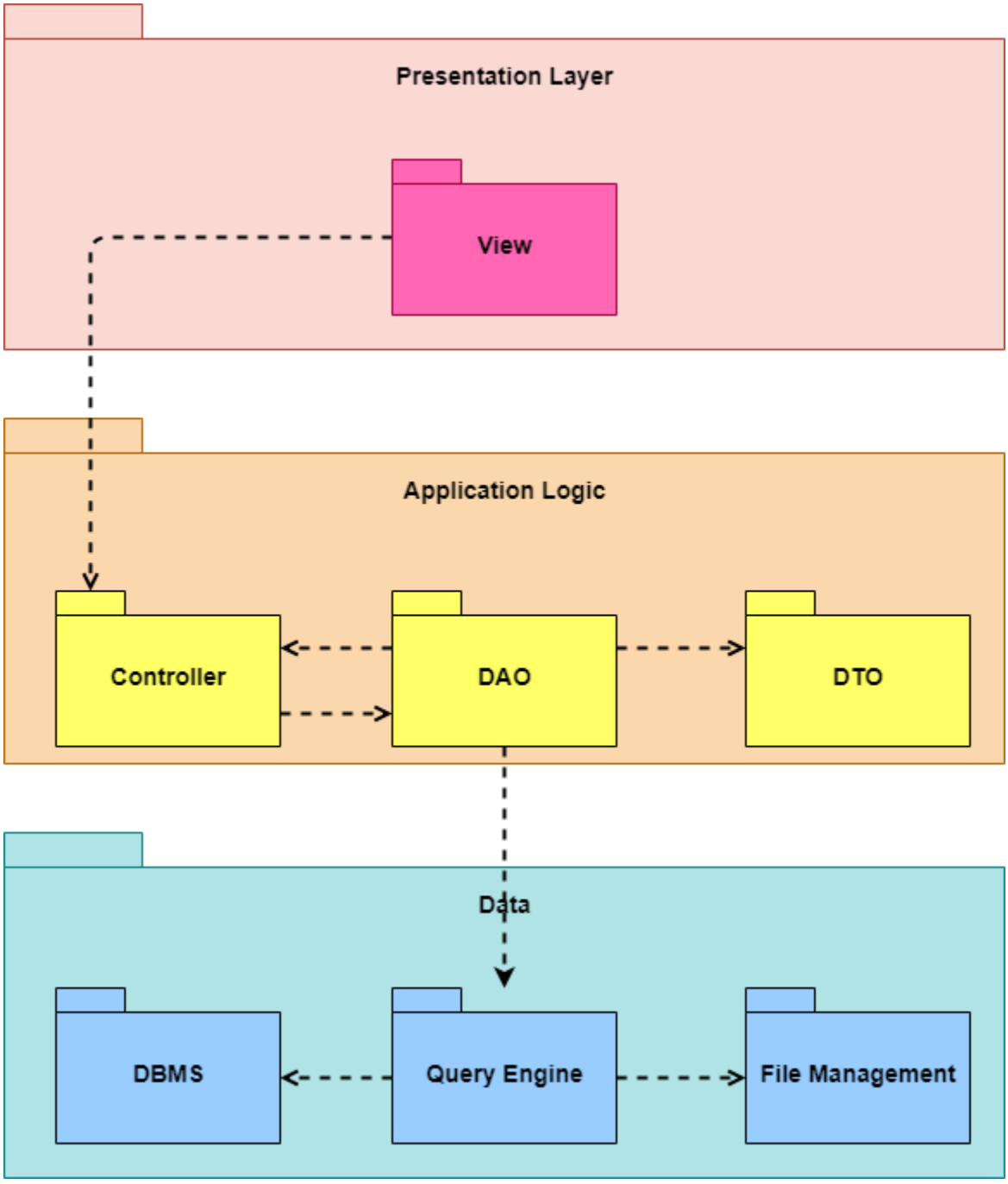
Software Design Document

– Ho Chi Minh, May 2022 –

Table of Contents

I. Overview

1. Code Packages/Namespaces



Package descriptions & package class naming conventions

No	Package	Description
01	View	This is where the JSP pages do the work of displaying the user interface. The first word is in lowercase (camel case notation).
02	Controller	Functional controllers will be passed through the MainController to execute its functions. Capital letters at the beginning of each word , according to the syntax: Verb+Controller
03	DAO	Contains functions that execute commands related to handling the database. Nouns, capital letters for each word. Write whole words, avoid abbreviations.
04	DTO	Contains all DTO classes

2. Database Schema

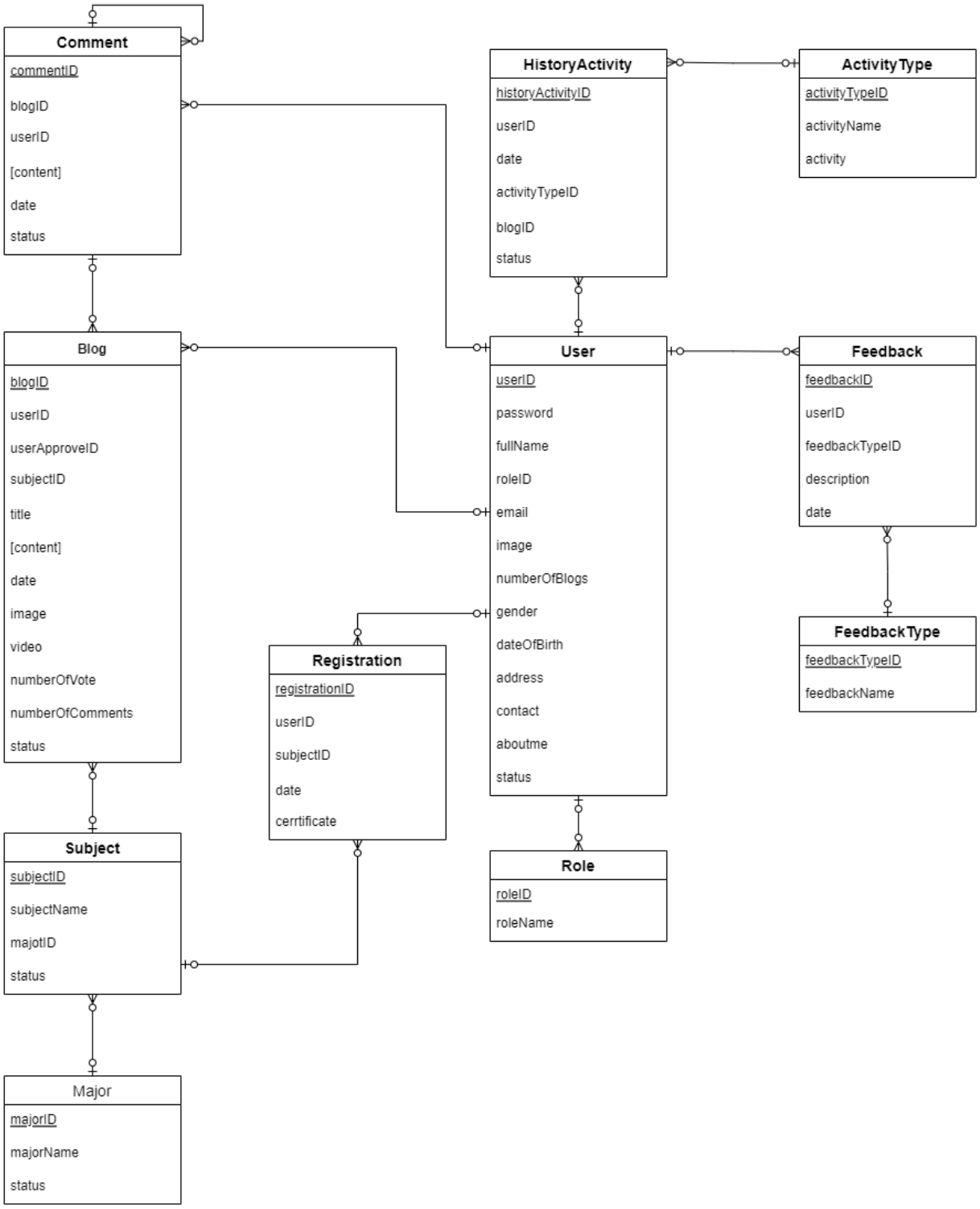


Table descriptions & package class naming conventions are as below

No	Table	Description
01	*dbo.Blog*	- The table contains information about a specific blog, that subject name such as blog's id, author's id, comment id, subject's id and mentor approver's id, the title of blog, date viết, image/video, number of vote, number of comment and status of blog - Primary keys: BlogID - Foreign keys: UserID, CommentID, SubjectID
02	*dbo.User*	- The table contains information about a particular user such as user's id, and role id, email , password, full name, image, number of the blogs , gender, date of birth, address , contact, information of user and the status - Primary keys: userID - Foreign keys: roleID
03	*dbo.Major*	- The table contains information about a major cụ thể such as major's id, tên của major and status. - Primary keys: majorID - Foreign keys: None
04	*dbo.Subject*	The table contains information about a specific subject such as the subject's id, the name of that subject, the major to which that subject belongs, and the status of that subject. - Primary keys: subjectID - Foreign keys: majorID
05	*dbo.Comment*	The table contains information that the comment should have such as the id, the id of the blog to which the comment belongs, who owns this comment, the content of the comment, the date the user left the comment, and the status of that comment. - Primary keys: commentID - Foreign keys: blogID, userID, commentID
06	*dbo.Registration*	- The table contains information about a Registration such as registration' id, user's id, subject's id , date register and certificate - Primary keys: registrationID - Foreign keys: userID, subjectID
07	*dbo.Feedback*	- The table contains information about feedback such as feedback id and author's id, type feedback id, description of feedback and date post feedback. - Primary keys: feedbackID - Foreign keys: userID
08	*dbo.Role*	- The table contains information about a role of user such as role's id and role name - Primary keys: roleID - Foreign keys: None
09	*dbo.FeedbackType*	- The table contains information about a type of feedback such as type feedback id and feedback name. - Primary keys: feedbackTypeID - Foreign keys: None
10	*dbo.HistoryActivity*	- The table contains information about a history activity such as history activity id, activity type id, user's id, date of activity, blog id and status - Primary keys: historyActivityID - Foreign keys: userID, activityTypeID

11	*dbo.ActivityType*	- The table contains information about a activity type such as activity type id, activity Name, activity - Primary keys: activityTypeID - Foreign keys: userID, blogID

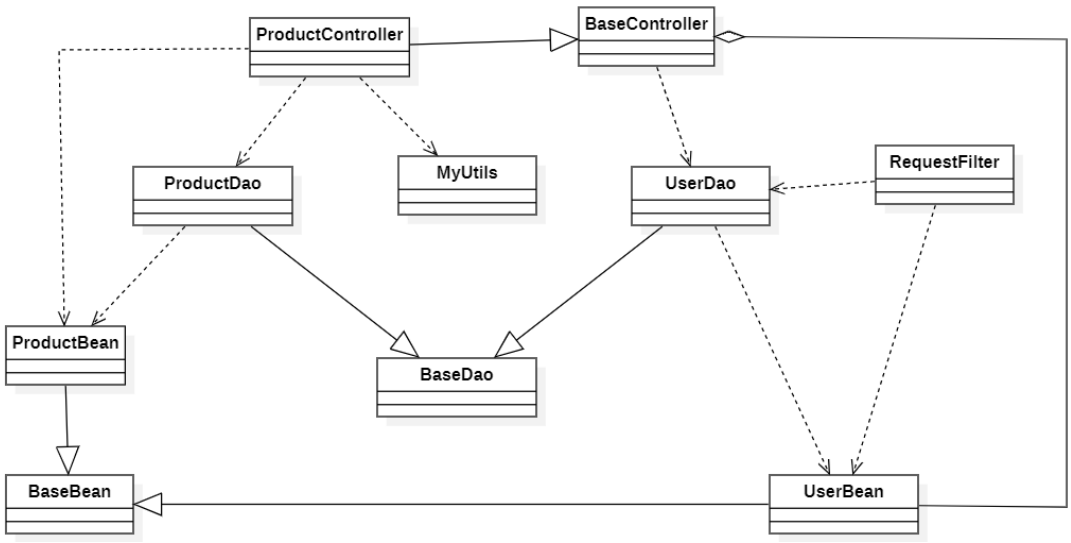
II. Code Designs

1. <Feature/Function Name1>

[Provide the detailed design for the function <Feature/Fun ction Name1>. It include Class Diagram, Class Specifications, and Sequence Diagram(s)]

a. Class Diagram

[This part presents the class diagram for the relevant feature]



b. Class Specifications

DBUtils Class

No	Method	Description
01	*public static Connection getConnection()*	*This is a commonly used method to connect with the database. In this method, we need to provide a class name, URL, user, and password for connecting with the database. If connected successfully, return an object in connection type, else null.*

GoogleUtils Class

No	Method	Description
01	*public static String getToken(final String code)*	*This is a commonly used method to create an access token for signing in with a Google account. In this method, we need to provide a code and process google information such as google client ID, google client secret, etc for the response. If created successfully, return an access token with the String type, else null.*

02	*public static GooglePojo getUserInfo(final String accessToken)*	*This is a commonly used method to get user information from a Google account. In this method, we need to provide an access token for creating a link to request a response. If returning a response, return a Google Pojo object, else null.*

ActivityDAO Class

No	Method	Description
01	*public List<ActivityDTO> getAllActivities(int userID)*	*This method is used to get all activities of an account. In this method, we need to provide a user ID to get all the activities of this account, create an object to store all properties of the activity, and add it to the list of the activity. If the size of the list is not null, return the list of all activities, else null.*
02	*public boolean deleteActivity(int historyActivityID)*	*This method is used to delete an activity of an account by setting a new status. In this method, we need to provide an activity ID to set a new status for this activity of this account. If set successfully, returns true, else false.*
03	*public boolean findVoteActivity(int blogID, int userID)*	*This method is used to find the Vote activity for checking the Vote activity of an account. In this method, we need to provide a blog ID and a user ID to find the Vote activity of this account. If finding a Vote activity successfully, return true, else false.*
04	*public boolean updateActivity(int blogID, int userID, String date)*	*This method is used to create an activity of an account. In this method, we need to provide a blog ID, a user ID, and the date that the user did this activity and insert this activity into the database. If inserted successfully, return true, else false.*
05	*public boolean deleteUpdate(int blogID, int userID)*	*This method is used to delete an activity of an account by setting a new status. In this method, we need to provide a blog ID and a user ID to delete this activity of this account. If deleted successfully, returns true, else false.*
06	*public List<ActivityDTO> SearchActivitiesByName(String searchName, int userID)*	*This method is used to search the user's activity history, we need to provide the search information and the userID to search the activity history. If Search successfully, returns list of matching activities*

BlogDAO Class

No	Method	Description
01	*public List<BlogDTO> getAllBlogs()*	*This method is used to get all blogs in the database for display on the homepage screen. In this method, we create an object to store all properties of a blog with the status "approved" and add it to the list of the blog. If the size of the list is not null, return the list of all blogs, else null.*
02	*public boolean updateVote(int blogID, int numberOfVotes)*	*This method is used to update the number of votes on a blog. In this method, we need to provide a blog ID, a new number of votes, and update this number for this blog. If updated successfully, returns true, else false.*

03	*public List<BlogDTO> getAllApproveBlogs(int loginUserID)*	*This method is used to get all approved blogs in the database for display on the mentor page. In this method, we need to provide a user ID to select the blogs that have the owner differ from this user and create an object to store all properties of a blog with the status "waiting" and add it to the list of the blog. If the size of the list is not null, return the list of all blogs, else null.*
04	*public List<BlogDTO> searchByTitle(String search)*	*This method is used to search for approved blog posts from the database and display it on the Home page. In this method, we need to provide part or all of the blog post title to search for matching blog posts . If Search successfully, returns list of matching blogs*
05	*public BlogDTO BlogDetail(int blogID)*	*This method is used so that users can view the details of the blog post in the most complete way on the Blog detail page. In this method, we need to provide a blog ID so that the information retrieved from the database is correct with the blog post the user needs to see details. If matching successfully, returns detailed content of the blog post*
06	*public boolean postBlog(int userID, int subjectID, String title, String content, String date, String image)*	*This method is used to post user's blog posts to the system with "waiting" status. In this method, we need to provide the information of the blog post such as user ID, subject ID, title, content,.. to create a complete blog post. If posting successfully, returns notice posted successfully*
07	*public BlogDTO getBlogByID(int blogID)*	*This method is used to find the exact blog post that users want to visit. In this method, we need to provide the blog ID to the system from which the system will get the data of the blog post matching that ID to display to the user*
08	*public boolean draftBlog(int userID, int subjectID, String title, String content, String date, String image)*	*This method is used to post user's blog posts to the system with "draft" status. In this method, we need to provide the information of the blog post such as user ID, subject ID, title, content,.. to create an incomplete blog post. If posting successfully, returns notice posted successfully*
09	*public boolean deleteBlog(int blogID)*	*This method is used to delete a blog by setting the status from "approved" to "disabled". In this method we need to provide a blog ID to set a new status for this blog. If successful, return true, else false.*
10	*public static int editBlog(int blogID, int subjectID, String title, String content, String date, String image, String video)*	*This method is used to edit a user's blog posts to the system with "waiting" status. In this method, we need to provide the information of the blog post such as user ID, subject ID, title, content,.. to edit a complete blog post. If editing successfully, returns notice posted successfully*
11	*public boolean approveBlog(int blogID)*	*This method is used by the mentor to approve certain quality blog posts and change the status to "approved" for that blog post. In this method, we need to provide the blog ID so that the system can correctly identify the blog post so that the mentor can approve it. If approved successfully, returns the notice approved successfully and blog post will be displayed in the home page*

12	*public boolean rejectBlog(int blogID)*	*This method is used by mentors to reject blog posts that don't match the requirements and change the status to "rejected" for that blog post. In this method, we need to provide the blog ID so that the system can correctly identify the blog post so that the mentor can reject it. If the refusal is successful, return the message of the successful refusal and provide the student with the reason.*
13	*public List<BlogDTO> getAllPersonalBlogs(int userID)*	*This method is used to get all personal blogs in the database for display on the personal page screen. In this method, we create an object to store all properties of a blog with the status "approved" and add it to the list of the blog. If the size of the list is not null, return the list of all blogs, else null.*

CommentDAO Class

No	Method	Description
01	*public static CommentDTO getCommentbyBlogID(int blogID)*	*This method is used to get all comments of the blog by blog id. In this method, we need to provide a blog id. The result will return a list of comments whose blog id is the blog id entered*
02	*public static int insertComment(int blogID, int userID, String content,String date)*	*This method is used to insert comments into the blog. In this method, we need to provide a blog id, user id , content of the comment and date to insert the comment. If successful, the comment will be updated to the database and return a non-zero int value*

FeedbackDAO Class

No	Method	Description
01	*public List<FeedbackDTO> getAllFeedback()*	*This method is used to get all feedback in the database for display on the feedback manager. In this method if the size of list is not null, it will return list of user feedback, else null*
02	*public FeedbackDTO getFeedbackByID(int id)*	*This method is used to get details of feedback by feedback id . In this method, we need to provide the feedback id and the system will return the detail of the feedback*
03	*public List<FeedbackTypeDTO> getAllFeedbackTypes( )*	*This method is used to get all feedback types in the database for display on the feedback screen. In this method, we create a list to store all the feedback names and add them to the list of the feedback type. If the size of the list is not null, return the list of all feedback types, else null.*
04	*public boolean giveFeedback(int userID, int feedbackTypeID, String description, String date)*	*This method is used to create feedback for a student, mentor, or system. In this method, we need to provide a user ID (who wrote this feedback), a feedback type ID (1 for the system, 2 for the student, and 3 for the mentor), the description, the date that this feedback was sent, and insert into the database. If inserted successfully, returns true, else false.*

MajorDAO Class

No	Method	Description

01	*public List<MajorDTO> getAllMajors()*	*This method is used to get all majors in the database for display on the manage majors screen. In this method, we create a list to store all the major names and add them to the list of the major. If the size of the list is not null, return the list of all majors, else null.*
02	*public int editMajor(int majorID, String majorName)*	*This method is used to update details of major in the database. In this method, we need to provide the major id, major name. The result is returns non-zero int if edit major succeeds*
03	*public boolean deleteMajor(int majorID)*	*This method is used to delete a major by setting a new status. In this method, we need to provide a major ID to set a new status for this major. If set successfully, returns true, else false.*
04	*public int createMajor(int majorID, String majorName)*	*This method is major to create a new major. In this method, we need to provide a major name, of this major and insert this major into the database. If inserted successfully, return true, else false.*

SubjectDAO Class

No	Method	Description
01	*public static ArrayList<SubjectDTO > getSubject()*	*This method is used to get all subjects in the database for display on the manage subjects screen. In this method, we create a list to store all the subject names and add them to the list of the subject. If the size of the list is not null, return the list of all subjects, else null.*
02	*public int editSubject(int subjectID, int majorID, String subjectName)*	*This method is used to update details of a subject in the database. In this method, we need to provide the subject id, major id, and subject name. The result is returns non-zero int if edit major succeeds*
03	*public boolean deleteSubject(int subjectID)*	*This method is used to delete a subject by setting a new status. In this method, we need to provide a subject ID to set a new status for this subject. If setting successfully, returns true, else false.*
04	*public int createSubject(int subjectID, int majorID, String subjectName)*	*This method is subject to create a new subject. In this method, we need to provide a subject name, of this subject and insert this subject into the database. If inserted successfully, return true, else false.*

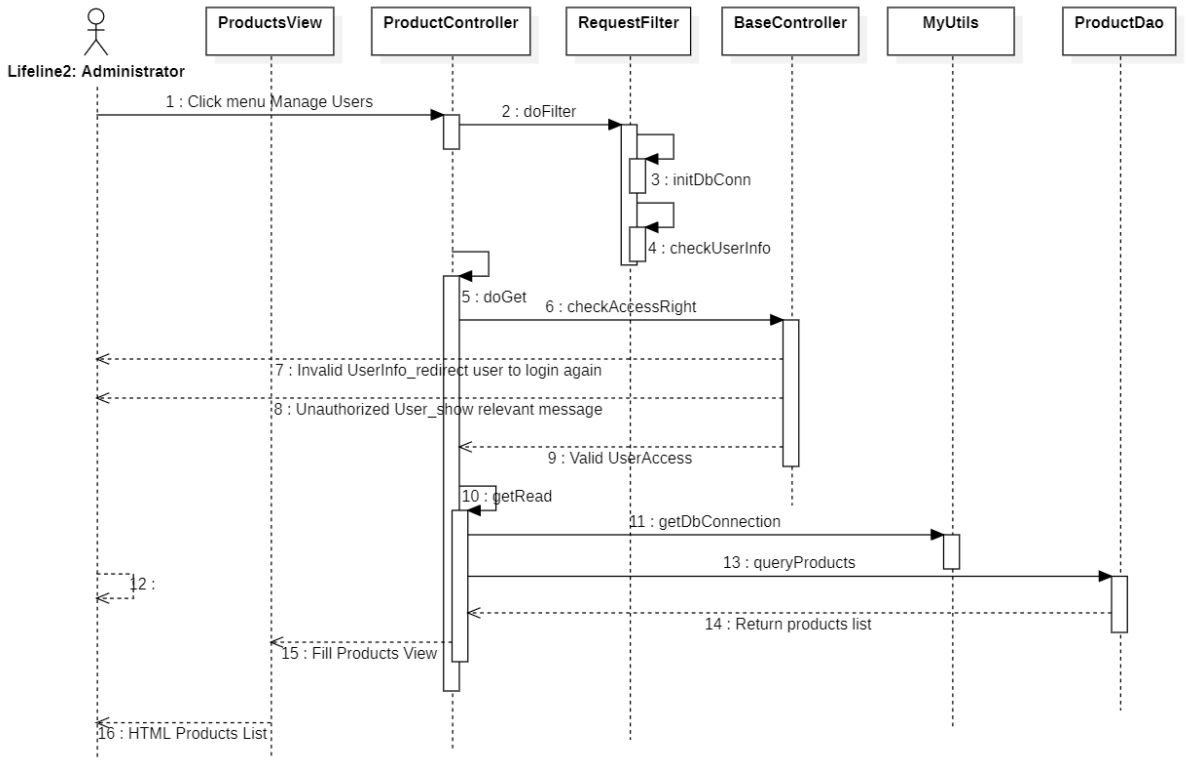
UserDAOClass

No	Method	Description
01	*public UserDTO checkLogin(String email, String password)*	*This method is used to check the existence of an account for logging in by email and password. In this method, we need to provide an email and a password to find this account. If finding an account successfully, return user information with UserDTO type, else null.*
02	*public UserDTO checkLoginByEmail(String email)*	*This method is used to check the existence of an account for logging in by Google accounts. In this method, we need to provide an email to find this account. If finding an account successfully, return user information with UserDTO type, else null.*

03	*public String checkRole(int roleID)*	*This method is used to check the role of an account for logging in by Google accounts. In this method, we need to provide a role ID to find the name of this role. If finding the role name successfully, return role name, else null.*
04	*public boolean createUser(String fullName, String email, String image)*	*This method is used to create a new user for logging in by Google account. In this method, we need to provide a full name, an email, and the image of this account and insert this user into the database. If inserted successfully, return true, else false.*
05	*public UserDTO GetUserByID(int id)*	*This method is used to find user information for displaying them on the profile page. In this method, we need to provide a user ID and create an object to store all user information. If finding an account successfully, returns the user information with UserDTO type, else null.*
06	*public List<UserDTO> getAllUser()*	*This method is used to get all users to display them on the manage account page. In this method, we need to provide a user ID and create an object to store all user information. If finding an account successfully, returns the user information with UserDTO type, else null.*
07	*public static int updateStatusUser(int userID, String oldStatus)*	
08	*public static int editUser(int userID, String fullName, String image, String gender, String dateOfBirth, String address, String contact, String aboutme)*	*This method is used so that users can customize information such as: userID, fullName, image, gender, ... in the profile page screen. In this method, we need to provide the information that we need to edit such as: userID, fullName, image, gender, ... from there editing personal information is complete . If editing a profile successfully, returns the user's new user profile , else error.*

c. Sequence Diagram(s)

[Provide the sequence diagram(s) for the feature, see the sample below]



d. Database queries

1. Blog DAO class

No	Method	Database Queries
01	*public List<BlogDTO> getAllBlogs()*	*SELECT blogID,Blog.userID,userApproveID,subjectID,title,content,date, Blog.image,video, numberOfVotes,numberOfComments,Blog.status, fullName FROM Blog JOIN [USER] ON Blog.userID = [User].userID WHERE Blog.status LIKE 'approved'*
02	*public List<BlogDTO> searchByTitle(String search)*	*SELECT blogID,Blog.userID,userApproveID,subjectID,title,content,date, Blog.image,video,numberOfVotes,numberOfComments,Blog.status, fullName FROM Blog JOIN [USER] ON Blog.userID = [User].userID WHERE title LIKE ? AND Blog.status LIKE 'approved'*
03	*public BlogDTO getBlogByID(int blogID)*	*SELECT blogID,Blog.userID,userApproveID,subjectID,title,content,date, Blog.image,video, numberOfVotes,numberOfComments,Blog.status, fullName FROM Blog JOIN [USER] ON Blog.userID = [User].userID WHERE blogID = ?*
04	*public boolean postBlog(int userID, int subjectID, String title, String content, String date, String image)*	*INSERT INTO [Blog](userID, userApproveID, subjectID, title, content, date, image, video, numberOfVotes, numberOfComments, status) VALUES(?,null,?,?,?,?,?,null,0,0,'waiting')*

05	*public boolean updateVote(int blogID, int numberOfVotes)*	*UPDATE Blog SET numberOfVotes = ? WHERE blogID = ?*
06	*public List<BlogDTO> getAllApproveBlogs(int loginUserID)*	*SELECT blogID,Blog.userID,userApproveID,subjectID,title,content,date, Blog.image,video,numberOfVotes,numberOfComments,Blog.status, fullName FROM Blog JOIN [USER] ON Blog.userID = [User].userID WHERE Blog.status LIKE 'waiting' AND Blog.userID != ?*
08	*public boolean deleteBlog(int blogID)*	*UPDATE Blog SET status = 'disable' WHERE blogID = ?*
09	*public static int editBlog(int blogID, int subjectID, String title, String content, String date, String image, String video)*	*UPDATE Blog set subjectID=?, title=? , content=?,date=?,image=?, video=? where blogID=?*
10	*public boolean approveBlog(int blogID)*	*UPDATE Blog SET status= 'approved' WHERE blogID = ?*
11	*public boolean rejectBlog(int blogID)*	*UPDATE Blog SET status= 'rejected' WHERE blogID = ?*
12	*public List<BlogDTO> getAllPersonalBlogs(int userID)*	*SELECT blogID,Blog.userID,userApproveID,subjectID,title,content,date, Blog.image,video, numberOfVotes,numberOfComments,Blog.status, fullName FROM Blog JOIN [USER] ON Blog.userID = [User].userID WHERE Blog.userID = ?*
13	*public boolean draftBlog(int userID, int subjectID, String title, String content, String date, String image)*	*INSERT INTO [Blog](userID, userApproveID, subjectID, title, content, date, image, video, numberOfVotes, numberOfComments, status) VALUES(?,null,?,?,?,?,?,null,0,0,'draft')*

2. ActivityDAO Class

No	Method	Description
01	*public List<ActivityDTO> getAllActivities(int userID)*	*SELECT historyActivityID, date, activity FROM HistoryActivity JOIN ActivityType ON HistoryActivity.activityTypeID = ActivityType.activityTypeID WHERE userID = ? AND status = 1*
02	*public boolean deleteActivity(int historyActivityID)*	*UPDATE HistoryActivity SET status = 0 WHERE historyActivityID = ?*
03	*public boolean findVoteActivity(int blogID, int userID)*	*SELECT historyActivityID FROM HistoryActivity WHERE blogID = ? AND userID = ? AND activityTypeID = 1*

04	public boolean updateActivity(int blogID, int userID, String date)	INSERT INTO HistoryActivity(userID, date, activityTypeID, blogID, status) VALUES(?,?,1,?,1)
05	public boolean deleteUpdate(int blogID, int userID)	DELETE HistoryActivity WHERE userID = ? AND blogID = ? AND activityTypeID = 1
06	public List<ActivityDTO> SearchActivitiesByName(String searchName, int userID)	SELECT historyActivityID,userID, date, activity FROM HistoryActivity h JOIN ActivityType a ON h.activityTypeID = a.activityTypeID WHERE a.activity like ? AND h.userID = ? AND h.status = 1

3. UserDAOClass

No	Method	Description
01	public UserDTO checkLogin(String email, String password)	SELECT userID, fullName, roleID, image, numberOfBlogs, gender, dateOfBirth, address, contact, aboutme, status FROM [User] WHERE email like ? AND password like ?
02	public UserDTO checkLoginByEmail(String email)	SELECT userID, password, fullName, roleID, image, numberOfBlogs, gender, dateOfBirth, address, contact, aboutme, status FROM [User] WHERE email like ?
03	public String checkRole(int roleID)	SELECT roleName FROM Role WHERE roleID = ?
04	public boolean createUser(String fullName, String email, String image)	INSERT INTO [User](password, fullName, roleID, email, image, numberOfBlogs, gender, dateOfBirth, address, contact, aboutme, status) VALUES(?,?,2,?,?,0,null,null,null,null,null,1)
05	public UserDTO GetUserByID(int id)	SELECT password, fullName, roleID,email, image, numberOfBlogs, gender, dateOfBirth, address, contact, aboutme, status FROM [User] WHERE UserID = ?
06	public List<UserDTO> getAllUser()	SELECT userID, password, fullName, roleID, email, image, numberOfBlogs, gender, dateOfBirth, address, contact, aboutme, status FROM [User]
07	public static int updateStatusUser(int userID, String oldStatus)	UPDATE [User] SET status = ? Where userID = ?
08	public static int editUser(int userID, String fullName, String image, String gender, String	UPDATE [User] SET fullName=?, image=?, gender=?, dateOfBirth=?, address=?, contact=?, aboutme=? " WHERE userID=?

	dateOfBirth, String address, String contact, String aboutme)	

4. MajorDAO Class

No	Method	Description
01	*public List<MajorDTO> getAllMajors()*	*SELECT majorID, majorName, status FROM Major WHERE status = 1*
02	*public int editMajor(int majorID, String majorName)*	*UPDATE Major SET status = 0 WHERE majorID = ?*
03	*public boolean deleteMajor(int majorID)*	*UPDATE Major SET status = 0 WHERE majorID = ?*
04	*public int createMajor(int majorID, String majorName)*	*INSERT INTO Major(majorID, majorName, status) VALUE(?,?,1)*

5. SubjectDAO Class

No	Method	Description
01	*public static ArrayList<SubjectDTO > getSubject()*	*SELECT subjectID, subjectName,majorID,status FROM Subject WHERE status = 1*
02	*public int editSubject(int subjectID, int majorID, String subjectName)*	*UPDATE Subject SET status = 0 WHERE subjectID = ?*
03	*public boolean deleteSubject(int subjectID)*	*SET subjectName = ?, majorID = ?* *WHERE subjectID = ?*
04	*public int createSubject(int subjectID, int majorID, String subjectName)*	*INSERT INTO Subject(subjectID, majorID, subjectName, status)* *VALUE(?,?,?,1)*

6. CommentDAO Class

No	Method	Description
01	public static CommentDTO getCommentbyBlogID(int blogID)	SELECT commentID, userID, content, date, status FROM Comment WHERE blogID = ? AND status = 1

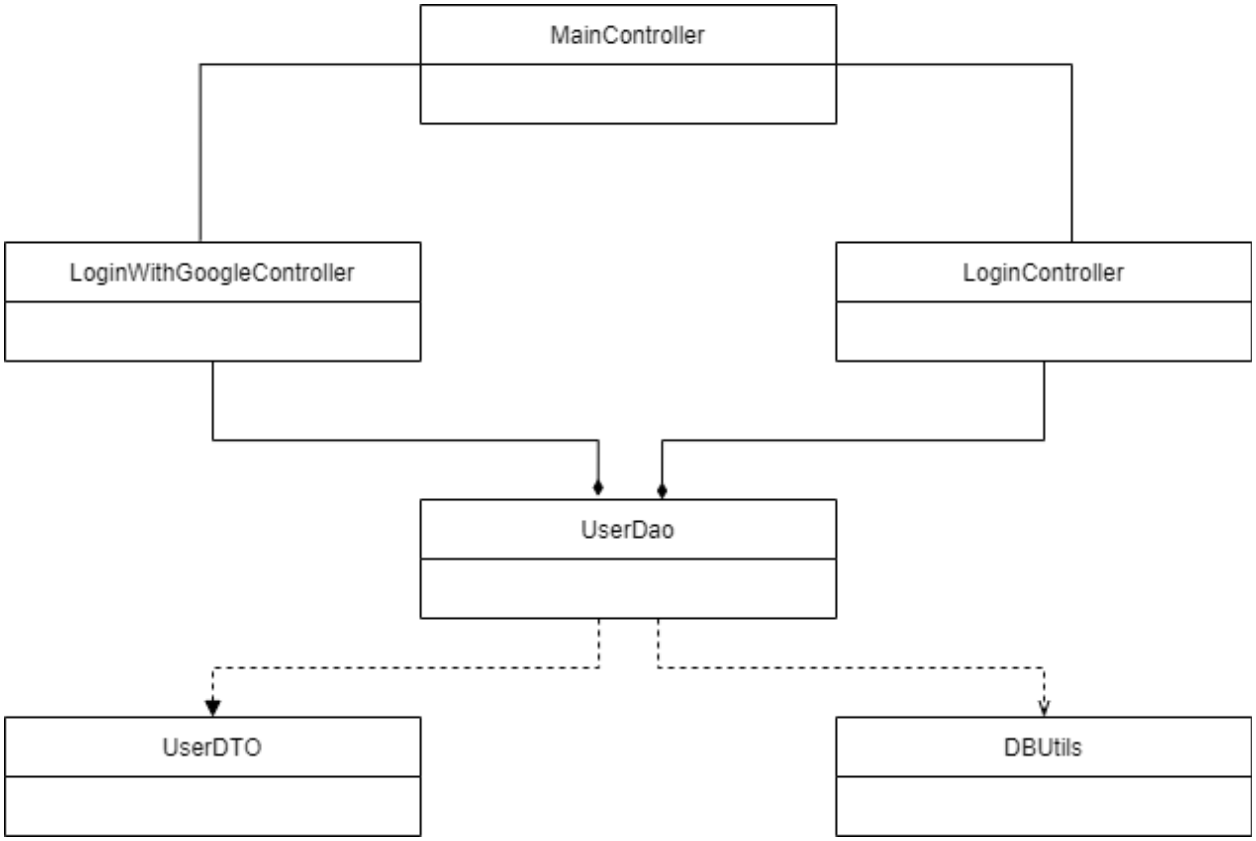
02	*public static int insertComment(int blogID, int userID, String content,String date)*	INSERT INTO Comment(blogID,userID,content,date,status) VALUES(?,?,?,?,1)

7. FeedbackDAO Class

No	Method	Description
01	*public List<FeedbackDTO> getAllFeedback()*	*SELECT [feedbackID],[userID],[feedbackTypeID],[description],[date] FROM [ABF].[dbo].[Feedback]*
02	*public FeedbackDTO getFeedbackByID(int id)*	*SELECT [feedbackID],[userID],[feedbackTypeID],[description],[date] FROM [ABF].[dbo].[Feedback]*
03	*public List<FeedbackTypeDTO> getAllFeedbackTypes( )*	*SELECT feedbackTypeID, feedbackName from FeedbackType*
04	*public boolean giveFeedback(int userID, int feedbackTypeID, String description, String date)*	*INSERT INTO Feedback(userID, feedbackTypeID, description, date) VALUES(?,?,?,?)*

1. Welcome

a. Class Diagram



b. Class Specifications

GoogleUtils Class

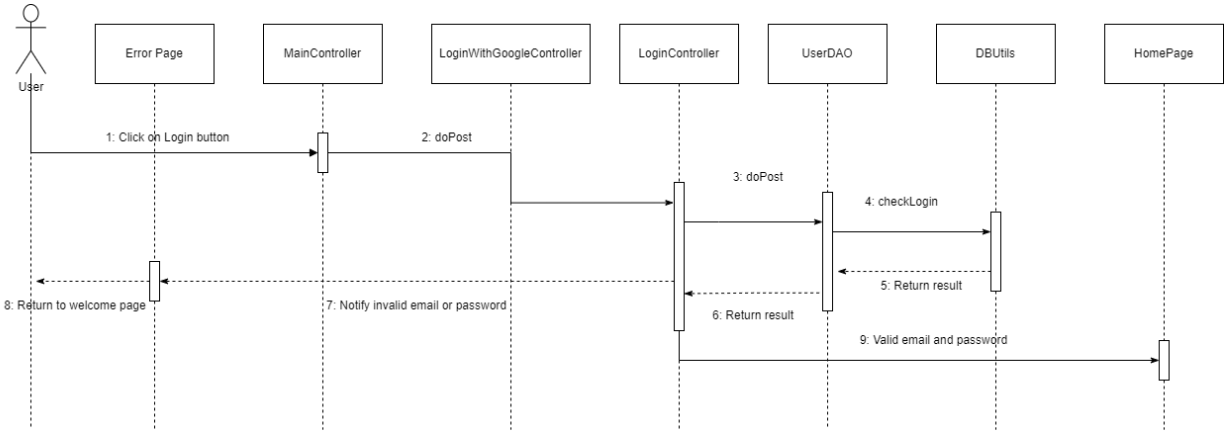
No	Method	Description
01	public static String getToken(final String code)	This is a commonly used method to create an access token for signing in with a Google account. In this method, we need to provide a code and process google information such as google client ID, google client secret, etc for the response. If created successfully, return an access token with the String type, else null.
02	public static GooglePojo getUserInfo(final String accessToken)	This is a commonly used method to get user information from a Google account. In this method, we need to provide an access token for creating a link to request a response. If returning a response, return a Google Pojo object, else null.

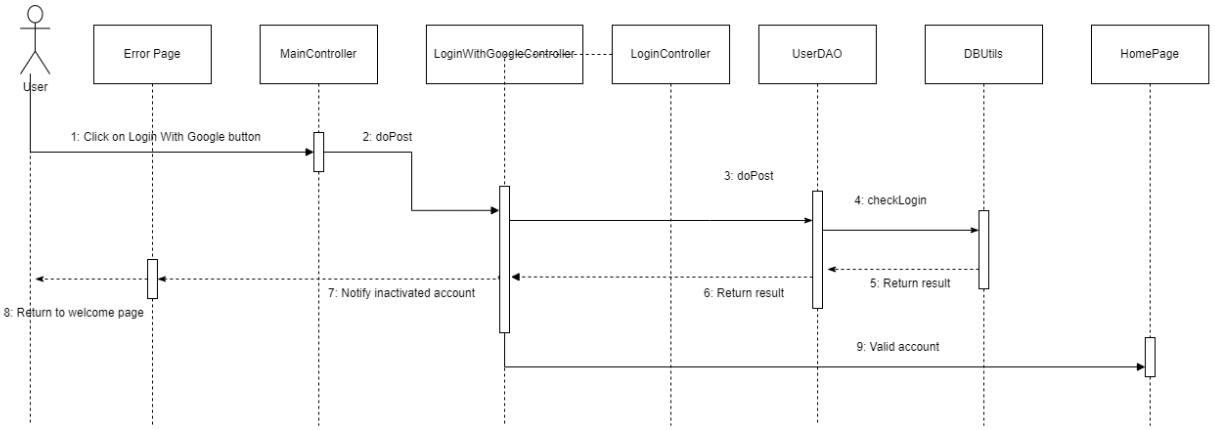
UserDAOClass

No	Method	Description
01	public UserDTO checkLogin(String email, String password)	This method is used to check the existence of an account for logging in by email and password. In this method, we need to provide an email and a password to find this account. If finding an account successfully, returns user information with UserDTO type, else null.

02	public UserDTO checkLoginByEmail(String email)	This method is used to check the existence of an account for logging in by Google accounts. In this method, we need to provide an email to find this account. If finding an account successfully, return user information with UserDTO type, else null.
03	public String checkRole(int roleID)	This method is used to check the role of an account for logging in by Google accounts. In this method, we need to provide a role ID to find the name of this role. If finding the role name successfully, return role name, else null.
04	public boolean createUser(String fullName, String email, String image)	This method is used to create a new user for logging in by Google account. In this method, we need to provide a full name, an email, and the image of this account and insert this user into the database. If inserted successfully, return true, else false.

c. Sequence Diagram(s)





d. Database queries

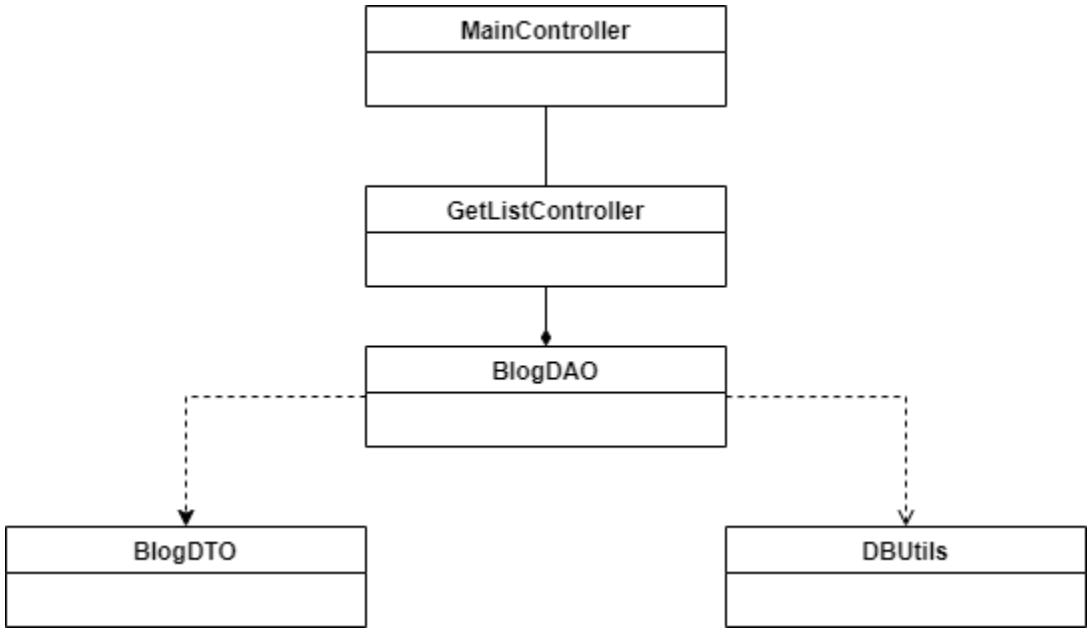
UserDAOClass

No	Method	Description
01	public UserDTO checkLogin(String email, String password)	SELECT userID, fullName, roleID, image, numberOfBlogs, gender, dateOfBirth, address, contact, aboutme, status FROM [User] WHERE email like ? AND password like ?

02	public UserDTO checkLoginByEmail(String email)	SELECT userID, password, fullName, roleID, image, numberOfBlogs, gender, dateOfBirth, address, contact, aboutme, status FROM [User] WHERE email like ?
03	public String checkRole(int roleID)	SELECT roleName FROM Role WHERE roleID = ?
04	public boolean createUser(String fullName, String email, String image)	INSERT INTO [User](password, fullName, roleID, email, image, numberOfBlogs, gender, dateOfBirth, address, contact, aboutme, status) VALUES(?,?,2,?,?,0,null,null,null,null,null,1)

2. Home page

a. Class Diagram



b. Class Specifications

BlogDAO Class

No	Method	Description
01	public List<BlogDTO> getAllBlogs()	This method is used to get all blogs in the database for display on the homepage screen. In this method, we create an object to store all properties of a blog with the status “approved” and add it to the list of the blog. If the size of the list is not null, return the list of all blogs, else null.

c. Sequence Diagram(s)

d. Database queries

Blog DAO class

No	Method	Database Queries

01	public List<BlogDTO> getAllBlogs()	SELECT blogID,Blog.userID,userApproveID,subjectID,title,content,date, Blog.image,video, numberOfVotes,numberOfComments,Blog.status, fullName FROM Blog JOIN [USER] ON Blog.userID = [User].userID WHERE Blog.status LIKE 'approved'

3. Write new blog

a. Class Diagram

b. Class Specifications

Blog DAO class

No	Method	Database Queries
01	public boolean postBlog(int userID, int subjectID, String title, String content, String date, String image)	INSERT INTO [Blog](userID, userApproveID, subjectID, title, content, date, image, video, numberOfVotes, numberOfComments, status) VALUES(?,null,?,?,?,?,?,null,0,0,'waiting')
02	public boolean draftBlog(int userID, int subjectID, String title, String content, String date, String image)	INSERT INTO [Blog](userID, userApproveID, subjectID, title, content, date, image, video, numberOfVotes, numberOfComments, status) VALUES(?,null,?,?,?,?,?,null,0,0,'draft')

ActivityDAO Class

No	Method	Description
01	public boolean updateActivity(int blogID, int userID, String date)	INSERT INTO HistoryActivity(userID, date, activityTypeID, blogID, status) VALUES(?,?,1,?,1)

c. Sequence Diagram(s)

d. Database queries

Blog DAO class

No	Method	Database Queries
01	public boolean postBlog(int userID, int subjectID, String title, String content, String date, String image)	INSERT INTO [Blog](userID, userApproveID, subjectID, title, content, date, image, video, numberOfVotes, numberOfComments, status) VALUES(?,null,?,?,?,?,?,null,0,0,'waiting')
02	public boolean draftBlog(int userID, int subjectID, String	INSERT INTO [Blog](userID, userApproveID, subjectID, title, content, date, image, video, numberOfVotes, numberOfComments, status) VALUES(?,null,?,?,?,?,?,null,0,0,'draft')

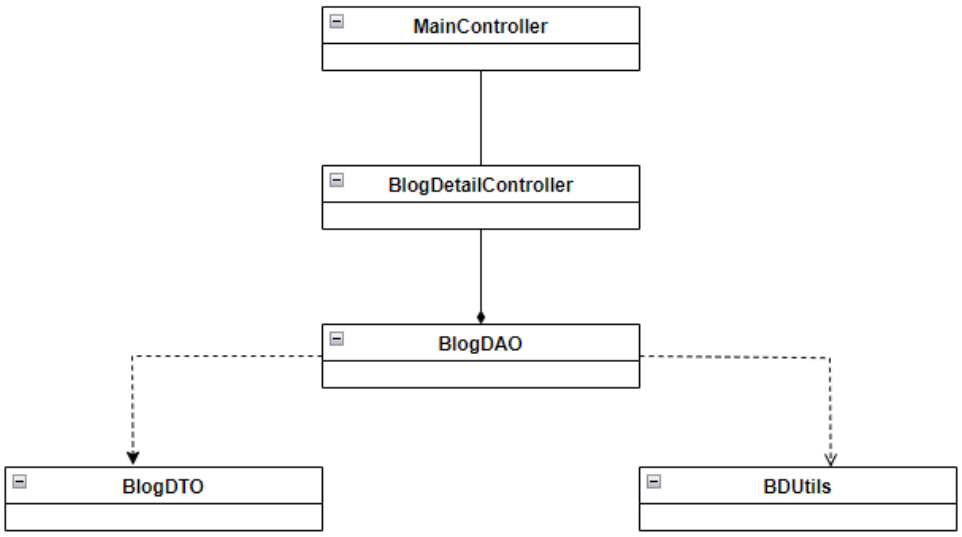
	title, String content, String date, String image)	

ActivityDAO Class

No	Method	Description
01	public boolean updateActivity(int blogID, int userID, String date)	INSERT INTO HistoryActivity(userID, date, activityTypeID, blogID, status) VALUES(?,?,1,?,1)

4. View blog

a. Class Diagram



b. Class Specifications

BlogDAO Class

No	Method	Description
01	public BlogDTO BlogDetail(int blogID)	This method is used so that users can view the details of the blog post in the most complete way on the Blog detail page. In this method, we need to provide a blog ID so that the information retrieved from the database is correct with the blog post the user needs to see details. If matching successfully, returns detailed content of the blog post
02	public boolean deleteBlog(int blogID)	This method is used to delete a blog by setting the status from "approved" to "disabled". In this method we need to provide a blog ID to set a new status for this blog. If successful, return true, else false.
08	public boolean draftBlog(int userID,	This method is used to post user's blog posts to the system with "draft" status. In this method, we need to provide the information of

	int subjectID, String title, String content, String date, String image)	the blog post such as user ID, subject ID, title, content,.. to create an incomplete blog post. If posting successfully, returns notice posted successfully

c. Sequence Diagram(s)

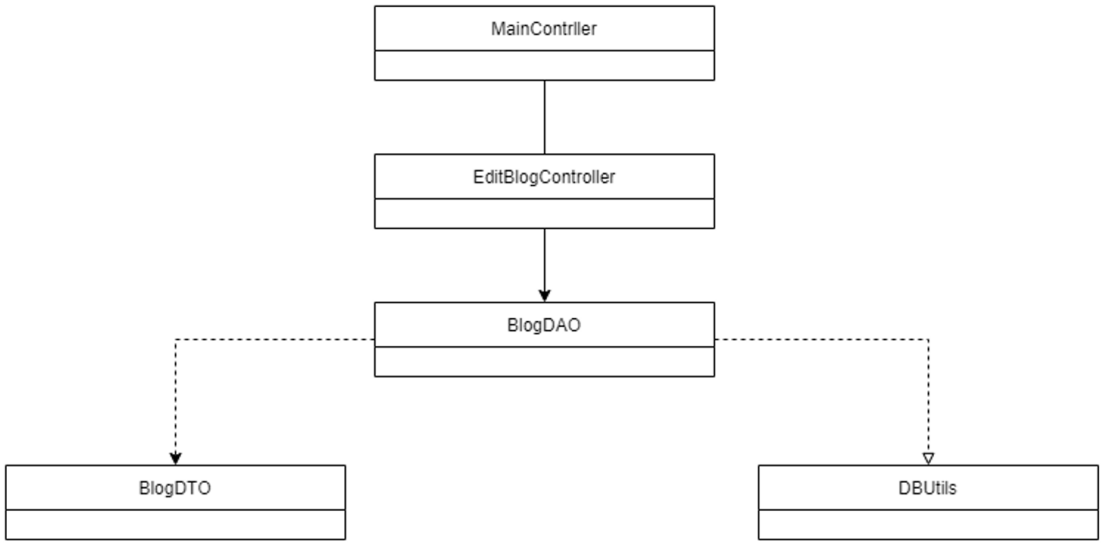
d. Database queries

Blog DAO class

No	Method	Database Queries
01	public BlogDTO BlogDetail(int blogID)	SELECT blogID,Blog.userID,userApproveID,subjectID,title,content,date, Blog.image,video, numberOfVotes,numberOfComments,Blog.status, fullName FROM Blog JOIN [USER] ON Blog.userID = [User].userID WHERE blogID = ?
02	public boolean deleteBlog(int blogID)	UPDATE Blog SET status = 'disable' WHERE blogID = ?
03	public boolean draftBlog(int userID, int subjectID, String title, String content, String date, String image)	INSERT INTO [Blog](userID, userApproveID, subjectID, title, content, date, image, video, numberOfVotes, numberOfComments, status) VALUES(?,null,?,?,?,?,?,null,0,0,'draft')

5. Edit blog

a. Class Diagram



b. Class Specifications

BlogDAO class

No	Method	Description
01	public static int editBlog(int blogID, int subjectID, String title, String content, String date, String image, String video)	This method is used to edit a user's blog posts to the system with "waiting" status. In this method, we need to provide the information of the blog post such as user ID, subject ID, title, content,.. to edit a complete blog post. If editing successfully, returns notice posted successfully

ActivityDAO Class

No	Method	Description
01	public boolean updateActivity(int blogID, int userID, String date)	This method is used to create an activity of an account. In this method, we need to provide a blog ID, a user ID, and the date that the user did this activity and insert this activity into the database. If inserted successfully, return true, else false.

c. Sequence Diagram(s)

d. Database queries

Blog DAO class

No	Method	Database Queries
01	public static int editBlog(int blogID, int subjectID, String title, String content, String date, String image, String video)	UPDATE Blog set subjectID=?, title=? , content=?,date=?,image=?, video=? where blogID=?

ActivityDAO Class

No	Method	Description
01	public boolean updateActivity(int blogID, int userID, String date)	INSERT INTO HistoryActivity(userID, date, activityTypeID, blogID, status) VALUES(?,?,1,?,1)

6. Give Feedback

a. Class Diagram

b. Class Specifications

FeedbackDAO Class

No	Method	Description

03	*public List<FeedbackTypeDTO> getAllFeedbackTypes( )*	*This method is used to get all feedback types in the database for display on the feedback screen. In this method, we create a list to store all the feedback names and add them to the list of the feedback type. If the size of the list is not null, return the list of all feedback types, else null.*
04	*public boolean giveFeedback(int userID, int feedbackTypeID, String description, String date)*	*This method is used to create feedback for a student, mentor, or system. In this method, we need to provide a user ID (who wrote this feedback), a feedback type ID (1 for the system, 2 for the student, and 3 for the mentor), the description, the date that this feedback was sent, and insert into the database. If inserted successfully, returns true, else false.*

c. Sequence Diagram(s)

d. Database queries

FeedbackDAO Class

No	Method	Description
01	*public List<FeedbackTypeDTO> getAllFeedbackTypes( )*	*SELECT feedbackTypeID, feedbackName from FeedbackType*
02	*public boolean giveFeedback(int userID, int feedbackTypeID, String description, String date)*	*INSERT INTO Feedback(userID, feedbackTypeID, description, date) VALUES(?,?,?,?)*

7. Admin page

a. Class Diagram

b. Class Specifications

FeedbackDAO Class

No	Method	Description
01	public List<FeedbackDTO> getAllFeedback()	This method is used to get all feedback in the database for display on the feedback manager. In this method if the size of list is not null, it will return list of user feedback, else null

UserDAOClass

No	Method	Description
06	public List<UserDTO> getAllUser()	This method is used to get all users to display them on the manage account page. In this method, we need to provide a user ID and create an object to store all user information. If finding an account successfully, returns the user information with UserDTO type, else null.

SubjectDAO Class

No	Method	Description
01	public static ArrayList<SubjectDTO > getSubject()	This method is used to get all subjects in the database for display on the manage subjects screen. In this method, we create a list to store all the subject names and add them to the list of the subject. If the size of the list is not null, return the list of all subjects, else null.

MajorDAO Class

No	Method	Description
01	public List<MajorDTO> getAllMajors()	This method is used to get all majors in the database for display on the manage majors screen. In this method, we create a list to store all the major names and add them to the list of the major. If the size of the list is not null, return the list of all majors, else null.

c. Sequence Diagram(s)

d. Database queries

FeedbackDAO Class

No	Method	Description
01	public List<FeedbackDTO> getAllFeedback()	SELECT [feedbackID],[userID],[feedbackTypeID],[description],[date] FROM [ABF].[dbo].[Feedback]

UserDAOClass

No	Method	Description
01	public List<UserDTO> getAllUser()	SELECT userID, password, fullName, roleID, email, image, numberOfBlogs, gender, dateOfBirth, address, contact, aboutme, status FROM [User]

SubjectDAO Class

No	Method	Description
01	public static ArrayList<SubjectDTO > getSubject()	SELECT majorID, majorName, status FROM Major WHERE status = 1

MajorDAO Class

No	Method	Description
01	public List<MajorDTO> getAllMajors()	SELECT subjectID, subjectName,majorID,status FROM Subject WHERE status = 1

8. Manage Account

a. Class Diagram

b. Class Specifications

UserDAOClass

No	Method	Description

01	public List<UserDTO> getAllUser()	This method is used to get all users to display them on the manage account page. In this method, we need to provide a user ID and create an object to store all user information. If finding an account successfully, returns the user information with UserDTO type, else null.
02	public static int updateStatusUser(int userID, String oldStatus)	
03	public static int editUser(int userID, String fullName, String image, String gender, String dateOfBirth, String address, String contact, String aboutme)	This method is used so that users can customize information such as: userID, fullName, image, gender, ... in the profile page screen. In this method, we need to provide the information that we need to edit such as: userID, fullName, image, gender, ... from there editing personal information is complete . If editing a profile successfully, returns the user's new user profile , else error.

c. Sequence Diagram(s)

d. Database queries

UserDAOClass

No	Method	Description
01	public List<UserDTO> getAllUser()	SELECT userID, password, fullName, roleID, email, image, numberOfBlogs, gender, dateOfBirth, address, contact, aboutme, status FROM [User]
02	public static int updateStatusUser(int userID, String oldStatus)	UPDATE [User] SET status = ? Where userID = ?
03	public static int editUser(int userID, String fullName, String image, String gender, String dateOfBirth, String address, String contact, String aboutme)	UPDATE [User] SET fullName=?, image=?, gender=?, dateOfBirth=?, address=?, contact=?, aboutme=? " WHERE userID=?

9. Feedback List

a. Class Diagram

b. Class Specifications

FeedbackDAO Class

No	Method	Description
01	*public List<FeedbackDTO> getAllFeedback()*	*This method is used to get all feedback in the database for display on the feedback manager. In this method if the size of list is not null, it will return list of user feedback, else null*
02	*public List<FeedbackTypeDTO> getAllFeedbackTypes( )*	*This method is used to get all feedback types in the database for display on the feedback screen. In this method, we create a list to store all the feedback names and add them to the list of the feedback type. If the size of the list is not null, return the list of all feedback types, else null.*

c. Sequence Diagram(s)

d. Database queries

FeedbackDAO Class

No	Method	Description
01	public List<FeedbackDTO> getAllFeedback()	SELECT [feedbackID],[userID],[feedbackTypeID],[description],[date] FROM [ABF].[dbo].[Feedback]
02	public List<FeedbackTypeDTO> getAllFeedbackTypes( )	SELECT feedbackTypeID, feedbackName from FeedbackType

10. Feedback Details

a. Class Diagram

b. Class Specifications

FeedbackDAO Class

No	Method	Description
01	public FeedbackDTO getFeedbackByID(int id)	This method is used to get details of feedback by feedback id . In this method, we need to provide the feedback id and the system will return the detail of the feedback

c. Sequence Diagram(s)

d. Database queries

FeedbackDAO Class

No	Method	Description

01	public FeedbackDTO getFeedbackByID(int id)	SELECT [feedbackID],[userID],[feedbackTypeID],[description],[date]\n" + " FROM [ABF].[dbo].[Feedback]

11. Manage Major

a. Class Diagram

b. Class Specifications

MajorDAO Class

No	Method	Description
01	public List<MajorDTO> getAllMajors()	This method is used to get all majors in the database for display on the manage majors screen. In this method, we create a list to store all the major names and add them to the list of the major. If the size of the list is not null, return the list of all majors, else null.
02	public int editMajor(int majorID, String majorName)	This method is used to update details of major in the database. In this method, we need to provide the major id, major name. The result is returns non-zero int if edit major succeeds
03	public boolean deleteMajor(int majorID)	This method is used to delete a major by setting a new status. In this method, we need to provide a major ID to set a new status for this major. If set successfully, returns true, else false.
04	public int createMajor(int majorID, String majorName)	This method is major to create a new major. In this method, we need to provide a major name, of this major and insert this major into the database. If inserted successfully, return true, else false.

c. Sequence Diagram(s)

d. Database queries

MajorDAO Class

No	Method	Description
01	public List<MajorDTO> getAllMajors()	SELECT majorID, majorName, status FROM Major WHERE status = 1
02	public int editMajor(int majorID, String majorName)	UPDATE Major SET status = 0 WHERE majorID = ?
03	public boolean deleteMajor(int majorID)	UPDATE Major SET status = 0 WHERE majorID = ?
04	public int createMajor(int	INSERT INTO Major(majorID, majorName, status) VALUE(?,?,1)

	majorID, String majorName)	

12. Manage Subject

a. Class Diagram

b. Class Specifications

SubjectDAO Class

No	Method	Description
01	public static ArrayList<SubjectDTO > getSubject()	This method is used to get all subjects in the database for display on the manage subjects screen. In this method, we create a list to store all the subject names and add them to the list of the subject. If the size of the list is not null, return the list of all subjects, else null.
02	public int editSubject(int subjectID, int majorID, String subjectName)	This method is used to update details of a subject in the database. In this method, we need to provide the subject id, major id, and subject name. The result is returns non-zero int if edit major succeeds
03	public boolean deleteSubject(int subjectID)	This method is used to delete a subject by setting a new status. In this method, we need to provide a subject ID to set a new status for this subject. If set successfully, returns true, else false.
04	public int createSubject(int subjectID, int majorID, String subjectName)	This method is subject to create a new subject. In this method, we need to provide a subject name, of this subject and insert this subject into the database. If inserted successfully, return true, else false.

c. Sequence Diagram(s)

d. Database queries

SubjectDAO Class

No	Method	Description
01	public static ArrayList<SubjectDTO > getSubject()	SELECT subjectID, subjectName,majorID,status FROM Subject WHERE status = 1
02	public int editSubject(int subjectID, int majorID, String subjectName)	UPDATE Subject SET status = 0 WHERE subjectID = ?
03	public boolean deleteSubject(int subjectID)	SET subjectName = ?, majorID = ? WHERE subjectID = ?

04	public int createSubject(int subjectID, int majorID, String subjectName)	INSERT INTO Subject(subjectID, majorID, subjectName, status) VALUE(?,?,?,1)

15. Approve Blog

a. Class Diagram

b. Class Specifications

BlogDAO Class

No	Method	Description
01	public boolean approveBlog(int blogID)	This method is used by the mentor to approve certain quality blog posts and change the status to "approved" for that blog post. In this method, we need to provide the blog ID so that the system can correctly identify the blog post so that the mentor can approve it. If approved successfully, returns the notice approved successfully and blog post will be displayed in the home page
02	public boolean rejectBlog(int blogID)	This method is used by mentors to reject blog posts that don't match the requirements and change the status to "rejected" for that blog post. In this method, we need to provide the blog ID so that the system can correctly identify the blog post so that the mentor can reject it. If the refusal is successful, return the message of the successful refusal and provide the student with the reason.

c. Sequence Diagram(s)

d. Database queries

BlogDAO Class

No	Method	Database Queries
10	*public boolean approveBlog(int blogID)*	*UPDATE Blog SET status= 'approved' WHERE blogID = ?*
11	*public boolean rejectBlog(int blogID)*	*UPDATE Blog SET status= 'rejected' WHERE blogID = ?*

16. Manage profile

a. Class Diagram

b. Class Specifications

UserDAOClass

No	Method	Description
01	public UserDTO GetUserByID(int id)	This method is used to find user information for displaying them on the profile page. In this method, we need to provide a user ID and create an object to store all user information. If finding an account successfully, returns the user information with UserDTO type, else null.
02	public static int editUser(int userID, String fullName, String image, String gender, String dateOfBirth, String address, String contact, String aboutme)	This method is used so that users can customize information such as: userID, fullName, image, gender, ... in the profile page screen. In this method, we need to provide the information that we need to edit such as: userID, fullName, image, gender, ... from there editing personal information is complete . If editing a profile successfully, returns the user's new user profile , else error.

c. Sequence Diagram(s)

d. Database queries

8. UserDAOClass

No	Method	Description
01	public UserDTO GetUserByID(int id)	SELECT password, fullName, roleID,email, image, numberOfBlogs, gender, dateOfBirth, address, contact, aboutme, status FROM [User] WHERE UserID = ?
02	public static int editUser(int userID, String fullName, String image, String gender, String dateOfBirth, String address, String contact, String aboutme)	UPDATE [User] SET fullName=?, image=?, gender=?, dateOfBirth=?, address=?, contact=?, aboutme=? " WHERE userID=?

17. Manage Activity

a. Class Diagram

b. Class Specifications

ActivityDAO Class

No	Method	Description
01	*public List<ActivityDTO> getAllActivities(int userID)*	*This method is used to get all activities of an account. In this method, we need to provide a user ID to get all the activities of this account, create an object to store all properties of the activity, and add it to the list of the activity. If the size of the list is not null, return the list of all activities, else null.*
02	*public boolean deleteActivity(int historyActivityID)*	*This method is used to delete an activity of an account by setting a new status. In this method, we need to provide an activity ID to set a new status for this activity of this account. If set successfully, returns true, else false.*
03	*public boolean findVoteActivity(int blogID, int userID)*	*This method is used to find the Vote activity for checking the Vote activity of an account. In this method, we need to provide a blog ID and a user ID to find the Vote activity of this account. If finding a Vote activity successfully, return true, else false.*
04	*public boolean updateActivity(int blogID, int userID, String date)*	*This method is used to create an activity of an account. In this method, we need to provide a blog ID, a user ID, and the date that the user did this activity and insert this activity into the database. If inserted successfully, return true, else false.*
05	*public boolean deleteUpdate(int blogID, int userID)*	*This method is used to delete an activity of an account by setting a new status. In this method, we need to provide a blog ID and a user ID to delete this activity of this account. If deleted successfully, returns true, else false.*
06	*public List<ActivityDTO> SearchActivitiesByName(String searchName, int userID)*	*This method is used to search the user's activity history, we need to provide the search information and the userID to search the activity history. If Search successfully, returns list of matching activities*

c. Sequence Diagram(s)

d. Database queries

ActivityDAO Class

No	Method	Description
01	*public List<ActivityDTO> getAllActivities(int userID)*	*SELECT historyActivityID, date, activity FROM HistoryActivity JOIN ActivityType ON HistoryActivity.activityTypeID = ActivityType.activityTypeID WHERE userID = ? AND status = 1*
02	*public boolean deleteActivity(int historyActivityID)*	*UPDATE HistoryActivity SET status = 0 WHERE historyActivityID = ?*
03	*public boolean findVoteActivity(int blogID, int userID)*	*SELECT historyActivityID FROM HistoryActivity WHERE blogID = ? AND userID = ? AND activityTypeID = 1*

04	*public boolean updateActivity(int blogID, int userID, String date)*	*INSERT INTO HistoryActivity(userID, date, activityTypeID, blogID, status) VALUES(?,?,1,?,1)*
05	*public boolean deleteUpdate(int blogID, int userID)*	*DELETE HistoryActivity WHERE userID = ? AND blogID = ? AND activityTypeID = 1*
06	*public List<ActivityDTO> SearchActivitiesByName(String searchName, int userID)*	*SELECT historyActivityID,userID, date, activity FROM HistoryActivity h JOIN ActivityType a ON h.activityTypeID = a.activityTypeID WHERE a.activity like ? AND h.userID = ? AND h.status = 1*

19. User Details

a. Class Diagram

b. Class Specifications

c. Sequence Diagram(s)

d. Database queries

20. Manage Draft Blog

a. Class Diagram

b. Class Specifications

No	Method	Description
01	public boolean draftBlog(int userID, int subjectID, String title, String content, String date, String image)	This method is used to post user's blog posts to the system with "draft" status. In this method, we need to provide the information of the blog post such as user ID, subject ID, title, content,.. to create an incomplete blog post. If posting successfully, returns notice posted successfully

c. Sequence Diagram(s)

d. Database queries

No	Method	Description
01	public boolean draftBlog(int userID, int subjectID, String title, String content, String date, String image)	INSERT INTO [Blog](userID, userApproveID, subjectID, title, content, date, image, video, numberOfVotes, numberOfComments, status) VALUES(?,null,?,?,?,?,?,null,0,0,'draft')

21. Mentor register

a. Class Diagram

b. Class Specifications

c. Sequence Diagram(s)

d. Database queries

III. Database Tables

1. dbo.User

#	Field name	Type	Size	Unique	Not Null	PK/FK	Notes
1	userID	int		X	X	PK	
2	password	nvachar	50		X		
3	fullName	nvachar	250		X		
4	roleID	int			X	FK	
5	email	nvachar	255		X		
6	image	nvachar	MAX				
7	numberOfBlogs	int			X		
8	gender	nvachar	50				
9	dateOfBirth	nvachar	50				
10	address	nvachar	250				
11	contact	nvachar	50				
12	aboutme	nvachar	50				
13	status	nvachar	10		X		

2. dbo.Blog

#	Field name	Type	Size	Unique	Not Null	PK/FK	Notes
1	blogID	int		X	X	PK	
2	userID	int			X	FK	
3	userApproveID	int					
4	subjectID	int			X	FK	
5	title	nvarchar	50		X		
6	content	nvarchar	MAX		X		
7	date	nvarchar	100		X		
8	image	nvarchar	MAX				
9	video	nvarchar	MAX				
10	numberOfVotes	int			X		
11	numberOfComments	int			X		
12	status	nvarchar	10		X		include "approved", "rejected", "waiting", "disable", "draft"

3. dbo.Role

#	Field name	Type	Size	Unique	Not Null	PK/FK	Notes
1	roleID	int		X		PK	
2	roleName	nvarchar	50				

4. dbo.Comment

#	Field name	Type	Size	Unique	Not Null	PK/FK	Notes
1	commentID	int		X	X	PK	
2	blogID	int			X	FK	
3	userID	int			X	FK	
4	content	nvarchar	MAX		X		
5	date	nvarchar	50		X		
6	image	nvarchar	MAX				
7	video	nvarchar	MAX				
8	status	nvarchar	10		X		

5. dbo.Feedback

#	Field name	Type	Size	Unique	Not Null	PK/FK	Notes
1	feedbackID	int		X	X	PK	
2	userID	int			X	FK	
3	feedbackTypeID	int			X	FK	
4	description	nvarchar	MAX		X		
5	date	nvarchar	50		X		

6. dbo.FeedbackType

#	Field name	Type	Size	Unique	Not Null	PK/FK	Notes
1	feedbackTypeID	int		X	X	PK	
2	feedbackName	nvarchar	50		X		

7. dbo.HistoryActivity

#	Field name	Type	Size	Unique	Not Null	PK/FK	Notes
1	historyActivityID	int		X	X	PK	
2	userID	int			X	FK	
3	date	nvarchar	50		X		
4	activityTypeID	int			X	FK	
5	blogID	int			X	FK	
6	status	int			X		

8. dbo.ActivityType

#	Field name	Type	Size	Unique	Not Null	PK/FK	Notes
1	activityTypeID	int		X	X	PK	
2	activity	nvarchar	50		X		

9. dbo.Major

#	Field name	Type	Size	Unique	Not Null	PK/FK	Notes
1	majorID	int		X	X	PK	
2	majorName	nvarchar	50		X		
4	status	nvarchar	10		X		

10. dbo.Registration

#	Field name	Type	Size	Unique	Not Null	PK/FK	Notes
1	registrationID	int		X	X	PK	
2	userID	int			X	FK	
3	certificate	image			X		

11. dbo.Subject

#	Field name	Type	Size	Unique	Not Null	PK/FK	Notes
1	subjectID	int		X	X	PK	
2	subjectName	nvarchar	250		X		
3	majorID	int			X	FK	
4	status	nvarchar	10		X		