

Team k final project

CS 2005



Ian bardos

omar mohamed

trei solis

yuchen zhang

1. Project Management

1.1 Management structure 2

1.2 work assignments and individual credit 2

1.3 Gantt chart 3

1.4 Rankings 4

1. Requirements document
   1. Functional features and team chosen feature 4
   2. Use case and User story 5
2. Design document
   1. High level design structure 6
   2. Module and class details 6
3. Prototype implementation 10
4. Test suites 10
5. State of prototype
   1. User prototype documentation 11
   2. Project review from a developer perspective 12

**2.1 Management structure**

Team members

**Omar Mohamed**: Project Manager, document writer, back end developer.

**Ian bardos**: Database and backend developer

**Trei Solis**: Full stack Developer (Front and back end)

**Yuchen Zhang**: Back end Development and high-level design architect.

**Marapana, Koswatta Arachchige**: Dropped the course, inactive member.

Communication:

My team used slack as a primary form of communication for team meeting confirmation, small file sharing and updates. For our project source code, diagrams, and documents we used a git-hub repository.

Initially we met once a week in the library and we increased the frequency of the meetings as we approached the deadline time.

Minutes were also taken during decision-based meetings. Rest of the meetings were work oriented and had no major discussions worth noting. We met on the 27th,29th and April 2nd, 4th and 5th.

|  |  |  |  |
| --- | --- | --- | --- |
| **March 1 Meeting 1**  Attendees: Omar, Ian, Yuchen, Chulan  Absent: Trei  **Discussion**: Personal introductions, initial contact meeting and discussion. Task/role allocation and future plan completed. Social media created ( slack and github)  **Action items**: Work on the gant chart(Ian), Create the requirements document(omar),Start working on the high level design ( Yuchen). | **March 8, Meeting 2**  Attendees: Omar, Ian,Yuchen, Trei.  Chulan: dropped out  Action Item:Wrap up the full writing for design requirement (Omar and Trei),  Ask the professor about the data base.  Create a skeleton front-end website pages (Omar and Trei).  Work on the python back end (yuchen,Omar, Ian). | **March 15 Meeting 3**  Attendees: Omar, Ian, Yuchen, Trei  **Action items**: confirm with the professor about dropping a requirement due to team member resignation.  Work on the UML class diagram (yuchen)  Start assembling the project components (Omar)  Finalize the database ( ian)  Back end development(Trei) | **March 22, Meeting 4**  Attendees: Omar,Ian,Yuchen,Trei  Action Items: Continue working on the UML class diagrams (yuchen)  Continue assembling the project components and start the test suites (Omar)  Wrap up the database (ian)  Polish the front end and gant chart (Trei) |

**2.2 Work assignment, individual credit and ranking**

**Omar Mohamed:**

Total work load: Project management and leadership, Assisted Ian with data base design, assisted Trei with final touches for the front-end components. Developed the login logout module. Assembled and edited the final report for submission. Assisted in unit testing

Individual work load: Project management, Gantt chart, Final report writing and login module.

**Ian Bardos:**

Total work load: Data base implementation, subscribed user notification module development, assist with front end development, general back end development and flask implementation. Assisted in Unit testing

Individual work load: Data base implementation and user subscription and notification modules.

**Trei Solis:**

Total work load: Front end development, diagram development assistance, user ability to upvote a post, and general back end assistance. Assisted in unit testing

Individual work load: Front end components and user ability to up vote posts (team picked requirement)

**Yuchen Zhang:**

Total work load: Create the required tables/diagrams. Assist in the general back end development and flask implementation. Create the user ability to post/edit. Assist in general report writing. Assisted in unit testing

Individual workload: Created all the required UML diagrams. Developed the user post module(user’s ability to post) .

**2.3 Gant Chart**: Please refer to the attached file for the Gantt chart

**2Requirements Document**

**2.1 Functional features:** Refer to the requirements document for full details**.**

**User password authentication**: This functional requirement allows the user to log in successfully into the web application.

Description: Our team decided to take a simple approach, we created a database where we can store log in credentials in. The user would then enter the user name and password and it would compare the entry to the data stored in the database for each the username and the password.

Due to time constraints we were unable to have a sign-up feature where a user would sign up, we currently must add the username and password to the database manually.

Validation: Use case: Final Project, Team k

Verify by: successful log-in and prompted error message upon wrong credential entry.

Author: Team K; date: Sun March 4th 2018. 18:30:13 NST 2018

**User ability to post**: This functional requirement allows the user to post new entries under the desired lecture topic. The user can post useful links and comments.

Description: It was implemented such that once the user picks the desired topic they want to post under, they can click the add post button and they are then prompted to word editing tools in the post box. The user then clicks post once they are ready to submit it.

This requirement works as intended but if time was given, we would have added more security features to what the user can and cannot post. This module works as intended.

Validation: Use case: Final Project, Team K

Verify by: successful creation of a new post under a topic.

Author: Team K; date: Sun March 4th 19:05:36 NST 2018;

**User ability to edit personal posts:** This functional requirement allows the user to only edit their own posts after it has been posted.

Description: After a post has been added successfully, the user may want to omit or add more to the original post. This function was implemented such that the new string replaces the existing one in the database. These changes cannot be undone once posted. This module works as intended.

Validation: Use case: Final Project, Team K

Verify by: successfully edited a pre-existing post written by the user. The user was unable to edit someone else’s post.

Author: Team K ; date : sun March 4th 19:15:43 NST 2018.

**User subscription and notification:** This functional requirement allows the user to subscribe to a post and be alerted when a new entry is posted under it.

Description: while the user is subscribed, if any new posts are posted the user is prompted with a pop-up message to go look at the new post. This was implemented via the notification module. Due to time constrains the team was unable to add direct links to the newly posted entry under a lecture.

Validation: Use case: Final Project, Team K

Verify by: user getting a notification when a new post has been added.

Author: Team K; date: sun March 4th 19:20: 09 NST 2018.

**Team chosen feature**

**User ability to up vote a post under a topic:** This functional requirement allows the user to upvote posts to highlight its importance versus other posts under the lecture.

Description: Any user may upvote a link or post under the lecture to highlight its importance, this will maximize study efficiency by making sure students don’t spend too much time reading filler articles. The team wanted to let admins have a heaver weighted vote but due to time constraints that was put aside.

Validation: Use case: Final Project, Team k

Verify by: User upvoting a link posted under a lecture.

Author: Team K; date: sun March 4th 19:30:53 NST 2018.

**2.1 UserStory and Usecase:** Refer to the attached file TeamKUseCaseandStory

**3 Design document**

**3.1 High level design structure:** Please refer to the attached UML diagrams for the architecture used in this project.

**3.2 Module and class details**

**App.py**: Is the “main” function that connect all modules. It’s a collection of interfaces for all the modules. App.py also renders the page templates.

**Comment.py**: Is a module that describes the Comment class and some of its features relate to this class. The Comment class can be used to push comment object into the database. It has accessor methods that return the attributes, it also contains:

Upvote: It increments the vote count.

Downvote: it Decrements the vote count.

**LectureTopic.py:** Contains the class LectureTopic( LTid,title,creator,type,) and it describes the LectureTopic objects. It has accessor mtehods that return the attributes.!

**Notification.py:** Contains the class Notification(Nid, subscription) and it describes the notification objects. It has accessor methods that return the attributes.

**User.py:** Contains the class User(username,password,Uid) and it describes the user objects. It has accessor methods that return the attributes.

**Subscription.py**: Contains the class Subscription(Sid,Ltid, user) and it describes the subscription objects. It has accessor methods that return the attributes.

**Login\_Logout\_Handle.py**: This module contains the functions of log\_in and log\_out.

**log\_in**: This function is related to login interface that is in app.py.

1. It takes the information submitted from the form on the login page.
2. It then uses retrieve from the persistence module to retrieve the matching user name from the database to the input submitted on the form. The same is done for the entered password.
3. If no user is found, it flashes a “login failed” message.
4. If the user is found but the password pulled from database associated with the user name does not match to the submitted password, then flash “login failed” message, and user is promted back to login.
5. If the user is found and the password matches, then the user’s saved data is retrieved by: setting the session [“logged\_in”] to true, session [“user\_id”] to the id of the user, [“user\_name”] to the username of the user, and session[“user\_subcriptions”] to a list of the LTids of the users subscriptions which is retrieved from the database(could be empty list if the user is not subscribed any lectures or topics).
6. A ”You are now logged in” message flashes and the user is redirected to the home page.

**Log\_out:** This function is related to logout interface(which uses the is\_logged\_in function in app.py to checks if the user is logged in for the session).

It clears the session and flashes “You are now logged out” and redirect to login page.

**Notification\_handle.py**: Contains the function myNotification

**myNotification**: This function is related to notifications interface in app.py.

1. It looks at the session[“user\_subscriptions”] and retrieves the subscription object from the database which is associated with the Lecture/Topic ids in session[“user\_subscriptions”].
2. If the sessions user is associated with those subscriptions, it appends that subscription to a list called all\_subs.
3. Then it looks at all the subscriptions in all\_subs and retrieves any notifications that are related to that subscription.
4. If there are notifications append them to a list called myNot and delete the notifications from the database.
5. Then retrieve the lectures associated with the notifications and append them to a list called postToNot.
6. Return that list to the interface which then renders the Notification page with a link to all the posts that the user has notifications for.
7. If the list is empty, then page displays a paragraph saying no new notifications.

**presisitence.py**: Contains functions related to the retrieval, persistence, deletion, and updating of object stored in database. These are explained in more detail below.

**\_reset**: It connects to the database and then reads from the file Schema.sql to create the table in the database, then it commits to the database, and closes the connection.

**Persist**: This is an interface for persist functions.

1. It takes the argument o, which is an object to be persisted.
2. looks up the correct orm (Object rotational map) for the class of object o.
3. Connects to the database, and does the sql\_persist(conn,o) for that orm which persists the object to the correct table.
4. Then commits to the database and closes the connection and returns o.

**Update**: This is an interface for persist functions.

1. It takes the argument o, which is an object to be updated.
2. Looks up the correct (orm) (Object rotational map) for the class of object o.
3. Connects to the database, and do the sql\_update (conn,o) for that orm which updates in the correct table.
4. Then commits to the database and closes the connect and returns o.

**Delete**: This is an interface for the delete functions.

It takes the (arguments class, searchattr, and searchvalue); where class is the class of the object you are trying to delete, Search attribute specifies an attribute in the table of the object to delete, and searchvalue compares the value with the stored values in the database.

Then it looks up the ORM of the class then it connects to the database and it uses sql\_delete (conn, searchattr,searchvalue) which finds the corresponding row in the table to delete and removes the desired object and stores it in a temp variable o, then commits to the database then closes the connection and returns o.

**Retrieve:** This is an interface for the retrieve functions.

It takes the (arguments class, searchattr, and searchvalue); where class is the class of the object you are trying to retrieve, Search attribute specifies an attribute in the table of the object to retrieve, and searchvalue compares the value with the stored values in the database.

1. Then it looks up the ORM of the class then it connects to the database and it uses sql\_retrieve( conn, searchattr,searchvalue).
2. Finds all data in the database that have the matching search values.
3. Stores that in a list called matches.
4. Commits to the database and closes the connection and the returns matches.

**PostingComment.py**

Contains the class postingCForm(Form) this is just an extension to the class Form which is from an imported open source pip package called WTForms. This class contains the attribute body which is the text area field.

**postComment**

This function takes in the argument’s type, form and LTid. Where type is either Lecture or Topic. Form is postingCForm and LTid is the topic/lecture ID you are posting the comment on.

1. It assigns the text to the variable body and then it creates a new comment object using the session user ID, the body (holding 0 votes) and the LTid for the corresponding lecture/topic. It then persists the new post and then it retrieves the subscriptions for the lecture topic ID and stores in a list called subscriptions.
2. For all subscriptions within the list subscriptions, it creates a notification object for the subscription and persists that notification.
3. Then the “ comment created” message is flashed and redirect to the specific topic/discussion page.

**postingLT.py:** contains the class postingForm, postLT and editLT

**postingForm**(Form)

This is just an extension to the class Form which is from an imported open source pip package called WTForms. This class contain the attribute body which is the text area field and the attribute title which is a string.

**postLT**(type, form): This function takes in the argument’s type and form; Where type is either Lecture or Topic And form is a form object imported from WTForms.

1. It creates title which pulls from the form.Title attribute.
2. Then it creates body which pulls from the form.Body attribute.
3. A new lectureTopic object is created using the title that was made above, the session[user\_id], the type (topic or lecture) and the body attribute.
4. The new lecture topic object is then persisted.
5. A “ topic/lecture created” message is flashed and the page is redirected to the specific topic/Lecture page.

**editLT**(form,toEdit)

Form is a form object imported from WTForms.

toEdit is the lecture topic object that requires editing.

1. It creates title which pulls from the form.Title attribute.
2. A new body is created by pulling from the form.Body attribute.
3. A new lectureTopic object is created using the toEdit.getLTid() and the title that was created above, and it gets the user who created it from toEdit.getCreator() and the type from toEdit.getType and the text from body.
4. Then it updates the object, if toEdit.getType = Topic/lecture.
5. It flashes topic/lecture edited and the page is redirected to the specific topic/Lecture page.

**Subscription\_Handle.py:** Contains the functions sub and unsub

**sub**(type, id)

Type is either lecture or topic and id is the LTid you are subscribing to.

1. A new subscription object is then created, using id and session[user\_id].
2. It then persists, then appends the Id to session[user\_subscriptions].
3. A “ you are now subscribed” message is flashed and the page is redirected to the specific topic/Lecture discussion page.

**unsub**(type,id)

Type is either lecture or topic and id is the LTid you are subscribing to.

1. It creates a new subscription object, using id and session[user\_id].
2. First it retrieves all the subscriptions for the specific lecture, and appends that to a list called sub.
3. Then it iterates through the list and checks for the user by comparing the session[user\_id] to the stored id in the list.
4. Then the stored subscription id that is to be removed is then stored in the variable toUnsub. Then the delete function is used with the Subscription ID and and it removes that ID from session[user\_subscriptions].

**vote\_handle.py:** Contains the functions uVote and dVote.

**uVote**(type,id,Cid)

Type is the lecture/topic, ID is LTid and Cid is the comment ID.

1. Cid retrieves the comment to be upvoted from the database, and uses the function upvote from Comment.py to increase the vote count allocated to the comment.
2. The object is then updated in the data base.
3. The user is redirected to the specific topic/Lecture discussion page.

**dVote**(type,id,Cid)

Type is the lecture/topic, ID is LTid and Cid is the comment ID.

1. Cid retrieves the comment to be downvoted from the database, and uses the function downvote from Comment.py to decrease the vote count allocated to the comment.
2. The object is then updated in the data base.
3. Then the page is redirected to the specific topic/Lecture discussion page

**4 Prototype implementation:** Please refer to the attached source code.

**5 Unit testing:** Please refer to the attached test.py files. And the Readme for an explanation on how to run the tests.

**6 State of prototype:**

**6.1 User prototype documentation:**

The user accesses the website’s URL and is then prompted by a log in page that has a form. This login page is a simple HTML document that is connected to the database, it pulls the data from the user table; this table contains the username, user-ID and password.

After successful verification the user is then taken to the main HTML homepage which contains links to the lectures and topic page. The user then has multiple navigation options, which contain: lecture page, Topics page, notifications and logout. The lectures and topics pull from the database and list the lectures or topics depending on the page. The notification shows the user any newly posted entries under a specific subscribed topic or lecture because when the comment is added The module postingComment.py then checks for subscriptions, it pushes to the notification table on the database and shows the notification to the user. While no new notifications exists, it displays “no notifications” since the notifications are deleted after the user views them.

When the user accesses the Lecture or Topic page, they have the option to add a new entry to that page, which will take them to the add Lecture/topic page which allows them to see a form that creates and pushes an object to the database. This then appends it to the html document as a new entry.

When the user goes to a specific Lecture/Topic it pulls that post from the database and it displays the lecture material along with information about the posting author. If and only if the user for the session was the poster, they have the ability to edit or delete the post. This functions by using the PostingLT.py module which uses the edit function on the data base which then in turn goes to the Persistence module which matches the lecture and topic ID and updated the information stored there.

Under any lecture/topic the user may subscribe to this specific post, which then the Subscription\_Handle.py which then creates a new subscription object and persists it to the database, along with the post ID as part of the object. If a user unsubscribes it then deletes that entry from the database.

The user may post their own comment which works in the same manner as posting a new lecture/Topic mentioned above. If there are no comments it displays a message saying: “no comments”. If there are comments for that topic, it pulls them from the database and puts them in a list on the HTML page. Where the user can upvote or downvote comments accordingly. The vote mechanism works such that the vote\_handle.py pulls the comment object from the database and uses the method inside Comment.py which performs the function on the object and pushes it back on the database and updates it.

Once the user is done, they can click the logout button which terminates the session under login\_logout\_Handle.py and redirects the user to the login page.

**6.2 Project Review from developer perspective**:

Current state

* Users can login with a manually generated username and password by the team.
* Users have the ability to view posted lecture notes.
* Users are able to post a comment under the discussions.
* Users are allowed to edit their own comments after posting but not anyone else’s.
* Users can upvote the posts in-order to maximize the importance of the comments.

Recognized implementation concerns

* There exists no sign-up option and all the log in credentials have to be manually generated into the database.
* The notification pop-ups do not link directly to the post. The notification was simplified to just display where the update occurred.
* The private/public discussion group requirement was scraped due to a team member dropping the course. Therefore, the 5th requirement was dropped.
* Sub-comment feature is cancelled because its function can be achieved with the regular comment feature.
* Cross-platform support feature is cancelled due to technical and time constraints.

Next steps:

* Implement a sign-up function.
* Upgrade the notification module to link the user directly to the updated post.
* Add security features to what can and cannot be posted as a comment under the lecture discussions.
* Implement group discussions with public and private features.
* Add a profile page where the user can add a photo, edit their bio, manage their profile and its personal information such as the password and email.
* Restrict the user from voting more than once.

Summary:

The program is fully functional with the requirements working as intended, with minor implementation sacrifices due to time constraints.