OurTieba Proposal Documentation

1. A brief description of our project

Contributors:

Boyan Li
Juncheng Dong
Ruhao Xin
Zihang Xia

Project Background:

Nowadays there are a lot of online forum systems that provide a platform for users to discuss something freely with each other. In China we have Baidu Tieba, and outside China we have Reddit. This kind of forum system actually plays one of the most important ways for us to get information and provoke thoughts. However, this freedom provided by this kind of platform may not encourage more people to engage in communication. We think that one of the reasons is that some people passively accept others' perspectives and fail to raise their own opinions, in most cases due to the lack of information. As a result, our system is designed to try to solve this issue of information shortage for some people during the discussion by not only providing them a relatively free platform for discussion, like most of the forum systems, but also providing them relevant information as much as possible. To achieve this, our system is an integration of a news platform which can push information to you, and a forum platform. We think if we design a forum platform in this way, it may encourage more people to engage in the discussion and assure the relative high-quality of discussion.

Stakeholders:

From now on, we have implemented two stakeholders for this system. One is users, including both not-logged-in guests and logged-in users, and the other is the admin.

2. The process followed by our team

Methodology we use:

Agile Methodology, eXtreme Programming

The process:

Actually, the process of our project can be divided into two parts. The first part is a design part, which means building a basic model in theory. The second part is an implementation part, which requires us to do particular code programming to accomplish our design.

For the first design part, we do our design in three ways:

1. At the beginning of the project, we started to design what our system looks like by drawing many crafts of html pages on paper. Also we started to design the use cases. In this way, we can determine how many html pages do we need to build this system and how the html pages interact with each other.
2. Then we started to design the system. Firstly we decided to choose Python as our basic programming language and use python flask to accomplish interaction between server and clients. Also we chose MTV as our system's model. Then we started to design the database model. We choose SQLAlchemy as our basic database model. After this, we started to design the class diagram and the specific database table.
3. Next we started to design the 'view' parts. Firstly we designed the GET URL function on the server side. These functions are used to return html pages by giving the URL. Then we designed the POST URL function on the server side, which, in other words, is the API part. These functions are used to return json messages and they can only be run by html inner interaction, such as clicking a button.

After we passed the first design part, we had a clear model about our final system. Then we were able to enter the second part: Implementation part.

The process of our implementation part can be described as a loop: Firstly we tried to implement the functions based on our previous design. Then someone came up with a new idea about the system so that we need to review and revise our previous design. After that we tried to implement the functions based on a new design. As a result, this loop continuously keeps going until we finish our final system.
Testing is also a very important process we need to do in the implementation. At the beginning of our implementation, we did white-box testing by ourselves: Based on the code we have written, we give them some input and check whether the output is the same as our expected value calculated by ourselves. After we implemented most of our system, we started to implement testing code to help us automatically test the system.

3. Requirements & Specifications of our project

Use cases:

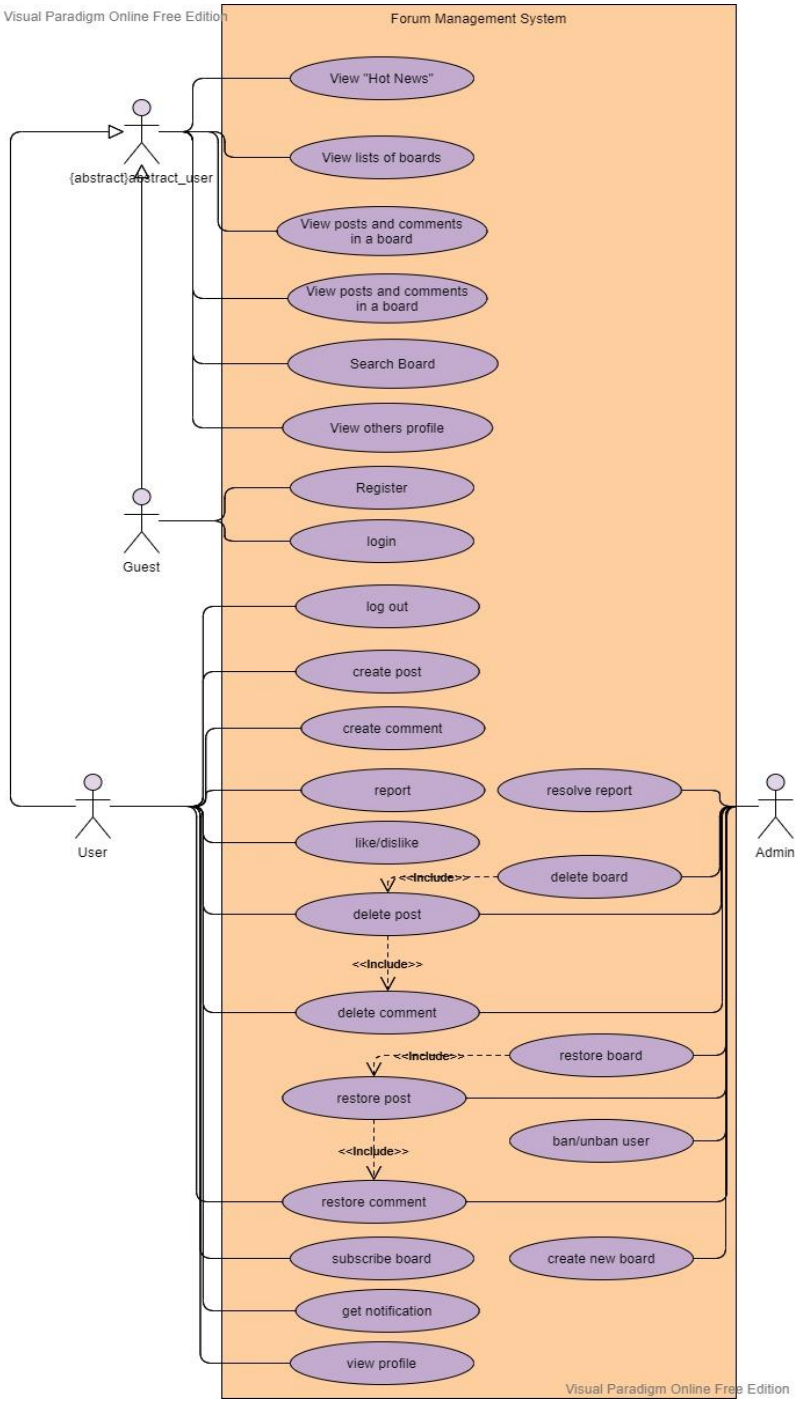

Figure 3.1: Our system's use cases

User stories:

1. As a general user (abstract user), I can:
    - View the hot news the system pushed to me
    - View the board the system pushed to me so that I can view the posts in this board
    - View the post in a board together with the comments
    - Search a board based on some keywords so that I can view the board
    - View other users' profile page
2. As a not-logged-in guest, I can:
    - Register my account by giving my unique username, password and nickname
    - Login based on my username and password so that I can become a logged-in user
3. As a logged-in user, I can:
    - Add post in a board
    - Add comment under a post
    - Like/dislike posts/comments
    - Report posts/comments
    - Delete the posts/comments that created by me
    - Restore the posts/comments that previous deleted by me
    - Subscribe boards so that later I can directly find them
    - Get notifications when others like/dislike my posts/comments or the admins delete/restore my posts/comments or the admins ban/unban me.
    - Log out from the system so that I will become a guest
    - View my profile so that I can:
        - Change my nickname
        - Add more information about myself such as date of birth, gender, email address, phone number and address
        - View my previously created posts/comments
        - View the boards that I have subscripted
        - View my browse history of posts
        - Change my avatar
4. As an admin, I can:
    - Delete any board/post/comment based on their Bid/Pid/Cid
    - Ban/Unban any user based on their Uid
    - Resolve a report so that after the resolution I will not see this report again
    - Create new board

4. Architecture & Design

Class diagram:

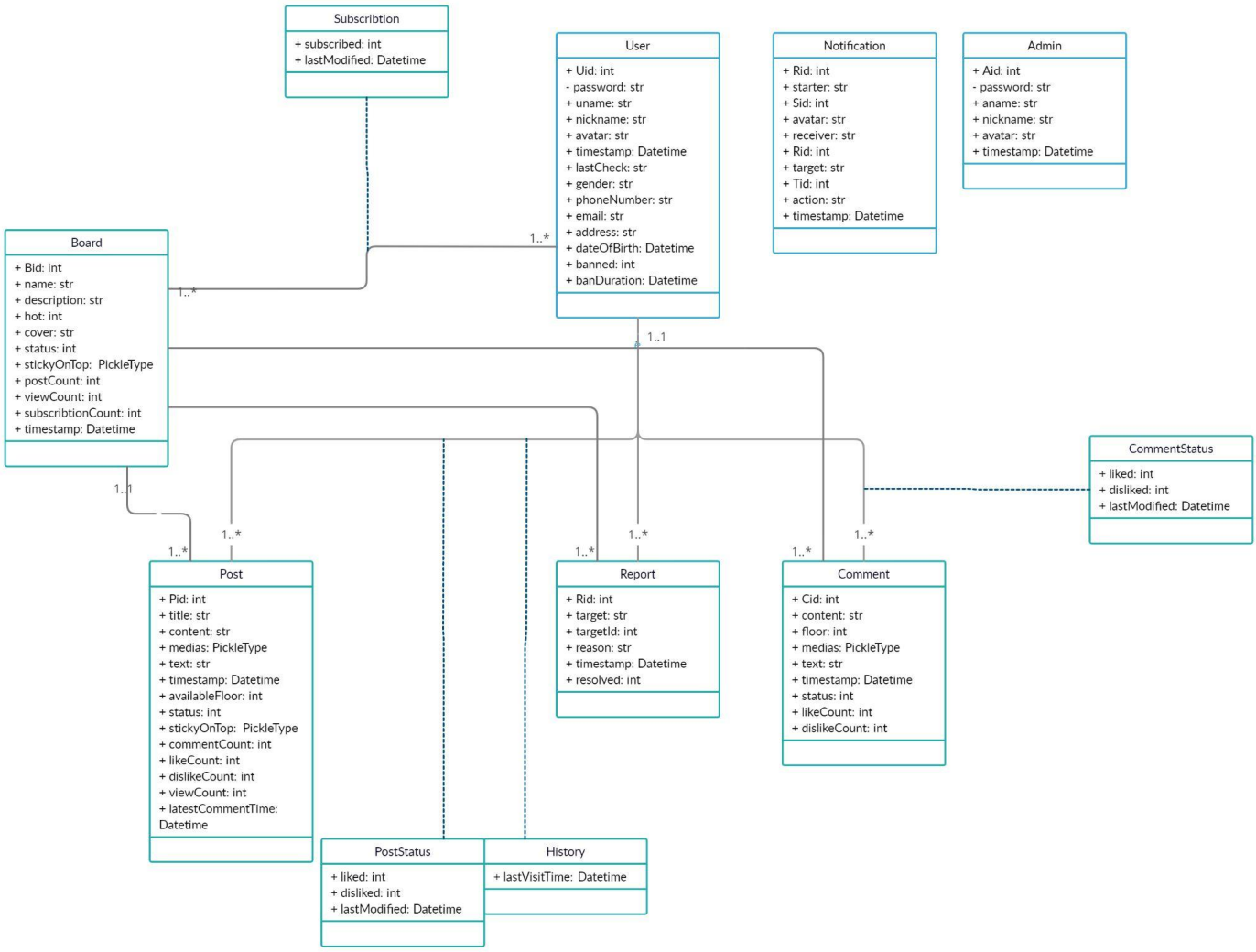

Figure 4.1: Our system's class diagram

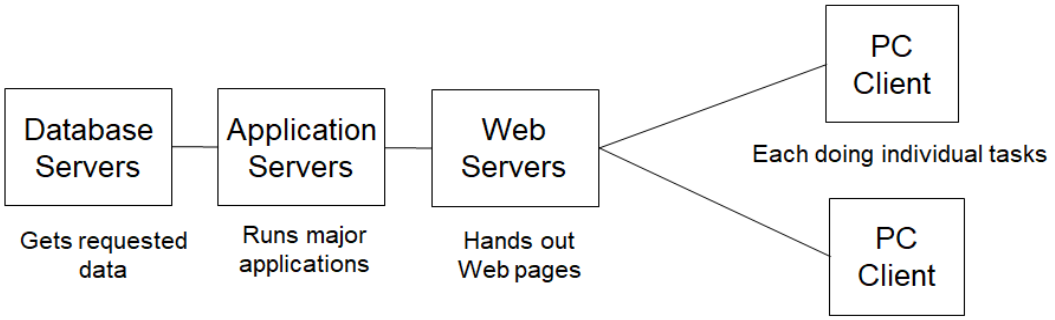

Figure 4.2: client/server architecture (from slides "Software Architecture" in class)

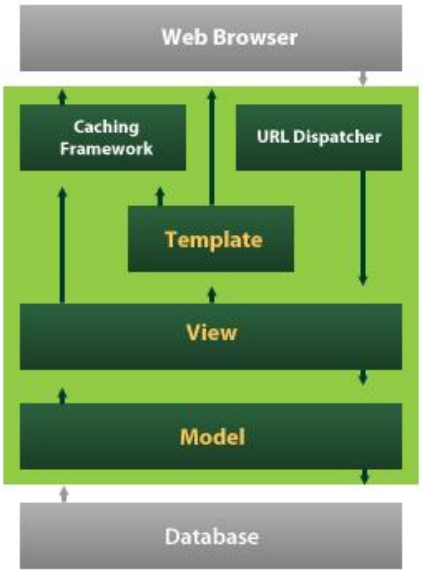

Figure 4.3: MTV pattern

Architecture:

Our system basically follows a typical client/server architecture, and is mainly implemented by web framework Flask. Flask is based on the Model-Template-View pattern. Our models follow this MTV pattern to interface with the database. The basic work-flow is as follows: Once a client sends a request, the view layer will process this request and fetch data from the model layer who interacts with the database. Then, a make-response request and optional data will be sent to the template layer. This layer will use Flask engine to render a HTML with the data (not JavasScript) and send a response with this HTML or other type of data if specified as response body back to the client. The client's browser will render the document and request other resources if needed.

The major components of our system are listed in the class diagram in Figure 4.1. The active entities are admin and (abstract) user. They will perform certain actions on the passive entities like post, comment, board etc. And as a result, some relations will be created like postStatus, commentStatus, subscription, report, history etc. Notifications will be sent upon triggering of the actions. Also, there will be interaction between admin and user, that is ban/unban. The consequence will be reflected in the user's attribute field. Many design details focus on improving user experience, such as allowing users to delete/restore their posts in the personal profile page, displaying more read-friendly timestamps.

The choice of the Flask framework certainly has a big impact on our design. We have to follow the MTV pattern which does not clearly separate frontend and backend. Also, the model layer requires ORM performed by SQLAlchemy, which certainly has a negative

impact on performance of database actions, although it is more “pythonic” and mitigates SQL injections.

5. Reflections and Lessons Learned

Boyan Li:

At the beginning of this project, I was assigned the role of “front end”, however I found myself unable to make much contribution in terms of code, partly because of procrastination, partly because of me kind of waiting for a “what do I do”. In the end, I wrote the test case that uses Selenium, as well as provided many minor fixes and bug reports. I’m grateful that my fellow teammates can swiftly respond to, and implement fixes for, the various bugs I threw at them.
Some of the things that I learned or reinforced during the project include (in no particular order):

- Browsers can be messy.
- Sometimes seemingly irrelevant things can affect what you are doing.
- Blackbox testing might be intuitive at times, but can also miss some critical (but obscure) bugs. (When writing my test case, I needed some way to reference an element on the page, and when I looked into that element, I found that if a user has a particular nickname, then it can prevent the user from posting things due to an HTML element ID conflict.)
- Whitebox testing might allow you to get deeper, but can also be overwhelming. (Also when I was writing my test case, I found a bug with the like counter. I tried to read the relevant source code to see what went wrong, but the logic is rather confusing. If I had to whitebox-test this piece of code, I would certainly have thought the logic being “plausible” and completely miss the bug.)
- Don’t wait until someone assigns you a task before you start doing things. Especially when contribution percentage matters. I’d imagine this would be more important when I leave college and get a job at a company?
- Keep engaged. Being disconnected for a long period of time means you will have a lot to catch up, and you can easily fail to understand things.
- When it’s your chance to speak up (and you have something to say), speak up. That’s exactly how I failed to give my own project idea a fair chance during the proposal phase.

Looking back at the project as a whole, the final product we arrived at differed quite a bit from the initial idea. Compared to the initial idea, the final product both lacked some features (most importantly, being able to incorporate relevant news articles into every page - we ended up having 3 news articles on the home page and nowhere else, and the rest of the website is basically a standard forum/tieba implementation), and had extra features. By implementing an almost-standard forum, I’d say we technically made a piece of software, but on the other hand we missed part of the spirit of the project - to make a product that makes a difference in some way, say solves a particular problem.

Juncheng Dong:

I took on most of the programming work in this project, including front-end and back-end development. The most difficult part of the project was to learn many new things. For the front-end part, to incorporate a rich text editor, I had to study and modify the source code of it (including the .min.js file), so that the functionalities work. To display photos more clearly and enable video play, I studied fullpage.js and dplayer. To overwrite some Bootstrap style, I had to study this framework as well. Also, almost all the requests in javascript have to be written in ajax style, and this took me a lot of time to study jQuery implementation of it. For the back-end part, I had to learn many advanced usages of Flask to filter requests, register routes, config apps etc. SQLAlchemy was also hard at first because without directly writing SQL queries, you have to get familiar with the methods of the database session it provides and encapsulate it in your own database class. Database was no easy task because you always have to think of what columns might be useful for future features. In total, I do not think our project is a good software product. It's more like a pure web application with client-server architecture. Many designs are done ad-hoc. There's still a lack of testing and documentation in the end. However, the process really made me learn a lot about time management and the importance of communication between developers. Even with VCS, communication is essential not to mess up your code unconsciously. This project was also my first experience of developing something big, and I really got to know the time and effort needed to develop a successful software.

Ruhao Xin:

My work in this project is to design the structure of this system at the beginning of the project. In addition, in the process of the development, I also do many things in the back-end, like modeling the database, building the algorithm of the calculation of the hot of a board, implementing many functions based on user stories and so on. In the process of the development of our system, I started to realize that there is much difference between the theory and the reality, which means that for some functions, I think it can be very simple in theory but in practice it occurs a lot of unexpected problems. For example, in my experience of building the function of deleting, I think that in theory it is a very simple function. But in reality, I meet two serious problems. 1. The cascade function cannot work during the deletion so I can only use a very foolish algorithm that when deleting the parent object, searches for all children objects and deletes them one by one. 2. My initial thought about the deletion is to delete the object from the database. But when I tried to implement this, there is a serious problem: Because we allowed users to upload images and videos in their posts and comments, the deletion process should also delete the images and videos. However, the deletion of the videos is forbidden until the video finished loading. I tried a lot of methods to solve this problem but none of them worked. Finally, I have to implement a very foolish algorithm: make the deletion process keep trying to delete until it succeeds. Fortunately, after the interaction with my team members, we

realized that we can do a ‘fake deletion’, which means that the deletion process, instead of deleting the object from the database, can update the object status to ‘delete’. In this way, the deletion process can be very simple. Moreover, based on this algorithm, we can start to implement a new function of restoring, though we have to alter other functions to still meet the initial user stories. As a result, after doing this project, I learned that when doing software engineering, many things are much harder in practice than you think. In this case, pair programming is very useful to solve unexpected problems.

Zihang Xia:

In this project, my primary responsibility is front end development. At first we decided to use vanilla HTML, CSS and as little JS as possible, but we encountered too many functional limitations as well as aesthetic issues. The main functional limitations is async requests to the backend which is not available without ajax request. To solve this issue, I incorporated jQuery library for posting request. To improve the aesthetics of the website, I used Bootstrap for styling. Although there are no artists in our group, the frontend is at least clear and to-the-point if not extremely aesthetic pleasing. To solve the time zone issues, I used the flask-moment module which is an integration of moment.js with flask.
Another part of my work is deployment and testing. For testing, I set up the pytest and selenium for UI testing though most test cases are written by other members. And for deployment, I packaged the OurTieba and uploaded the project to PyPI and used a production server (Gunicorn) for deploying OurTieba on Tencent Cloud.
What I have realized from this project is that setting up a new component like testing or bootstrap is time-consuming but developing using a framework already set up is much easier. There are many issues that are inevitable so a good programmer should have multiple solutions to the same problem in case at certain times one does not work.