CSC 431

IT JOB FORUM

System Architecture Specification (SAS)

Team 12

yuxin song	Scrum Master
zilin xu	Web Developer
siyuan chen	Web Developer

Version History

Version	Date	Author(s)	Change Comments
1.0.0	4.1	Yuxin Zilin	First Draft
2.0.0	4.28	Yuxin	Second Draft
2.0.1	5.1	Yuxin Siyuan	Update system diagrams
3.0.0	5.5	Yuxin	Final Draft; Update diagrams

Table of Contents

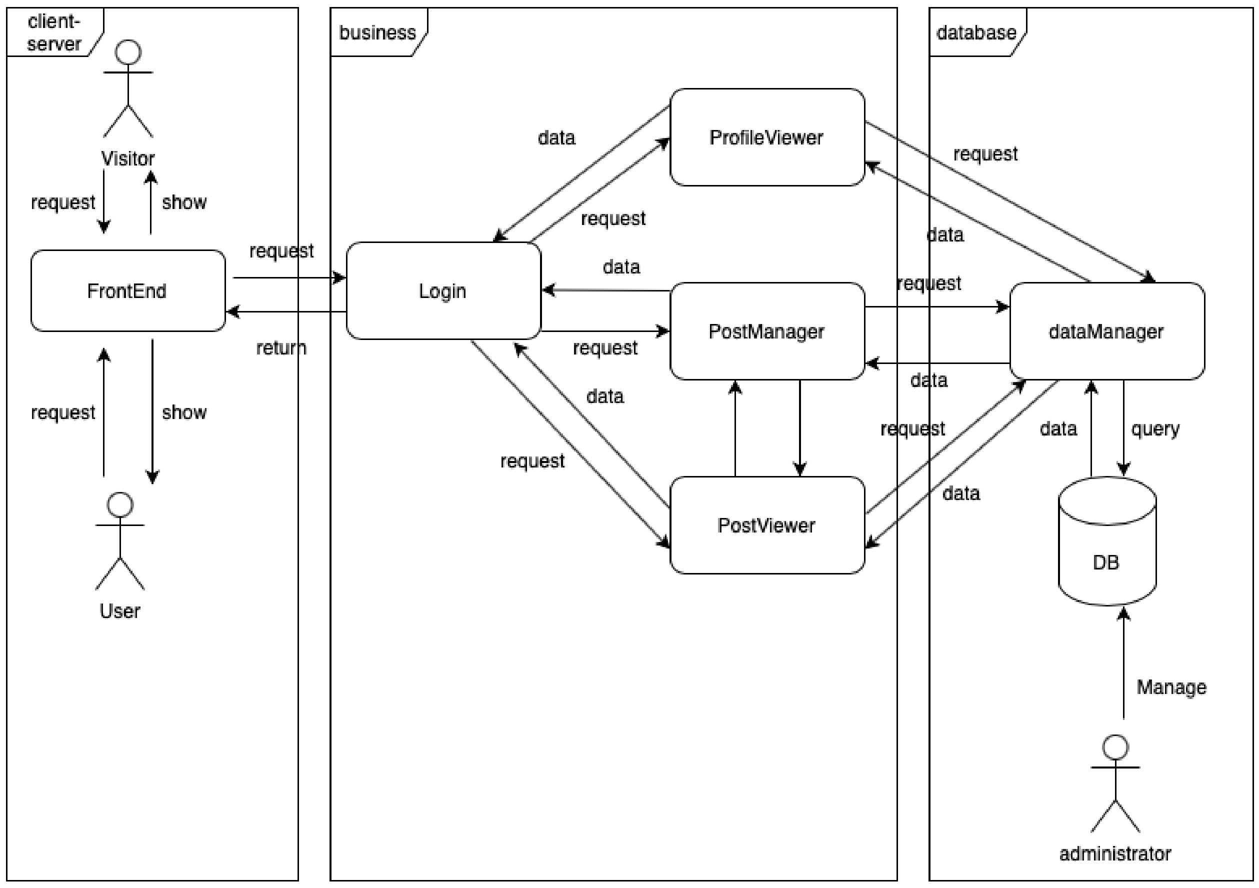
Table of Figures

1. System Analysis

1.1 System Overview

The system will consist of 5 main components: FrontEnd, Login, ProfileViewer, PostManager, PostViewer, and data Manager. All visitors can send a request to Login. ProfileViewer will send a request from users to dataManager, and transfer the data from dataManager back to the login system. PostManager is for users to do some operations including delete posts, add posts, and comment posts. PostViewer helps show which post do users want to see based on different algorithms. Normally it will show posts based on users' prefered companies or fields, or the users can also change to only see the posts created by some other users who they followed. DataManager can receive all the requests from other parts and use sql to get the desired data.

1.2 System Diagram



1.3 Actor Identification

There are three actors in our system.

a. Visitors

For visitors, they only have the permission to look through some general information of the thread like the title or the general content of the thread. They do not have the permission to use the calendar or check others personal information or personal homepage.

b. User

Once visitors create their own account, they become users. The users have the permission to check everything in the front end of the system and they also have the permission to do all operations of the system, but they cannot check other user's sensitive information.

c. Administrator

The administrators are also called superusers. They have all the permissions to edit everything including frontend backend and database structure.

1.4 Design Rationale

1.4.1 Architectural Style

The three-tier architecture will be used for this project. The client-server tier allows the website to handle the request from users, and show what the user wants to see. Specifically, it will receive requests from users and send them to the business layer, and, when the business layer sends data back, it shows the data to users. The database-centric, it queries the database to obtain the data. It helps store the data from the business layer, and also send the desired data back to the business layer. The business layer is for transferring data and requests. It receives requests from the front-end, does corresponding operations, and requests data from the database-centric layer. It might also send successful information back to the front-end. For postViewer in the business layer, it can use the algorithms to get the user's favorite contents whose data are located in the database-centric layer and send them back to the front-end. For postManager in the business layer, it can send requests to the database-centric layer to change or ask for the data. For profileViewer, it can receive requests from client-server layer, and use algorithms to select particular content and send them back.

1.4.2 Design Pattern(s)

We assume that we need following design patterns to make the system easier.

Command:

Command can help turn a request into a stand-alone object that contains all information about the request. The forum needs to show all buttons in one website, but the current problem is that it has a bunch of buttons of different operations like post, comment, like, calendar, etc. With the Command pattern, it should extract all of the request details. And commands will become a convenient middle layer that reduces coupling between the frontEnd and business layers.

Singleton:

This pattern allows objects to have a single instance that it uses a private variable to store that instance and we can use lazy loading to ensure that there is only one instance of the class. It will prevent multiple instances from being active at the same time. One reason we need it is for our logger. This pattern makes it easier to keep track of the activities with one instance.

Strategy:
With this pattern, different implementations will be decided during the run time based on the client. One potential case in which we need this pattern is during the payment process. In the future we plan to give users the opportunity to pay for broadcasting their posts. So the system has to support different payment options for users so the implementation of the payment should also be different including credit card or WeChat or other payment methods.

1.4.3 Framework

Backend:

The forum will use SpringBoot + MyBatis + Maven for basic structure of the forum. And it will use MySQL for our DBMS.

Frontend:

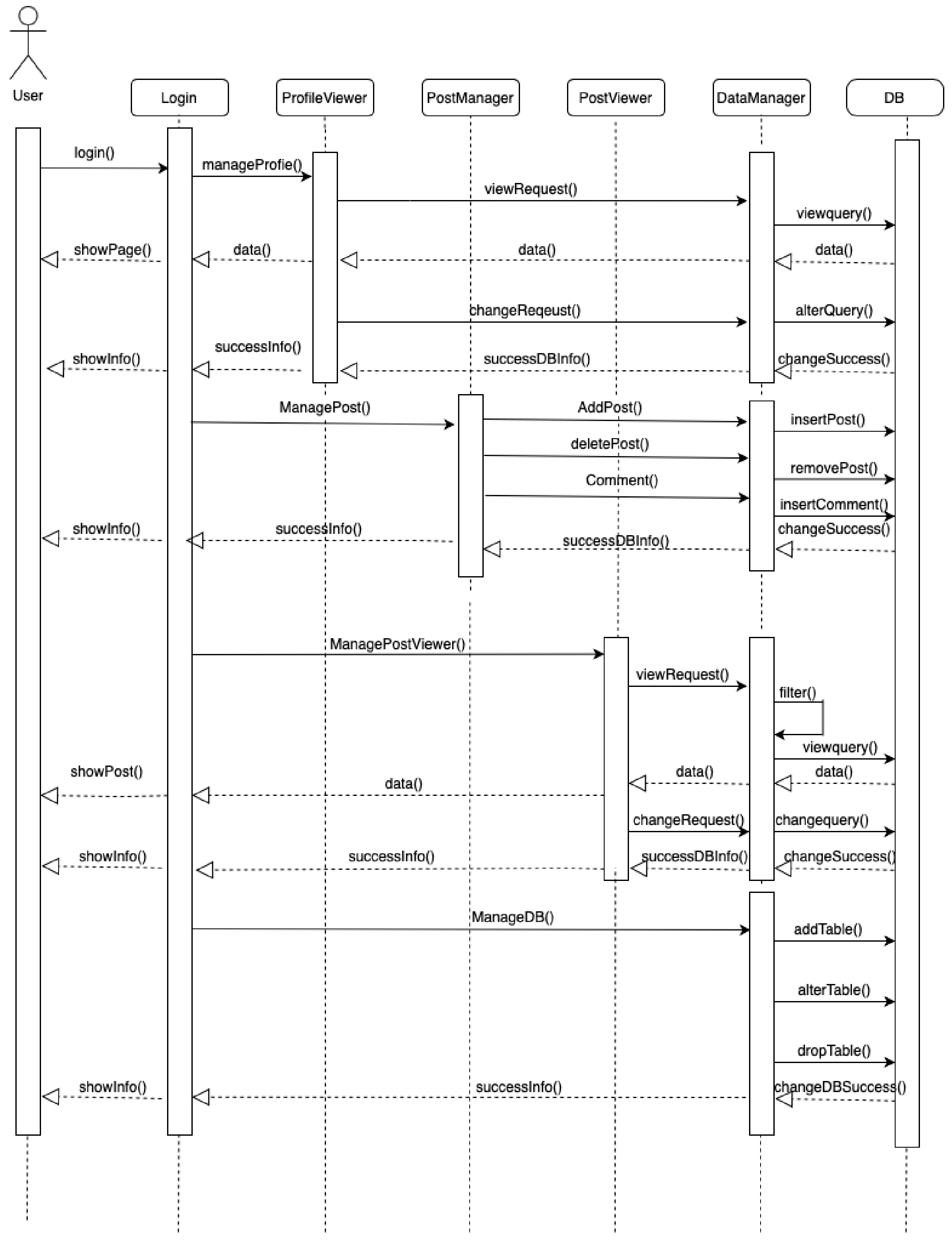
The forum will use Thymeleaf + Layui. Thymelaeaf can offer the basic template. Layui is a sentimental front-end UI framework written in its own module specification. It follows the writing and organization of native HTML/CSS/JS. The threshold is extremely low and ready to use.

Website of Thymeleaf: https://www.thymeleaf.org

Website of Layui: https://www.layui.com66`,

2. Functional Design

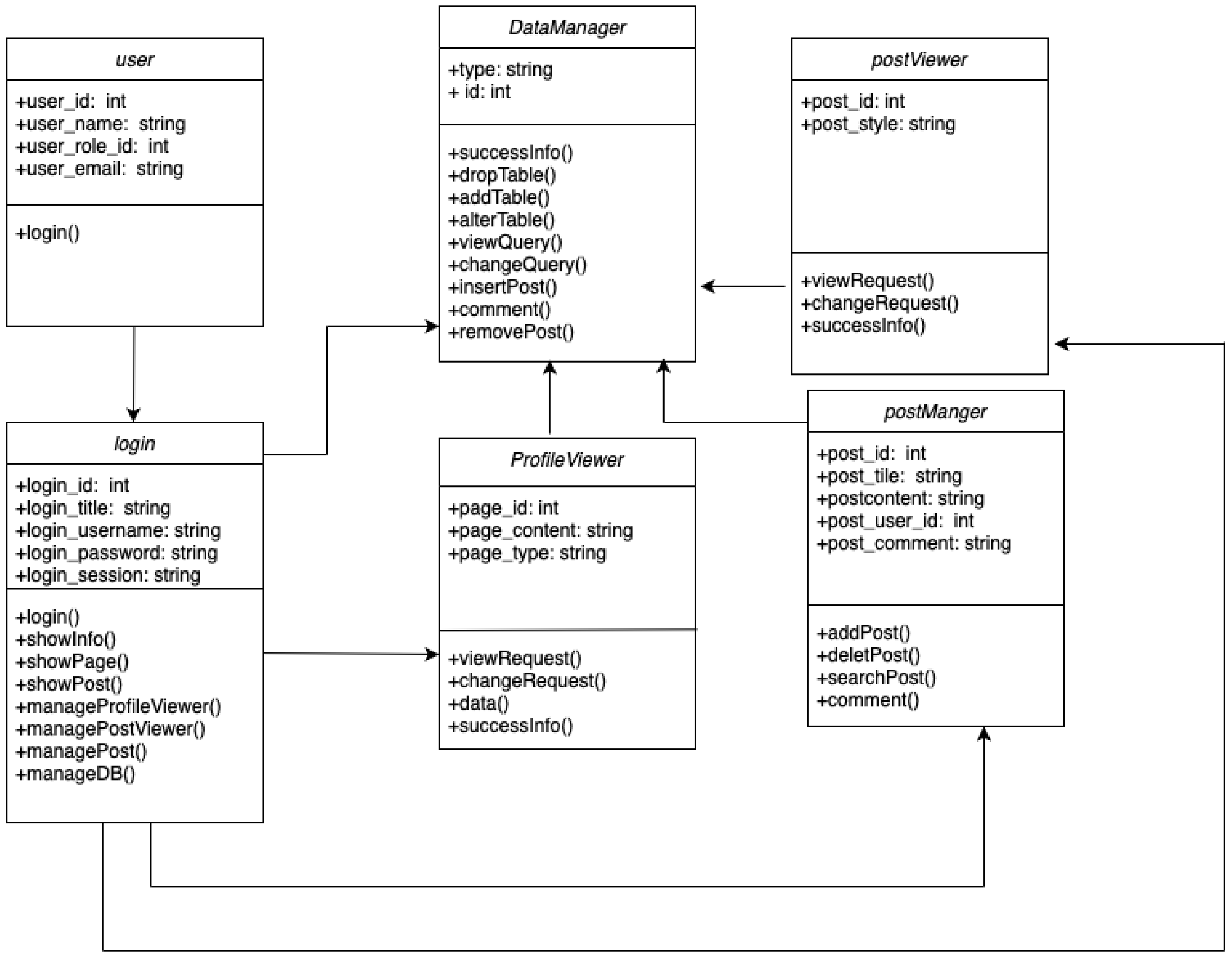
2.1 System sequence diagram



- When a user can successfully login, he or she can do these operations.
- Users can do manageProfile(), to use all the operations in the profileViewer.
    - Users can use viewRequest() to let dataManager use queries for retrieving desired data for viewing pages. The dataManager will use viewQuery() to do the query part.
    - Users can use changeRequest() to let dataManager use queries for changing raw data for pages. The dataManager will use alterQuery() to do the query part.
    - When Database returns data or success information, it will send them back all the way to the front-end.
- Users can call managePost() to do some operations related to posts.
    - Users can do addPost(), deletPost(), or comment().
    - And the datamanager will do insertPost(), removePost(), or insertComment() to save all data to the database.
    - Once the dataManager successfully saves these changes to the database and it will back a successful information to tell the PostManager that all actions are accepted.
    - When Database returns data or success information, it will send them back all the way to the front-end.
- Users can do managePostViewer(), to use all operations in PostViewer.
    - Users can use viewRequest() to let dataManager use queries for retrieving desired data for viewing posts.
    - PostViewer will call filter() to use some algorithms to filter the posts and send the data back all the way to the user.
    - Users can use changeRequest() to let dataManager use queries for changing raw data for desired posts. The dataManager will use changeQuery() to do the query part.
    - When Database returns data or success information, it will send them back all the way to the front-end.
- SuperUsers can do manageDB(), which might include changing the query or other related process. It can do addTable(), alterTable(), or dropTable() to change the structure of the database. And it will send back successful information.

3. Structural Design

3.1 Class Diagram


user
+user_id: int
+user_name: string
+user_role_id: int
+user_email: string
+login()
DataManager
+type: string
+ id: int
+successInfo()
+dropTable()
+addTable()
+alterTable()
+viewQuery()
+changeQuery()
+insertPost()
+comment()
+removePost()
postViewer
+post_id: int
+post_style: string
+viewRequest()
+changeRequest()
+successInfo()
login
+login_id: int
+login_title: string
+login_username: string
+login_password: string
+login_session: string
+login()
+showInfo()
+showPage()
+showPost()
+manageProfileViewer()
+managePostViewer()
+managePost()
+manageDB()
ProfileViewer
+page_id: int
+page_content: string
+page_type: string
+viewRequest()
+changeRequest()
+data()
+successInfo()
postManger
+post_id: int
+post_tile: string
+postcontent: string
+post_user_id: int
+post_comment: string
+addPost()
+deletPost()
+searchPost()
+comment()