Final Report

Hou Shizheng, Zheng Ying

I. Overview

Kanban is a workflow management method for defining, managing and improving services that deliver knowledge work. Deriving from a Japanese word “kanban”, it was first developed and applied by Toyota as a scheduling system (Fig.1) for just-in-time manufacturing in the late 1970s. Because of the properties of full-transparency workflow, planning flexibility, real-time communication, and efficiency maximization, Kanban is quickly spread all around the world. Agile software development teams today are deeply dependent on this amazing method.

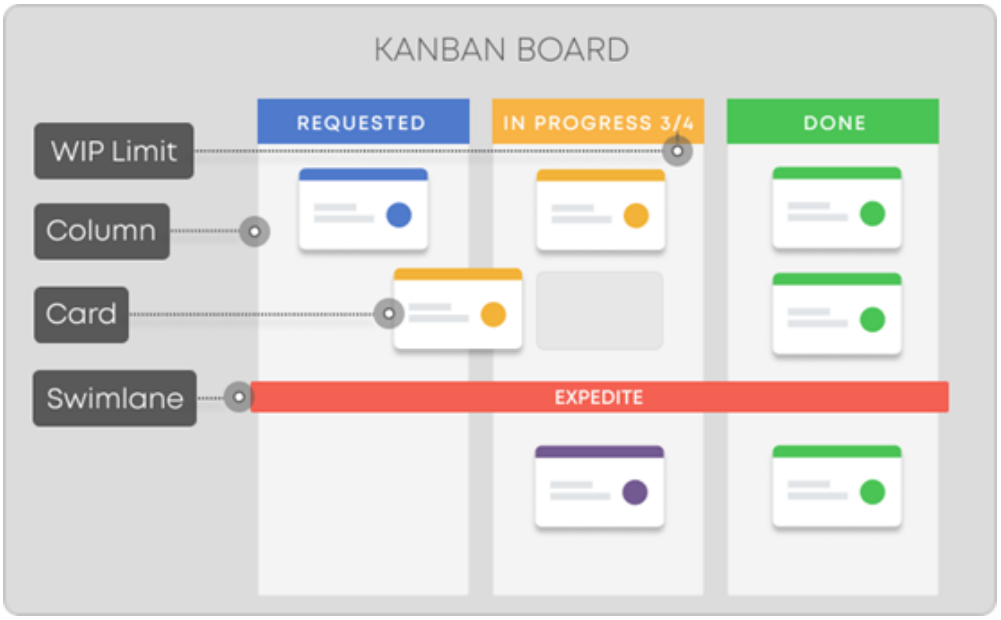


Fig.1 Kanban Board

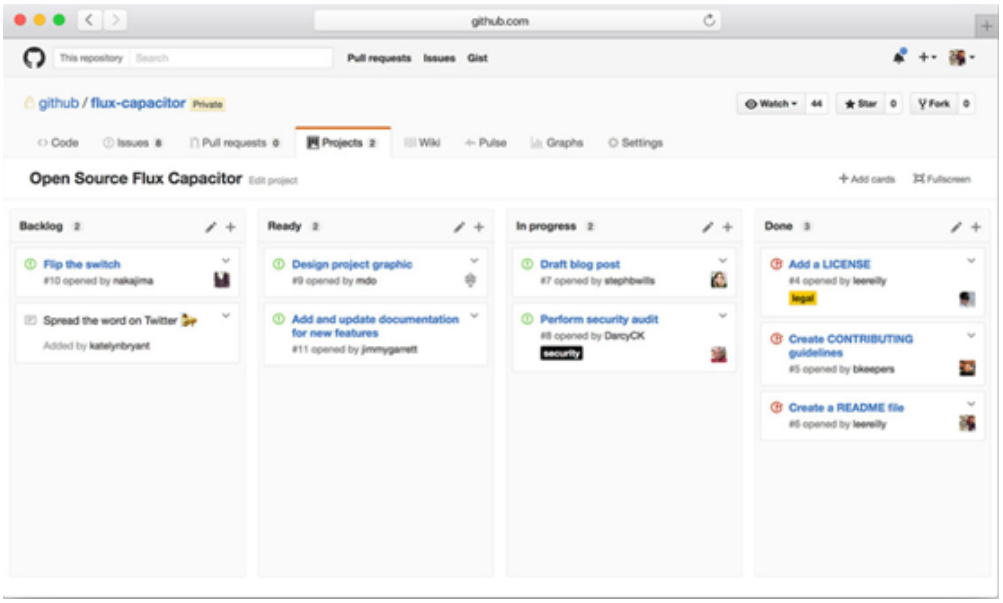


Fig.2 “Project” Board on GitHub (Kanban on GitHub)

Our project is inspired by GitHub's Kanban, a new function released last year and shown as the "project" board (Fig.2). Together with "issue" and "pull request", GitHub helps us manage projects more organized. However, different from production and manufacture, which are clear and with universal certainty on the workflow, software development has a lot of uncertainty due to the design stage. So, voting together with Kanban is like icing on the cake, which makes users easier to provide feedback on the frequent new development ideas. Just click a button, your support on ideas is well expressed.

And the application "Padlet" (Fig.3) also gives us some ideas. Padlet is an application that we use to ask questions on the other lecture this semester. Compared to the chat box in zoom, the message inside Padlet won't disappear with the end of lecture. All the questions can be checked every time we want which is convenient. But there are also some functions that can be improved. For example, all the questions are sorted by time, so the teacher cannot catch the most common question if time is limited; questions cannot be marked, so it is hard for the teacher to find out which questions have already been asked and which do not quickly. If the voting function is developed here, the above troubles can be solved immediately. Every student can vote for the question posted by others that he is also confused about, so how common one question is will be shown by the number of votes. Then, if all the questions are sorted by the number of votes, everyone can get the most common question. What's more, teachers can also mark a question as solved by voting for it.

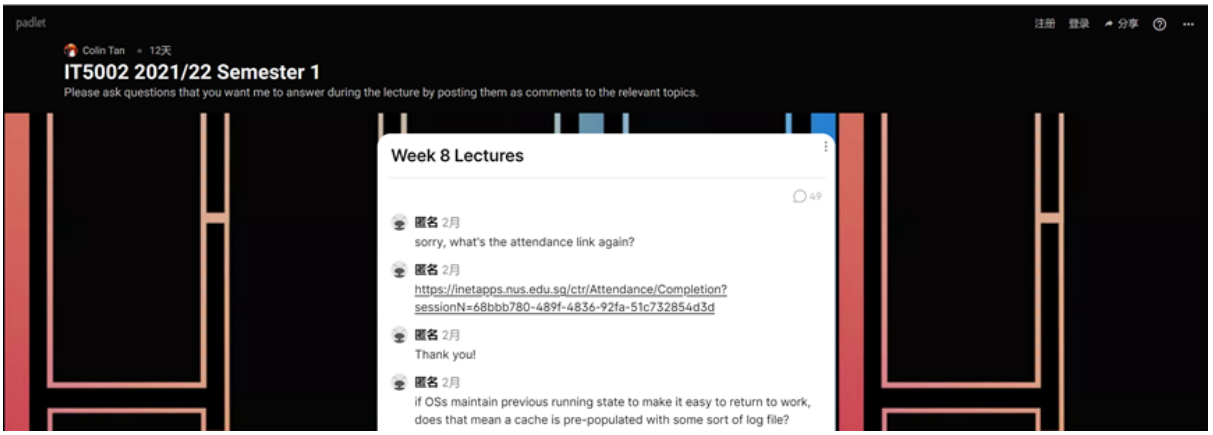


Fig.3 Padlet

Therefore, our system VoTeam occurs. It not only can be used for agile software development as said before, but also can be used as only a simple voting system, like determining where to go in teaming building activity, ordering the prioritization of some issues depending on the support number and so on. What's more, away from the concept of repository on GitHub which always confuses non-developers, VoTeam is a more user-friendly system for everyone.

Once users have registered and logged into the system, they can create their own topics on the Dashboard page. When a topic is created, its content, column number, anonymity option and maximum voting number for a single user can be customized. And the encrypted topic link can be shared with other users for voting on or posting ideas. What is worth mentioning is that anonymity really brings great convenience to all the users. It allows users who can

obtain the topic link to vote for an idea without logging in, which makes the cumbersome steps like registration and login optional, so no one needs to worry about his identity disclosure. Of course, some measures must be taken to prevent topics from the anonymous user's malicious behaviors.

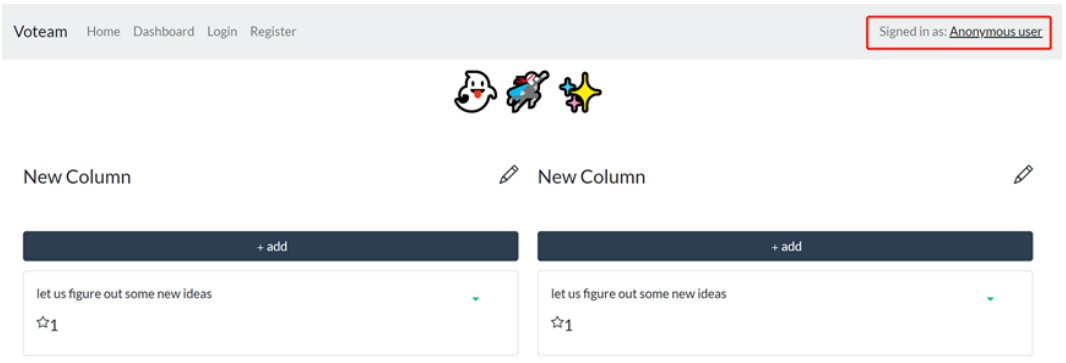
II. Features

1. Password-free Login

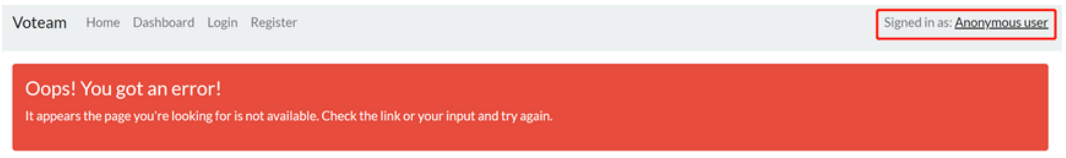
When an account is logged in, a Json Web Token (JWT) will be generated by the server automatically and stored in localStorage. In two days, no repeat login is needed, instead, the account will be logged in automatically.

2. Anonymous User

Anonymity mode will be triggered when you enter VoTeam through another account's sharing link. When you access the sharing URL, the back end will check your browser's localStorage. If there is nothing in it, an anonymous user will be registered automatically, and a token will be generated as well. Then the anonymous user will be logged in automatically and be used to vote. If the topic supports anonymity, you will see the topic page shown as Fig.4(a) and you can vote using an anonymous account, otherwise, the system will route to the error page shown as Fig.4(b).



(a)



(b)

Fig.4 Anonymous User Page

3. Operation Permission Restriction

To prevent topics from malicious modification, there are some operating restrictions for anonymous users (Fig.5). Anonymous users cannot route to the Dashboard page. If an anonymous user clicks the “Dashboard”, the Login page will be displayed. What’s more, anonymous users cannot edit topic name, column name and idea name, so the related button is not shown here. But the new idea addition function is available for anonymous users to post their opinions.

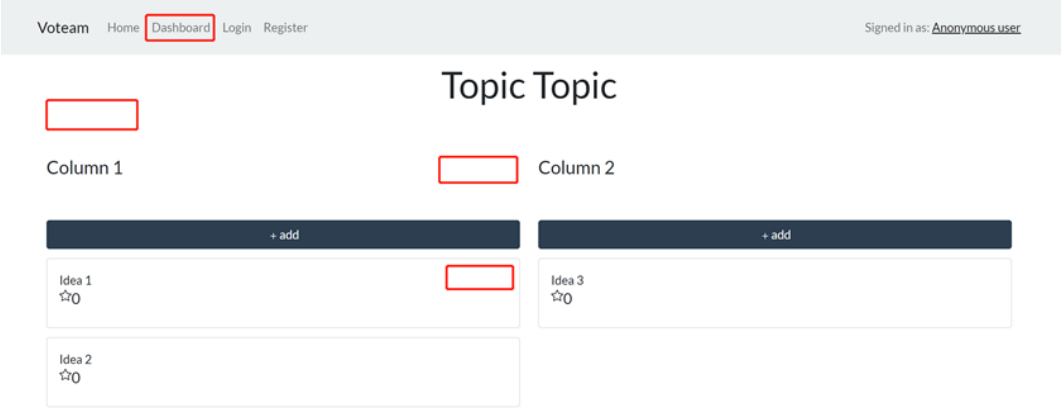
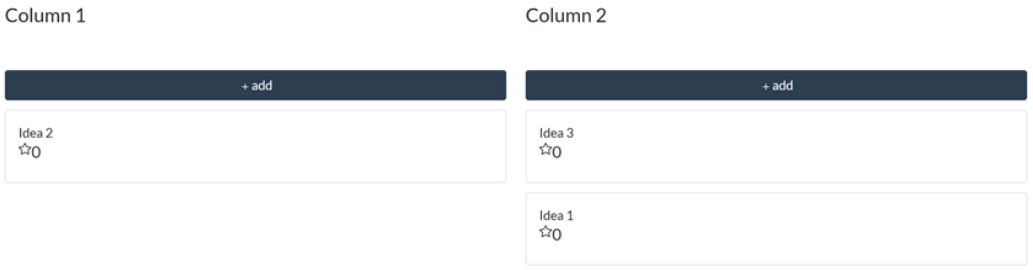


Fig.5 Page for Anonymous Users with Operating Restrictions

4. Idea Card Dragging and Dropping

Every idea card can be dragged and dropped from one column to another. Fig.6 shows the operating process. (a) shows the page before dragging and dropping operation, (b) shows the page under the process of dragging and dropping and (c) shows the page after dragging and dropping operation.

(b)



(c)

Fig.6 Idea Card Dragging and Dropping

5. Topic Encryption

Instead of showing the real topic ID that the database stores, a “fake” ID that generated by Advanced Encryption Standard (AES) algorithm is used in the URL, which can prevent users from accessing our backend URL of the topic and make a protection for our back end..

6. Sorted Idea Card

After each voting, the displayed idea cards will be resorted by the number of voting.

Fig.7 Idea cards before and after sorting

7. Topic Paging and sorted topic

At the bottom of the dashboard page, all the topics created before will be displayed. To make pages more clear and neat, paging technique is used here. At most 5 topics are shown on each page (Fig.8). And also, the topic will sort by update time, so it is convenient for user to find the most recent topic.

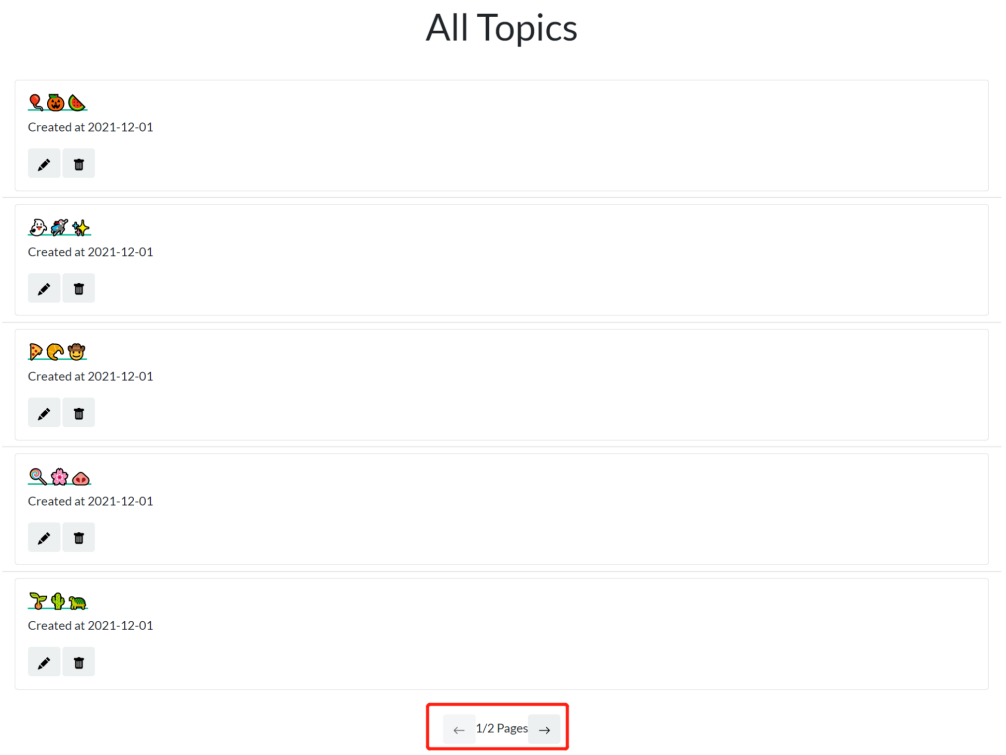


Fig.8 Topic Paging

8. Emoji input

Emoji input is supported in the text box.

III. Project Schedule and Member Roles

Tab.1 Task Allocation

Tasks	Priority	Deadline	Assignee	Remarks
Basic frontend frameworks, routers	High	Week 6	Hou Shizheng	
Frontend home, login, register and dashboard pages and related APIs	High	Week 6	Zheng Ying	
Frontend topic page and related APIs	High	Week 8	Hou Shizheng	
Basic backend settings, database models, database settings	High	Week 6	Hou Shizheng Zheng Ying	
User API (controllers and related data model)	High	Week 9	Hou Shizheng	
Topic API (controllers and related data model)	High	Week 9	Zheng Ying	

Idea API (controllers and related data model)	High	Week 9	Zheng Ying	
Column API (controllers and related data model)	High	Week 9	Hou Shizheng	
Vote API (controllers and related data model)	High	Week 9	Zheng Ying	
Other APIs (If necessary)	Medium	Week 10	Hou Shizheng Zheng Ying	
Split the controllers into controllers, services and repositories.	Medium	Week 10	Hou Shizheng Zheng Ying	
Support drag and drop on frontend topic page	Medium	Week 10	Hou Shizheng	
Support real-time vote, edit and update on frontend topic page	Low	TBC	Hou Shizheng Zheng Ying	Cannot be guaranteed
Testing and debug	High	Week 12	Hou Shizheng Zheng Ying	
Deploy and release	Medium	Week 13	Hou Shizheng Zheng Ying	

IV. Sample Mock-Ups

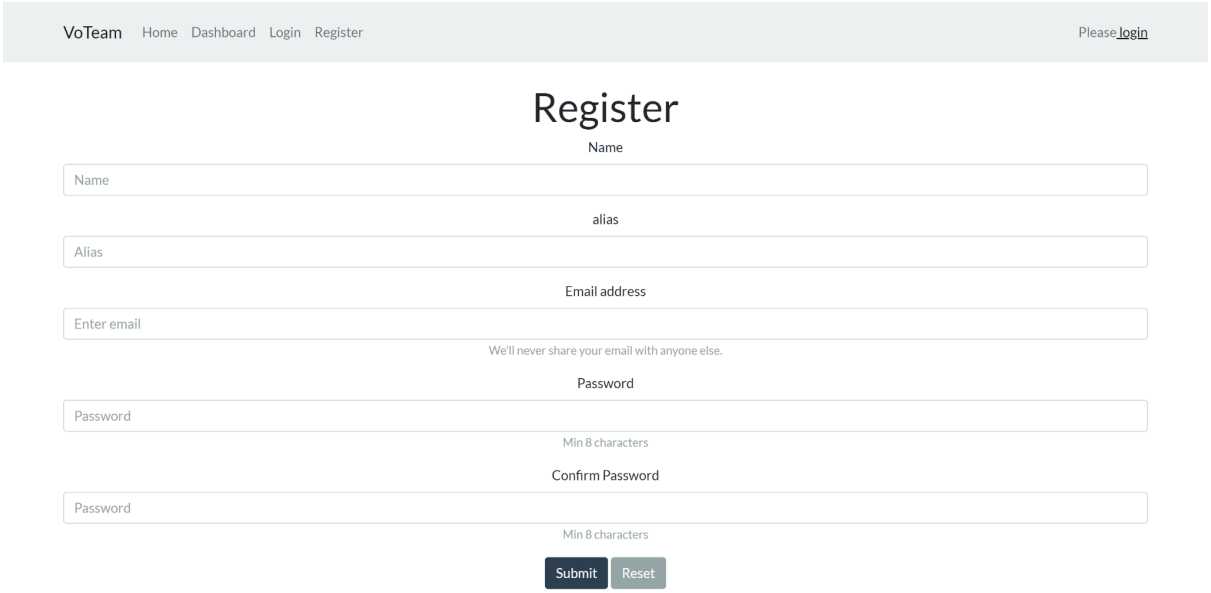


Fig.9 Registration Page

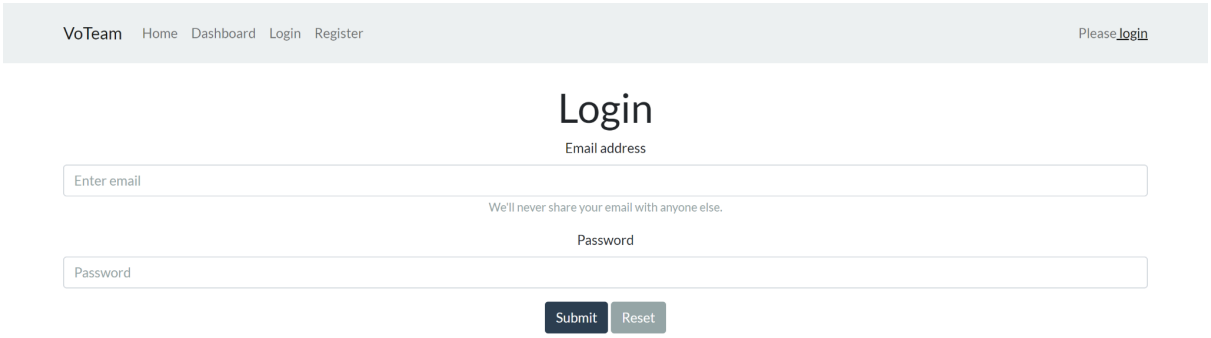


Fig.10 Login Page

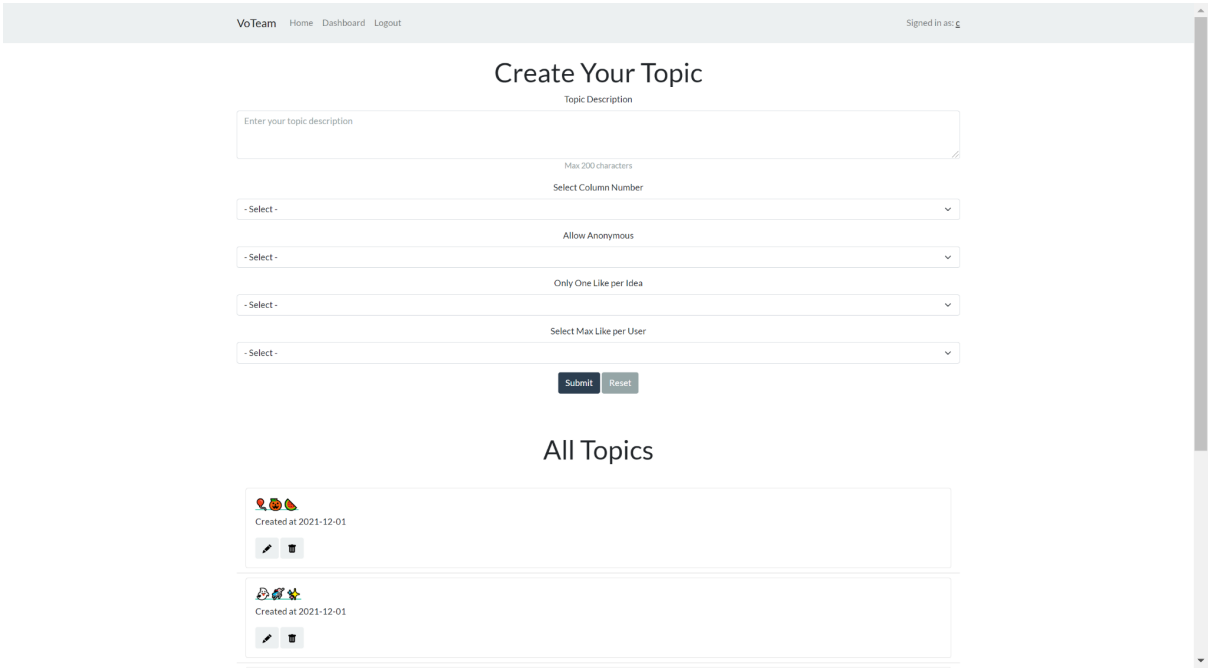


Fig.11 Dashboard Page

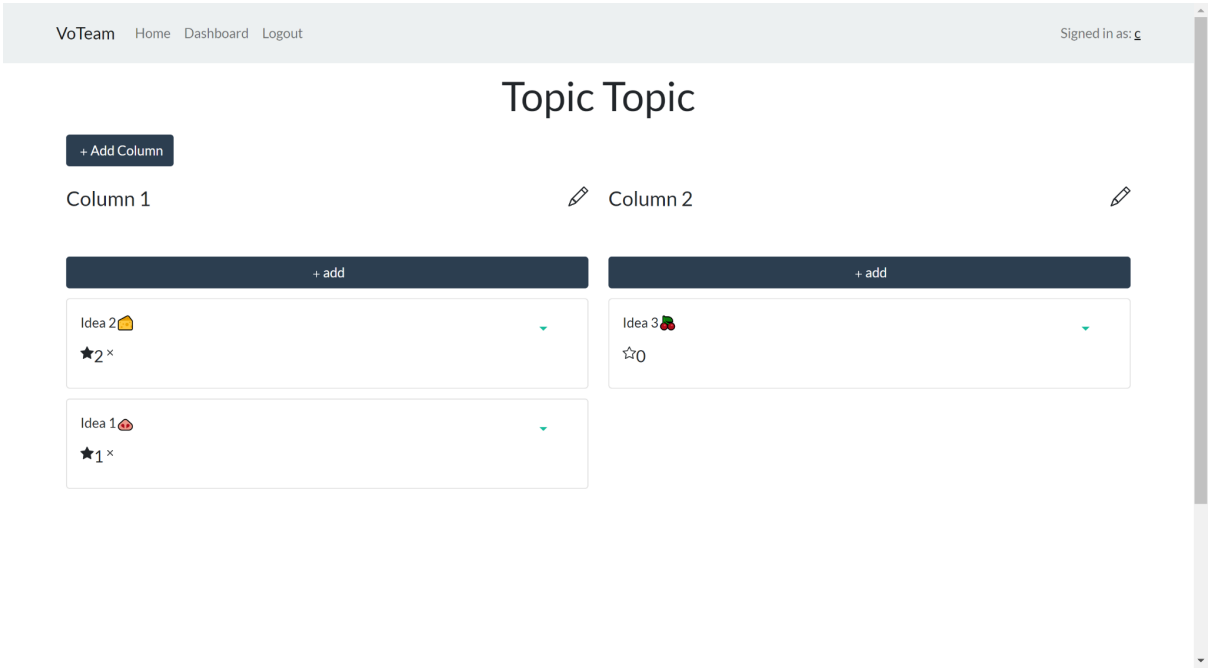


Fig.12 Topic Page

As shown in Fig.9 and Fig.10, Users can register accounts at the registration page and can log in to their accounts at the login page.

The dashboard page (Fig.11) is for users to create their topics including topic description, whether to allow anonymity, column numbers, max vote per user. All topics created by a user are also displayed on the dashboard page as a list of cards, and users can edit, delete, and get links for topics by clicking the related buttons shown there.

The topic page (Fig.12) displays all ideas created in a specific topic. There are different columns, and also different cards representing different ideas below the columns. All non-anonymous users and anonymous users can vote for any ideas and can post their ideas in any column, and the idea will be displayed as a card in corresponding columns. Topic creators can also add or delete columns, change the topic description of the topic.

V. Architecture Design

i. Initial proposal

For our project, we are planning to stick to the MERN stack (Fig.13). Namely, the ReactJS library will be used in our front end, the Express framework based on NodeJS will be used in our backend, MongoDB will be used as our database. We will follow the style of front and back end separation to implement our project.

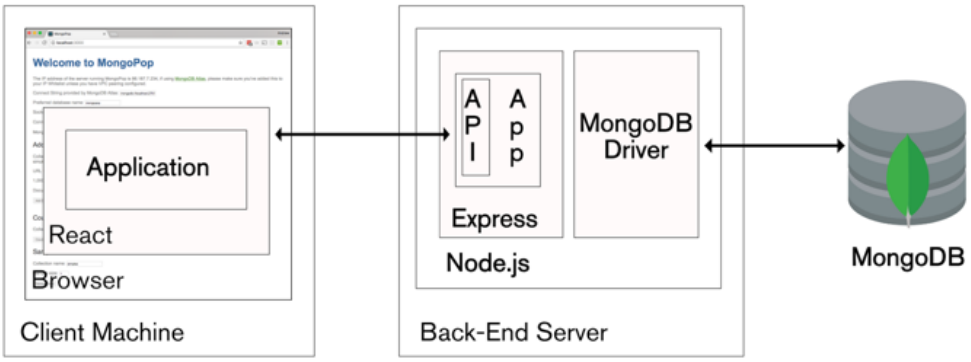


Fig.13 Overall Architecture (Source: www.mongodb.com)

1. Frontend

To be more specific, for the frontend, we will design various React components for different pages, especially as many functional components as possible, and cooperate with Hooks to handle events and update datas and states. React routers will be used for navigating between different pages. Context and Redux may also be introduced when necessary. Axios will be used to communicate between the front-end and back-end APIs. Material Design or React Bootstrap may be used as our design system. We will also use some encryption algorithms and authentication mechanisms to protect our topic and authenticate anonymous and non-anonymous users.

2. Backend

For the backend, We will follow a RESTful style. Namely, we will design our backend as many RESTful APIs, using URLs to locate resources and HTTP verbs (GET, POST, DELETE, PUT) to describe operations, thus making our application architecture more concise, more hierarchical, easier to implement mechanisms such as caching. For our project, we will design five basic database models (Fig.14).

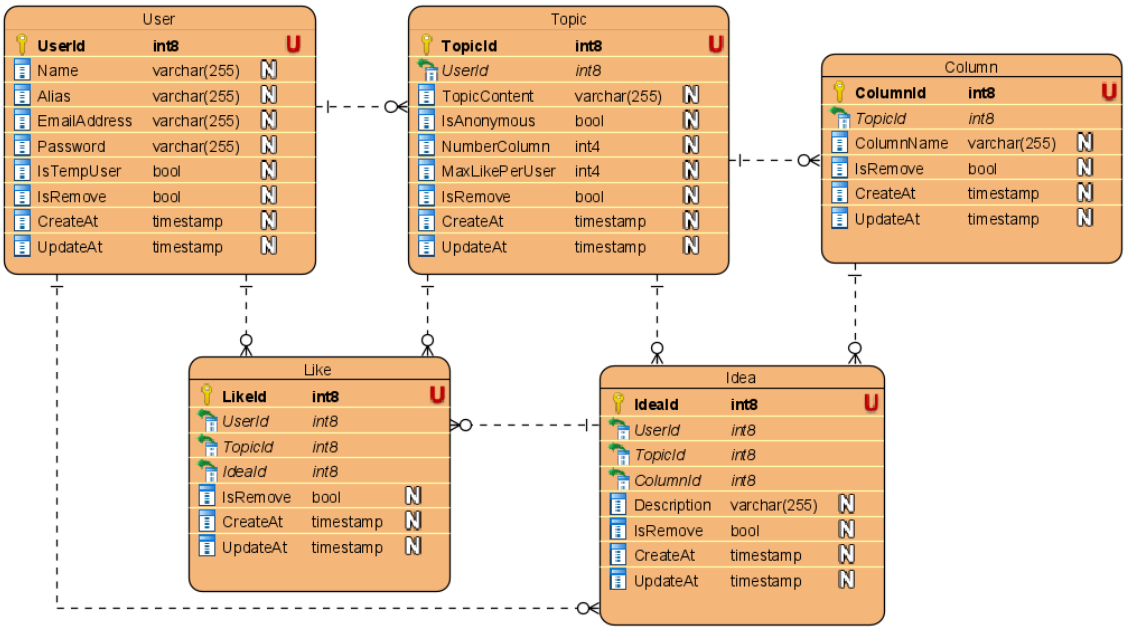


Fig.14 Basic Database Models

For security and convenience reasons, we may gradually add different DTOs, such as DisplayDTO, EditDTO, RegisterDTO, for our data models.

We will design different controllers corresponding to different data models to implement our business logic. Thus, business logic will be translated to CRUD operations with the database. For extensibility and flexibility of our code, we may consider splitting the controllers into controllers layer, services layer and repositories layer. Thus, the controllers layer will only be responsible for dealing with HTTP requests from different APIs, the services layer will deal with business logic and user authentication, the repository layer will communicate with the database to do all CRUD operations.

Finally, our backend will be presented as REST APIs for the frontend to consume.

ii. Updates to the Architecture

1. Frontend

For the frontend, the implementation basically matches the original proposal and the mock-ups, but now we can introduce the features more specifically.

For the UI part, we chose React Bootstrap as our basic design system. React-Bootstrap is a complete re-implementation of the Bootstrap components using React. It has no dependency on either bootstrap.js or jQuery, and provides a more reliable solution by incorporating Bootstrap functionality into React's virtual DOM. And we used the Flatly theme from Bootswatch as our basic UI theme since it balances simplicity and focus, making it easy to use. The icons we used were mainly from react-icons, a popular icon library for bootstrap.

For better code readability and code extensibility, we carefully designed our code architecture and code modules. For example, we designed a uniform API module with AXIOS, a popular AJAX library, to communicate with the backend. We write a createAPIEndpoint method in the module and call it by passing the required parameters to it, thus we can easily send http requests. We also added an interceptor to add authorization information to our request header before sending requests, eliminating the need to manually add authorization information every time we send a request.

For the main functional part, function components are designed and implemented by us. The main reason we chose function components rather than class components is that function components are simpler, easier to read, easier to maintain, and more recommended by React, compared with class components. For better code logic and architecture, we separate the main logic part to hooks folder, and remain the ui part in the main part of components. We also created a NavBar component to realize the navigation bar one the page top, and we reuse it in all relevant components, thus reducing the code redundancy. And since the topic component is much more complex, apart from the main component, we also create a separate folder called Kanban to deal with it.

For more concise and clear URLs and routing logic, we adopted the react routers in our project. Each URL path is bound to the corresponding route and component. Thus we have a concise architecture.

In order to achieve uniform and standardized user authentication logic, we choose the JWT (JSON Web Token) token as our authentication tools. Once a user has logged in, the user will receive a corresponding token and save it to the Local Storage of the browser. Then we put the token in the Authorization part of our request header on every follow up request, thus the backend can identify the request user and do some related verification and permission control logic. Since the JWT token has a signature part, which is encrypted by the backend server, this method has a relatively high security level and can prevent attacks from malicious users.

For anonymous users, we also use the same method to mark and identify the users. We put an encrypted userId as the payload part of the token, so the backend can identify the user, and do related permission control for the user. For example, anonymous users can only post their ideas and vote for others' ideas, but cannot create topics, cannot change columns of a specific topic, cannot change the contents of idea cards posted by other users. Only after the user registered as a formal user, he/she can have access to all functions.

As we mentioned previously, our application can be used as a Kanban application, so the drag and drop function is very important. We implement the drag and drop function by a popular library called Dragula. After importing the dragula library to our code, we can create a dragula instance. Then we put the required HTML elements into the container attributes of the dragula instance. Every time we drag and drop an idea card, by calling specific methods of the dragula instance, the dragula instance will track the element id, source column id, target column id, thus we can use the ids to constitute the request body, and send the corresponding request to backend. Therefore, the drag and drop feature is not only a frontend feature, but also needs backend operations.

2. Backend

For the backend, the overall architecture is basically the same as previous proposoal, we follow the specifications of the RESTful to design our APIs to process get, post, put and delete requests and related CRUD operations. Our implementation of the Model is basically the same as previous proposal. However, there are some differences with the proposal in details. Since JavaScript is the programming language we used in our backend, and JavaScript is not a strongly typed language, so it is not necessary for us to set up DTOs (Data transfer models) in our backend. Another different is that for convenience and better manage our code, we did not adopt the same architecture layering scheme as mentioned in the previous proposal, but we still layered the architecture, which will be talked follow up.

For almost all requests from frontend except the user related requests, we design a uniform layered architecture. Firstly, in order to make the overall architecture clearer, we designed a global router middleware and used the router middleware to distribute different requests to different routers, for example, all user related requests will be directed to the user route, all topic related requests will be directed to the topic router. In every route of every router, the route will exactly match the corresponding HTTP requests, for example, a put request with url end with "`api/topics`" will be processed by the "`router.put("/", auth, topicValidator.updateTopic, topicCtrl.updateTopic)`".

For almost every request, it will go through three middlewares to process. The whole process is uniform for almost every request. First middleware is authorization middle ware (i.e., auth), this middleware will get the JWT token from the request header, verify and decode the token, and put the decoded user information on request object for follow up middleware to consume. Second middleware is validator middleware (e.g., topicValidator), this middleware will verify the legality and validity of the data in request url and request body. This not only can detect invalid requests earlier, but also prevent some illegal and insecure requests (for example: SQL injection) be executed by our server. If all verification passes, the second middleware will get some data from the database and put the data on the request object for the next middleware to consume. Third middleware is controller middleware (e.g., topicCtrl), the middleware will perform the required CRUD operations, for example, it may get some data and perform filter and sort operations according to the request, and finally send the final json object to the frontend.

As we said in the frontend part, we choose JWT (JSON Web Token) token to realize uniform authentication login. JSON Web Tokens consist of three parts separated by dots (.), which are: Header, Payload, Signature. The header typically consists of two parts: the type of the token, which is JWT, and the signing algorithm being used, such as HMAC SHA256 or RSA. In the project, we used HS256 as our signing algorithm. The second part of the token is the payload, which contains the claims. We place the necessary information such as user id in the payload for verification. The third part is signature, which is the encrypted Header and Payload, and the secret key to do encryption was stored on the server. So, we can prevent falsifying tokens and protect the security of server and user data.

Once the backend receives the login request, after the verification process, the controller will generate a corresponding JWT token and send back the frontend, thus every follow up request will carry the token for the backend to verify and identify the user information. This process also applied to anonymous users, but for anonymous user, there is a additional

step, once we received a expired token, we will issue a new valid token for the anonymous user, and user information in the payload of the new token and the expired token are actually same, to ensure the consistency of the anonymous user.

For anonymous users, there are some additional operations in our backend. For example, when an anonymous user tries to get a topic content which does not allow anonymous users, the validator will identify the user as an anonymous user and reject the request. When an anonymous user tries to get a topic content which allows anonymous users, the controller will identify the user as an anonymous user and set some fields in the response data to false, to set required permissions control to the anonymous user.

VI. Performance (Tutorial 7)

(1) We measured the performance of the landing page and the topic page (our most critical page and operation) using the performance API, the main data we got as shown in Fig.15 and Fig.16

Fig. 15 Landing page

Fig. 16 Topic page

From the data we got, we can calculate the loading time for the landing page is 790.7ms, while the loading time for the topic page (one of the most data-intensive pages) is 2493.4ms. The main reason for the difference between the two is that the topic page has more HTML elements, and needs to send requests to backend and get response data from the backend, so the whole process will take some time.

(2) We measured the latency caused by the topic page at the server itself using postman instead of using the performance API, and the result we got as shown in Fig.17 as follows:

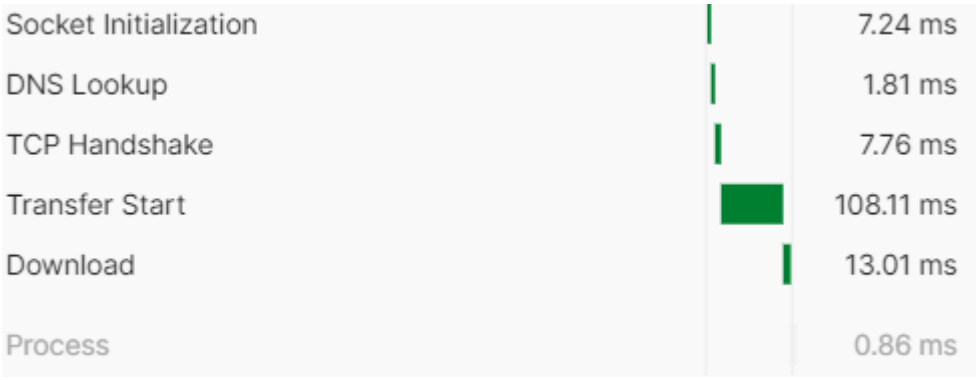


Fig. 17 Latency caused by the backend

From the figure above, we can see clearly that the actual latency caused by the backend server is very small, compared with the whole loading time we saw previously. Therefore, we can conclude that the main latency was caused by the frontend to parse data and render all HTML elements.

VII. Operating Instruction

Details can be seen in README.md in voting folder

i. Deploy

- Download code from GitHub repo: https://github.com/HOU-SZ/voting
- Install npm, mongoDB on server-side, install npm on client-side
- Install dependencies
- Get started

ii. Usage

- Click `Register` to register an account
- Click `Login` to log in your account
- Click `Dashboard` to create your topic, then you can explore to use all the functions
- Click `Voteam` logo or `Home` to return to the homepage
- Click `Logout` to log out your account

Notice: when you do Drag and Drop on idea cards, you may see the below error: `NotFoundError: Failed to execute 'removeChild' on 'Node': The node to be removed is not a child of this node.` Do not have to care about it. This error is caused by the dragula library. It is visible only in development. It will not appear if the app crashes in production.