TECHNISCHE UNIVERSITÄT DRESDEN

Department of Computer Science Institute for Systems Architecture, Chair of Computer Networks

Master Thesis

GRAPHICAL DISCUSSION SYSTEM

Kaijun Chen

Born on: 18th September 1990 in China
Matriculation number: 3942792
Matriculation year: 2013

to achieve the academic degree

Master of Science (M.Sc.)

Supervisor

Tenshi Hara

Iris Braun

Supervising professor

Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill

Submitted on: 18th February 2015

TECHNISCHE UNIVERSITÄT DRESDEN

Department of Computer Science Institute for Systems Architecture, Chair of Computer Networks

Statement of authorship

I hereby certify that I have authored this Master Thesis entitled *Graphical Discussion System* independently and without undue assistance from third parties. No other than the resources and references indicated in this thesis have been used. I have marked both literal and accordingly adopted quotations as such. They were no additional persons involved in the spiritual preparation of the present thesis. I am aware that violations of this declaration may lead to subsequent withdrawal of the degree.

Dresden, 18th February 2015

Kaijun Chen

TECHNISCHE UNIVERSITÄT DRESDEN

Department of Computer Science Institute for Systems Architecture, Chair of Computer Networks

ABSTRACT

A discussion of the teaching content or the educational material is always essential for both tutors and students in the teaching activities. In traditional way, a discussion can only be performed normally after courses also requires the absence of the students as well as the tutors.

The traditional approach of discussing shows its limitations. Inefficiency in knowledge acquisition: not all the students have the same question and the tutor is able to offer explanation for only one question at same time; time-consumption: ; low interactivity:

Thus, a discuss system with intense interactivity as well as in crowdsourcing way is highly needed. To achieve high interactivity, a discuss system with graphical tool and real-time data communication is proposed. Students are able to contribute their questions and answers to get to the bottom of his deficiencies of teaching content and the educational material. And students who has the same questions can instantly acquire the best solution which is recommended and approved by the community.

In order to validate and evaluate the concepts of this approach, an implementation of the proposed solution is developed on top of modern web technologies. Moreover, a usability questionnaire survey is proposed and delivered for a quantized evaluation of the client application. The performance of this application is also evaluated at the same time through the created simulation scenarios.

CONTENTS

1 INTRODUCTION

With the rapid development and popularization of internet and technology, the traditional educational activities are moving to the online platforms. MooCs like ..... have taken the responsibilities and ... to .

Tell some histories!

1.1 MOTIVATION

What's the situation now?

Pain?

1.2 GOALS AND RESEARCH QUESTIONS

What's the features of the new system?

What's the problems/question am i solving?

1.3 THESIS OUTLINE

Outline

2 BACKGROUND AND RELATED WORKS

2.1 ONLINE Q.A. SYSTEMS

2.2 MOOCS

3 STATE OF THE ART

The following chapter gives an overview of state of the art. To achieve the high interactivity and responsiveness in the graphical discussion system, a lot of modern web technologies should be applied.

However, there are always plenty of alternatives for each technology which could differ from system to system. So it is important to investigate and analyse the existing solutions and capture an overview about different alternatives of technologies. It's is also vital to understand the benifit and drawback of the technologies used. In addition, a

First of all, the general modern web technology and development workflow will be introduced, which goes through our whole development and has a great impact on the development efficiency. The next part describes the different graphical technologies on web, and which fits our system best. Then an overview of the real time communication technologies is illustrated and evaluated. At last, a collection of modern technologies applied within the backdend server is listed.

3.1 MODERN WEB DEVELOPMENT

3.1.1 EVOLUTION

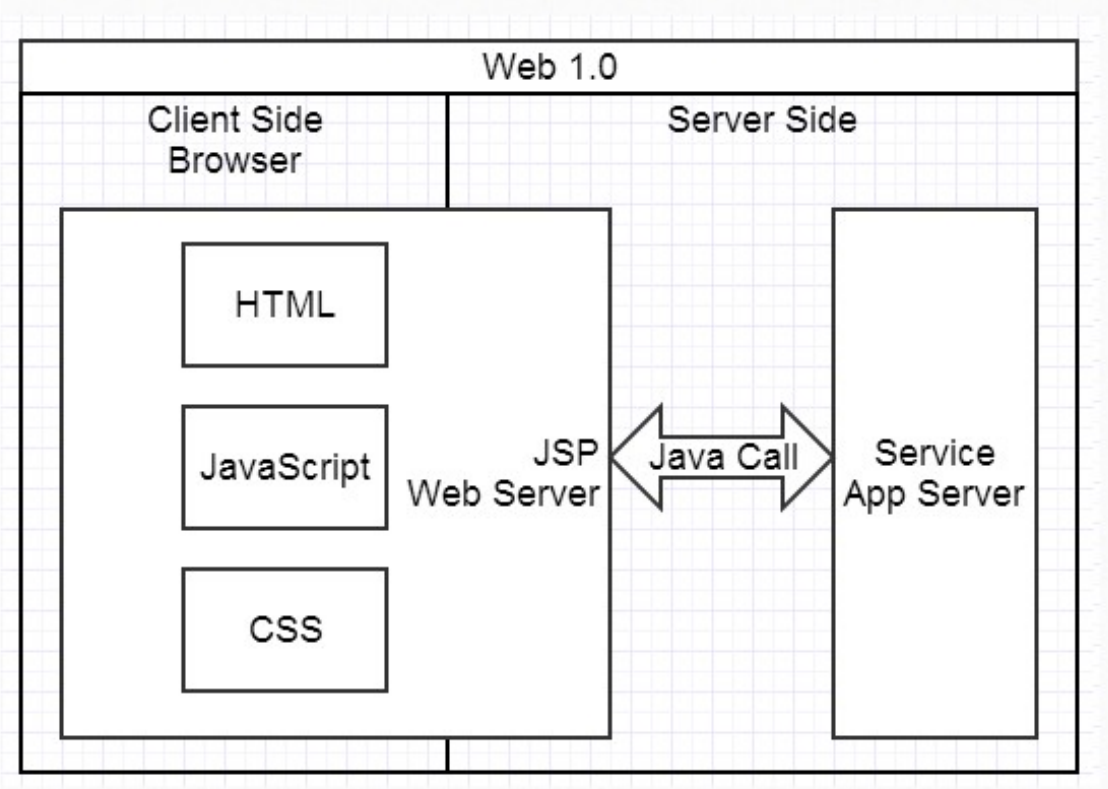
With the increasing complexity of a web app and high demand of agile development, people are always thinking about how to improve patterns of the workflow in web development as well as the architecture of a web app. To achieve better scalability, maintainability and ubiquity of a web app, the architecture of an entire web app has been envolving in the past several years .

EARLY AGE

At the early age of web development, the backend did all jobs for both client side(browser) and server side, such as rendering, calling system service, composing data, etc. Figure 3.1 shows the overview of web architecture: The advantage of this pattern is clear: The development and deployment are simple and straightforward, if the bussiness logic doesn't become complexer. But with the increasing complexity of the product, the problems shows up:

1. Development of frontend heavily depends on the whore development environment. Developers have to start up all the tools and services for testing and debugging only some small changes on view. In most cases, the frontend developer who isn't familar with the backend needs help while intergrating the new views into the system. Not

Figure 3.1: Web architecture in early age



only the efficiency of development, but also the cost of communication between frontend developer and backend developer are huge problems.

2. The own responsibility of front-end and back-end mess up, which could be expressed by the commingled codes from different layers, for example there is no clear boundry from data processing tp data representing. The maintainability of the project becomes worse and worse with the increasing complexity.

It's really significant to improve the maintainability of code, as well as the efficiency and resrationality of division of work from both front-end and back-end in the whole web development phase. In the section below, a evolution of the technical architecture will reveal how these problems are solved.

WEB 2.0

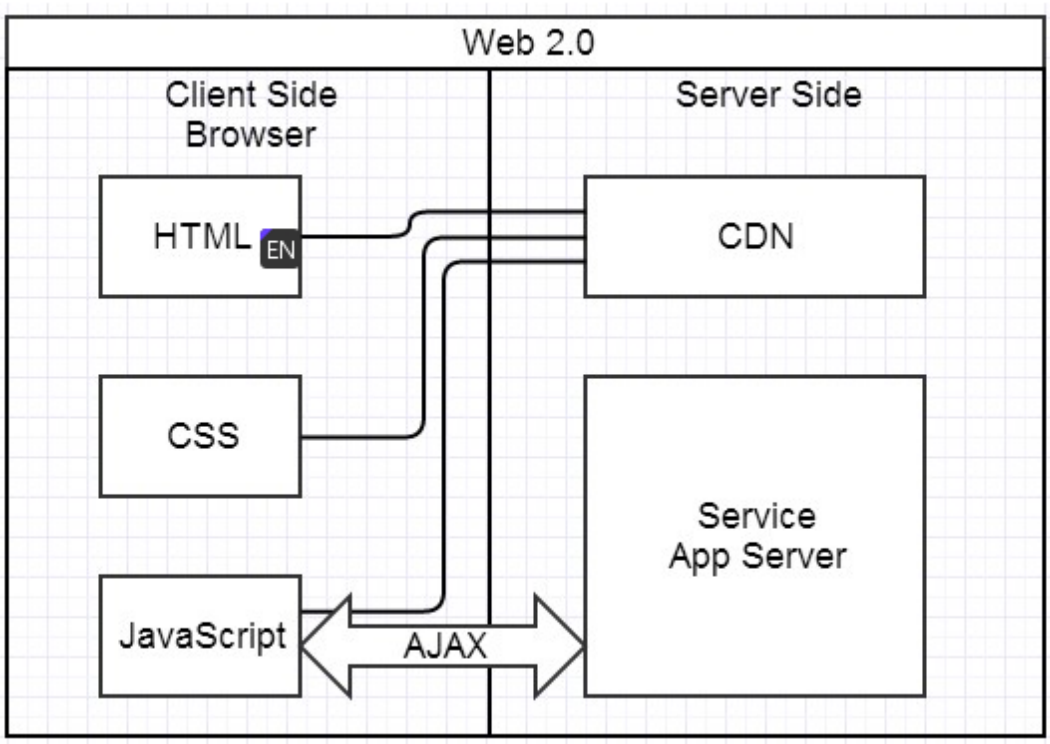
Along with the birth of Gmail[1] in 2004, which is noted for its pioneering use of Ajax, the web application started to behave more interactively. Browser began to take over the job of data fetching, processing, rendering, such a sequence of workflow which could only be done by the server side formerly.

The architecture in Web 2.0 generation is presented in figure 3.2.

By using Ajax, the client has the ability to fetching data stream asynchronously, after which the client will consume the data and render it into the specific section of view. Usability was dramatically improved, because the entired view represented to users will

[1] https://mail.google.com/

Figure 3.2: Web 2.0 architecture



not be refreshed and the front-end is able to process and render data in its own intension, which means more flexible control of the consumption of data.

SINGLE PAGE APP

With the evolution of web technologies and promotion of these technologies in modern browswers by brower vendors, a new web development model called SPA was proposed and caught the developers' eye. The back-end is no more responsible for rendering and view controling, it only take charge of providing services for the front-end.

The structure in figure 3.3 shows that the client side has the full control of view rendering and data consumption after data acquisition through web services which is released by back-end with promised protocol. All rendering tasks was stripped off from the server side, which means that the server side achieves more efficiency and concentrate more on the core bussiness logics.

But more responsibility in front-end means more complexity. How to reduce the complexity and increace the maintainability of a front-end project becomes a significant problem. Developers come up with an new envolved variant of SPA as demonstrated in figure 3.4.

In general, the architecture is componentized and layered into template, controller and model. Each component is isolated and has its own view as well as correlated logics. Front-end frameworks like EmberJS, AngularJS, ReactJS are providing such a approach and development pattern for developers to build modern web apps. With this approach, a giant and complex front-end app is broken up into fine grained components, therefore, components are easy to reuse if the components are well abstrated in a proper way. In addition, the maintaince of each component is also effortless.

Figure 3.3: SPA architecture

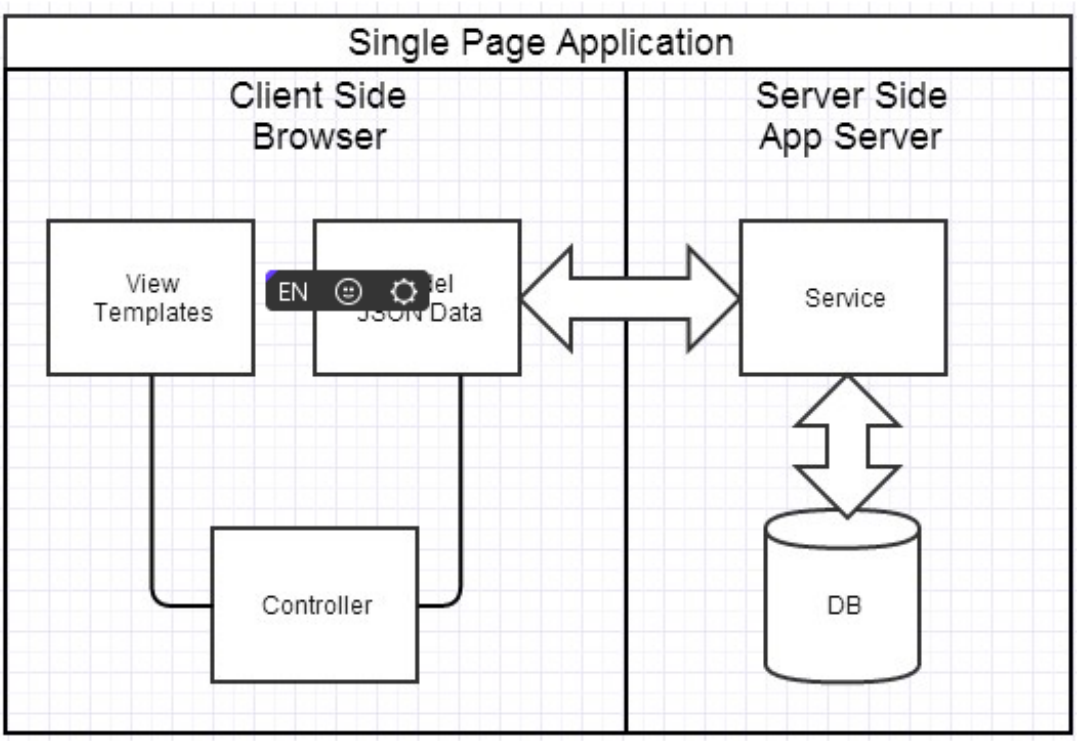
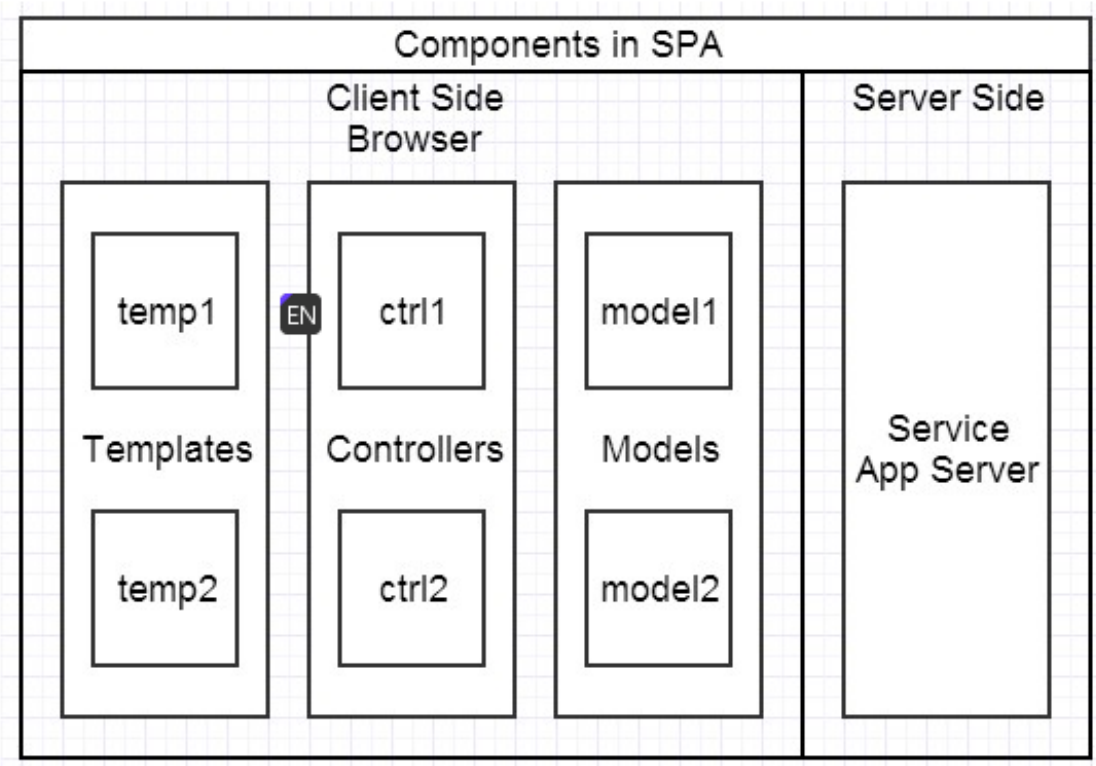


Figure 3.4: Components in SPA



TRADE-OFF

In summarize, a single-page app has a lot of benefits:

1. Rational seperation of works from front-end and back-end: client takes charge of view rendering and data representation, as well as slight data processing if needed; the server focus on providing services of the core logics, persistance of data, and also computational tasks.
2. High interactivity and user experience in client side: asynchronous data fetching and view rendering implies no more need of hard reloading the page which user is viewing and the current states of the page could also be preserved.
3. Efficiency in server side: rendering tasks are stripped off from server side.
4. Ubiquity: with the seperation of services provided by server side, not only the web browser but also other clients in other platforms such as Android, iOS apps are able to access and comsume the services.

But SPA also has its deficiencies:

1. SEO unfriendly: because the page are not directly rendered by server side, and the web crawlers are not able to run JavaScript codes like a browser does, the site could not be crawled properly under normal circumstances. So if SEO results really matter for the app, SPA is obviously not the best choice.
2. Excessive http connections: all the data is acquired from different services through diversed APIs, thus multiple HTTP connections are established and performed parallelly, whose initial time of connections for partial data could be much more than a single connection in the traditional way. So it's highly needed to merge the services and find a balance between data model complexity and time consumption.

3.1.2 RESTFUL INTERFACE

3.1.3 CONTINUOUS INTEGRATION

3.1.4 OPERATING-SYSTEM-LEVEL VIRTUALIZATION

3.2 GRAPHICS ON THE WEB

3.3 REAL-TIME COMMUNICATION

3.4 EFFICIENT CLIENT SIDE

3.5 EFFICIENT SERVER SIDE

3.6 CONCLUSION

TBD

4 CONCEPTION

!!! In this chapter a conception of the discuss system including both client and server side will be described. In the first section, all the essential requirements are presented. After that, the general conception or workflow of the application will be proposed. The more concrete details and definitions of the conception will also be outlined. In addition, the focal points behind the conception and the proper solutions of the difficulties will be illustrated.

4.1 AIMS AND OBJECTIVES

Before starting to concept the graphical discuss system, it's necessary to analyse requirements and objectives behind the origin motivation in the first place. It should be defined at first, what kind of functionalities should be achieved and how the system behaves.

4.1.1 BASIC FUNCTIONALITIES

As a graphical discuss system for the educational purpose, the system should contain basic functionalities on the prototype of a forum which could be organized by classes. So class management, question management and answer management are the three essential parameters to be designed at the start.

COURSE MANAGEMENT

Each question should have a certain domain of its content, so the questions are organized by classes initially. The features of course management should be:

1. Create Course: The user who is identified as a tutor is able to create course and maintain the course he created. While creating the course, the tutor can define the name of the course and upload an image as a background of the course for better recognization. In addition, concret description of the course could also be added to the description area.
2. Search Course: After a course is created, a corresponding unique identifier code for the course will be generated at the same time. The students are able to find the course through the identifier code.
3. Favor Course: If a student is interest to a certain course, he is capable to add the course to his favor list so that it's easy to find and access the course he liked later.

Figure 4.1: placeholder

Figure 4.2: placeholder

Figure 4.3: placeholder

QUESTION MANAGEMENT

1. Submit/Edit/Withdraw Question: The student who has confusion with the teaching content can submit his own question with detailed description in a certain course. The user is also permitted to edit the question if he want to add more precise informations or modify the unclarity he made to the question. Withdrawing of his own question is also possible, but only when there're no contributes made to the question.

Figure 4.4: placeholder

2. Upvote/Downvote Question: An assessment of a question is decisive for building a better community with high-quality contents. So the user is able to upvote or downvote of a question and determine if the question is helpful for other members in the community or not.

Figure 4.5: placeholder

3. Favor Question: If the student consider the question as a helpful and useful content and want to review this question in the future, he can favor the question and locate it in a certain list.

4. Accept Answer: The owner of the question has the right to accept the most useful answer in his opion which will be shown up at the top of the answer list.

Figure 4.6: placeholder

Figure 4.7: placeholder

ANSWER MANAGEMENT

1. Submit/Modify/Remove Answer: User who has experence with the question can submit his answer to the question. After the submission, the modification or removal of the user's own question is possible.

Figure 4.8: placeholder

2. Upvote/Downvote Answer: As mentioned above in section of question functionality, a similar idea of assessment should also be applied to answers. Answer with higher vote will be listed at first.

Figure 4.9: placeholder

3. Quote Answer: Answers are able to be quoted so that the user can supplement informations on the top of original post or point out the deficiency of the contribute.

4.1.2 HIGH INTERATIVITY

Building with the basic functionalities is far not enough. To fit the system for educational purpose and improve the interactivity for arousing enthusiasm of students, a drawing tool and realtime functionality should be intergrated into the system.

Figure 4.10: placeholder

DRAWING TOOL

Normally, some of the thoughts can't be simply expressed by textual description, so a drawing tool should be designed to enables the user to compose not only text but also different components such like rectangle, circle, line and so on, which helps the user to express his question more precisely. The ideal drawing tool should have following features:

1. Drawing Diverse Components: Not only text but also diverse components could be drawn while posting a contribution. Styling of a component such as size tuning, color changing is also the essential, which will helps emphasize the important part the user expressing.

Figure 4.11: placeholder

2. Drawing History: During drawing, the user might make mistakes or change mind after placing a component or text. So a history list of drawing actions bundled with undo and redo functionalities will dramatically improve the usability of drawing process.

Figure 4.12: placeholder

REALTIME

How to ease the approach of content acquisition and improve the interactivity for arousing enthusiasm of students, is also a key point while designing the discuss system. So two major realtime functionalities are featured as follow:

1. Realtime Question List: Without requesting the question list initiatively, all new questions posted by other users will be pushed to user automatically. The user doesn't have to concern himself with acquisition of the new content anymore.

Figure 4.13: placeholder

2. Realtime Answer Ordering: Without refreshing the page, the answers will be re-ordered as new vote action is triggered.

4.2 GENERAL CONCEPT

Before the whole conception of the system, a general conceptual architecture of the system should be defined initially. In order to help understanding how the system works, the primary data flow between different domains will also be described.

Figure 4.14: placeholder

4.2.1 ARCHITECTURE

According to the analysis result of Single-Page-App in chapter 3, and considering the demand on high interactivity and ubiquity as well as scalability in the graphical discuss system, leveraging SPA architecture will benefit a lot and accelerate the implementation of the system.

In general, the entire system will be divided into two parts: namely client and server-side. Each side is basically full independent to the other and has its own responsibility.

1. Client: The client is totally responsible for initial view rendering and view re-rendering as the view model changes.
2. Server: The server is in charge of core business logic, data processing, data persistence and also provides the client interfaces for data acquisition.

Figure 4.15: placeholder

General architecture of the system is described in figure 4.15, the only bridge between the client and server-side is data transmission service. Complete separation of both sides will also accelerate the development flow in implementation phase. Once the protocol of

data transmission services is fully confirmed and defined, development of each side is able to performed parallelly. Furthermore, technical choices on both sides are more flexible. Both sides are able to apply the technologies which fit them most without coupling to each other, the only thing they should obey is to follow the protocol of data transmission.

4.2.2 COMMUNICATION

As mentioned above in section 4.2.1, data communication is the only coupling factor in the general architecture. In this system, there exist two different type of protocols: standard HTTP using REST architecture and WebSocket with persistent connection. Each data transmission protocol has its own responsibility and usage scenario.

1. **HTTP with REST architecture**: Data which is requested initiatively is transferred over HTTP. The HTTP connection will be closed as soon as the data is successfully transferred.

2. **WebSocket with persistent connection**: Reactive data with realtime need is transferred over WebSocket. After the persistent connection is established, client are able to receive the data at the first moment as the state of data is updated.

Figure 4.16: placeholder

As figure 4.16 shows, in case data for view model is acquired from server despite transferred over HTTP or WebSocket, the views are re-rendered. Comparing with HTTP, the specialty of data acquisition over WebSocket is: a listener for specific resource with unified process of data processing and automatic view re-rendering will be created.

4.3 DATA

In this section, data modelling of the system including definitions of data domain, data fields for each domain, and relation between domains will be performed in the first place. In the following subsection, APIs corresponding to related operations on data models will be assigned.

4.3.1 GENERAL DATA MODEL

DATA DOMAIN

In general, data in the system could be divided into 4 primary domains:

1. User: personal information as well as user identifier for accessing the system.
2. Course: a container with own course information as well as collection of questions classified to this context.
3. Question: data with information of questions submitted by users.
4. Answer: data with information of answers, also has graphical data within the data model.

Users are able to assess the contributions made by other users and mark it as useful or useless, which will affect the contributions' oder priority while rendering the views. Voters should also be aware of what kind of vote he has marked to the contribution. Therefore, additional 2 data models VoteAnswer and VoteQuestion should also be considered.

The relation between domains is illustrated in the following figure ??. Each data modal has an unique identifier, which is referenced in another data model when they have a connection.

Figure 4.17: placeholder

DATA FIELDS

More detailed definition and description of fields in each data model should be made for a better understanding of the structure as well as behaviour of a data model. Tabel ?? describes key fields of general domains.

4.3.2 RESTFUL API DEFINITIONS

As mentioned in section xxx, RESTful architecture is a an excellent technical choice for data transferring. Because of its simplicity and clear semantic description of HTTP methods comparing to other protocols such like SOAP, it will dramatically simplify and clarify our data transmission services.

Domain	Field	Description
User	email	unique identifier of an user for authentification, also as contact way to user.
	password	string for user authentication to prove identity or access approval
	username	user identifier shown to other users
	isTutor	flag which determines if the user is a student or tutor
Course	name	name of the course
	desc	description of the course
	creator	id of the user(tutor) who created this course
	code	unique code for quick search purpose which is generated automatically as the course is created
Question	title	title of the question
	content	content of the question
	course	id of the the course to which the question belongs
	vote	vote count of the question
	creator	id of the user who submitted this question
Answer	content	content of the answer
	question	id of the question for which the answer is made
	quoted	id of the original question which is quoted
	vote	vote count of the answer
	creator	id of the submitter
Vote	type	enum values of up-voting or down-voting actions
	handler	id of the hander
	question/answer	id of the question/answer to which the vote action is applied

Table 4.1: Fields for Each Data Domain

MAPPING OF HTTP METHODS TO DATA MODEL BEHAVIOUR

The data model defined above can directly map to the definition of resources in RESTful. The HTTP methods on each resource domain can also represent the data model behaviours, *User Model* is taken as an example:

Method	Operation of data model collection
GET	Query and return a specific user from user model collection.
POST	Create a new user entry and insert into user model collection.
PUT	Update a specific user in user model collection.
DELETE	Delete a specific user in user model collection.

Table 4.2: HTTP methods on User resource

In table **??**, resource entry in persistent storage can be executed with specific action while requesting resource URI through different HTTP methods. A semantic description of connection between CRUD and HTTP methods on RESTful will make the data transmission services more understandable and unified.

GENERAL RESTFUL API DEFINITIONS

Requesting a specific resource can only succeed through its URI, through which the client and server-side could connect to each other actually. Therefore, a definition of APIs which describes URI of the resources and its functional responsibility should be proposed in the first place.

1. User Authentication: the major actions of user authentication include signup, login, logout. To protect the user information, POST method which doesn't expose information via the URL, is highly recommended.

URI	Method	Description
/auth/login	POST	User login action, request with login information.
/auth/signup	POST	User signup action, request with registration information.
/auth/logout	GET	User logout action, no data submission is needed.

Table 4.3: User Auth APIs

2. Courses: acquisition of courses and new submission of a course is possible. In addition, CRUD operations on a specific course should also be achieved through a single URI with various HTTP methods.

URI	Method	Description
/courses	GET/POST	request the whole collection of courses; create course with data submitted
/courses/:courseId	GET/PUT/DELETE	request, modify, remove specific entry of course

Table 4.4: Course Resource APIs

3. Questions: in a real sense question resource is attached to course resource. According to the best practise of RESTful API design [reference xxx], question resource

could be touched under course URI, */courses/:courseId/questions/:questionId*. But in the real world, question resource has its own collection, and *questionId* is the unique identifier, through which a specific question entry could be selected without using *courseId*. So an optimized conception is simply using */questions* as URI instead. And pass *courseId* as a query parameter while requesting collection of question entries under a specific course.

URI	Method	Description
/questions?courseId=:id	GET/POST	request the whole collection of questions belonging to a specific course; create question under a specific course
/questions/:id	GET/PUT/DELETE	request, modify, remove specific entry of question
/questions/:id/vote/:type	POST	vote actions with different vote types applied to specific question

Table 4.5: Question Resource APIs

4. **Answers**: the general API design of answer is totally same as the approach applied in question resource. A independent API for voting functionality should also be designed. And multiple possibilities of vote types could also be passed through the API.

URI	Method	Description
/answers?questionId=:id	GET/POST	request the whole collection of answers belonging to a specific question; create answer under a specific question
/answers/:id	GET/PUT/DELETE	request, modify, remove specific entry of answer
/answers/:id/vote/:type	POST	vote actions with different vote types applied to specific answer

Table 4.6: Answer Resource APIs

Once all APIs with different HTTP methods are defined, a more concrete data structure over the APIs between two sides should be promised and confirmed. By following defined APIs and promised data structure, developments on both client and server-side could be executed parallelly.

4.3.3 WEBSOCKET DEFINITIONS

As the requirements defined in section 4.1, users could be informed as new question is posted or the order of answers with rating priority changes. Basically, only two different types of listeners are needed in this case: one for listening to the new questions under a specific class and one for responsive order of answers under a specific question. With WebSocket, URI should also be defined as an identifier for the persistent connection between client and server. And different events within a connection of one URI should also be designed. Table 4.7 defines these two WebSocket specifications.

URI	Event	Response
/ws/courses/	new-question	data of new question posted by others
/ws/questions/	order-changed	data of answers in new order

Table 4.7: Answer Resource APIs

4.4 GRAPHICAL DATA CONVERSION

Graphical content is the most efficient and intuitive way to deliver the explanation of an answer to other users comparing with pure textual content. In this section, Choices for graphical technologies on the Web will be analysed in order to figure out which is the ideal and fit the graphical discussion system most. And a feasible approach of storing graphical data will be proposed. At last, a drawing tool will also be designed to offer user interfaces for drawing elements on the drawing board.

4.4.1 CANVAS OVER SVG

It seems that both Canvas and SVG are good candidates as a graphic technology for the graphic discussion system. Both of them provides native methods to render rich varieties of elements like path, circle, rectangle and so on. Which would be a better choice above the context of discussion system, should be analysed at first.

EFFICIENCY

As mentioned in section 3.2, the rendering efficiency is the primary deficiency happening to SVG.

And graphic technologies are not only simply used for rendering a static graphical content in the system. Dragging, resizing or deleting an element are the basic features of a drawing tool, which provides input of graphical content. All these features are only able to be accomplished by re-rendering the elements on the drawing board.

Considering that all contributions in the system are made with graphical contents, efficiency plays a significant role especially on mobile devices with old hardware. Choosing Canvas will give users better usability while viewing the contributions as well as using the efficient drawing tool without janky feeling.

EXTENSIBILITY

TDB

4.4.2 STORABLE AND REVERSIBLE CANVAS DATA

A focus point in the thesis is how to store the graphical content submitted by users. In the traditional way, graphical data could only be stored either in file system or in the database with encoded format, for example Base64 encoded images.

DEFICIENCY OF CANVAS - STORABLE IMAGE DATA

Canvas provides a native method called *getImageData()* to export the whole Canvas context including the size of Canvas and pixels on Canvas to an image data. The image data exported by canvas represents the underlying pixel data with the format of *Uint8ClampedArray*.

The *Uint8ClampedArray* typed array represents an array of 8-bit unsigned integers clamped to 0-255 [reference], which implies the position of the pixel as the index of array and the color of pixel as value from 0 to 255. A example is taken in figure 4.18 shows a Canvas with content and its simplified ImageData.

Figure 4.18: placeholder

Restoring with the exported ImageData is also possible in Canvas by using its native method called *putImageData()*. Basically, the concept of *putImageData()* is traversing the ImageData exported by Canvas, and re-drawing each pixel at the position according to the index in the *Uint8ClampedArray* and applying the color to the pixel based on the value from 0 to 255 stored in the array.

Natively exported result of image data could basically meet the storing demand, however, the data redundancy in the native exported format representing the properties of each pixel is still very huge. Storing such kind of data will cause high demand on storage space when plenty of graphical contributions are made in there system.

Additionally, it is expected that users are able to remove, resize and modify the elements in the canvas while quoting others' contributions. However, natively exported result of image data could only describe each pixel but not each element on the Canvas, which means modification on the elements is not possible even though the whole canvas could be reproduced with graphical content by others.

Therefore, a workable solution should be concepted and new data model describing the graphical content in Canvas should be designed to meet the demands mentioned above.

SOLUTION - OBJECTIFICATION OF ELEMENTS IN CANVAS

Even though Canvas has native methods to draw different shapes of elements, but Canvas only render them pixel by pixel, it knows nothing of the shapes that are drawn. Therefore, removal or modification of a already drawn element is not possible. In this condition, a feasible solution is to wrap the Canvas and objectify the original elements which could be stored and persisted in a stack. Furthermore, the wrapper on Canvas should also provide methods to render, modify, remove the custom elements.

The general conception of the wrapped Canvas is illustrated in figure 4.19.

The wrapped Canvas has a list of objectified elements which are visible on Canvas. Now there are new definitions for rendering, modification of an objectified element:

1. Rendering Canvas: the list of elements maintained by wrapped Canvas will be traversed and each element will be rendered by calling the native drawing method of

Figure 4.19: placeholder

Canvas. Position and style of the element in Canvas refer to the properties of its object.

2. Modification or Removal: In case the methods for modifying or removing provided by element object in wrapped Canvas are fired, the whole Canvas will be re-rendered in the same way of rendering the Canvas initially.

Now rendering means that the list of elements maintained by wrapped canvas is traversed and each elements are

DATA MODEL EXPORTED BY WRAPPED CANVAS

Since the wrapped Canvas maintains a list of objects for elements, which also contain the properties such as position, color, size and so on, so the exporting of image data is now really simple. A composition of all objectified elements' properties is already enough to describe the whole canvas. Figure 4.20 reveals the approach and data model output from wrapped canvas.

Figure 4.20: placeholder

After converting all elements in wrapped canvas, the new data model is much more efficient for storing comparing to the raw image data. The wrapped Canvas will also provides a method to restore the output data, create new objectified elements by giving the properties of the data model, and re-render the elements into original Canvas pixel by pixel. With this approach, the feature of modification on elements while quoting other contributions could also be achieved.

4.4.3 DRAWING TOOL

After the conception of a feasible wrapped canvas as base of the container for all elements, a drawing tool which provides user interfaces to draw variable shapes as well as texts should be designed in the next step.

First of all, user interfaces for selecting different drawing mode like drawing circle, rectangle or line should be designed. Buttons for toggling various drawing modes are the best choice in this case.

Because drawing elements in different shapes has distinct behaviors, so listeners for each specific drawing mode should be defined. When the button for toggling drawing mode is clicked by user and specific drawing mode is activated, the correlative listener will be initiated. Clicking events or moving events of the mouse on Canvas will be captured and processed. Meanwhile, drawing behaviors are also performed according the mouse events fired by user.

Figure 4.21: placeholder

Figure 4.21 illustrated the conception of drawing tool with user interfaces and life cycle of event listeners for each drawing mode.

4.5 REAL-TIME DEMAND

Realtime communication as mentioned in subsection 4.2.2 are used for reactive data, which requires WebSocket for establishing persistent connections to enable the bi-directional communication between client and server.

All users are able to subscribe arbitrary course for new submission of a question as well as arbitrary question for updated order of answers, and server could also push realtime data to those users who has subscribed the resource with specific identifier. However, only two WebSocket services: */ws/courses* and */ws/questions* are defined as the entry points

according to the definition in subsection 4.3.3. The approach how the server broadcast data precisely to the users who subscribe resources they require should be resolved.

A feasible solution is that the server maintains a list of user ids and resource ids subscribed by user. Afterwards, as a specific resource is updated, the server can get all users who has subscribed this resource from the maintained list using the resource's identifier.

Figure 4.22: placeholder

Figure 4.22 represents the whole workflow of establishing the connection over WebSocket. In the first place, the server start listening for requests in WebSocket protocol with specific URI. Then the user start a connection to server for subscribing the resource he requires. As soon as the connection is successfully established, the client will emit the resource id to the server side, and resource id from client will be mapped to the user id in a list maintained by the server.

After that, the server has the information of which user has subscribed which resource, and is able to emit realtime data precisely.

4.6 CONCLUSION

TBD

5 IMPLEMENTATION

In this chapter, an prototype of graphical discussion system will be created and make use of the previously developed approach as a "proof of concept". Firstly, ... the application domain will be presented and also foreshadow the prototype func- tionality. Afterwards, ... the development itself will take place, highlighting and documenting best practices in the subsequent sections.

5.1 GENERAL

On the whole, the implementation can be divided into two parts: client and server. Since they are fully separated, each part is considered and structured as a independent project. The implementation on client side is basically data fetching and template rendering, while data persistence and core business logic is implemented on the server side. For convenience, the graphical discuss system is named "Graphicuss", which stands for graphical plus discuss.

5.1.1 PLATFORM AND FRAMEWORK

To achieve a better performance of view rendering on client side running in browser, ReactJS[1] is taken as the front-end framework. Componentization, the main philosophy of ReactJS, also helps organize the views and view model logics. On the server side, ExpressJS[2] as a web framework is adopted for its efficiency and productivity of building RESTFul APIs.

Since both client and server projects are primarily implemented in JavaScript, NodeJS[3] is the single development environment for either project implementation or project management on both sides.

FILE STRUCTURE

To have a basic understanding of whole project including the server side and client side, a file structure of the project Graphicuss is listed in figure 5.1:

1. client/: independent front-end project built on top of ReactJS
2. server/: independent back-end project implemented by using ExpressJS

[1] https://facebook.github.io/react/
[2] http://expressjs.com/
[3] https://nodejs.org/

3. **dist/**: compiled back-end project integrated with compiled and compressed static view files from front-end project

4. **node_modules/**: source of referenced third party libraries

5. **package.json**: definition of third party libraries for client and server side

6. **webpack.config.json**: config of specific behaviours in automated development or building process

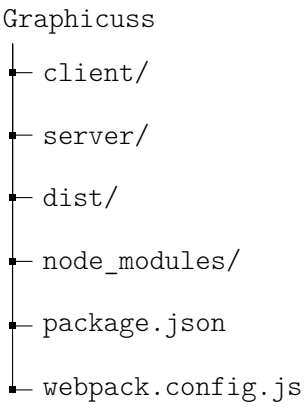


Figure 5.1: Overview of Graphicuss' file structure

MODULE MANAGEMENT

NodeJS provides native module management which is called *npm*[4]. Third party libraries, which are published in the official remote repository, can be installed conveniently by only using npm's command line. There is also a list of names of all modules and dependencies in the file *package.json*. Installing all dependencies and modules can be simply achieved by using only one command line *npm install*, which will significantly ease the setting up process of the project freshly on a new machine.

5.1.2 ARCHITECTURE

An architectural overview of Graphicuss is illustrated in figure 5.2.

While the client starts requesting a specific URL for data from the server, a HTTP connection will be established. The server program receives the HTTP request and forwards it to its router, in which rules for matching URLs have been pre-defined. By analysing headers of HTTP request, router will check if the request matches any pre-defined rules.

Not only the URL but also the parameters passed by client, for example the identifier of a resource, could also be extracted from HTTP headers. Server runs correlate business logics according to the rule of matched URL and executes operations of databases for data persistence. Afterwards, results are returned from server.

As soon as the data is successfully returned, the HTTP connection will be closed. The client processes data acquired from server, and represents it by re-rendering views. So far, a entire request over HTTP is accomplished.

[4]https://www.npmjs.com

Figure 5.2: placeholder

5.1.3 AUTOMATIZATION

To accelerate the developing as well as building process of the project, a automatization tool called *Webpack*[5] is used.

Webpack is a tool which could analyse the dependencies of the project and bundle modules with the app. In addition, it can also do tasks like compressing JavaScript codes to reduce the size of the client app, or compiling modern JavaScript as well as CSS codes to achieve the compatibility for old browsers.

AUTOMATED DEVELOPMENT PROCESS

To make the development of client app independent, it will start a dev server on its own for development purpose. However the dev server started by client app and the actual server are running on different ports. Which means that the communication between them will cause CORS problem.

CORS means, a resource makes a cross-origin HTTP request when it requests a resource from a different domain than the one which the first resource itself serves. For security reasons, browsers restrict cross-origin HTTP requests initiated from within scripts. cite!!!!!!!!!!

Through configuring the dev server started by Webpack, a proxy could established to forward request to the actual back-end server. As figure 5.3 shows, the client is able now to request APIs under the same domain, and requests will go though the dev server, after that they are forwarded to the actual server.

AUTOMATED BUILDING PROCESS

For the client app, multiple tasks are executed during the building process by using Webpack: transforming modern JavaScript code, pre-processing the modern CSS code and bundling the static files. All these tasks will significantly reduce the size of client app and also improve the compatibility of the app.

The usage of Webpack on server app is quite different. It will bundle all dependencies with the server app. After processing on both sides, the final output of the files will be

[5] https://webpack.github.io/

Figure 5.3: placeholder

extracted into the *dist* directory mentioned in 5.1, which is now ready for deploying and serving. The whole building process is represented in figure 5.4.

Figure 5.4: placeholder

5.1.4 STORAGE STRUCTURE

In section 4.3.3, data domains and fields of data domains have already been defined.

For all persistence storage of data on the server side a MongoDB[6] database is used. In figure 5.5, more concrete definition of data model table is defined. MongoDB is a non-SQL database, which uses document oriented storage and JSON style data model. That will make it easy to implement as well as scale data models. In addition, An ORM framework called Mongoose[7] is also applied to the implementation, which encapsulates the native database operations of MongoDB. With help of the ORM framework, definition of schema and query on database will be quite simple.

[6]https://www.mongodb.com/
[7]http://mongoosejs.com/

Figure 5.5: placeholder

5.2 SERVER OF GRAPHICUSS

This section gives the explanation of architectural pattern and more detailed implementation of the server side.

5.2.1 ARCHITECTURE

MVC PATTERN AND PROJECT STRUCTURE

To separate the different layers of model, view and controller, MVC pattern is used as the basic pattern of the architecture. In the model layer, all data model related concerns such as data shema definitions, data model validation as well as database operations are defined. And Controllers contain the core domain logics, process the data from model layer, and pass the result to view layer.

Since the templates are rendered on the client side, the view layer is just simply stripped. Therefore, basically the controllers response processed data to client side directly without rendering it to views. Figure 5.6 shows the overview of the server's file structure which is featured with MVC pattern.

```
server
├─ config/
│   ├─ index.js
│   └─ routes.js
├─ models/
├─ controllers/
├─ index.js
└─ ...
```

Figure 5.6: Overview of server app's file structure

1. index.js: the entry point of the whole server app. It will create a server instance and set up configurations for the server. In addition, a connection from server instance to database will be established. After all configurations are done, the server instance will start listening port and waiting for the requests from client.
2. config/index.js: config as well as constants for the server. It persists *apiConfig* for example the common prefix of API URL and version of the API. And config for database including the database URL will be defined here as well. In addition, keys for encryption are also stored in the config file.
3. config/routes.js: rules for URL matching. All URL matching rules are defined in this file. Controllers are referenced here and a dispatcher for router will be instantiated. If any request meets the defined rule, the request will be forward to a correlative controller.
4. controllers/*: controllers for processing specific requests.
5. models/*: data model definitions. Files under this directory are organized by different data domain.

ACHITECTURE OF SERVER

The figure 5.7 illustrates an overview of the server's architecture.

Figure 5.7: placeholder

5.2.2 MODEL LAYER IMPLEMENTATION

In subsection 5.1.4 an overview of the storage structure has been described. In following subsections, more concrete implementation of data model layer is explained and example codes are represented.

DATA MODEL SCHEMA

Not only the fields of each data model are defined, but also data type of each field should also be restricted. Mongoose will check the type of fields within a data model according to the definitions before a record is inserted into the database.

```
import mongoose from 'mongoose'
const userSchema = mongoose.Schema({
  username: String,
  email: {type: String, lowercase: true, trim: true, unique: true},
  password: {type: String, select: false},
  faculty: {type: String, default: ''},
  tutor: {type: Boolean, default: false},
  admin: {type: Boolean, default: false}
}, {timestamps: true});
```

Listing 5.1: Example: user schema definition within Mongoose

Code list 5.1 takes *UserSchema* definition as an example. Fields like *username*, *email*, *password* and *faculty* are defined as *String* type. Fileds like *tutor* and *admin* are using Boolean type for determining the role of a user. In addition, default value of an field could also be given if record is added without value on this field. And Mongoose also provides a set of configurations for processing the entry inserted. In the example, setting *lowercase* on *email* will transform all characters to lowercase, while setting *trim* will stripped space symbols out of a string. Mongoose will not only validate the type of data model, but also check the uniqueness of the data field in the entire database if *unique* is set to *true*.

METHODS ON MODEL LAYER

In most cases, various operations are executed to acquire data from the database, modify data and even remove data in the database. Therefore, those data fetching or data processing tasks are implemented in the data model layer, which will also achieve the goal of seperation of concern. A example of *Course* data model is taken in the following code list 5.2.

```
courseSchema.statics.list = function() {
  return this.find().sort('-createdAt').populate('creator').exec()
}
courseSchema.statics.load = function(_id) {
  return this.findById(_id).populate('creator').exec()
}
courseSchema.method.edit = function(fields) {
  return this.update(fields).exec()
}
courseSchema.method.remove = function() {
  return this.remove().exec()
}
```

Listing 5.2: Example: user schema definition within Mongoose

Other data models like *Question* and *Answer* are quite same as *Course*. Method *list()* is defined for querying all records of a collection under the data domain. In addition, sorting and referencing other model with foreign key will also be done before returning the result. *load()* method is for querying a specific entry. Methods *edit()* and *remove()* means modification or removal of an entry in database.

5.2.3 AUTHENTICATION

The goal of authentication is confirming the identity of an user and controll the access of resources by checking the privilege of an user. So an authentication system is also designed for security reasons.

JWT BASED AUTHENTICATION

JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. Reference!!!

Figure 5.8: placeholder

The basic idea of JWT based Authentication in the server side of Graphicuss is shown in figure **??**. After that the user sends request to login with its identifier and password and the the authentication is successfully verified, JWT will encode the user information with a secret key to a token.

```
1 var token = jwt.sign(userInfo, authConfig.jwtSecret)
2 response.cookie('token', token);
```

Listing 5.3: JWT encodes user information with secret key

After that the authentication is successfully verified and Cookies are written with JWT token, every request started from the client side will be sent with Cookies. All the requests will go through the *authMiddleware* and the tokens inside Cookies from requests will be decoded with the secret key. With the payload of user information which is decoded from the token, server is able to deal with resources for the specific user.

```
1 jwt.verify(token, authConfig.jwtSecret, function(err, decoded) {
2   var userInfo = decode
3 })
```

Listing 5.4: JWT decodes user information with secret key

ROLE CONTROL & ACCESS CONTROL

As defined in section 5.1.4, each user has two fields called *tutor* and *admin*. An admin has full control of all resources while a tutor is able to create a course and manage all resources under his course. Same as a normal user without any privileges, he could only maintain resources submitted by himself.

So a *if* statement will be executed before the operations on resources in order to determine the role of the user, or to verify if an user has the access to the specific resource.

5.2.4 WEBSOCKET IMPLEMENTATION

5.3 CLIENT OF GRAPHICUSS

5.4 DRAWING TOOL FOR GRAPHICUSS

5.5 DIFFICULTIES AND OPEN QUESTIONS

5.6 CONCLUSION

Conclusion!

6 EVALUATION

6.1 USABILITY

6.2 SYSTEM OVERLOAD

7 CONCLUSION AND FUTURE WORK

7.1 CONCLUSION

7.2 FUTURE WORK

LIST OF FIGURES

LIST OF TABLES