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

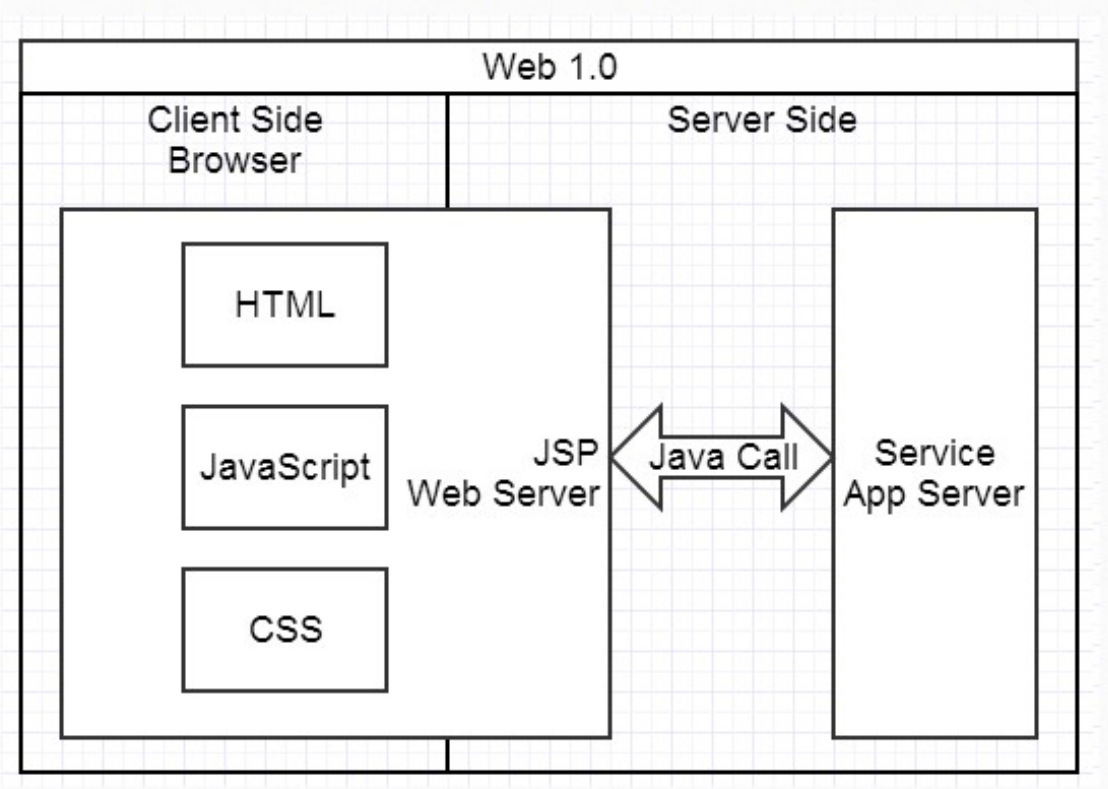
With the increasing complexity of a web app and high demand of agile development, people are always thinking about how to improve patterns of the workflow in web development as well as the architecture of a web app. To achieve better scalability, maintainability and ubiquity of a web app, the architecture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

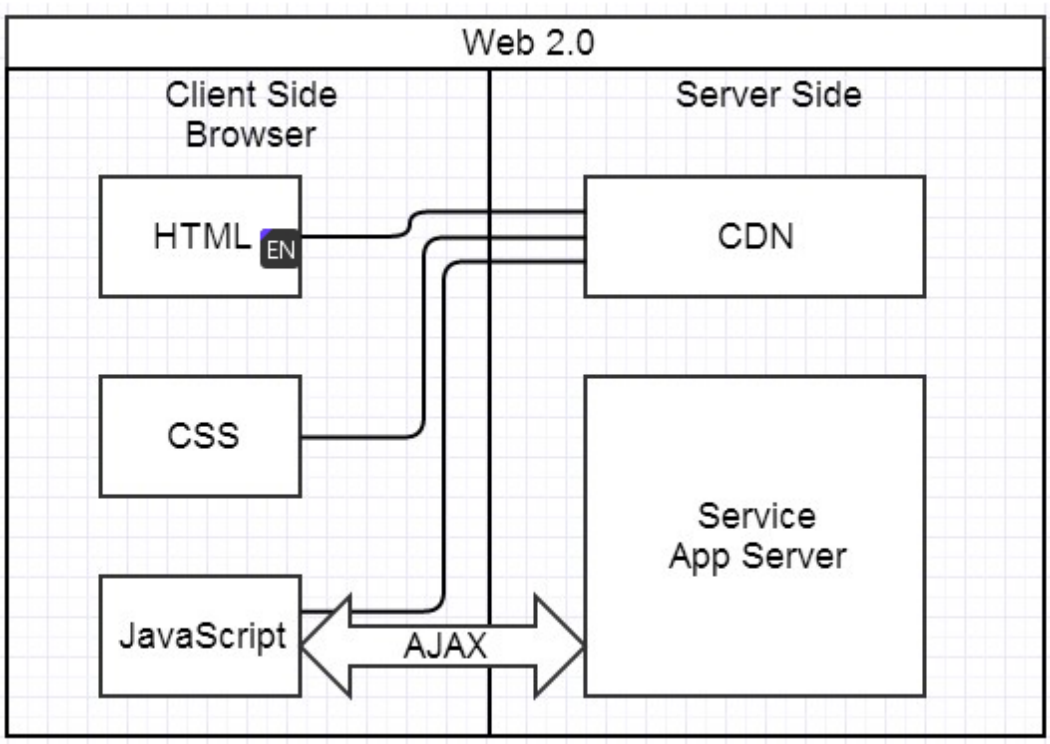
Along with the birth of Gmail[1] in 2004, which is noted for its pioneering use of Ajax, the web application started to behave more interactively. Browser began to take over the job of data fetching, processing, rendering, such a sequence of workflow which could only be done by the server side formerly.

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

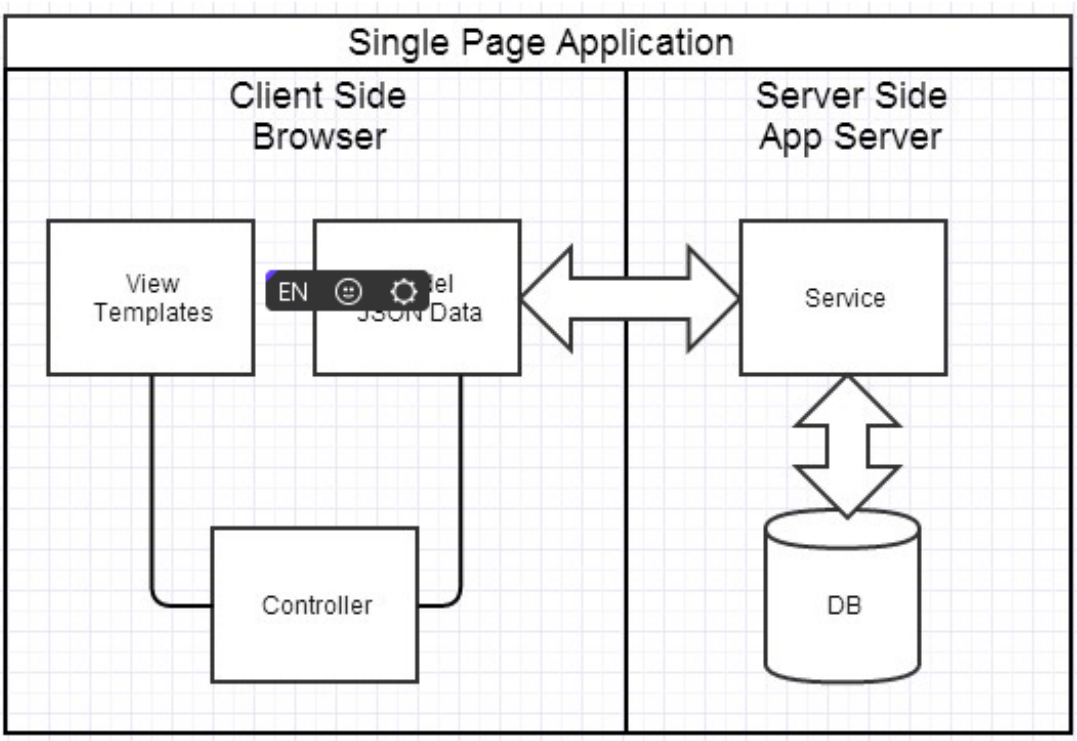
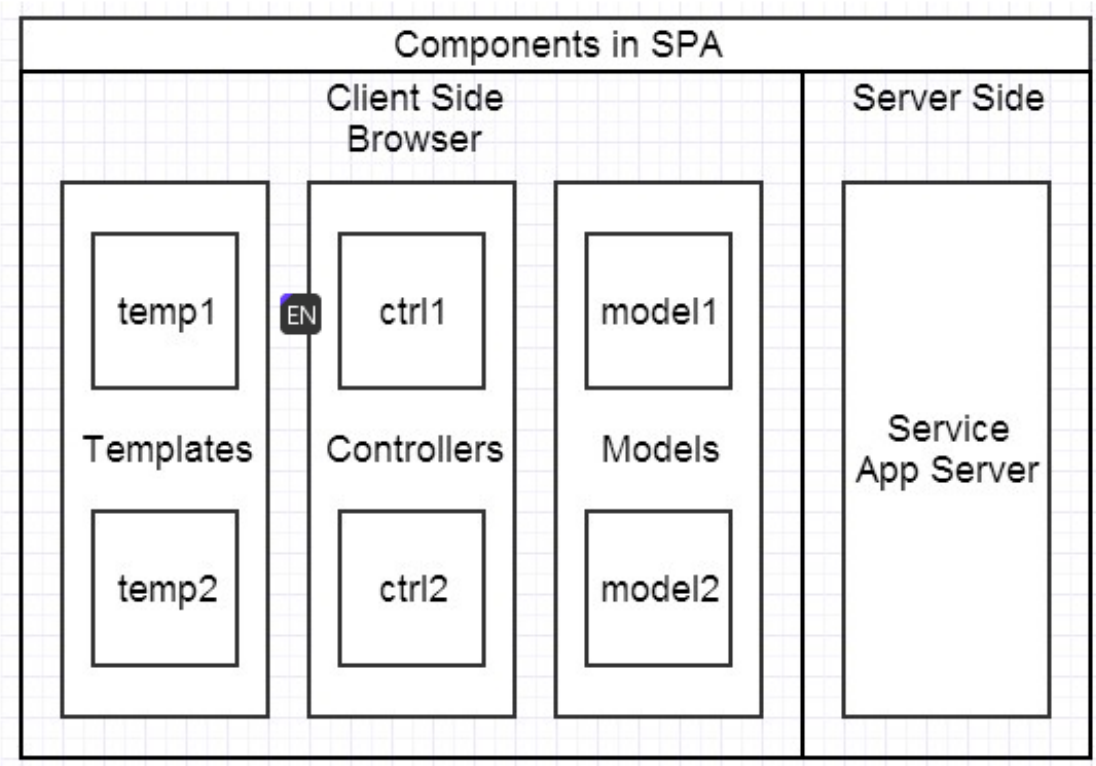


Figure 3.4: Components in SPA



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. Draw 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 template rendering and view re-render 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 XXX, 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 4.2.1, data communication is the only coupling factor in the general architecture.

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 sub section, 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.

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 4.1 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.

Figure 4.16: placeholder

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 4.3, 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

TBD

4.4 GRAPHICAL DATA CONVERSION

4.5 REAL-TIME DEMAND

4.6 CONCLUSION

TBD

5 IMPLEMENTATION

5.1 ARCHITECTURE

5.2 SERVER

5.3 CLIENT

5.4 DIFFICULTIES AND OPEN QUESTIONS

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