Relink Final Documentation

By:

Front-end: Mukesh Gande and Sahana Jain ,

Chat-service: Mingzhe Zhao, Songyu Li, Shuang Qu,

Back-end: Mingze Gao, Ranran Li, Nathan Pitchaikani

Table of Contents

Description

In many classes, the lectures are plain in the way that the form of the instruction is simply having the teacher render the course material. However, there has been plenty of researches conducted on instruction methodology and interactive classroom was upheld for better teaching quality. Therefore, our team come up with the idea to make a virtual classroom platform. In comparison to iClicker, a tool, the virtual classroom is a far more comprehensive platform. And in contrast to Piazza® and Blackboard®, this virtual classroom emphasizes on complementing real-life lecture and providing activities along with an on-going lecture.

Therefore, we aim at creating instant classroom session and providing chatting and quiz features. We set up two user types, instructor and students. Instructors can create classroom session (successful creation returns a valid room code). Students will join in the classroom via room code. Once join in the classroom, students can send message to ask question, discuss and communicate with classmates. They also have options to chat anonymously. We believe this option ensures participation with less pressure. Another significant part of the website is the quiz section. Instructors can create quiz that are visible by students in the classroom. After polling students' answers, the system generates statistics of multiple-choice questions to the instructor. The quiz creation, however, is not limited to multiple-choice format, it can be text form, too.

Process

We utilized the methodology of Extreme Programming "short as XP" (Beck 1). Extreme Programming is a software process methodology that is used to enhance software quality and responsiveness to a changing customer. It aims at modularizing units and urging frequent iterations to integrate units. XP is extremely useful because it splits up the development into short cycles, and builds bottom up. It starts with the most critical functionality, building up to additional and peripheral features. This avoids programming of features until they are actually needed. This improves productivity and often guarantees that so far the project works. It also enables the client to check the progress frequently. If any change is proposed, the team can quickly adjust the code base, without wasting too much of the progress. In practice, XP is realized as pair programming, in doing extensive code reviews, and unit test.

The properties and advantages of XP fit with the condition of our team. We have time and ability constraints. Employing XP helps us in making feasible progress and meet project requirements. We set up bi-week iterations and plan for each iteration. We meet every week to ensure the goal for the iteration will be met by the end. At each iteration meeting, we evaluate our progress. If we face any issue, or decide to change our plan, we update plan for future iterations. In practice, we cut some of the irrelevant features and focused on messaging and quiz features in the classroom platform. This rendered clearer theme. And we enriched the two parts with more manipulations. As a result, we used Extreme Programming to produce a workable web application. Under its help, we built the software bottom up and refined the proposal to a plan with clear user stories and concrete theme.

Requirement/Specification

Specifically, we expect our platform to enable account creation and login. User type determines permissions. Only instructors can create classroom. And the instructor(s) will become the admin of the classroom, in the way that they can post quizzes and end the class session.

Iteration 1 (Jan 23 - Feb 6)

actual	estimated	story description
4 units	2 units	write a proposal
3 units	4 units	form the team
4 units	2 units	prepare for Iteration 1 meeting

When creating an account, a user goes to the Relink homepage. He/She can click on the create account button. This will then redirect him to another page in which he/she will enter information like email address, password and name. He/She will also specify the status of this account, i.e., an instructor or a student. After successful account creation, the user can login as an instructor or a student.

Iteration 2 (Feb 6 - Feb 20)

actual	estimated	story description
5 units	4 units	Student can create student account
5 units	4 units	Student can log into platform
5 units	4 units	Instructor can create instructor account
6 units	6 units	User can update their profile

If an user login as instructor, he/she can create a virtual classroom session. The backend will generate a unique Id of the classroom. This will be used to identify the classroom. Our database saves data of each room session by its Id. And the Id allows students to join in the classroom. Instructors and students can enter the Id in blank for “join classroom” to be added in the room and forwarded to the classroom page.

Iteration 3 (Feb 20 - March 6)

actual	estimated	story description
8 units	6 units	Instructor can create a virtual classroom and get the classroom code
8 units	6 units	Student can enter the virtual classroom by its code
6 units	4 units	Instructor can shut down virtual classroom

Once in the classroom session, instructors and students can send public message. This message can be question, reply, comment or announcement. The real-time chatting service will broadcast this message to all the participants in the virtual classroom, including the instructors. Other

students who know the answer can choose to answer the question, or to discuss the question. So they can do online chatting classroom-wide. For the same contents, students can choose to send message anonymously. We make this feature to protect privacy and reduce participation pressure for students to post what they actually care about or have confusion on. The “anonymous” option is a button that can be clicked by the side of the “submit” button. Once the option is selected, message sender will appear as “Student Anonymous”.

Iteration 4 (March 6 - March 20)

actual	estimated	story description
15 units	12 units	Instructor can use quiz templates to create quiz used in class
5 units	4 units	Instructor can post real time topic for discussing
6 units	4 units	Student can send response to Instructor's quiz

So far, we have introduced a partial structure of the website platform. Essentially, the classroom is a chatting board if the instructor haven’t posted quiz (finished feature) or anything else (to be expected). In the virtual classroom, instructor can post quiz at anytime. Once the instructor decides to make a quiz. He/She clicks the “create quiz” button and get to a web page form.

Iteration 5 (March 20 - April 3)

actual	estimated	story description
5 units	4 units	Student can send their thought on instructor's topic
5 units	6 units	Instructor can view student's response to topics
5 units	6 units	Student can log out to a classroom and stop receiving discussion messages

The form acts as a template where instructor can fill in with questions and choices (if a multiple-choice or true-false question). Instructor can add more choices to a question and add questions of a quiz to be posted. Once the instructor finish entering information, he/she can click “submit” to post the quiz visible to students in the classroom.

Iteration 6 (April 3 - April 17)

actual	estimated	story description
7 units	6 units	Instructor can insert quiz into virtual classroom
6 units	6 units	Instructor can send response to instructor's quiz
6 units	6 units	Instructor get statistics about student's quiz responses

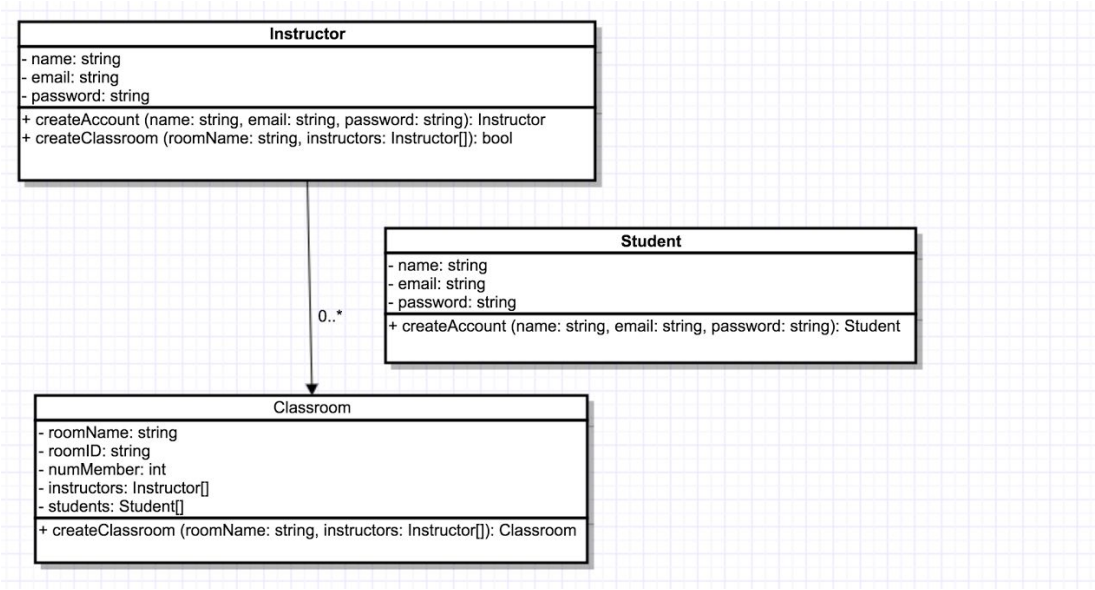
The backend will collect students’ response and generate statistics. The quiz feature is relatively independent from the classroom session. Instructors can create quizzes before the classroom session on live and save it. Students can save quizzes for later review, so the quizzes stay valid even after the classroom session ends.

There were many different frameworks that we used on this project. For the back-end we used Nodejs. Likewise for the Chat-service we used Socket.io. The last main part of the project, the front-end was done using React. We stored data in a MongoDB database. And the entire project based environment is Unix based.

Architecture/Design

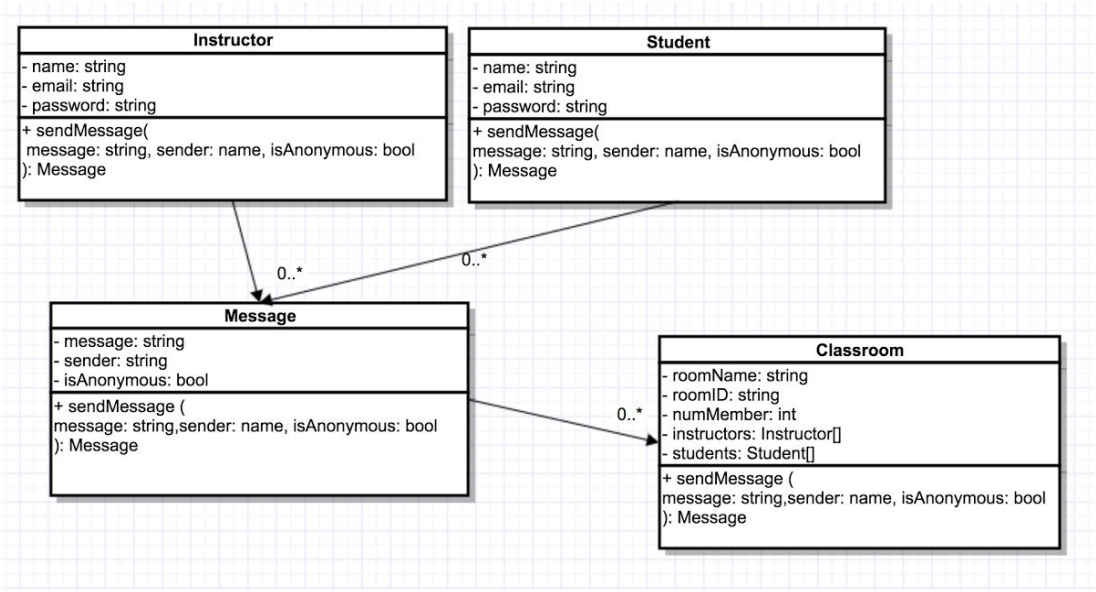
We divide this project in 3 major parts: frontend, backend and chat-service. The python to implement backend, that also incorporates Node.js and Socket.io. The chat-service heavily uses socket.io to receive messages from individual users and broadcast to the classroom. The front end uses React to design the webpage.

The structure of the classes are plotted as below. We have the classes of the users, instructor and student. We also define the class of the message and quiz, that inherits the class of questions.



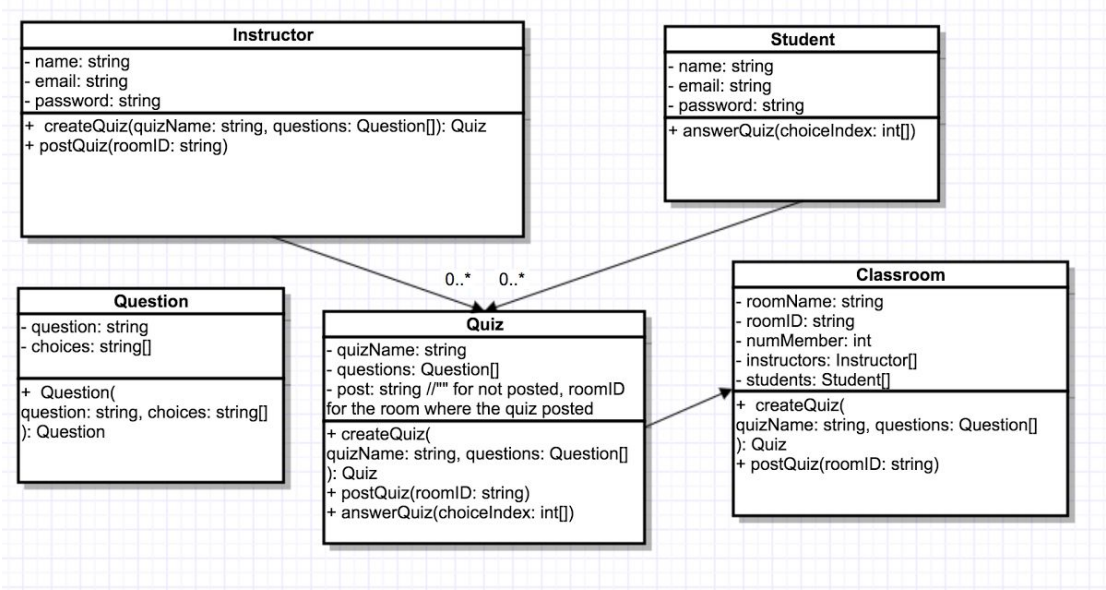
Class Diagram for Account Creation Feature

[Diagram Note: This class diagram indicates the constructor of user (Student or Instructor) and the constructor of the classroom. It shows the parameters needed and the private functions that create them. The permission is also clearly restricted to that only instructor can create classroom.]



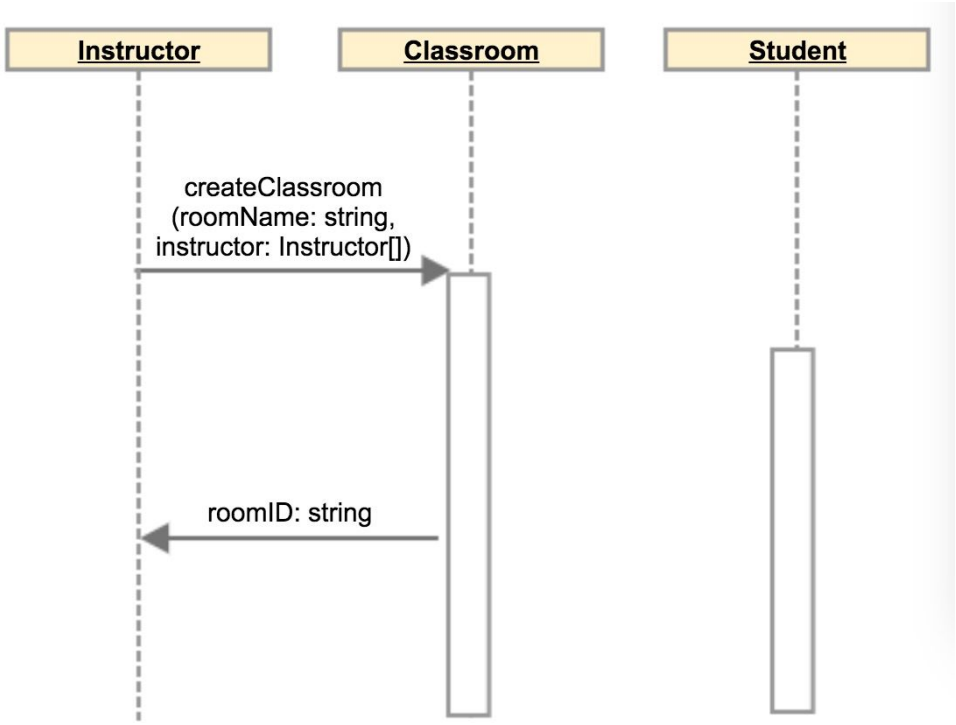
Class Diagram for Message Feature

[Diagram Note: This class diagram indicates the constructor of Message. It shows the parameters needed and the private functions that create them. The permissions are defined clearly that instructor and student can send Message. They have same permissions.]



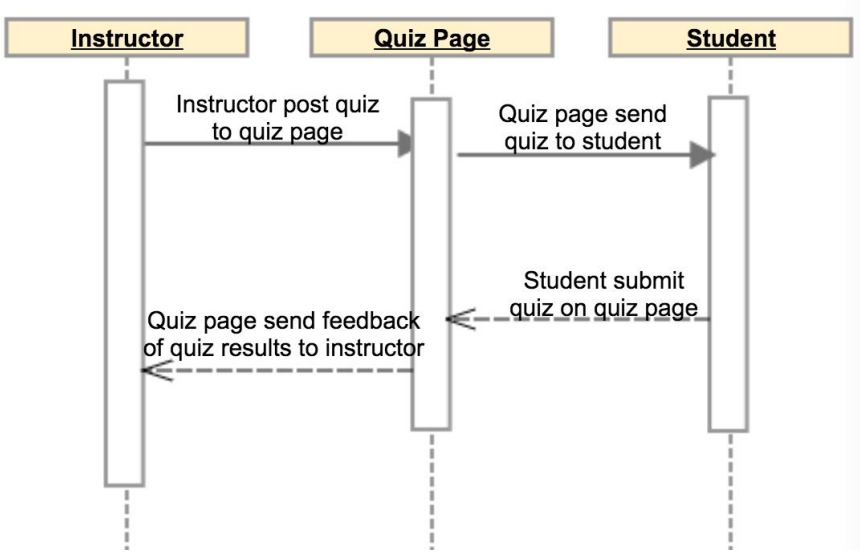
Class Diagram for Quiz Feature

[Diagram Note: This class diagram indicates the constructor of Question class and Quiz classes. It shows the parameters needed and the private functions that create them. The permission is clearly restricted to that only instructor can create Quiz. When an instructor creates a Quiz with any number of Questions, the individual question is constructed as a Question object.]



Sequence Diagram for Instructor Create Classroom

[Diagram Note: This sequence diagram shows the sequence to create a classroom. Instructor sends request to create a classroom. The Classroom class processes this request to create a classroom. If the creation is successful, it returns the roomID to instructor.]



Sequence Diagram for Quiz Activity

[Diagram Note: This sequence diagram shows the sequence of an entire quiz activity. Instructor sends request to create a quiz. The website enters the quiz page for instructor to enter quiz content. Once the instructor submits the quiz, this quiz page creates a quiz object. When instructor chooses to send it to the class, the quiz pops to students in the classroom. When students finish the quiz and submit, they get feedback.]

Reflection/Lesson learned

Nathan Pitchaikani (pitchai2):

I learned a lot from this project. I learned the basics of Django, which was more difficult than I initially thought it was. I also learned a lot about planning and time management. I feel I could have planned a bit better so I could accomplish more work. I spent a lot of time trying to get Azure to work however, once I got it working it crashed the next day and when I tried to re-config it; it didn't work. Because of this a lot of time spent went to waste. I should have moved on to a different hosting method earlier like Amazon Web Services. This would have saved me more time and I would have been able to get more work done.

Mukesh Gande (mgande2):

For this project, we choose React as the frontend framework. Since I had never worked with it before, I learned everything from the basics to advanced uses. This also thought me a valuable lesson about working with new technology. To build a sustainable code base you should focus on learning how to accomplish some functionality correctly rather than simply making it work. For example, I created an interface to create quizzes based on assumptions which were incorrect. I spent many hours debugging and eventually rewriting the entire interface. This situation could have been avoided had I simply spent the a few hours mastering the basics. Even though this was a setback, I felt that this lesson was important and worth learning.

Ranran Li (rli17):

I learned new skills and practiced Extreme programming process a lot. Before starting this project, I don't know Django at all. I learned and successfully wrote a backend for this project with my partner. Our team performed pair programming throughout the semester. Pair programming is a good technique and it really increased my efficiency on writing code. In addition, I learned how to work together with different groups. As a member of the backend team, I need to communicate with the front end team for change in demand and debugging. Therefore, I improved my communication skills as well.

Shuang Qu (squ7):

I think the software engineering II teaches me in principle and in practice about software development process. The concept of extreme programming is not complex, but in practice it makes crucial effect. The bi-week meet with TA urges us to be pertinent with plan. And I really learned about the uncertainty with doing project that we may encounter ill behaviors or bugs of the program. Look back, the final shape of the project deviates from the proposal. It is partially due to the ambiguity of the original proposal, and that our concentration on major figures cuts the less relevant ones. After all, from the first-hand experience, I learn to make better proposal and design valid user stories. The implementation part also teaches me about development programming.

Mingzhe Zhao (mzhao19):

This group project teaches me how the work grows exponentially when the team becomes larger and larger. The overhead in communication and negative work done is immeasurable when the team is not well coordinated, especially when the extreme programming process is not strictly followed. The software engineering process we learned from class make a lot more sense after the team work experience.

Sahana Jain(sjain46):

This group project taught me how even when people work separately on different parts in a group project there are a lot of parts that must be worked on together in the end to get all the different aspects to mesh together well. The pairs seperated the work and each pair worked on frontend, backend, and chat service but it was very important for us to touch base frequently to make sure that everything was working well together. I also learned a lot about new technology that I was uncomfortable with at first such as React and Socket I.O.

Songyu Li (sli111):

I explored many different areas in Software Engineering during this project, and the most valuable experience I got is how to maintain and develop a project from scratch. There are certainly a lot of things to consider if you want your project to be able to grow big. Extensibility, security and coordination between pairs are all important aspects a qualified engineer should comprehend. I also learned how to develop and deploy Node.js web application, and how to design a correct architecture with separate frontend and backend. These experiences together with topics I learned during lecture, such as Quality Assurance, Design and User Experiences, could be very useful in my future career.

Mingze Gao(mgao16):

I personally learned a lot through the development of our project. We practices pair programming throughout the whole development process and find it very effectively in resolving bugs and increasing productivity. I discover the significance of proper documentation and consistent API design, we would save a lot of time if we put more efforts on them before hand. I also learned some front end framework through coordinate with front end team and chat service team. I definitely value Software Engineering process more after development of this large project.

Appendix

Code Documentation

Backend:

Accounts/apps.py

```
Class AccountConfig(AppConfig):
```

Attributes:

```
name
```

Accounts/models.py

Classes:

```
Class VirtualClassroom(models.Model)
```

Attributes:

```
Date (models.DateTimeField)
Name (models.CharField)
instructorId (models.IntergetField)
```

Methods:

```
__str__(self)
    Returns Name + instructorId
```

Accounts/views.py

Methods:

```
index(request)
    Returns HttpResponse
register_view(request)
    Returns HttpResponse
login_view(request)
    Returns HttpResponse
logout_view(request)
    Returns HttpResponse
```

delete_user(request)
Returns HttpResponse
insert_room_to_mongo(room, instructor_id)
Returns None
create_classroom(request)
Returns HttpResponse or HttpResponseServerError
join_room_view(request, room_id)
Returns HttpResponse or HttpResponseServerError
send_message(request)
Returns HttpResponse or HttpResponseServerError
ensure_dir(file_path)
Returns None
create_quiz(request)
Returns HttpResponse or HttpResponseServerError
send_quiz(request)
Returns HttpResponse or HttpResponseServerError
posts_quiz(request)
Returns HttpResponse
list_all_quiz(request)
Returns HttpResponse
post_topic(request)
Returns HttpResponseServerError

Polls/apps.py

Classes:

PollsConfig(AppConfig):

Attributes:

Name

Polls/view.py

Methods:

index(request):

Returns HttpRepsonse

Chat_Service

database/room_apis.js

Methods:

existUserInRoom(data,cb)
Returns None
joinRoom(data,cb)
Returns None
leaveRoom(data,cb)
Returns None

database/room_data.js

Methods:

```
add_room(room_name, socketid)
    Returns None
join_room(room_name, socketid)
    Returns None
check_room(room_name)
    Returns None
```

utility/data_generator.js

Methods:

```
callback(err,res)
    Returns None
```

utility/utils.js

Methods:

```
resMsg(status,data)
    Returns status, data
```

App.js

Methods:

```
allowCrossDomain(req, res, next)
    Returns None
dbJoinRoom(duser, drid, cb)
    Returns None          dbCheckRoom(data, cb)
    Return None
onError(error)
    Return None
onListening()
    Return None
```

Frontend:

Components/AnswerInput.jsx

Classes:

```
class AnswerInput
```

Attributes:

```
questionCount
setAnswerValue
```

Methods:

```
SetAnswerValue(event)
    Returns: None
Render()
    Returns: HTML
```

Components/LabelInputs.jsx

Classes:

class LabeldInput

Methods:

render ()

Return: HTML

Components/LandingPage.jsx

Classes:

class LandingPage

Attributes:

navigate

Methods:

navigate(dst)

Returns: this.props.router.push(dst)

render()

Return: HTML

Components/LoadingStore.jsx

Classes:

class LoadingStore

Attributes:

State

Methods:

componentWillMount()

Return: None

render()

Return: HTML

Components/NavButton.jsx

Classes:

class NavButton

Attributes:

navigate

Methods:

navigate(dst)

Returns: this.props.router.push(dst)

render()

Return: HTML

Components/Quiz.jsx

Classes:

class Quiz

Attributes:

state

unsubscribe

Methods:

setQuizname(event)

Return: None
addQuestion()
Return: None
removeQuestion()
Return: None
saveQuiz()
Return: None
sumbitQuiz()
Return: None
componentWillUnmount()
Return: None
render()
Return: None

Components/QuizQuestionTemplate.jsx

Classes:

class QuizQuestionTemplate

Attributes:

addAnswer
removeAnswer
setValue
questionCount
Currstate

Methods:

componentWillUnmount()
Return: None
addAnswer()
Return: None
removeAnswer()
Return: None
setValue(event)
Return: None
render()
Return: HTML

Reducers/quiz.js

Public Methods:

questionHandler(state, action)
Return: state or assign object or answer

Scenes/CreateQuiz.jsx

Classes:

class createQuiz

Methods:

render()

Return: HTML

Scene/Home.jsx

Classes:

class Home

Attributes:

State
Navigate
Logout
leaveRoom
unsubscribe

Methods:

componentWillUnmount()
Return: None
navigate(dst)
Return: None
logout()
Return: None
leaveRoom()
Return: None
render()
Return: HTML

Scenes/Instructor.jsx

Classes:

class AddClass

Attributes:

createClass
createQuiz
postQuiz
onSumbit
setValue
navigate
close
proceed

Methods:

createClass(event)
Return None
close()
Return None
proceed()
Return None
createQuiz(event)
Return None
postQuiz(event)

```
                    Return None
              onSumbit(event)
                    Return None
              setValue(event)
                    Return None
              navigate(dst)
                    Return None
              render()
                    Return HTML
```

Scenes/Login.jsx

Public Method:

```
login(name, isInstructor)
      Return: (type,username,isInstructor)
```

Classes:

```
class Login
```

Attributes:

```
state
onSumbit
setValue
```

Methods:

```
onSubmit(event)
      Return None
setValue(event)
      Return None
render()
      Return HTML
```

Scenes/Quiz.jsx

Classes:

```
class createQuiz
```

Methods:

```
render()
      Return HTML
```

Scenes/Register.jsx

Classes:

```
class Register
```

Attributes:

```
state
onSumbit
updateInstructorState
setValue
```

Methods:

```
onSumbit(event)
      Return None
```

updateInstructorState(bool)
Return None
setValue(event)
Return None
render()
Return None

Scenes/Room.jsx

Classes:

class Room

Attributes:

storeState

Methods:

componentDidMount()
Return None
onSumbit(event)
Return None
exitRoom()
Return None
setValue(event)
Return None
setAnonymous(event)
Return None
postQuiz(event)
Return None
render()
Return HTML

Scenes/Student.jsx

Classes:

class Student

Attributes:

state
joinRoom
setValue

Methods:

joinRoom(event)
Return None
setValue(event)
Return None
render()
Return HTML

API Documenation

Back-end API

configuration:
python manage.py makemigrations
python manage.py migrate
python manage.py runserver

Home Page
accounts/index

GET
NO Parameters required

Register
accounts/register

POST: {
    username = "username"
    password = "password"
    lastname = "lastname"
    firstname = "firstname"
    isInstructor = "True" or "False" //default is False
}

Login
accounts/login

POST: {
username = "username"
password = "password"
}

Code: 200
Content: "Teacher login" or "Student login"

Log Out
accounts/logout

POST: {}

will redirect to accounts/index
accounts/delete_user
POST: {
email = 'email'
}

Join Room
accounts/classroom/id

GET request

Success:
Code: 200
Content: "find classroom: " + id
Fail:
Code: 500
Content: ""

Create Room
accounts/newroom
POST request to create new room

POST: {}

Success:
Code: 200
Content: "id"

Fail:
Code: 500
Content: ""
If the user is not logged in
Will be redirect to login page

Send Message
accounts/message

POST request to send a message to backend for authentication and forwarding to chat service{
"message": ...,
"room_id": ...,
}
Response{
"status": "200 ok" or "500 error",
"data": {
... //some detailed information
}
}

Create Quiz
accounts/createquiz

POST request to send a quiz to backend for authentication{
"questions": [
{
"question": "Some question",
"answers": ["Answer 1", "Answer 2", "Anwser 3"]
},
{
"question": "Some question",
"answers": ["Answer 1", "Answer 2", "Anwser 3"]
}
],

"quizname": "somename",

"answers": [0,1]
}

Response{
"status": "200 ok" or "500 error",
"data": quizid
}

Post Quiz
accounts/postquiz
POST request to send quizid to backend for authentication{

"quizname": ...
"instructor_id": ...
}
Response{
"status": "200 ok" or "500 error",
"data": [quizs, answer] (json list)
}

Send Quiz
accounts/sendquiz
POST request to send quizid to backend for authentication and then call chatservice{
"quizname": ...
"room_id": ...
}

Response{
"status": "200 ok" or "500 error",
"data": quiz content
}

Get all Quiz
accounts/listquiz
Get request to get all quiz names of an instructor

Response{
"status": 200 "ok or 500 "error",
"data": ['quiz_name_one', 'quiz_name_two' ...]
}

Chat Service RESTful API

Send message
POST sock/send
request{
"msg": ...,
"user": ...,
"room_id": ...
}
response{

"status": "200 ok" or "500 error",
"data": {
... //some detailed information
}
}

Send quiz

POST sock/sendQuiz

request{
"user": ..,
"room_id": ...,
"quiz_name": quiz file name
}

response{
"status":
"data": {
... // some detailed info
}
}

Chatroom

GET sock/room

response{
"status" : "jjblowd",
"data" : [all the rooms]
}

POST sock/createRoom

request{
"room_id": ...shoud be something meaningful and unique
"room_name": name, "" by default
}
response{
"status": "200 ok or 500 error",
"data": {
... // some detailed info
}
}

SOCKET

Join Room

frontend sends

emit('join', {room_id: 'some_id', user: 'username'});

server response

emit("error", {data: 'room_id does not exist'});

or

emit("ok", {data: 'joined room_id'});

Receive Message

backend sends

('message', {room_id})

forward message

emit('message', {'message': message, 'user': user})

forward quiz message

emit('commands', {type: 'quiz', name: 'quiz_name'})

MongoDB

room collection{

```
  "room_name": " ",
  "room_id": " ", //must be unique, otherwise too complicated to resolve conflicts
  //format: time+user_id, time accurate to millisecond
  "room_user": [
    {"user_id": " "},
    ...// contains all users in this room
  ]
}
```

experiental ones

post question{

```
  "message_msg": " ",
  "message_id": value
  "message_vote": 0
  "room_id": value
  "user_id": value
}
```

vote question{
    "message_id": value
    ... "message_vote": ++
}

Works Cited

Kent Beck and Cynthia Andres. “Extreme Programming Explained”,2nd Edition, 2005, pp 1-15.