CS546 Group 9: Data ReadMe

Hangyu Wang

Stevens Insitute of Technology

Introduction

This is a document for your to check the functions I provided.

Here, I will only indicate the mainly functions, including the input, output and errors I throw out. Of course, I will list the potential problems here. If any of you thinks these problems should be taken into consideration, please tell me and I will update my work. At the same time, if my output structures do not meet your requirement or I missed some functions you need, please tell me. Don't randomly modify the data folder without informing me.

1 DATA

The config and dataCHK will not be shown here.

1.1 candidates.js

All functions relating to candidates is listed below:

1.1.1 function 1:

registar(name, password): name should be name of candidate; password should be password of this candidate.

output: newUser = { % this is same as what stored in the db
_id: newId, %generated by mongodb
name: name, %candidate name given
password: password %candidate password given }

DB modification:
I removed the attribute **quizRecords**: [], since this is not convenient for updating.

errors:
(1) name: must be non-empty string. **potential problem: Should us allow duplicated name?**
(2) password: must be non-empty string. (NOT consider hashed yet. Coming soon!). **potential problem: more restriction should be considered or not? like length 8-12 and must includes num, letter and symbol etc.**
(3) info insertion: report if the data cannot be added to the collection.

1.1.2 function 2:

login(name, password): name should be name of candidate; password should be password of this candidate.

output: User = { % the information of this candidate.
_id: id,

name: name,
password: password }

errors:
(1) name: must be non-empty string. **potential problem: Should us allow duplicated name?**
(2) password: must be non-empty string. (NOT consider hashed yet. Coming soon!). **potential problem: more restriction should be considered or not? like length 8-12 and must includes num, letter and symbol etc.**
(3) match: if the name and password cannot match any instance in collection, report wrong User name or Password.

1.1.3 function 3:

infoUpdate(id, OldPassword, NewPassword): id should be id of candidate; OldPassword should be the former password of this candidate; NewPassword should be the new password of this candidate.
output: User = { % the information of this candidate.
_id: id,
name: name,
password: password % the new password. }

errors:
(1) id: If no such id, throw error.
(2) OldPassword and NewPassword: must be non-empty string. (NOT consider hashed yet. Coming soon!). **potential problem: more restriction should be considered or not? like length 8-12 and must includes num, letter and symbol etc.**

1.1.4 function 4:

getById(id): the id of a candidate. can be ObjectID or string. It's compatible for both type. The output is same as the other output. If no such id, throw error.

1.2 creators.js

All functions relating to creators is listed below:

1.2.1 function 1:

registar(name, password): name should be name of creator; password should be password of this creator.
output: newUser = { % this is same as what stored in the db
_id: newId, %generated by mongodb
name: name, %creator name given
password: password %creator password given }

DB modification:
I removed the attribute **quizQuestions**: [], since this is not necessary. We have enough information in the question collection.

errors:
(1) name: must be non-empty string. **potential problem: Should us allow duplicated name?**
(2) password: must be non-empty string. (NOT consider hashed yet. Coming soon!). **potential problem: more restriction should be considered or not? like length 8-12 and must includes num, letter and symbol etc.**
(3) info insertion: report if the data cannot be added to the collection.

1.2.2 function 2:

login(name, password): name should be name of creator; password should be password of this creator.
output: User = { % the information of this creator.
_id: id,
name: name,
password: password }

errors:
(1) name: must be non-empty string. **potential problem: Should us allow duplicated name?**
(2) password: must be non-empty string. (NOT consider hashed yet. Coming soon!). **potential problem: more restriction should be considered or not? like length 8-12 and must includes num, letter and symbol etc.**
(3) match: if the name and password cannot match any instance in collection, report wrong User name or Password.

1.2.3 function 3:

infoUpdate(id, OldPassword, NewPassword): id should be id of creator; OldPassword should be the former password of this creator; NewPassword should be the new password of this creator.
output: User = { % the information of this creator.
_id: id,
name: name,
password: password % the new password. }

errors:
(1) id: If no such id, throw error.
(2) OldPassword and NewPassword: must be non-empty string. (NOT consider hashed yet. Coming soon!). **potential problem: more restriction should be considered or not? like length 8-12 and must includes num, letter and symbol etc.**

1.2.4 function 4:

getById(id): the id of a creator. can be ObjectID or string. It's compatible for both type. The output is same as the other output. If no such id, throw error.

1.3 questions.js

All functions relating to questions is listed below:

1.3.1 function 1:

createQuestion(creator, content, answers, options): creator is the id of the creator can be string id or ObjectID; content should be non empty string; answers should be non empty array and each element should be valid string(not empty or only space); options are same format as answers input. And the total elements of answer and options are 4-6.
output: newQuestion = { % all information about the new question.
_id: newId, % generated by mongodbs
creator: creatorID, % the id of the question creator. This is stored as ObjectID.
content: content, % same as the input of content
answers: answers, % same as the input of answers
options: options % same as the input of options };

errors:
(1) creator: creator should exist.
(2) content: should not be a non-empty string.
(3) answers, options: should input non-empty array; each element should be valid(not empty or only space inside); total element number of them should be 4-6.
(4) **potential problem: should us allow duplicate question? exact same content and options.**

1.3.2 function 2:

viewQuestions(creator): creator is the id of the creator can be string id or ObjectID.
output: an array of all question created by this creator. Each question's format is shown as below:
Question = {
_id: id,
creator: creatorID,
content: content,
answers: answers,
options: options};

errors:
There should be a creator existing, else throw.

1.3.3 function 3:

getById(id): id is the id of a question. can be ObjectID or string. It's compatible for both type. The output format is same as the output of createQuestion. If no such id, throw error.

1.3.4 function 4:

updateQuestion(id, NewContent, NewAnswers, NewOptions): id is the id of a question; NewContent's requirement is same as the content in function createQuestion; NewAnswers's requirement is same as the answers in function createQuestion; NewOptions's requirement is same as the options in function createQuestion. NewContent, NewAnswers, NewOptions should not be null or undefined. When a creator is updating, we should not let them update without providing any information. So, we need to post all original information of this question on the page. Then, when we capture the information on page, we actually can capture all parts.
Question = {
_id: id,
creator: creatorID,
content: content, % the new content
answers: answers, % the new answers
options: options % the new options};

1.3.5 function 5:

deleteQuestion(id): id is the id of a question. can be ObjectID or string. It's compatible for both type. The output format is same as the output of createQuestion. If no such id, throw error.

1.3.6 function 6:

SearchByField(creator, field): creator is the id of the creator, can be string id or ObjectID; field is the keyword you offered to search in the content of the question. Here, the field can be empty (empty string or undefined), otherwise non empty string. Once empty, then all questions created by this creator will be shown.
output: an array of all question created by this creator with the keyword. Each question's format is shown as below:

Question = {
_id: id,
creator: creatorID,
content: content,
answers: answers,
options: options};

1.4 quizzes.js

Main functions relating to quizzes is listed below(actually more functions are created to support these three function. If you need any of them, please inform me.):

1.4.1 function 1:

genQuiz(id, field): creator is the id of the candidate, can be string id or ObjectID; field is the keyword you offered to search in the content of the questions. Here, the field can be empty (empty string or undefined), otherwise non empty string. Once empty, then question will be randomly picked from the whole question collection. Every time to generate a quiz, there will be 5 questions be shown. If not enough, then the limited question(s) will be listed.

output: it is an object. Q_id is the id of this quiz; Questions is the question(s) generated.

Quiz = {
Q_id: QUIZ._id,
Questions: outArr % outArr is an array containing all questions for candidate. And each element in this array is shown below.};
Question = {
_id: id,
creator: creatorID,
content: content,
answers: answers,
options: options};

DB modification:

(1) Here I update this part to a new collection since there will be updating problem when it treated as subdocument of the candidate.

(2) For a new collection, I give each quiz a _id for it and candidate attribute to record the candidate who took this quiz.

(3) quizName attribute: I name it according to the field given, field + 'QUIZ'.

(4) quizScore attribute: initialized as empty string.

errors:

(1) If no candidate can be found, throw an error.

(2) If outArr is empty, throw error to report that no relating question.

1.4.2 function 2:

grade(Quiz_id, Submission): Quiz_id is the id of this quiz; Submission is an array. Each element, called feedback, in this array is in the format below:

{
QuesId: QuesId, % this is the id of the corresponding question. You should provided which question it is.
answer: [] % An array containing the answers selected by candidates. Of course, if no option is selected this array is empty. }

output:
{

_id: the id of this quiz, the Quiz_id (ObjectID),
candidate: the id of the candidate (ObjectID),
quizName: name of this quiz (String),
quizScore: right#/total# (String)
}

errors:
There should be a Quiz_id existing, else throw.

1.4.3 function 3:

getAllQuiz(id): id is the id of a candidate. can be ObjectID or string. It's compatible for both type. The output format is same as the output of function grade. If no such id, throw error.