Quiz Manager Project

By Ajay Mungur

TABLE OF CONTENTS

Planning *5*

Requirements Analysis *9*

User Stories **10**

Authentication 10

Permission level – EDIT 11

Permission level – VIEW 13

Permission level – RESTRICTED 13

Decisions following analysis **14**

Settling on an API and separate front end 14

Software and dependencies choices 14

Assumptions 16

Design – Back End API *18*

Data & Database tables **18**

UML sequence diagrams for each endpoint **18**

Jwt filter 19

Quiz endpoints 20

Question endpoints 23

answer endpoints 26

Implementation – Back End API *29*

What went well **29**

Problems encountered **29**

Testing – Back End API *29*

Junit testing **29**

Manual testing **30**

login 30

Quiz endpoint 31

Question endpoint 38

Answer endpoint 45

Design – Front End *52*

mockups **52**

UML sequence diagrams **58**

login 58

quiz search 58

quiz view 60

Question view 62

Implementation – Front End *65*

What went well **65**

Problems encountered **65**

Testing – Front End *65*

Jest testing **65**

Manual testing **65**

All users 65

Users with edit permission 65

users with view permission 66

users with restriceted permission 66

Overall evaluation *66*

Areas for improvement **66**

User manual *67*

back-end setup - Windows **67**

Java 8 set up 67

Docker set up 68

back-end api setup - mac **68**

Java 8 download & set up 68

docker download & set up 69

Front-end set up windows/mac **69**

Running the app **70**

Step 1 – start up the back end API 70

step 2 – run the front end 71

home page & navigation **71**

Login **73**

Creating a quiz **74**

Quiz view page **75**

Question view page **79**

Searching **83**

View All quizzes **86**

Viewing the back-end API swagger documentation **88**

Default users **90**

permission level: edit 90

permission level: view 90

permission level: restricted 91

PLANNING

I began the project by reviewing the documentation outlining what would be required in terms of the app I would need to build and other areas of work I would need to conduct. I first planed how I would divide up my time so that I could ensure I managed my time effectively so that I could ensure I met the requirements and delivered what had been specified. The project length was set at 40 hours so I assessed that this would give me 5 days (8 hours per day). I initially thought that a good course of action would be to divide my time as follows:

Day 1 – 8hours

- Carry out requirements analysis
- Complete data designs including Entity Relationship Diagram (ERD) of the database tables
- Complete UML sequence diagrams for the back-end API
- Complete mock-ups of front-end pages
- Complete UML sequence diagrams for the front-end processes

Day 2 – 8hours

- Build Back end API
- Complete Junit tests
- Complete manual tests (would be carried out as I completed areas of the back end)

Day 3 – 8hours

- Build the Front End
- Complete manual tests (would be carried out as I completed areas of the back end)

Day 4 – 8hours

- Complete Jest tests

Day 5 – 8hours

- Document the process
- Complete user manual

After consideration I decided against this because I felt that carrying out 3 days of intense coding could lead to more mistakes due to fatigue so I revised my plan to:

Day 1

4 hours to:

- Carry out requirements analysis
- Complete data designs including Entity Relationship Diagram (ERD) of the database tables
- Complete UML sequence diagrams for the back-end API

4 hours to:

- Complete half of the work to build the API
- Carry out manual tests (would be carried out as I completed areas of the back end)

Day 2

4 hours to:

- Build the second half of the back-end API
- Complete Junit tests
- Carry out manual tests (would be carried out as I completed areas of the back end)

4 hours to

- Complete mock-ups of front-end pages
- Complete UML sequence diagrams for the front-end processes

Day 3

- Build the Front End
- Complete manual tests (would be carried out as I completed areas of the back end)

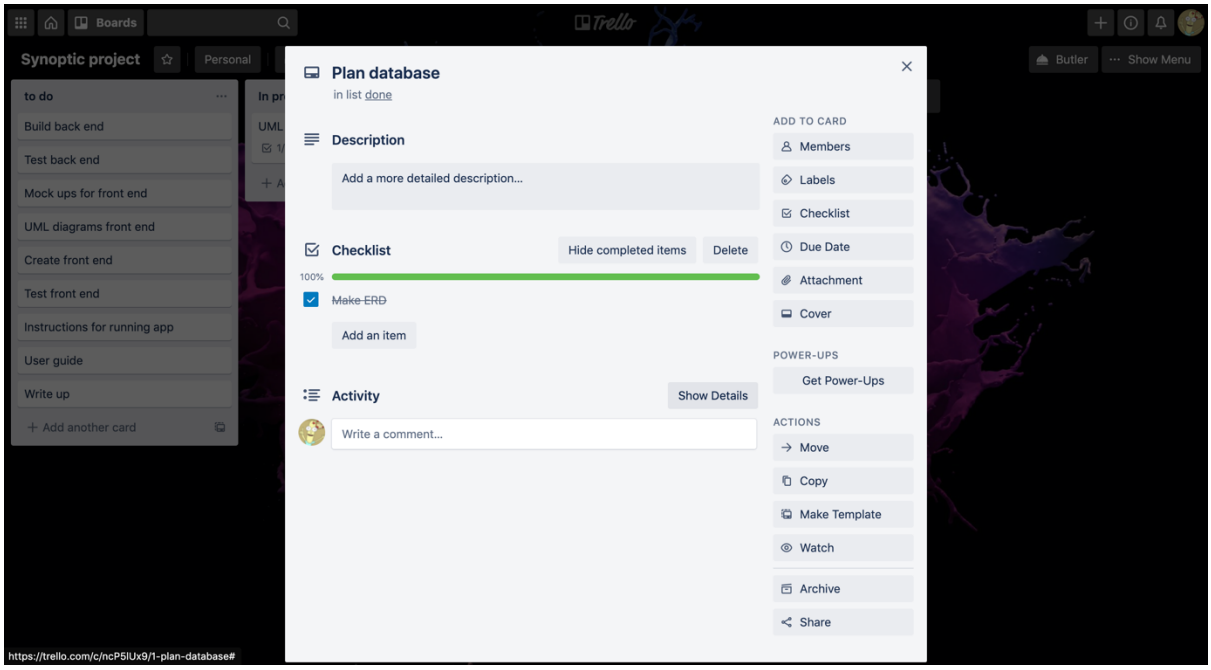
Day 4

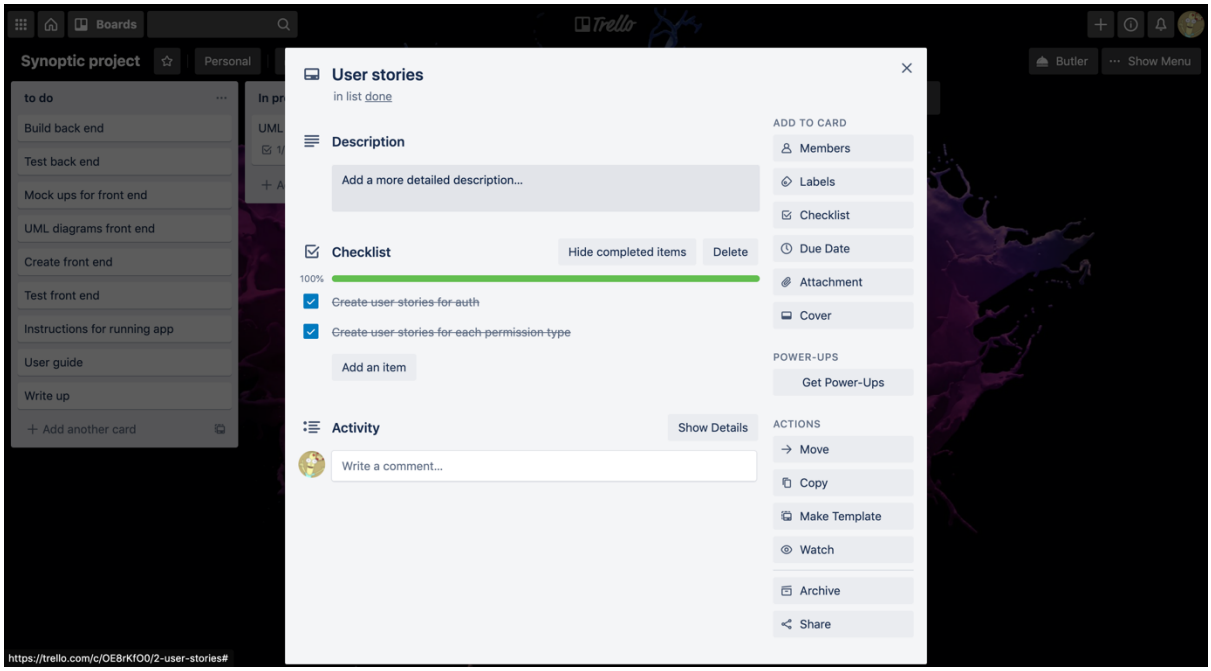
- Complete Jest tests

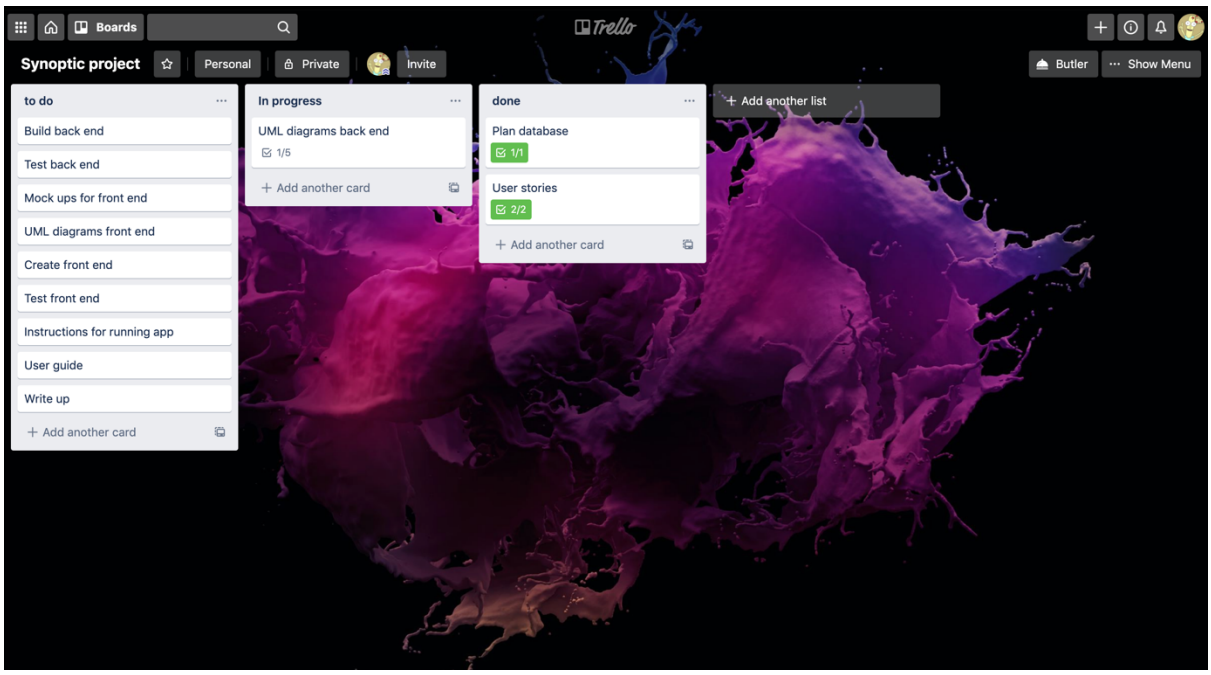
Day 5

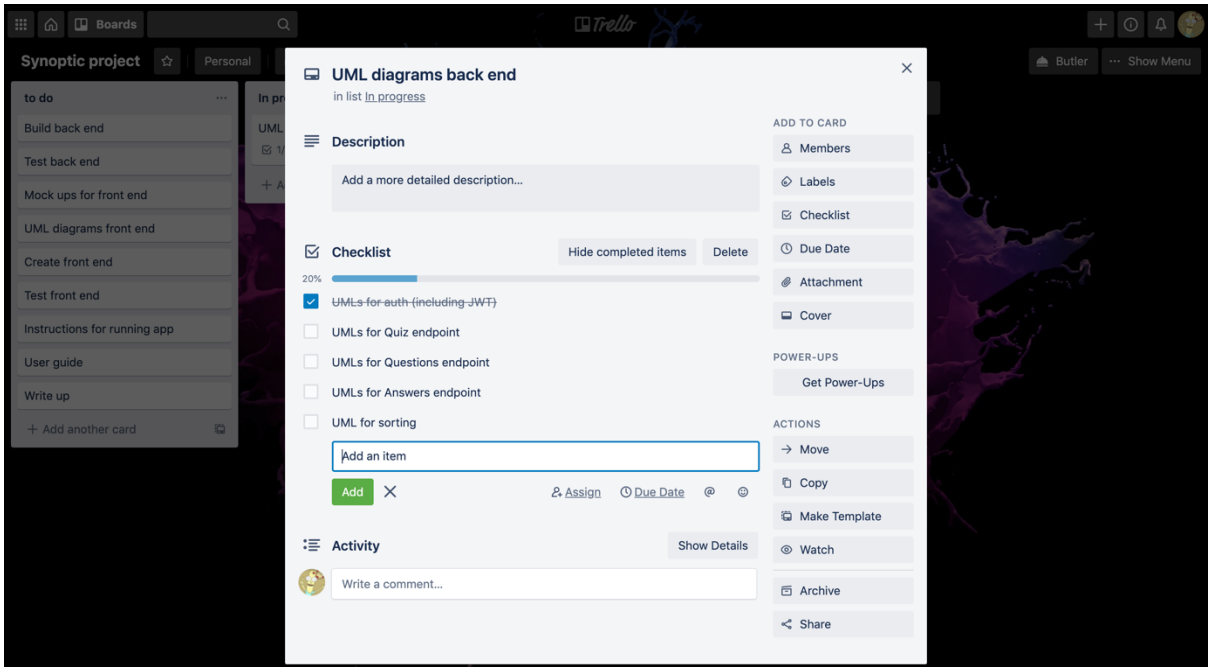
- Document the process
- Complete user manual

In terms of monitoring my work and further planning how I would carry out the work I used the Trello website to create a Kanban board with tickets so that I could monitor my progress and refer back to checklists I made within the tickets so that I could ensure I covered everything I needed to as efficiently as possible. This also ensured that I did not approach the building of the app haphazardly and e.g. trying to work on a section that relied on another section of the app which had not yet been built.









This was continually updated as I went along so that I could monitor my progress and so that I could account for any changes to my plans.

REQUIREMENTS ANALYSIS

After I had finished planning, I moved on to fully analysing the requirements. I read through the documentation and made notes of the features that had been specified.

User - Username
- password
- permission

login & logout

permission - Edit, View, Restricted

edit - add
delete
update
} question & answer

view - view Question & answer

Restricted - view question only (NOT ANSWERS)

Quiz - title

Question - Question number
- Description

N.B - Will have 3-5 answers associated with it

Answer - index value e.g. A, B, C, D etc
- description

10:00

USER STORIES

I then created the user stories to visualise the requirements into manageable stories.

AUTHENTICATION

As a User

I want my password to be hashed

So that it is stored securely

As a user

I want to be able to log in to the app

So that I can access the features relating to my account

As a user

I want a session to be created when I log in

So that I can access the features relating to my account without having to log back in

As a user

I want to be able to log out

So that my session is cleared and I am returned to the home page

PERMISSION LEVEL – EDIT

As a user with edit permission status

I want to have the option to add a quiz

So that I can create a quiz and save them for future use

As a user with edit permission status

I want to have the option to edit a quiz

So that I can update a quiz and save it future use

As a user with edit permission status

I want to have the option to delete a quiz

So that I can remove it from the database

As a user with edit permission status

When I delete a quiz, I want all associated questions and answers to be deleted

So that they are removed from the database at the same time

As a user with edit permission status

I want to have the option to add a question

So that I can add questions to quizzes

As a user with edit permission status

I want to have the option to edit a question

So that I can update either the question number or description

As a user with edit permission status

When I edit a question number, I want the question to be re-indexed

So that the questions list is automatically always in order

As a user with edit permission status

I want to have the option to delete a question

So that I can remove it from the database and, by extension, the quiz

As a user with edit permission status

I want to have the option to add answer

So that I can add answers to questions

As a user with edit permission status

I want to have the option to edit an answer

So that I can update either the answer index or the description

As a user with edit permission status

When I edit an answer index, I want the answer to be re-indexed

So that the answers list is automatically always in order

As a user with edit permission status

I want to have the option to delete an answer

So that I can remove it from the database and, by extension, the question

PERMISSION LEVEL – VIEW

As a user with view permission status

I want to be able to view questions

So that I can review them

As a user with view permission status

I want to be able to view Answers

So that I can review them

PERMISSION LEVEL – RESTRICTED

As a user with Restricted access

I should be able to view a question but not its associated answers

DECISIONS FOLLOWING ANALYSIS

SETTLING ON AN API AND SEPARATE FRONT END

Following on from the requirements analysis I had noted within the documentation that separate websites would be used to allow students to take quizzes and would manage their marks and grades. I decided that by splitting the app into 2 separate parts (a back end API and a separate front end) I would be providing the following benefits:

- The app would be easily scalable given that the data held would be easier to distribute so other services could easily make use of the data and services provided by the API endpoints e.g. the website that allowed students to take quizzes.

- The back end and front end could be worked on separately and in isolation e.g. If the front end needed to be converted to a different framework, the back end could be accessed to build this without any disruption to services provided.

I have also had experience working on an app that is separated into a back-end API and front end React app and I have seen the benefits of being to allow other apps to make use of the data and services that the API provides. Given that I had worked on this for number of months I was confident that this would be something that could be delivered within the timeframe and chose to build the.

SOFTWARE AND DEPENDENCIES CHOICES

Back-end language and Framework

My experience definitely informed and influenced my thinking in terms of what software and dependencies I wanted to use. I settled on Java as my language of choice for the API and I would use the Spring-boot framework to build the API in. Spring has a number of useful features and dependencies which I wanted to make use of such as:

- Spring initializr which I have used on the past to quickly set up a spring boot project.

- Spring security is a dependency that can be added at the initialization stage which would help with creating a login feature.

- Spring data JPA dependency which would allow me to create repositories for my classes. Each repository would make use of Hibernate which would handle the MySQL queries making it quicker and easier to do things like getting data from the database. E.g. instead of writing a MySQL query I could simply use the findById() method.

- The Lombok dependency is another dependency that I could get at the initialisation stage which would cut down on boilerplate code such as writing getter and setter methods for each class.

- There is also the Flyway database migration dependency which would allow me to add MySQL scripts which would be run when I run the app e.g. if I needed to add a table I could write a migration script in MYSQL to add the table, then if that scripts had not yet been run, the table would be added to the

database. This is very useful for keeping a log of the changes made to the database and at what stages the changes were made.

Back-end security

Prior to the project I had learned about how JSon web tokens (JWT) are used to secure API endpoints and had noted that the benefits of using them are:

- Session management is not required because the JWT will contain things like authentication, expiry time and user details such as username which can be used to authenticate each request to the API.
-
- The JWT can be used with multiple back ends so this helps to make the app much more scalable as other APIs/services are added and the app expands. As such, only one sign in would be required and could then allow access to multiple APIs
- No Cookies are required, so would be compatible with mobile apps which again helps with scaling up the app.

Due to this I chose to implement this feature to secure the API endpoints.

Database

For the database I chose MySQL as this was the database I had the most experience with. I could also then make use of the MySQL workbench app to monitor any changes to the database which would help with debugging and checking that features were working as intended. I chose to user Docker to run the database in a MySQL Docker container virtually as this seemed to be the practice at my place of work and I wanted to ensure that I conformed to industry best practices.

Back end documentation

From experience I have come across Swagger documentation which is something I have used at work to be able to quickly get a visual overview of all of the various endpoints available via an API. Swagger also allows users to try out the endpoints to see how they work and provides details of any required fields and their respective datatypes. This is very useful for developers who may want to make use of endpoints to build services so by including this the API would be very developer friendly which would help at the maintenance phase of the software development lifecycle, allowing new developers to quickly get up to speed, maintain and improve upon the API. I decided that due to this I would definitely need to implement this when building the back-end API.

Sorting

I noted that there was a requirement that the data is sorted so I planned to implement methods to carry out a sequence sort algorithm on lists of either questions or answers. This would sort them into the correct order before returning them to the front-end. I chose to go with a sequence sort because they are much more efficient at sorting smaller sets of data.

Front-end

In terms of the front end I decided to use Javascript as my language of choice and build the app using the React framework, due to my experience working with this at work. I noted that the benefits of using React were:

- I would be able to make use of my HTML skills as React uses JSX which is very similar which has helped me to get familiar with the framework.
- I would be able to build reusable components which would be able to re-render themselves independently of the main page they were on.
- I would be able to make use of Redux for data storage which would help to secure the data as opposed to storing it in session-storage which could potentially lead to unwanted access by users.
- It also has excellent support in terms of documentation online.
- The developer tools are also very useful.
- Create-React-App can be used to initialise a React app very quickly similarly to Spring Initializr; except the dependencies would need to be installed separately.

In terms of data storage, as touched on, I planned to use Redux. I thought that, in particular, this would be useful for storing a user's details and the associated JWT when a user logs in. I checked online and from the information I found, there seemed to be a consensus that storing user details would be better in redux as users would not be able to access them (even with the redux developer tool) once the app is packaged for production. In addition to this I planned to make use of the redux developer tool, which I had quite a bit of experience with as this would allow me to see what was being saved to the store at any given time. In turn this would help me to debug any problems I was having with that side of the build.

To help with the build I planned to make use of the following dependencies:

- Redux-forms, which I had learned would help to cut down on the boiler plate in terms of making action creators and reducers where multiple forms would be required.
- Axios would help with making network requests in addition to:
    - Providing support for older browsers
    - Providing automatic JSON data transformation
- React-router-dom for setting up routing and using React Links for page traversal
- Redux-thunk for calling functions prior do dispatching actions to update the redux store. This would be used to have the network requests made within action creators

ASSUMPTIONS

- Given that there was a requirement for the app to be straightforward to rebrand I assumed that the best course of action was to keep the design of the app as generic as possible so that branding and colour schemes could easily be applied.

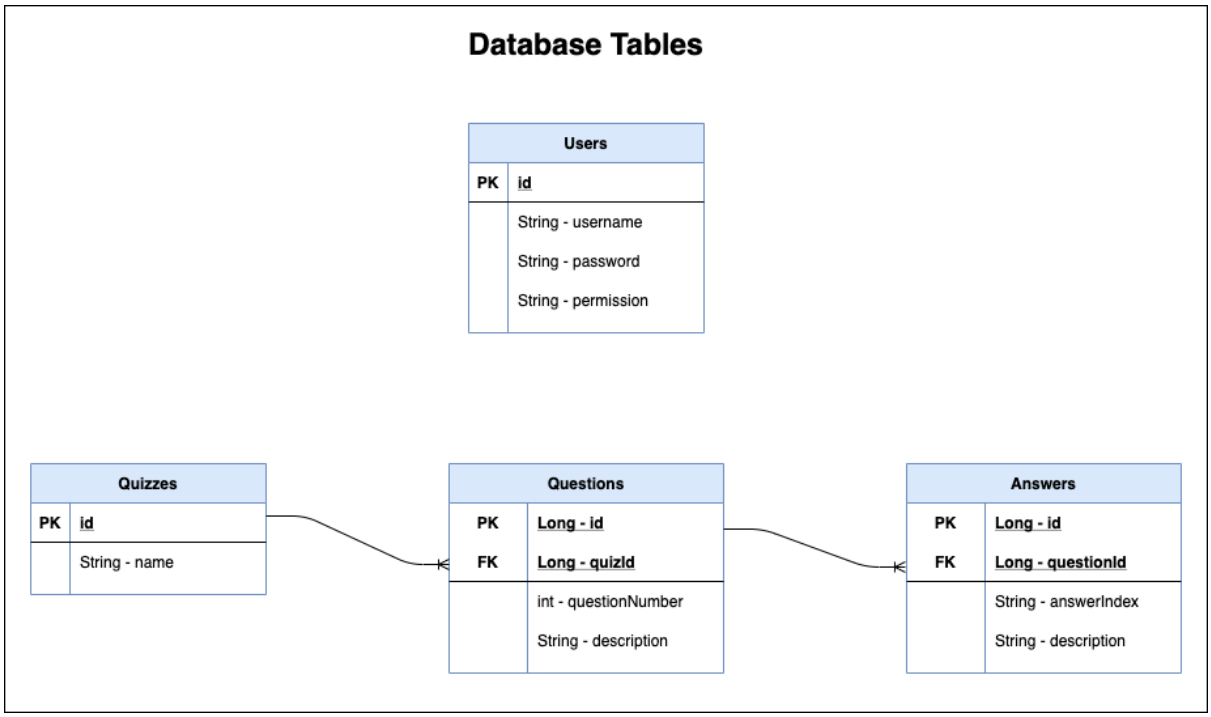
- The documentation specifies that a question will consist of between 3-5 answers. Due to this I have made the assumption that there will be no more than 5 answers for any questions so I decided to implement a feature that would.

- I have also assumed that the list of questions would not be large as I have not encountered any tests that make use of large lists of questions i.e. more than 50

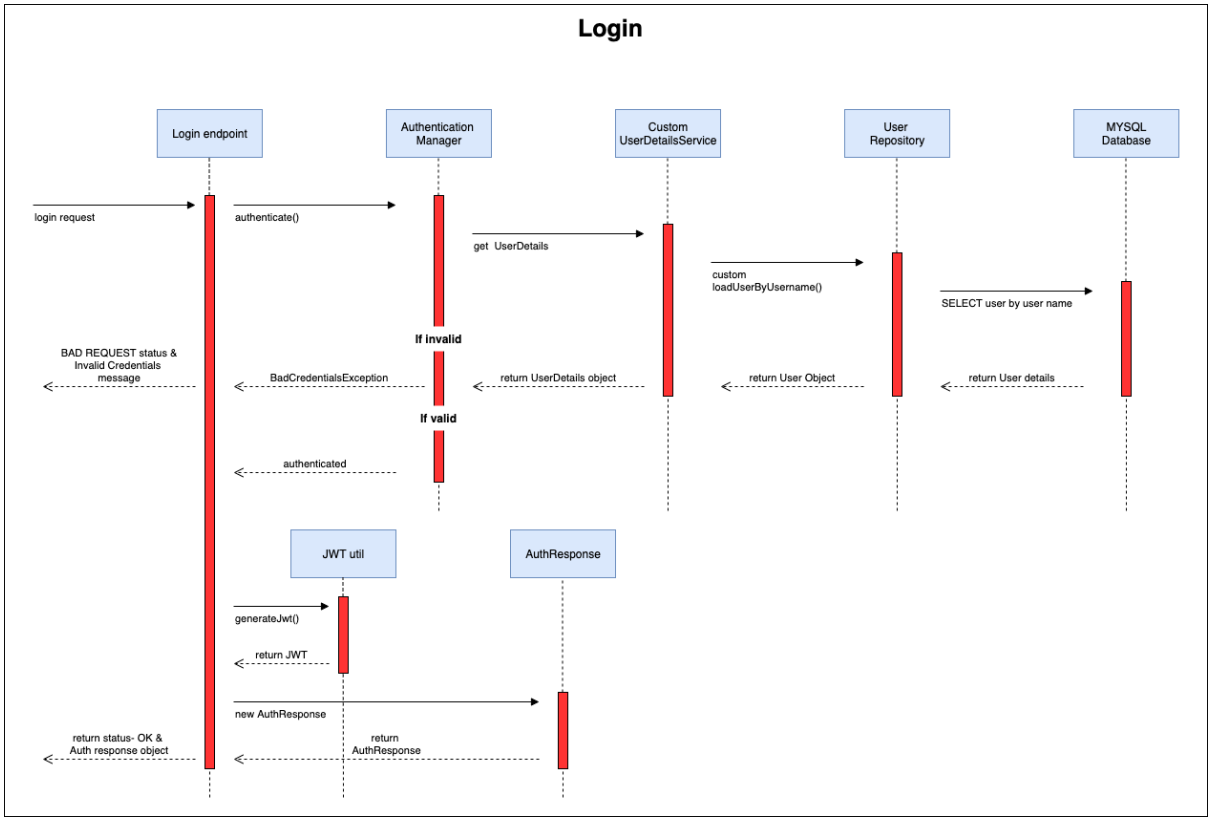
- I assumed that not quiz names would not necessarily be unique as no requirement for this was specified. In order to not restrict users, I decided to opt to provide functionality to search for a quiz name and anticipate more the possibility of receiving one or more results.

DESIGN – BACK END API

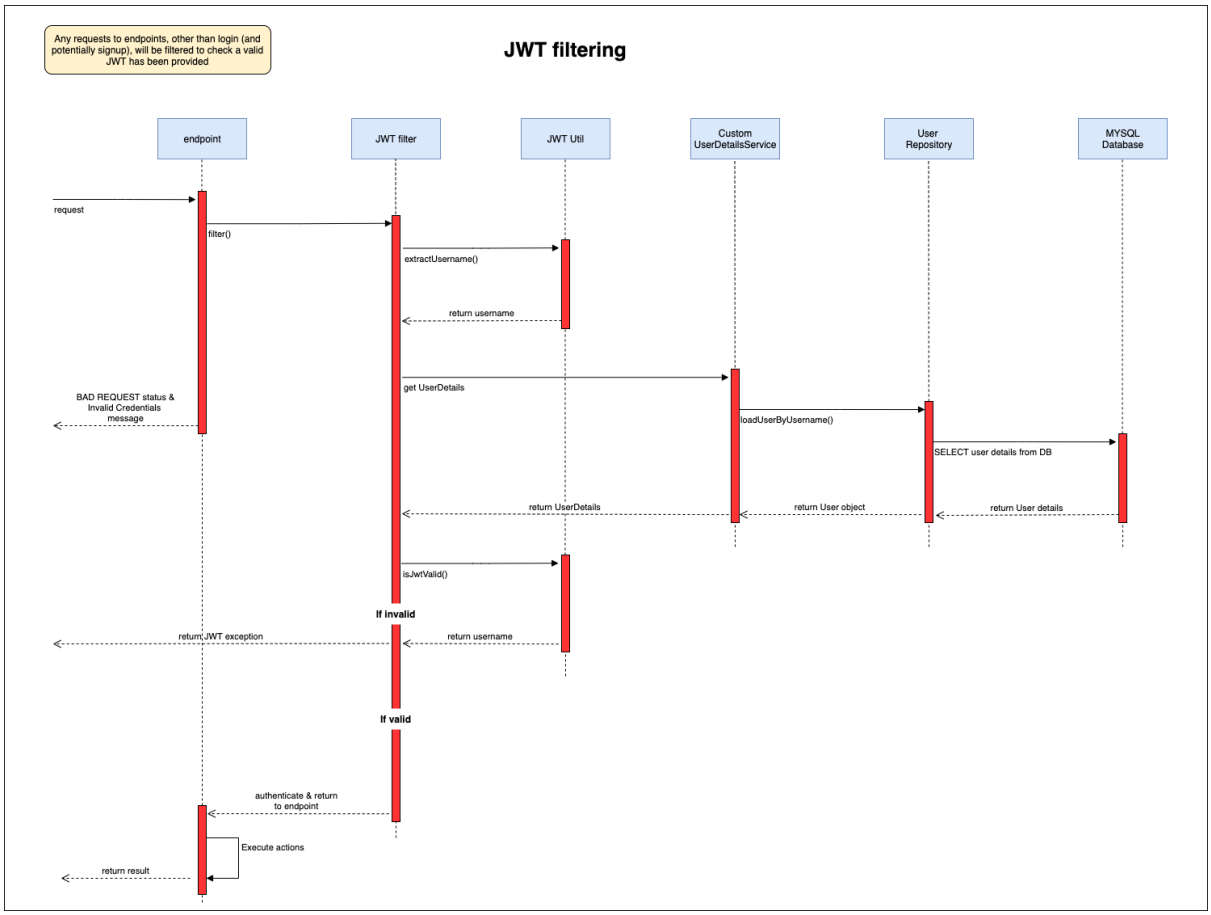
DATA & DATABASE TABLES



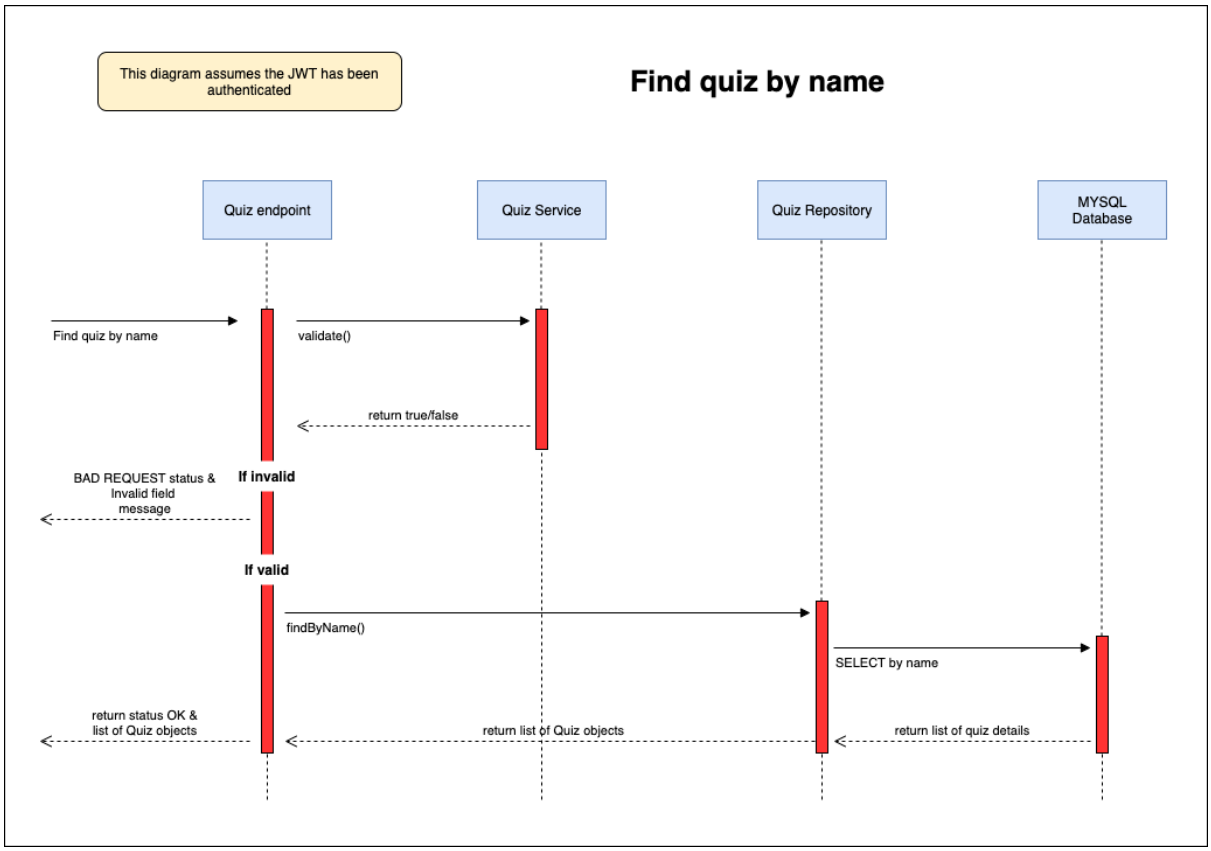
UML SEQUENCE DIAGRAMS FOR EACH ENDPOINT

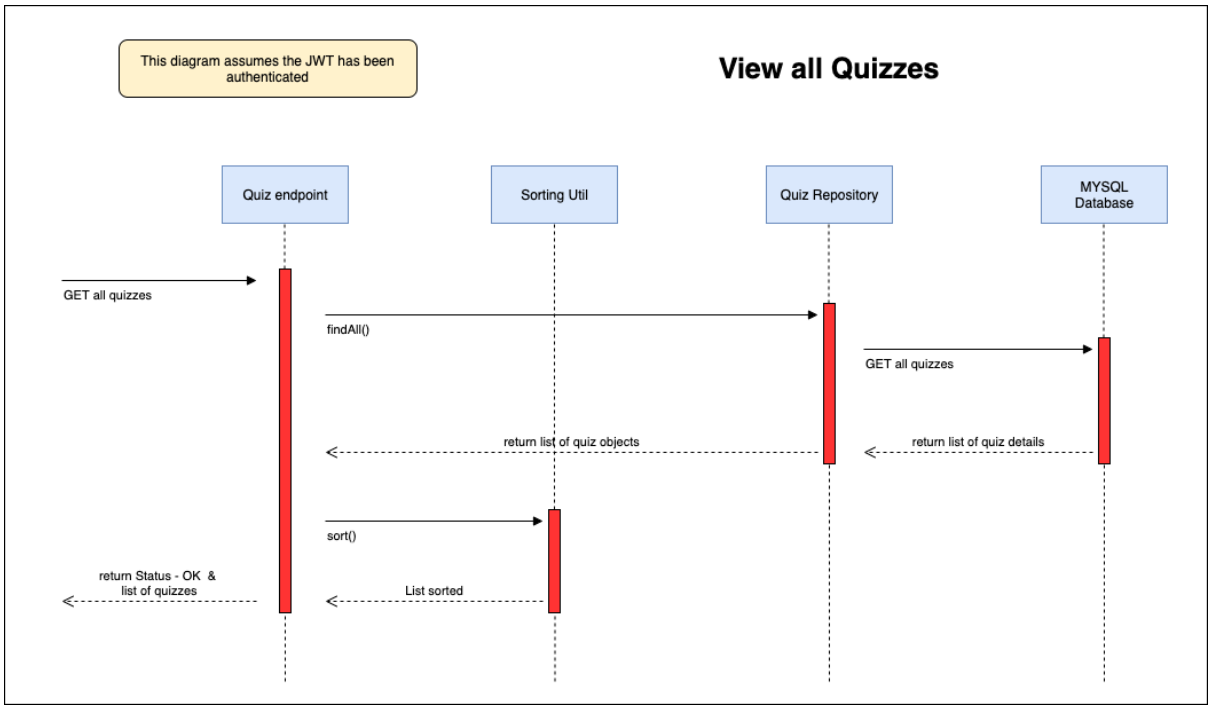


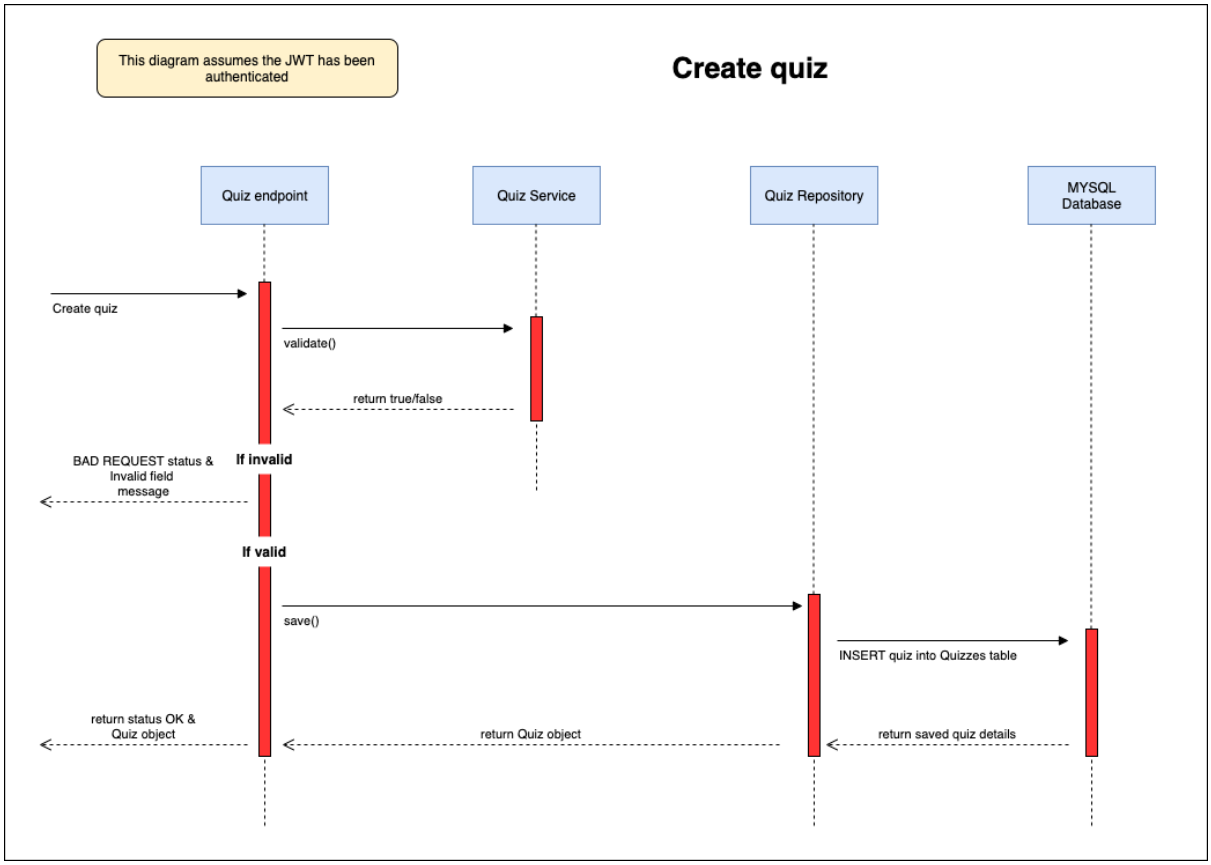
JWT FILTER

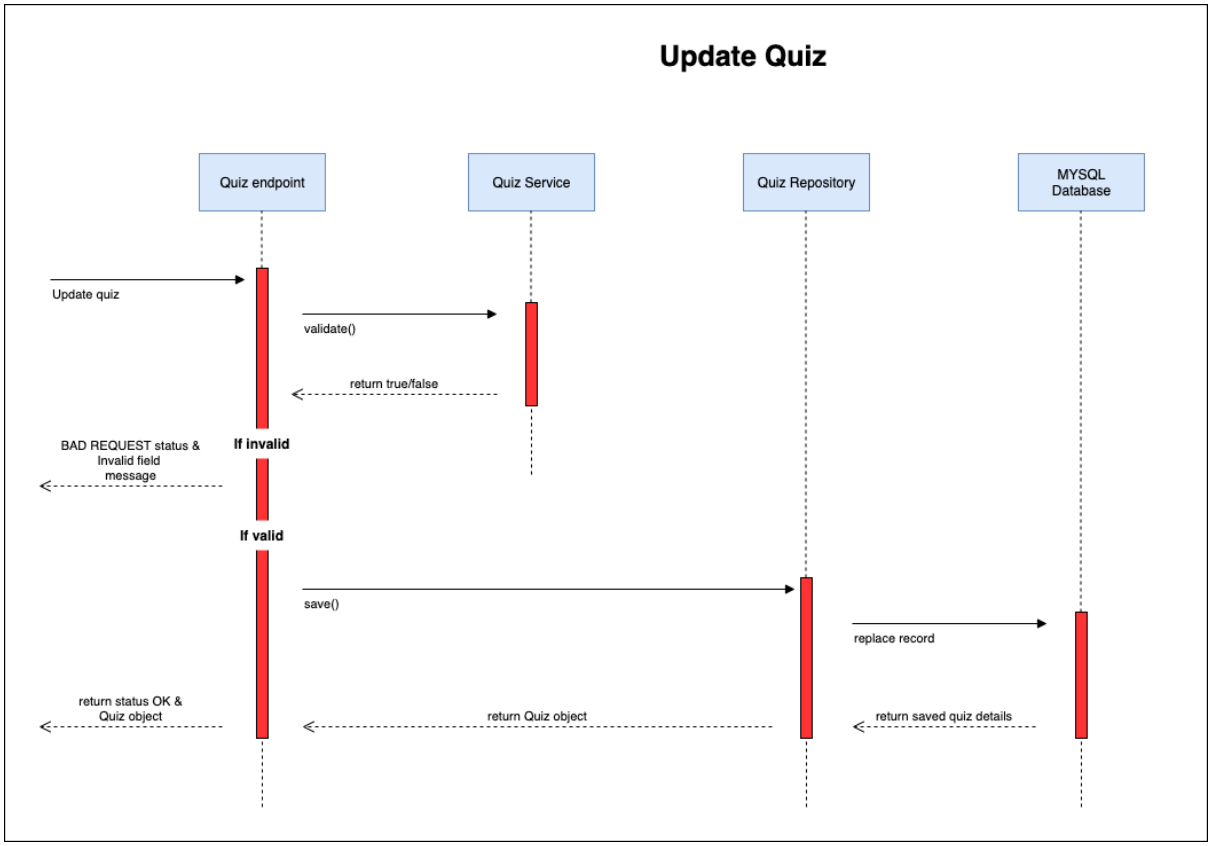

Any requests to endpoints, other than login (and potentially signup), will be filtered to check a valid JWT has been provided
JWT filtering
endpoint
JWT filter
JWT Util
Custom UserDetailsService
User Repository
MYSQL Database
request
filter()
extractUsername()
return username
get UserDetails
BAD REQUEST status & Invalid Credentials message
loadUserByUsername()
SELECT user details from DB
return UserDetails
return User object
return User details
isJwtValid()
If invalid
return JWT exception
return username
If valid
authenticate & return to endpoint
Execute actions
return result

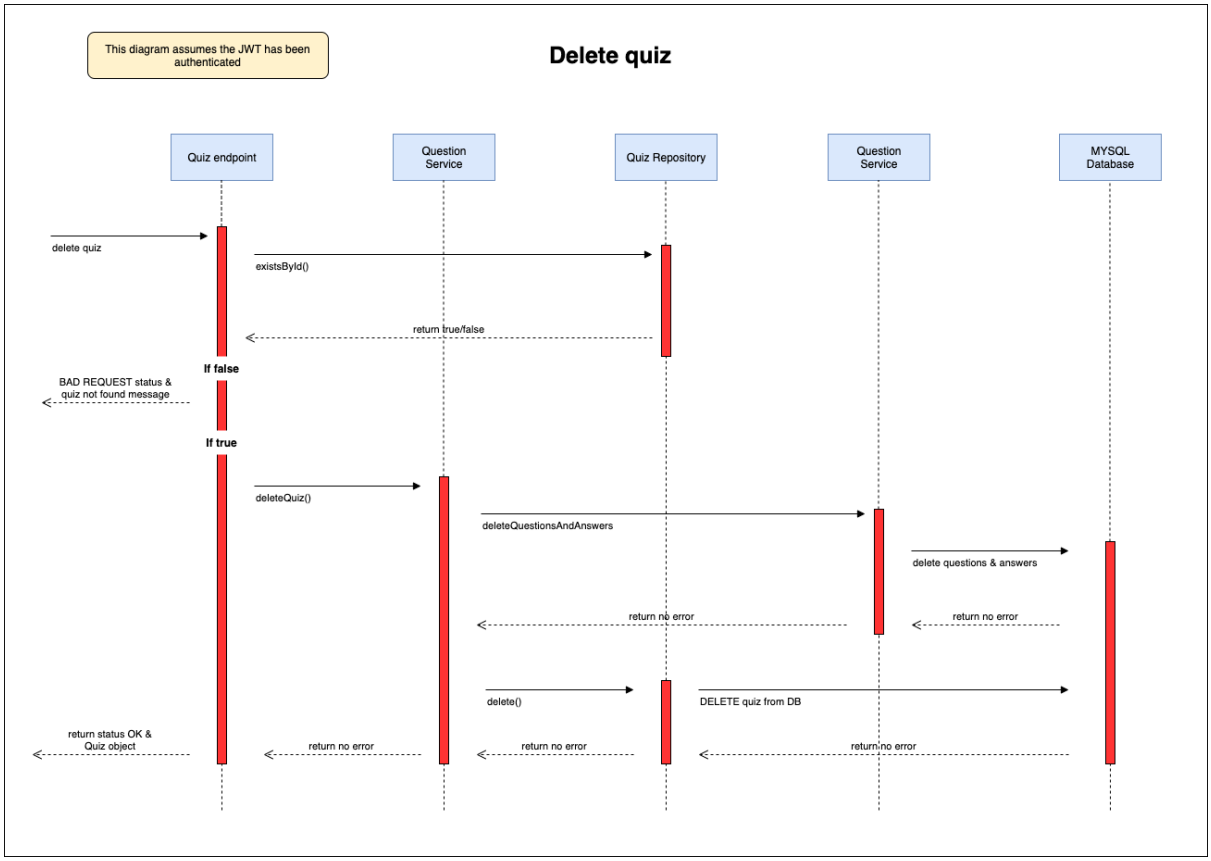
QUIZ ENDPOINTS



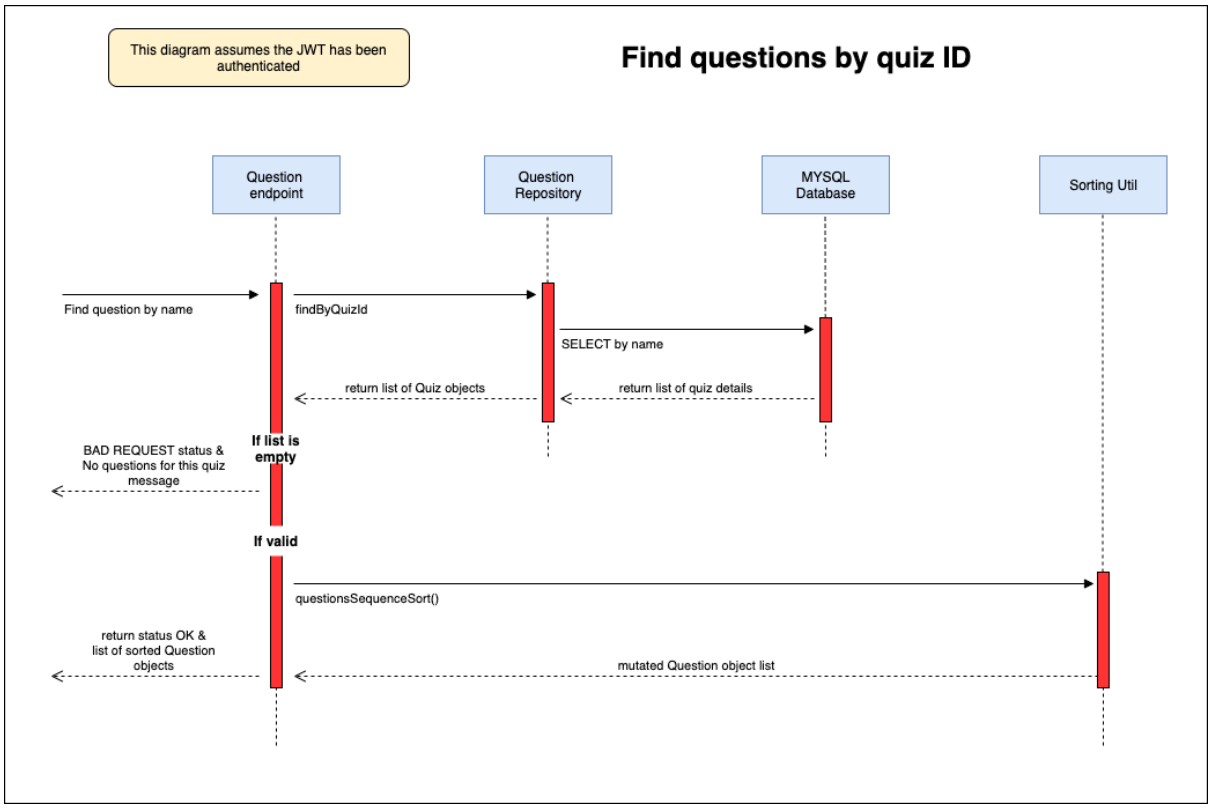



This diagram assumes the JWT has been authenticated
Create quiz
Quiz endpoint
Quiz Service
Quiz Repository
MYSQL Database
Create quiz
validate()
return true/false
BAD REQUEST status & Invalid field message
If invalid
If valid
save()
INSERT quiz into Quizzes table
return status OK & Quiz object
return Quiz object
return saved quiz details


Update Quiz
Quiz endpoint
Quiz Service
Quiz Repository
MYSQL Database
Update quiz
validate()
return true/false
BAD REQUEST status & Invalid field message
If invalid
If valid
save()
replace record
return status OK & Quiz object
return Quiz object
return saved quiz details

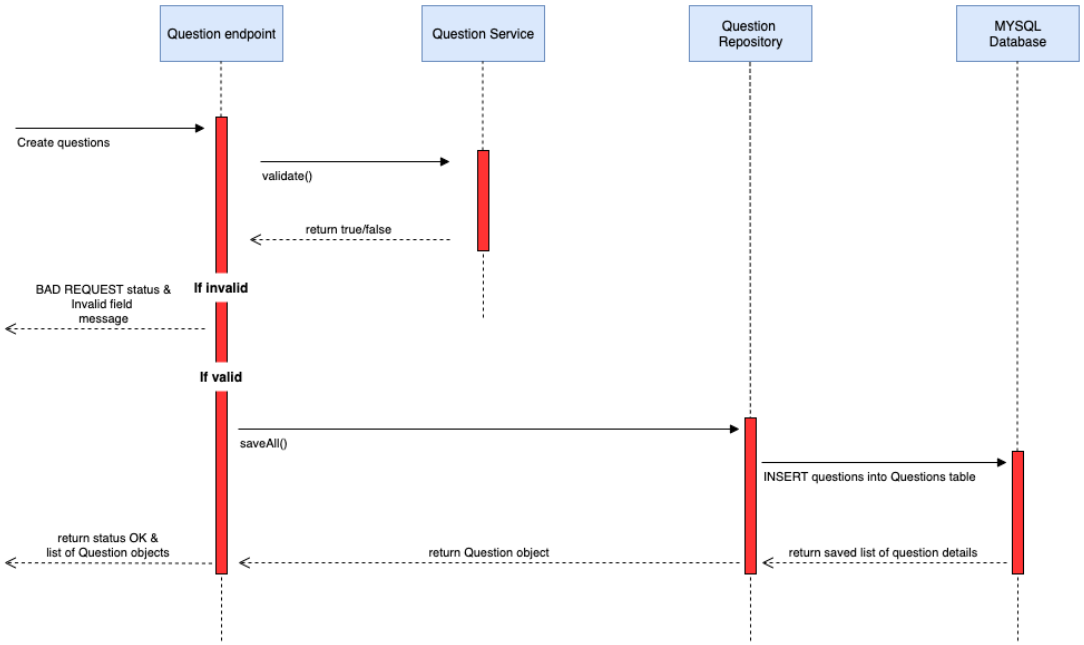

This diagram assumes the JWT has been authenticated
Delete quiz
Quiz endpoint
Question Service
Quiz Repository
Question Service
MYSQL Database
delete quiz
existsById()
return true/false
If false
BAD REQUEST status & quiz not found message
If true
deleteQuiz()
deleteQuestionsAndAnswers
delete questions & answers
return no error
return no error
delete()
DELETE quiz from DB
return status OK & Quiz object
return no error
return no error
return no error

QUESTION ENDPOINTS

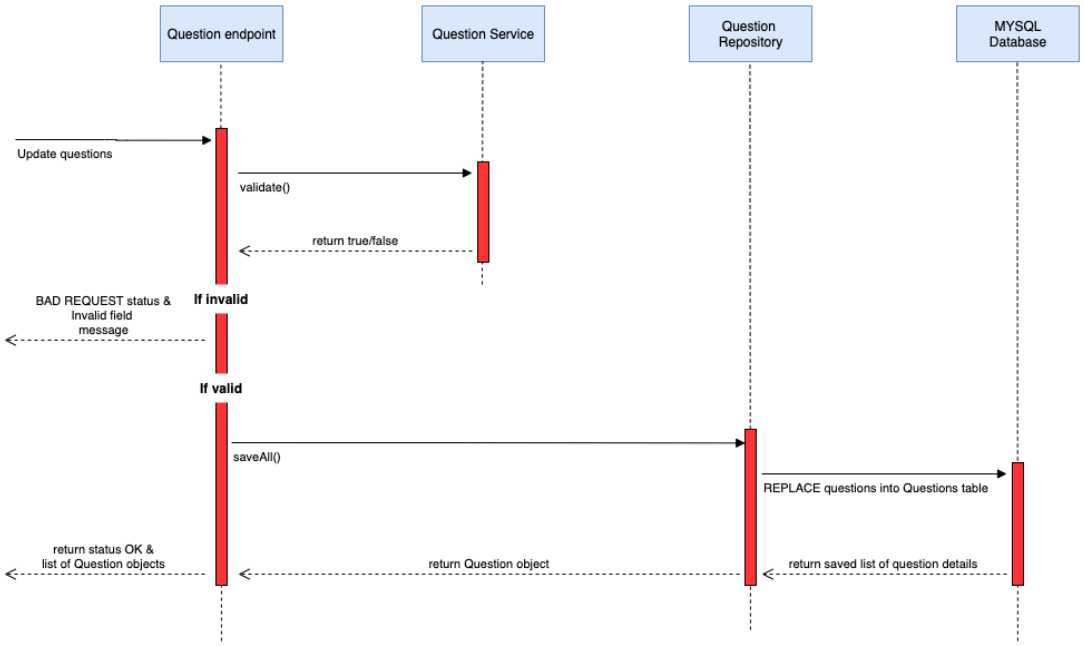


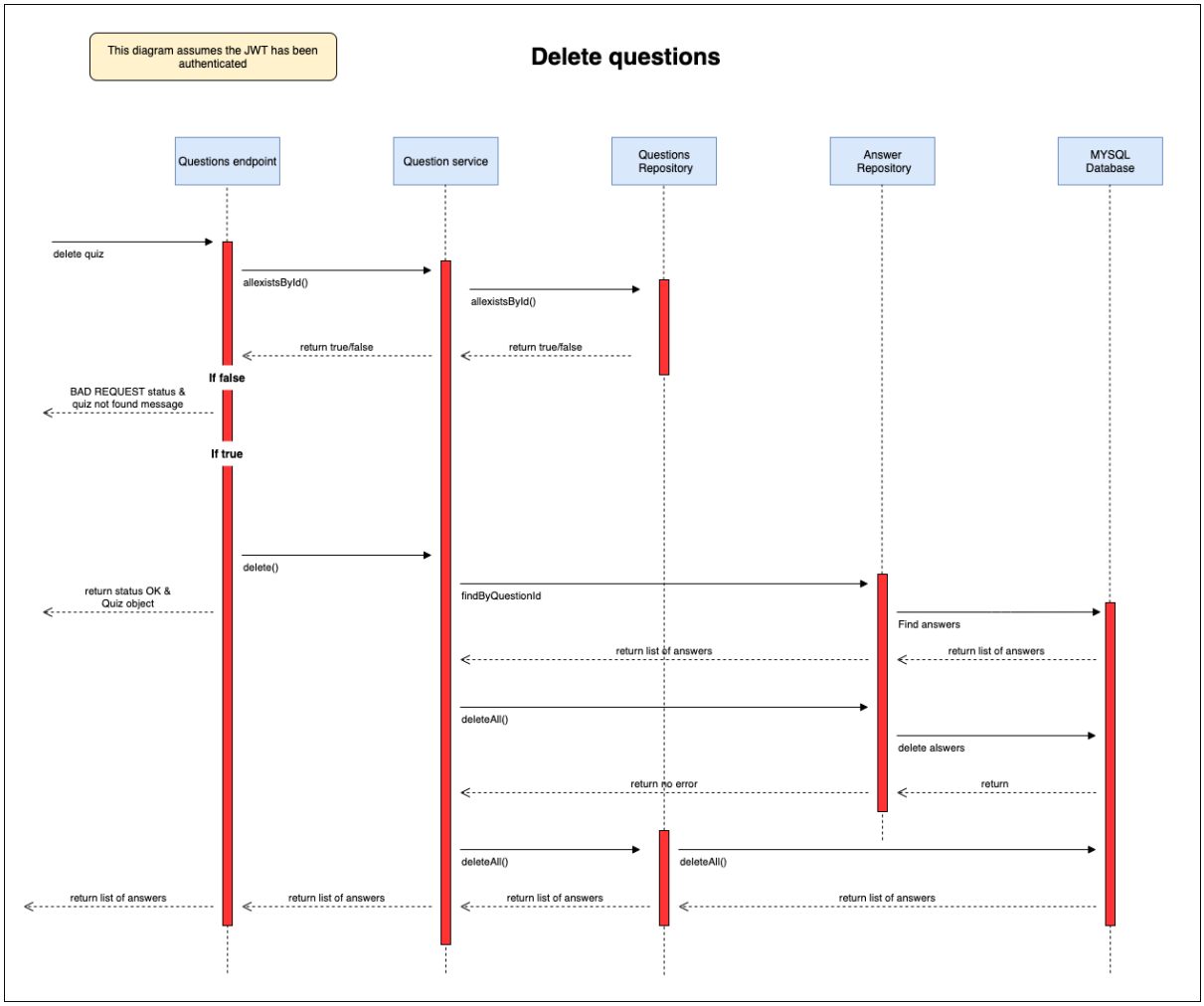
This diagram assumes the JWT has been authenticated

Create question

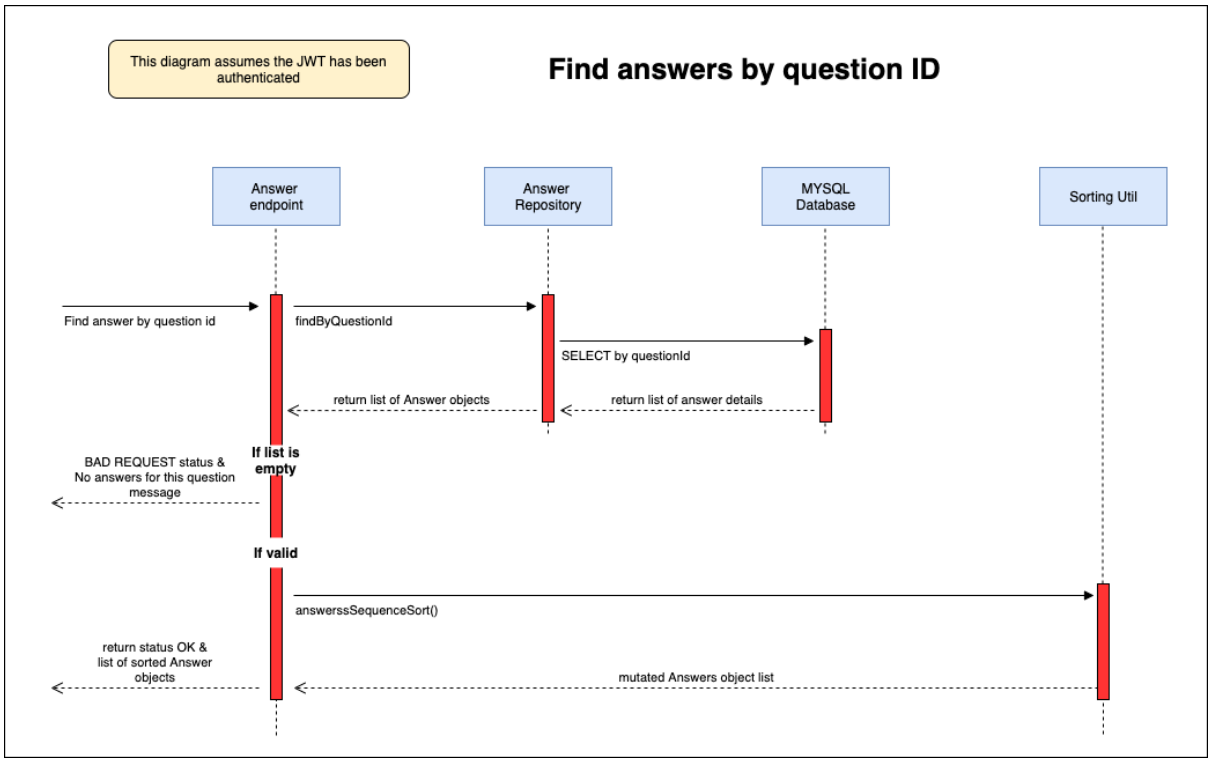


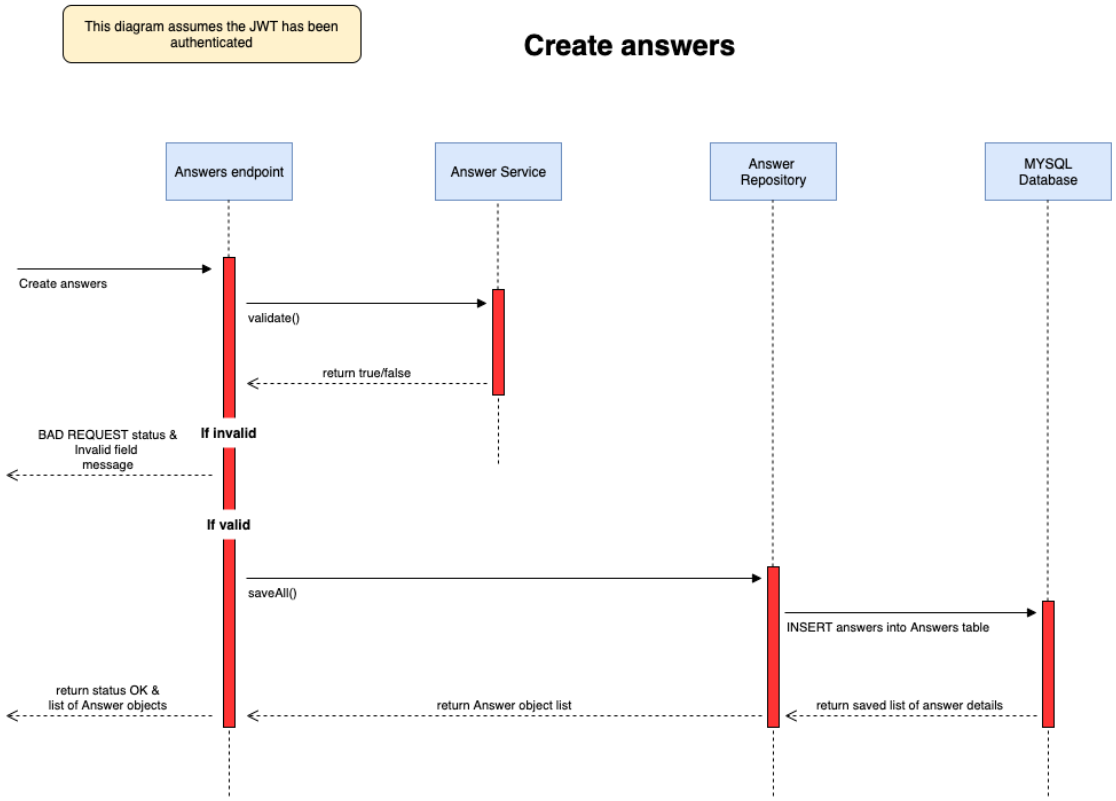
Update question

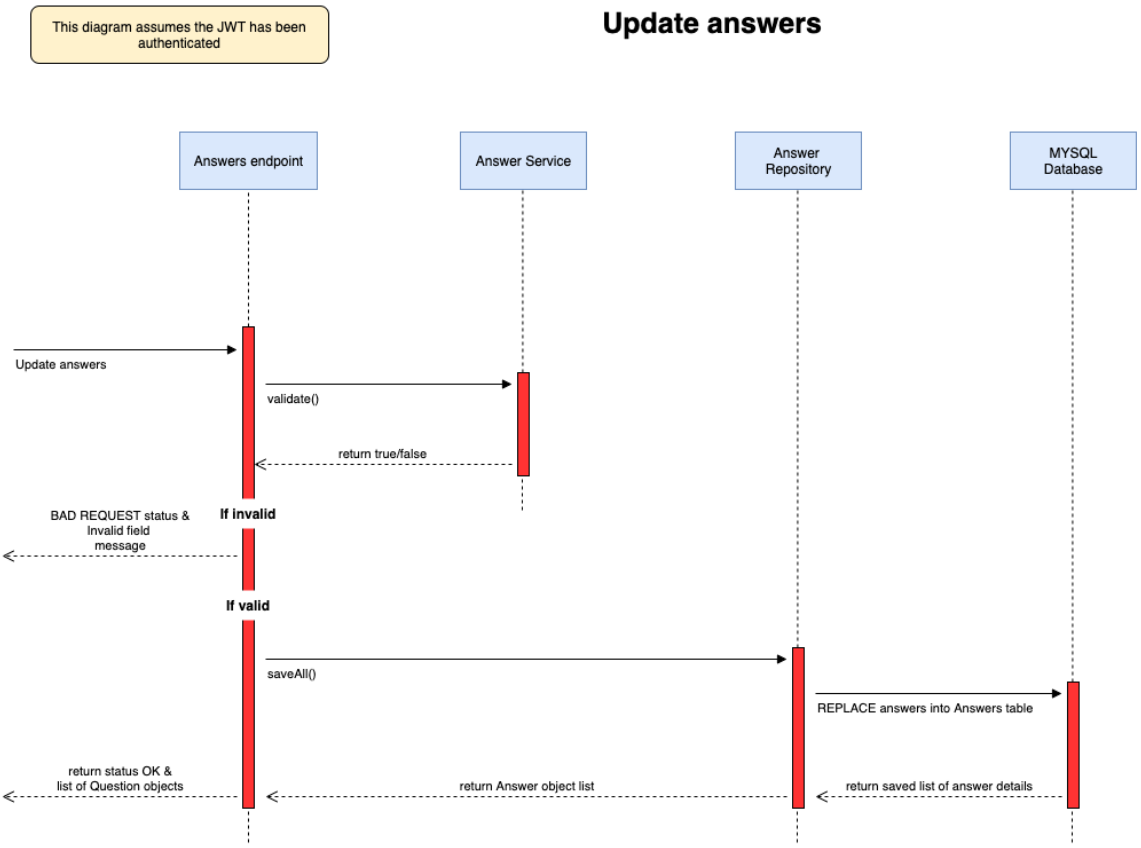


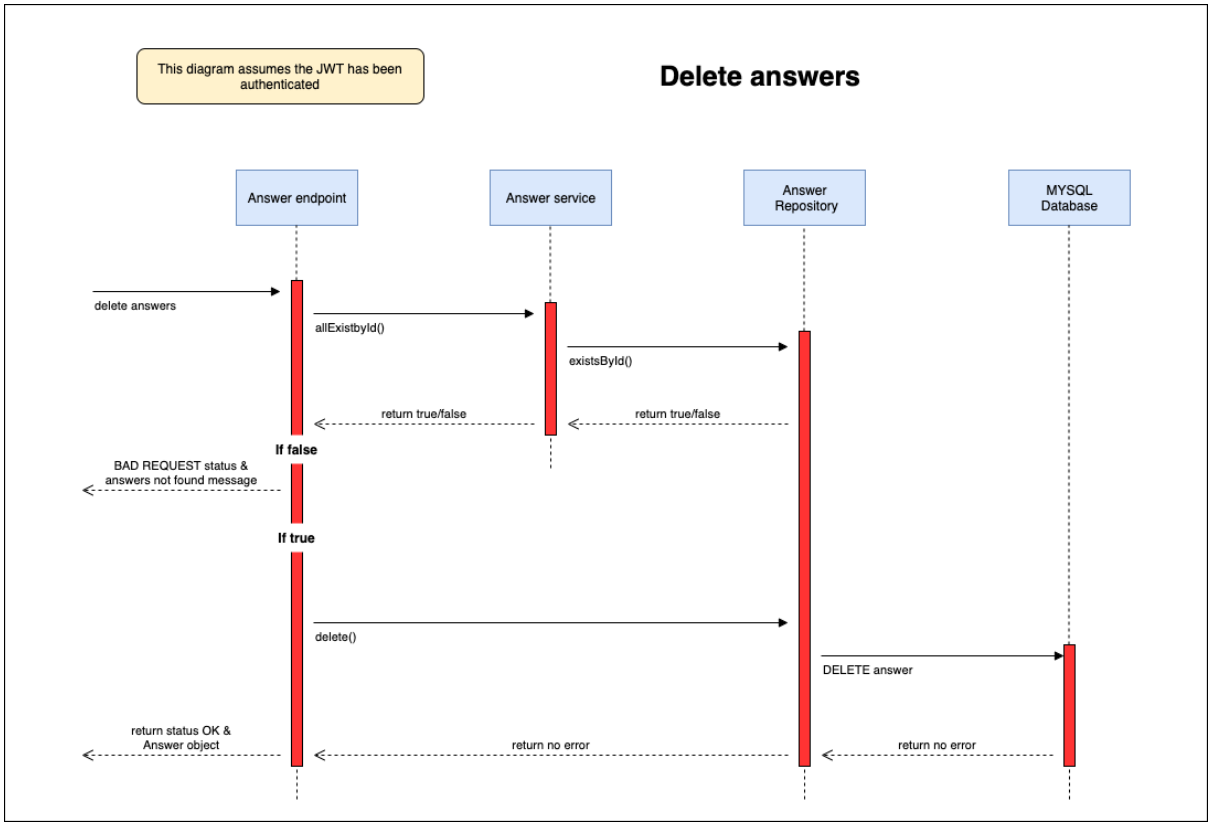

This diagram assumes the JWT has been authenticated
Delete questions
Questions endpoint
Question service
Questions Repository
Answer Repository
MYSQL Database
delete quiz
allexistsById()
allexistsById()
return true/false
return true/false
If false
BAD REQUEST status & quiz not found message
If true
delete()
return status OK & Quiz object
findByQuestionId
Find answers
return list of answers
return list of answers
deleteAll()
delete alswers
return no error
return
deleteAll()
deleteAll()
return list of answers
return list of answers
return list of answers
return list of answers

ANSWER ENDPOINTS




This diagram assumes the JWT has been authenticated
Create answers
Answers endpoint
Answer Service
Answer Repository
MYSQL Database
Create answers
validate()
return true/false
BAD REQUEST status & Invalid field message
If invalid
If valid
saveAll()
INSERT answers into Answers table
return status OK & list of Answer objects
return Answer object list
return saved list of answer details


This diagram assumes the JWT has been authenticated
Update answers
Answers endpoint
Answer Service
Answer Repository
MYSQL Database
Update answers
validate()
return true/false
BAD REQUEST status & Invalid field message
If invalid
If valid
saveAll()
REPLACE answers into Answers table
return status OK & list of Question objects
return Answer object list
return saved list of answer details


This diagram assumes the JWT has been authenticated
Delete answers
Answer endpoint
Answer service
Answer Repository
MYSQL Database
delete answers
allExistbyId()
existsById()
return true/false
return true/false
If false
BAD REQUEST status & answers not found message
If true
delete()
DELETE answer
return status OK & Answer object
return no error
return no error

IMPLEMENTATION – BACK END API

WHAT WENT WELL

App was built with all features I wanted to add. It provides all of the endpoints outlined in the requirements and they are all secured by a JWT filter which checks the Authorization field of every request header (except login) to ensure it contains a valid JWT. Any subsequent developers that work on the app will have the benefit of the Swagger documentation that has been provided so they will be able to see the endpoints and try them out which will help to familiarise them with the back end API very quickly.

I was also very happy I was able to complete all of the integration testing so all of the endpoints have been tested. I was also able to implement and test methods for conducting a sequence sort which will ensure that data is always provided in the correct order even if it is not saved in order in the database.

PROBLEMS ENCOUNTERED

Took longer than anticipated by approximately one hour because of errors that had to be resolved at the testing stage.

Not all Junit testing carried out. The tests that are outstanding are for the validation methods and repositories. There is not much to be tested for the repositories other than the custom methods; however these have technically been tested within the integration tests for each endpoint.

Found an issue where the 'No Content' status responses were not being accompanied by string text

TESTING – BACK END API

JUNIT TESTING

I carried out my tests for the back-end using JUnit and they consisted of integration tests and a number of unit tests. The integration tests tested that the endpoints returned the correct status, content type and JSON content. The following endpoints have integration tests:

- Authentication – Login
- Quiz
    - Find By Name
    - Get All
    - Create
    - Update
    - Delete
- Question
    - Find by quiz id
    - Create
    - Update
    - Delete

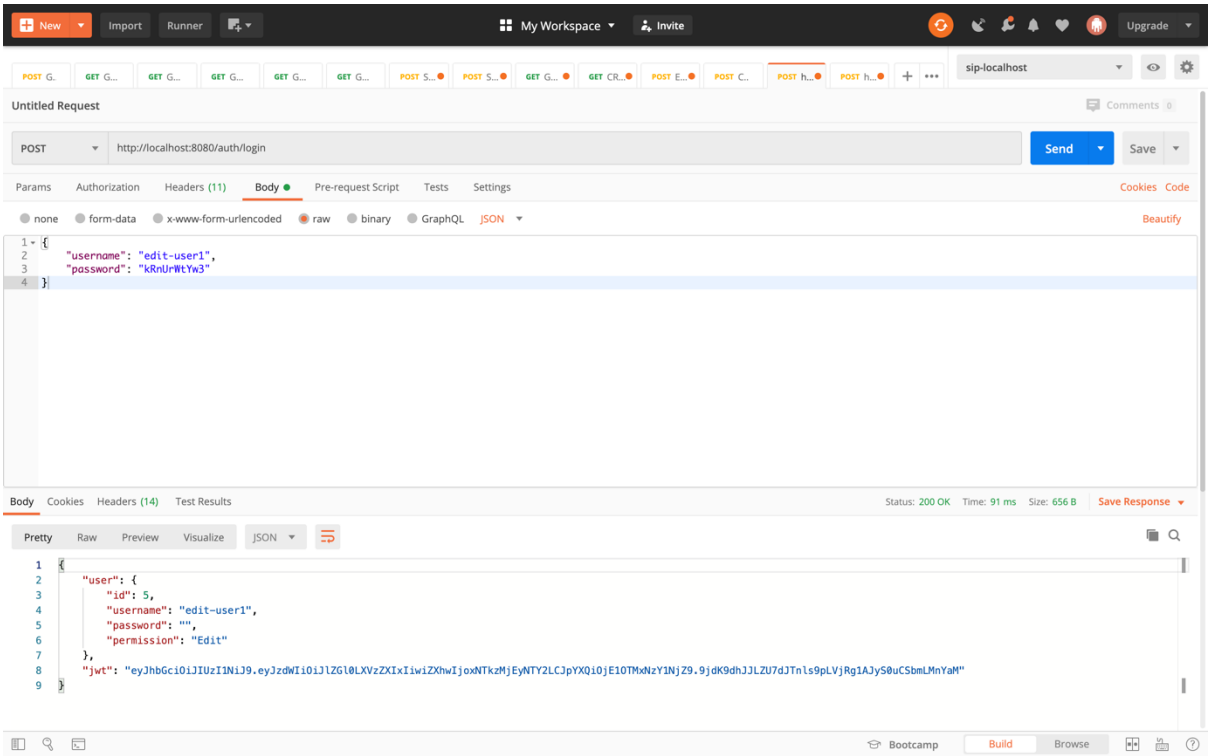
- Answer
    - Find by question id
    - Create
    - Update
    - Delete

MANUAL TESTING

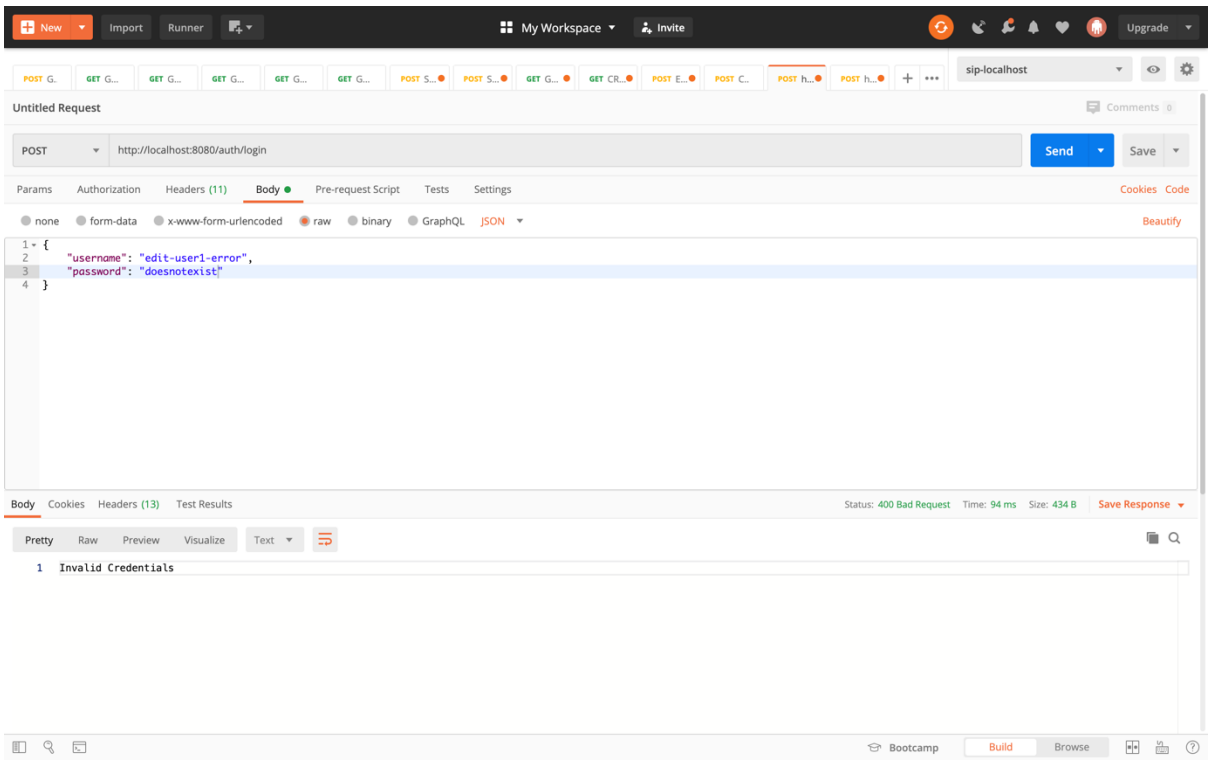
Testing has been carried out using Postman to access the endpoints and test for the expected results.

LOGIN

Endpoint	Test	Expected Outcome	Outcome	Pass/Fail
/auth/login	Log in with correct credentials	Should return a user object and JWT	Returned a user object and JWT	Pass
/auth/login	Log in with incorrect credentials	Should a string “Invalid Credentials”	Returned - string “Invalid Credentials”	Pass



Successful login – user details and JWT provided

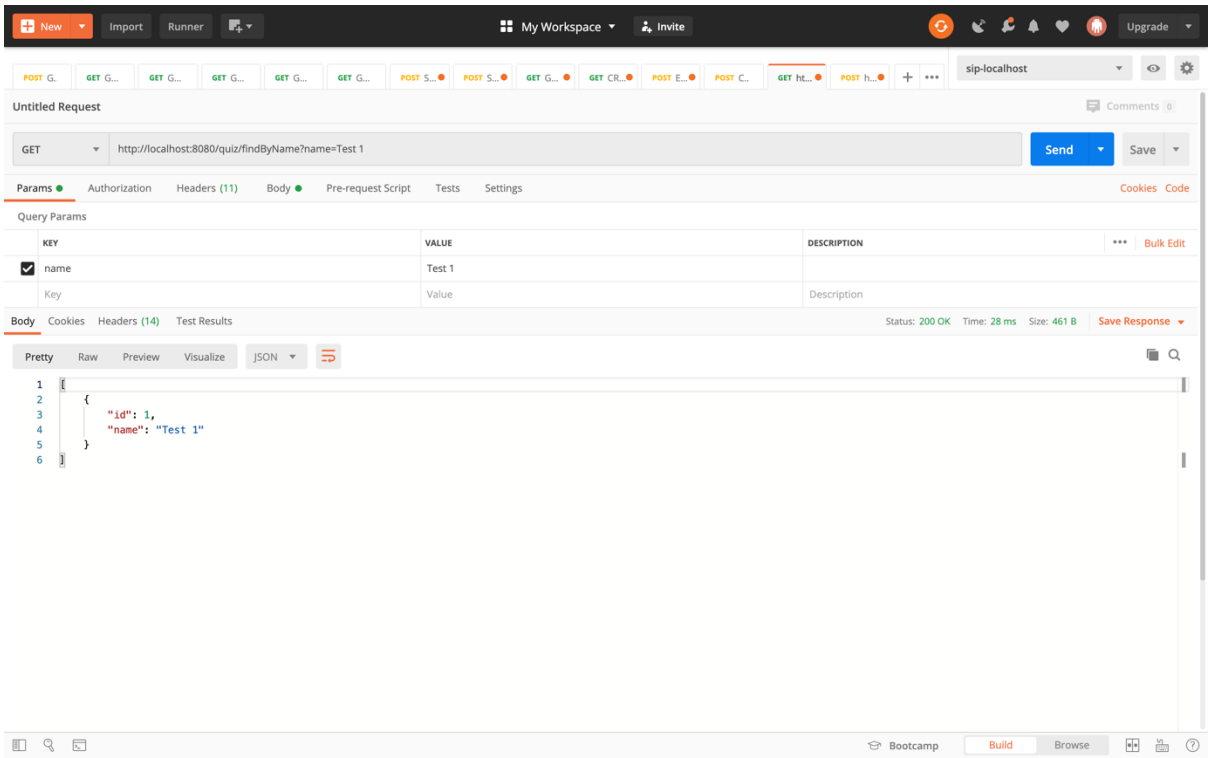


Invalid credentials error received

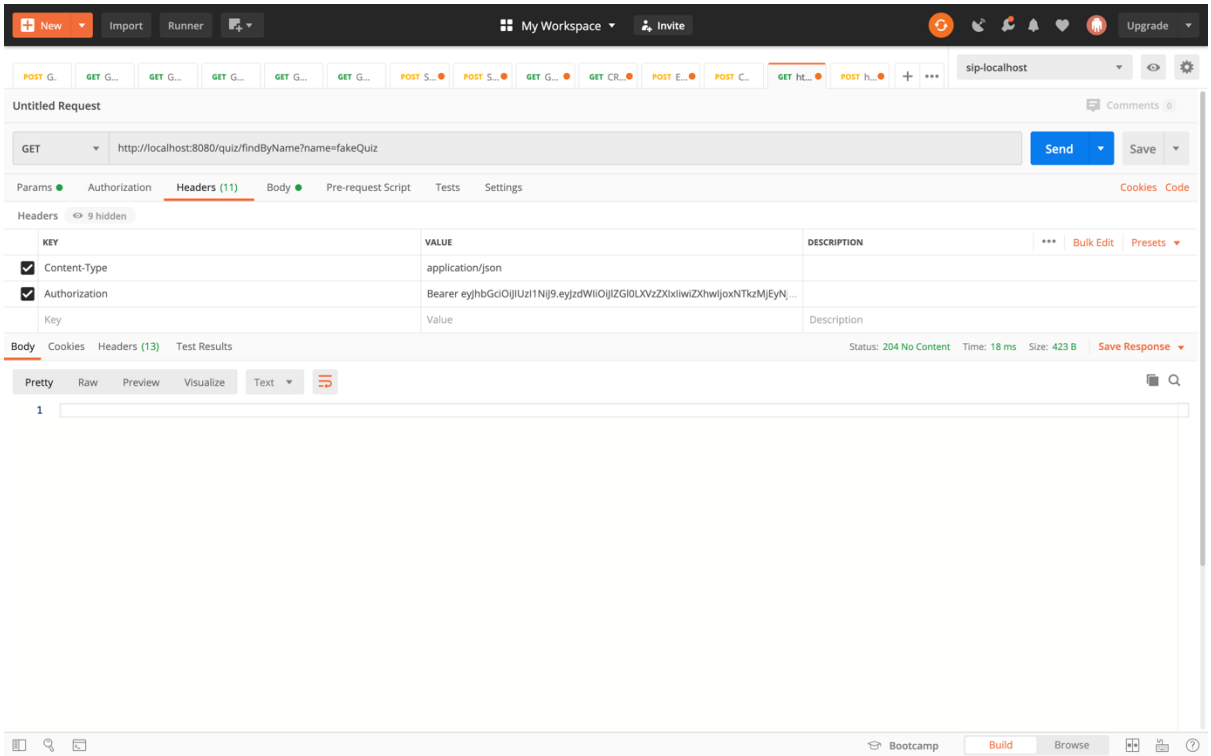
QUIZ ENDPOINT

Endpoint	Test	Expected Outcome	Outcome	Pass/Fail
/quiz/findByName	Find existing quiz	Return list of Quiz objects	Returned list of Quiz objects	Pass
/quiz/findByName	Find non-existent quiz	Return “No quizzes found by that name”	Status: 204 no content & no message	Fail
/quiz/findByName	Find non-existent quiz	Status: 204 no content & no message	Status: 204 no content & no message	Pass – After researching status, found that this does not provide string message so this will be handled by front end
/quiz/getAll	Find all	Return list of Quiz objects	Returned list of Quiz objects	Pass

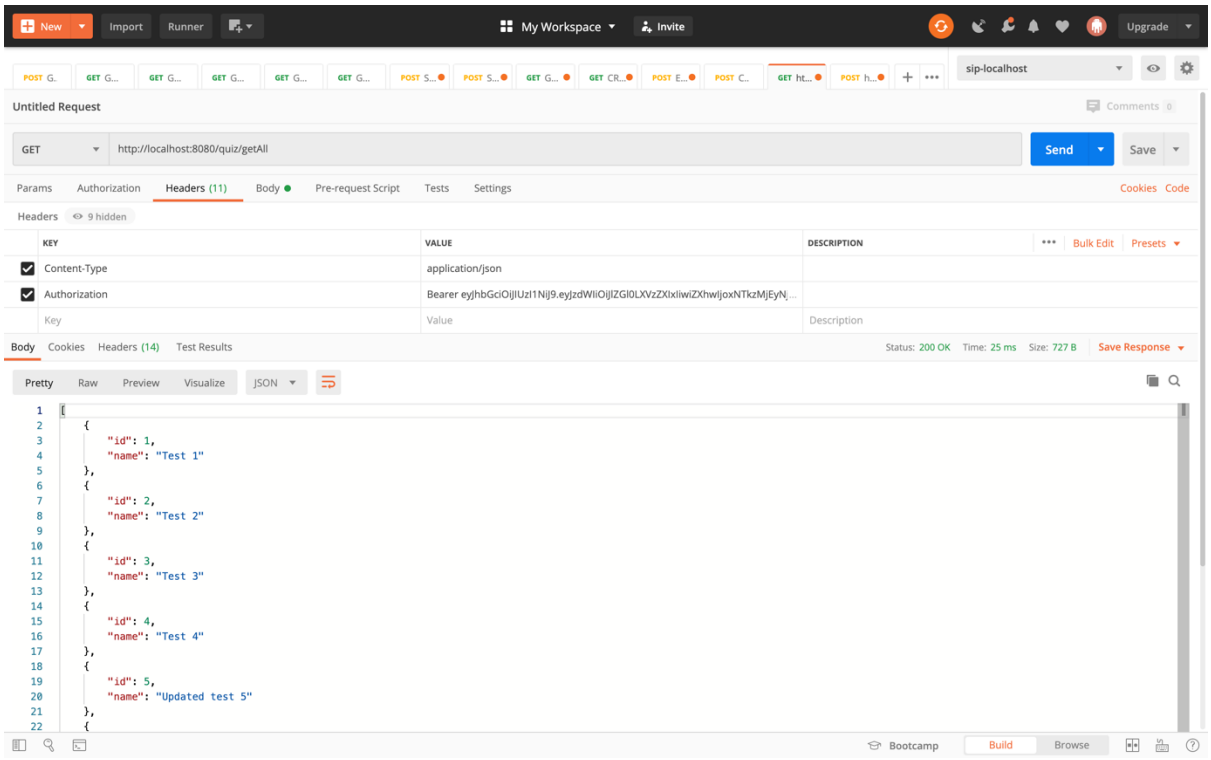
Endpoint	Test	Expected Outcome	Outcome	Pass/Fail
/quiz/getAll	DB empty	Status: 204 no content	Status: 204 no content	Pass
/quiz/create	Create quiz with valid fields	Return list of Quiz objects	Returned list of Quiz objects	Pass
/quiz/create	Create quiz with invalid fields	Return "Invalid values provided – Please try again"	Returned "Invalid values provided – Please try again"	Pass
/quiz/update	Update quiz with valid fields	Return list of Quiz objects	Returned list of Quiz objects	Pass
/quiz/update	Update quiz with invalid fields	String message "Quiz not found"	Status: 204 & no content	Fail
/quiz/update	Re-tested - Update quiz with invalid fields	String message "Quiz not found"	Returned String message "Quiz not found"	Pass – (After researching, changed the status to 400)
/quiz/delete	Delete existing quiz	Return "Invalid values provided – Please try again"	Returned "Invalid values provided – Please try again"	Pass
/quiz/delete	Delete non-existent quiz	Return "unable to delete-quiz not found"	Returned status 204 but no string message	Fail
/quiz/delete	Re-tested - Delete non-existent quiz	Return "unable to delete-quiz not found"	Returned "unable to delete-quiz not found"	Pass – (After researching, changed the status to 400)



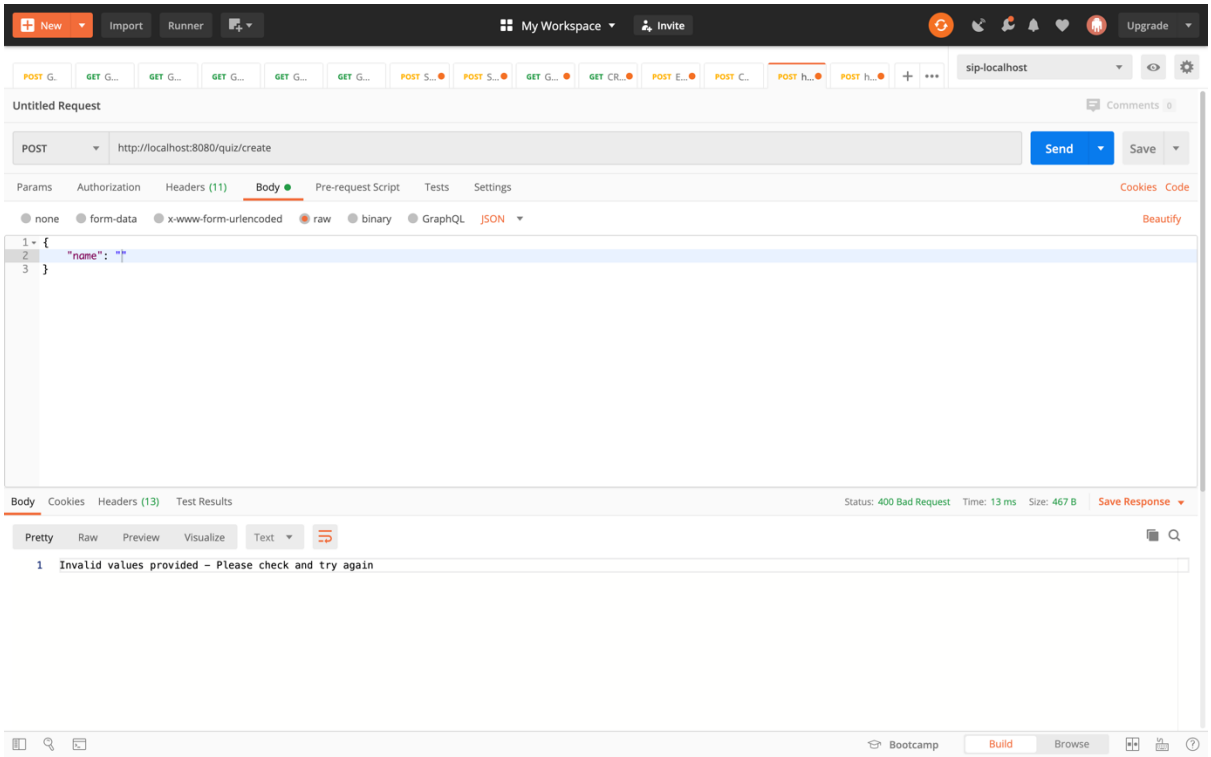
Successfully found a quiz by name



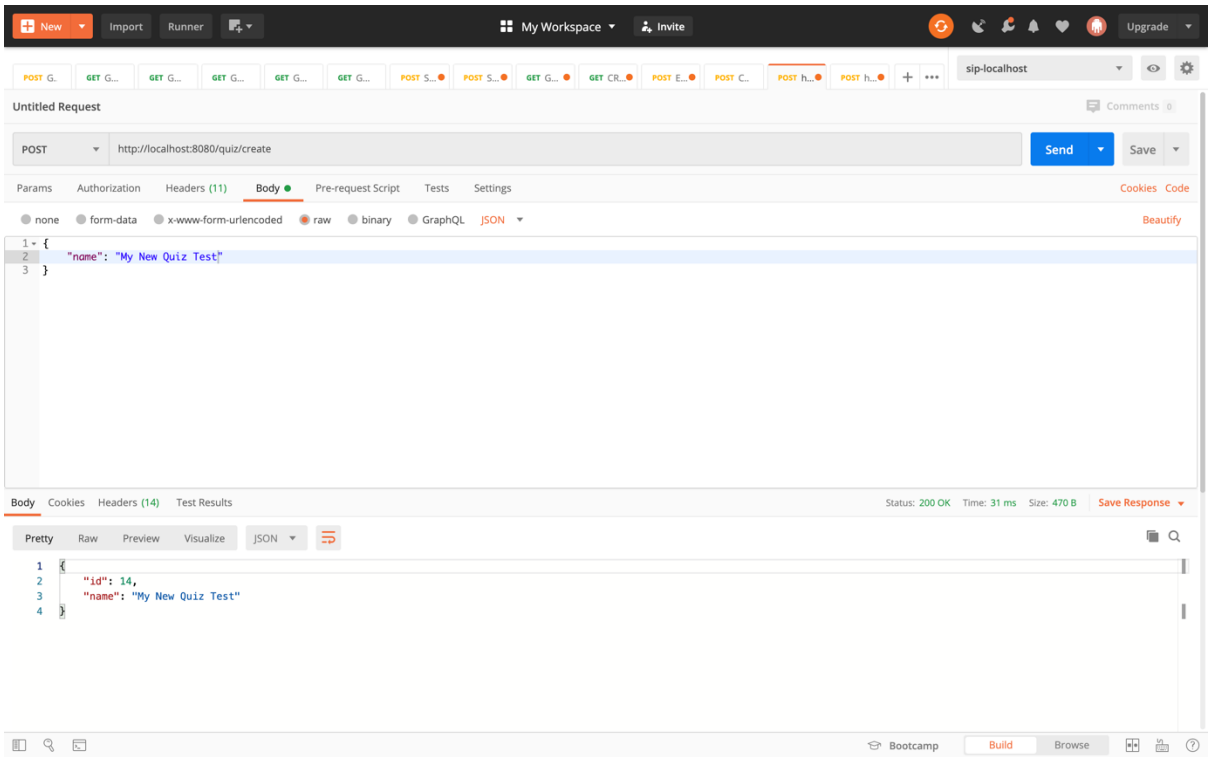
Status 204 when no content to return



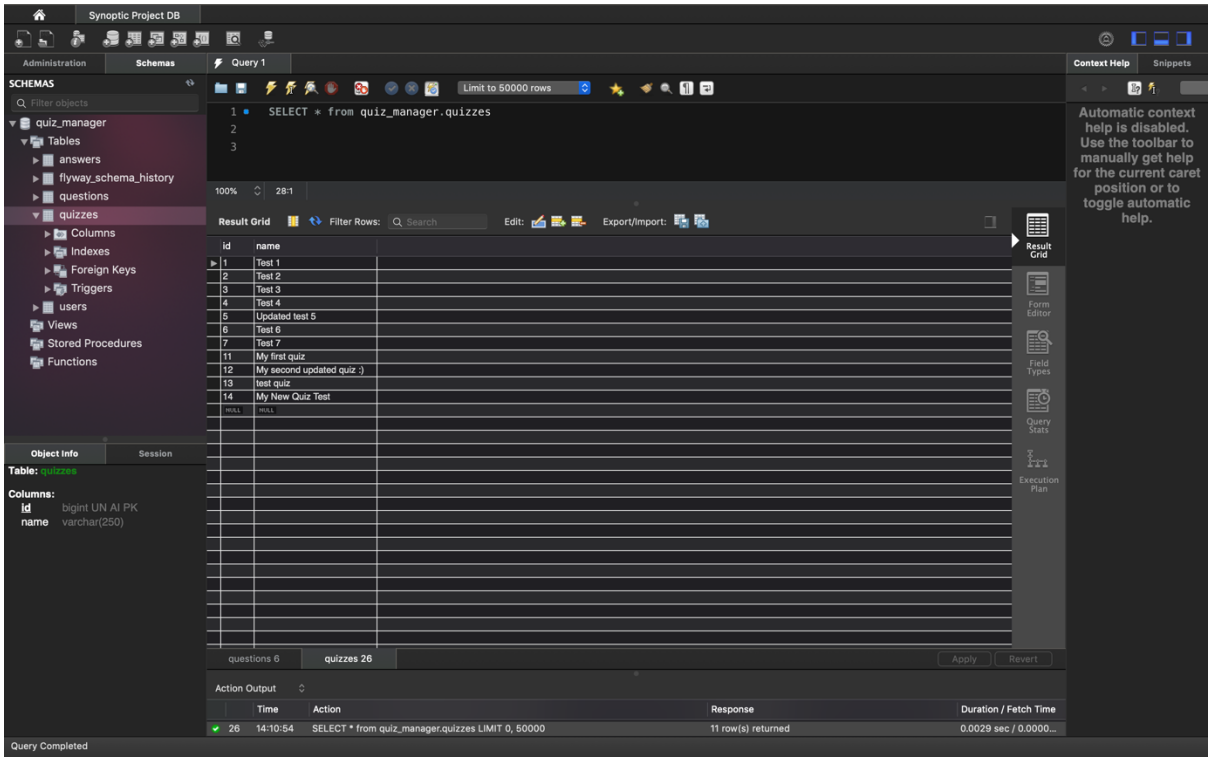
Returns list of all quizzes



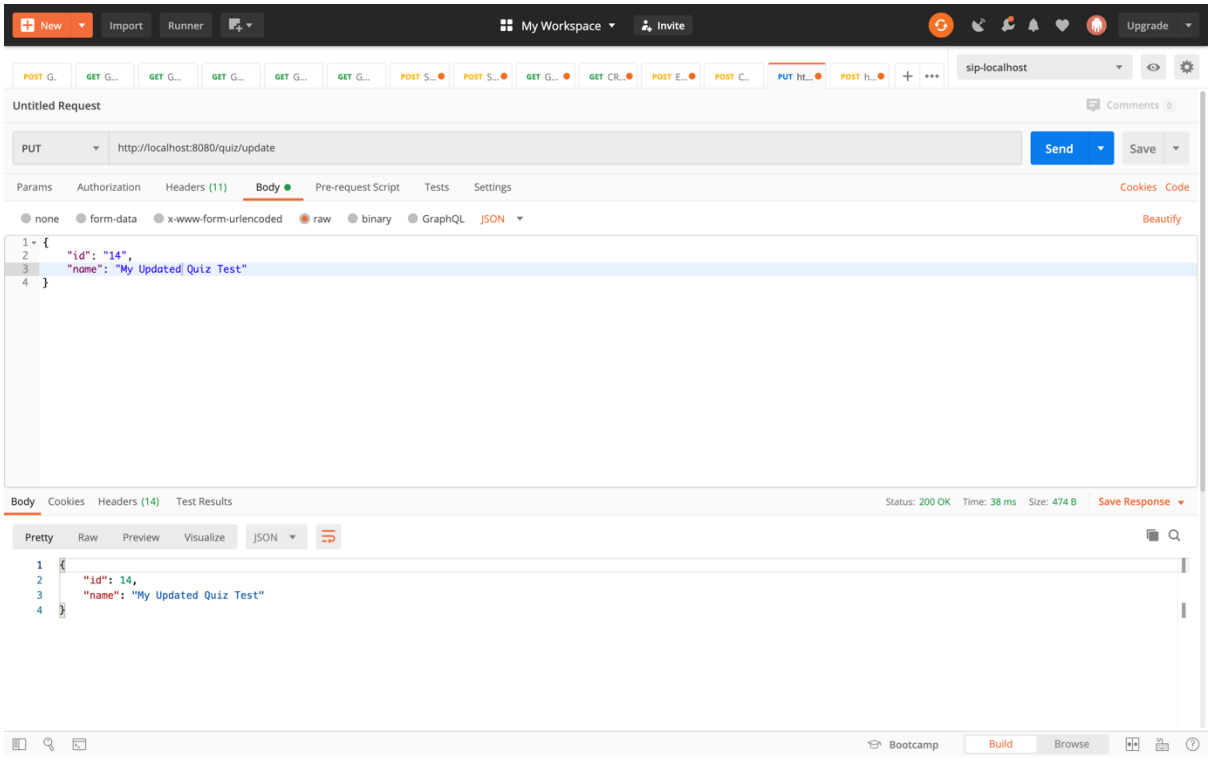
Returns invalid field message when attempting to create a quiz with invalid fields



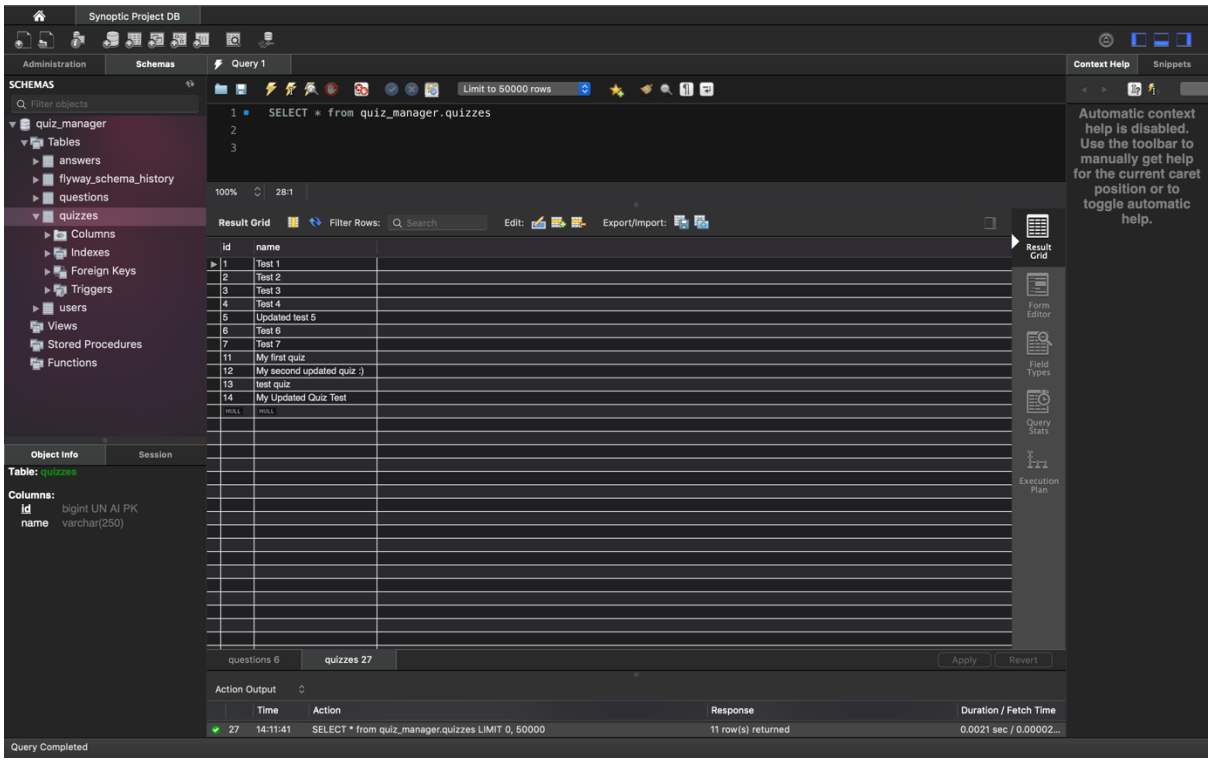
Returns quiz object of created quiz



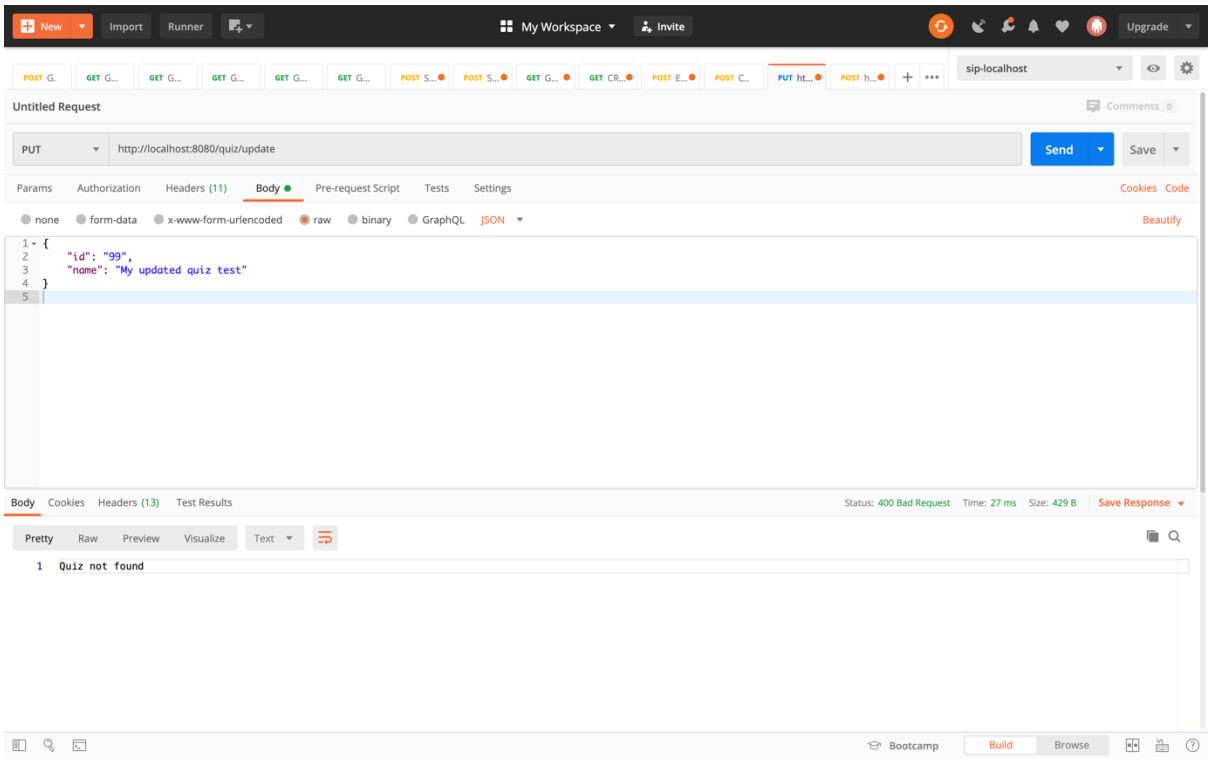
Quiz saved in DB



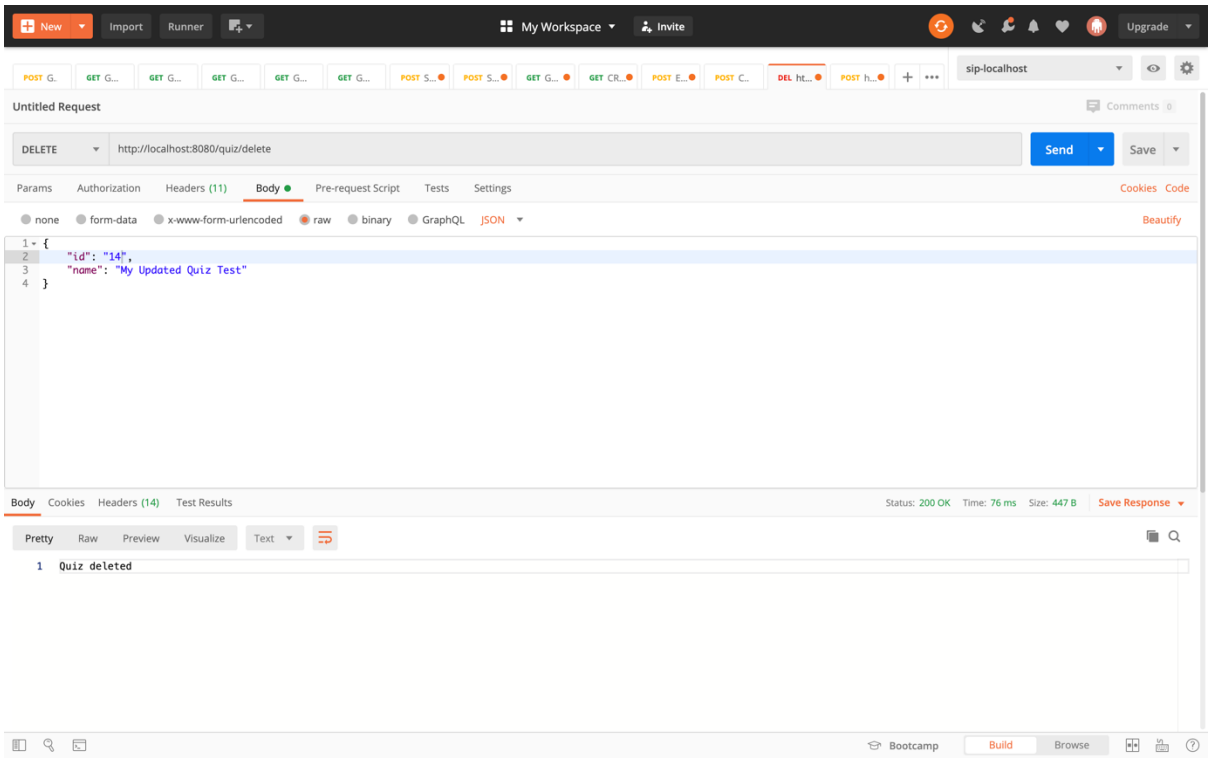
Updated quiz object returned



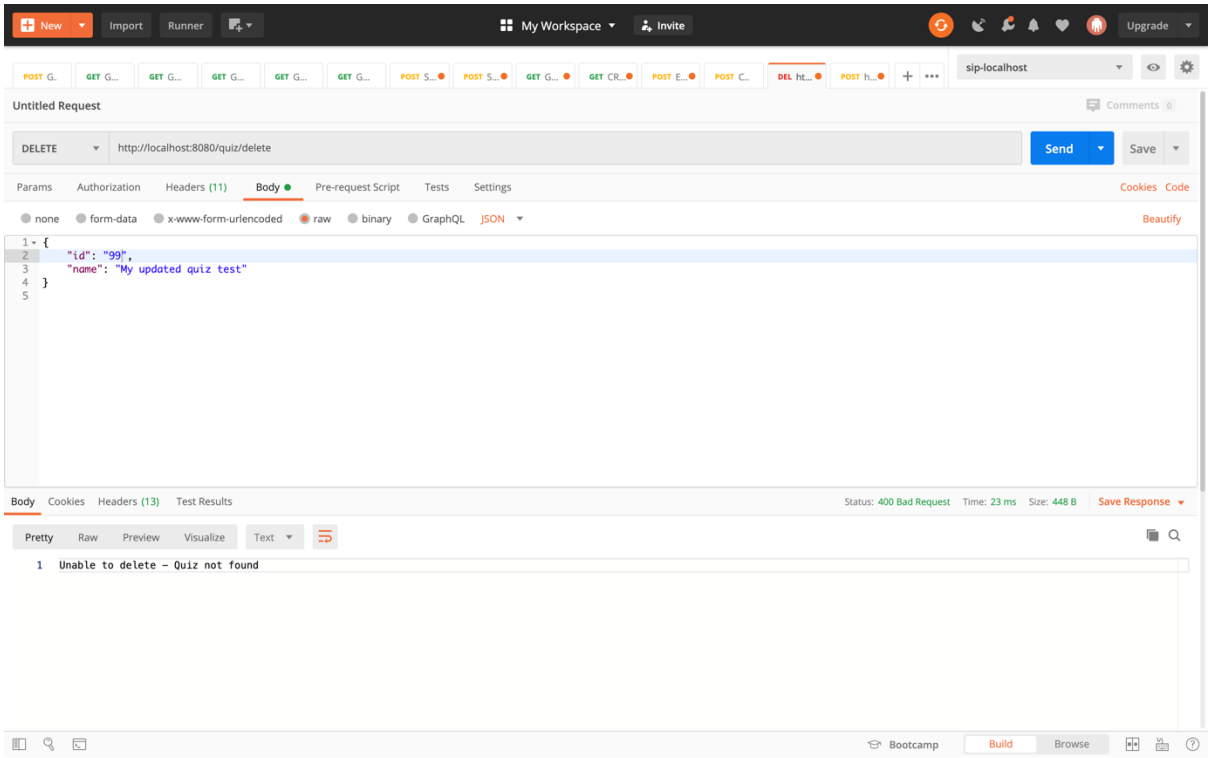
Updated quiz object saved in DB



Status 400 when quiz not found, along with string message



Confirmation message when quiz deleted

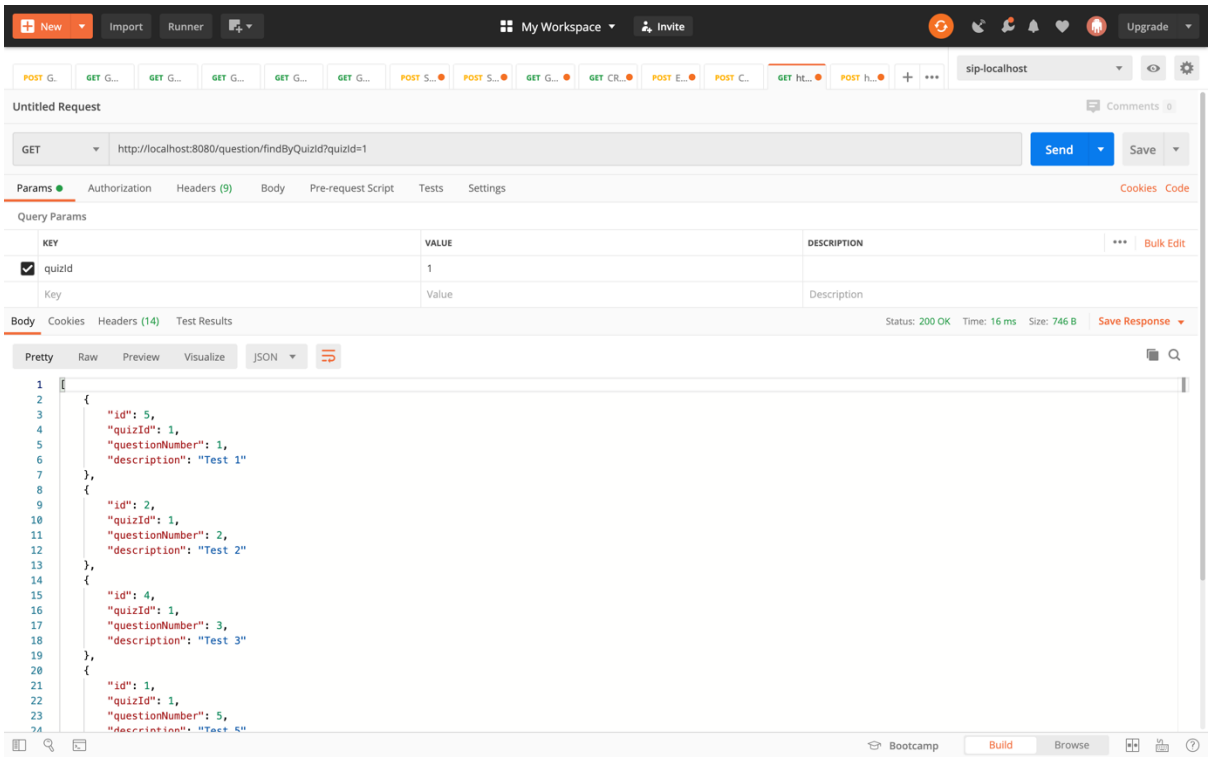


Error message and status 400 when trying to delete a quiz that does not exist

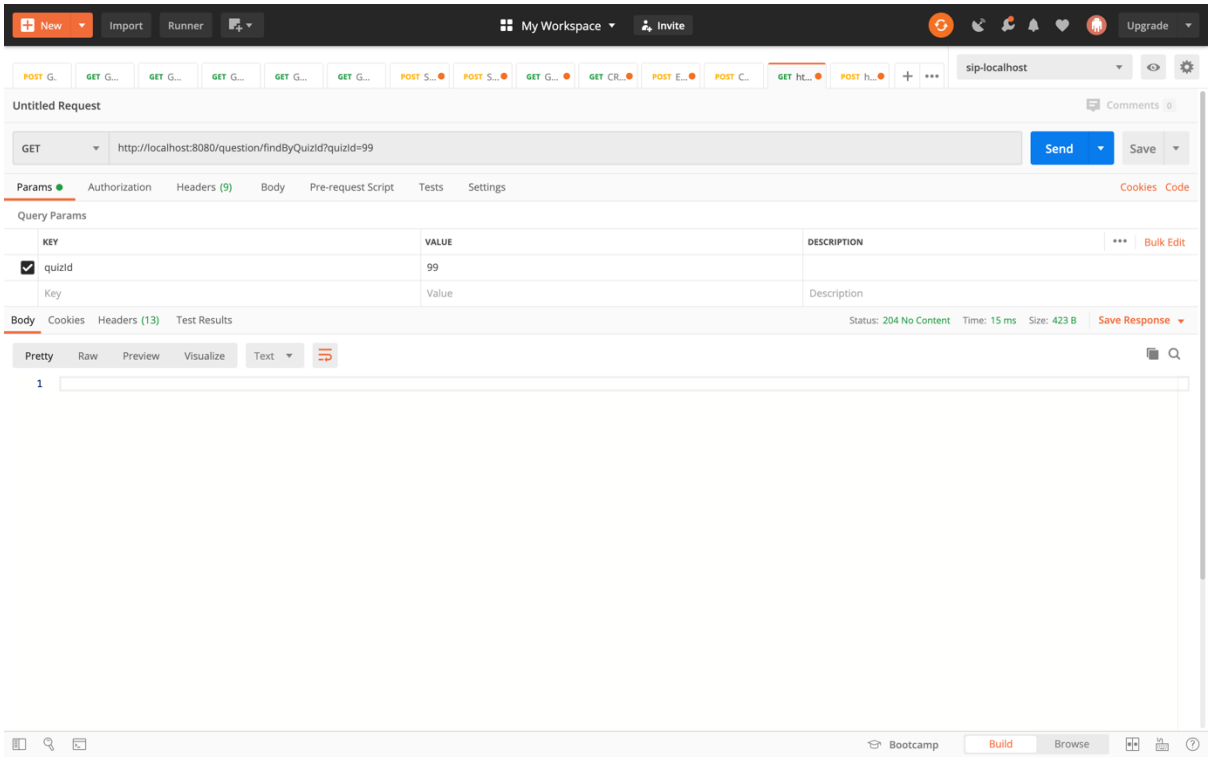
QUESTION ENDPOINT

Endpoint	Test	Expected Outcome	Outcome	Pass/Fail
/question/findByQuizId	Find questions using existing quiz id	Return list of questions	Returned list of questions	Pass
/question/findByQuizId	Find questions using non-existent quiz id	Return "This quiz currently has no questions"	Returned status: 204 no content	Fail
/question/create	Create question with valid fields	Return list of questions (one question in this case)	Returned list of questions (one question in this case)	Pass
/question/create	Create question with invalid fields	Return "Invalid values provided – Please try again"	Returned "Invalid values provided – Please try again"	Pass
/question/update	Update question with valid fields	Return list of questions (one question in this case)	Returned list of questions (one question in this case)	Pass

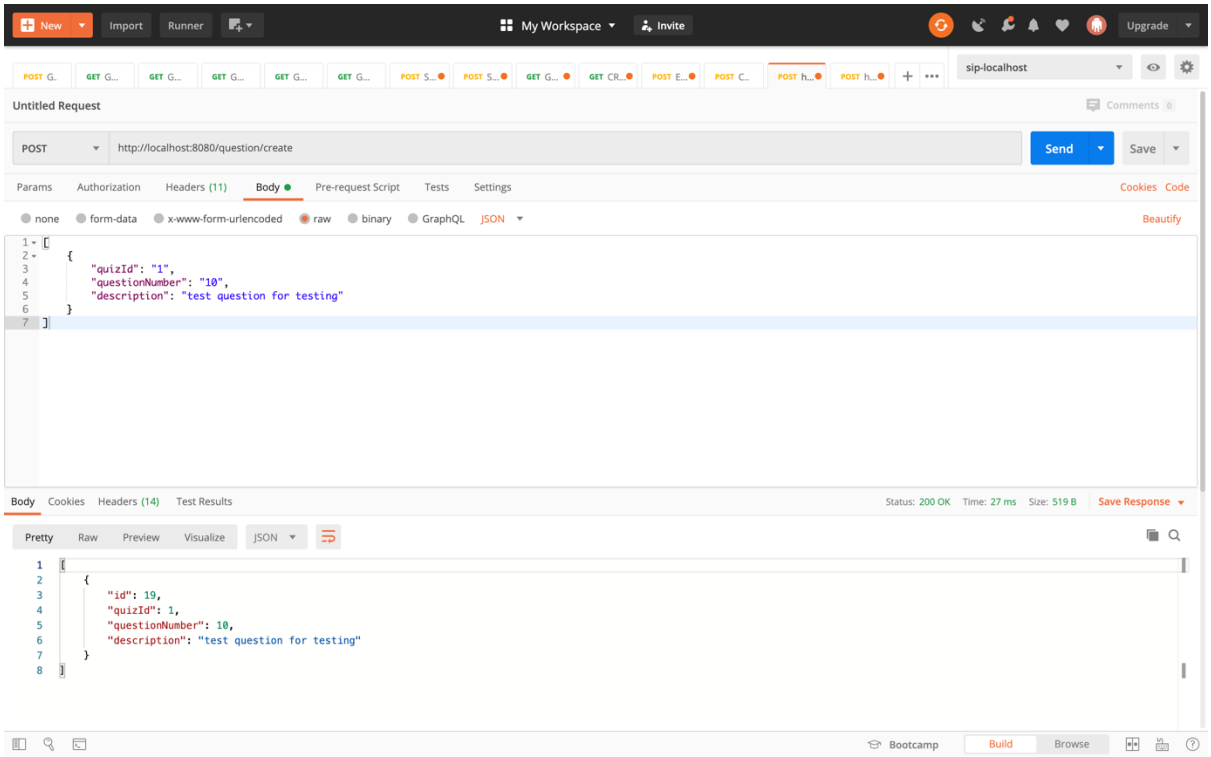
Endpoint	Test	Expected Outcome	Outcome	Pass/Fail
/question/update	Update question with invalid id	Return “questions not found”	Returned status: 204 no content	Fail
/question/update	Re-tested - Update question with invalid id	Return “questions not found”	Returned “questions not found	Pass – (After researching, changed the status to 400)
/question/delete	Delete existing question	Return “Question deleted”	Returned “Question deleted”	Pass
/question/delete	Delete non-existent question	Return “Questions not found”	Returned status: 204 no content	Pass
/question/delete	Re-tested - Delete non-existent question	Return “Questions not found”	Returned “Questions not found”	Pass – (After researching, changed the status to 400)



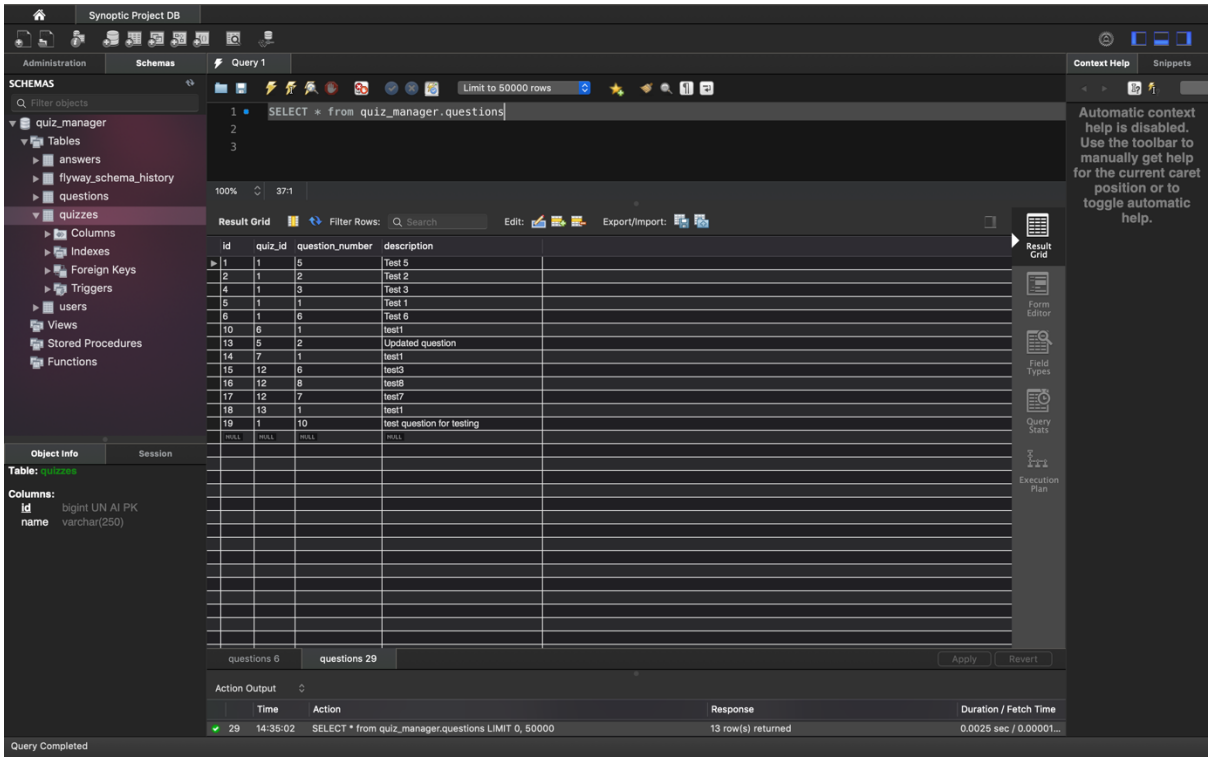
Question search results returned successfully



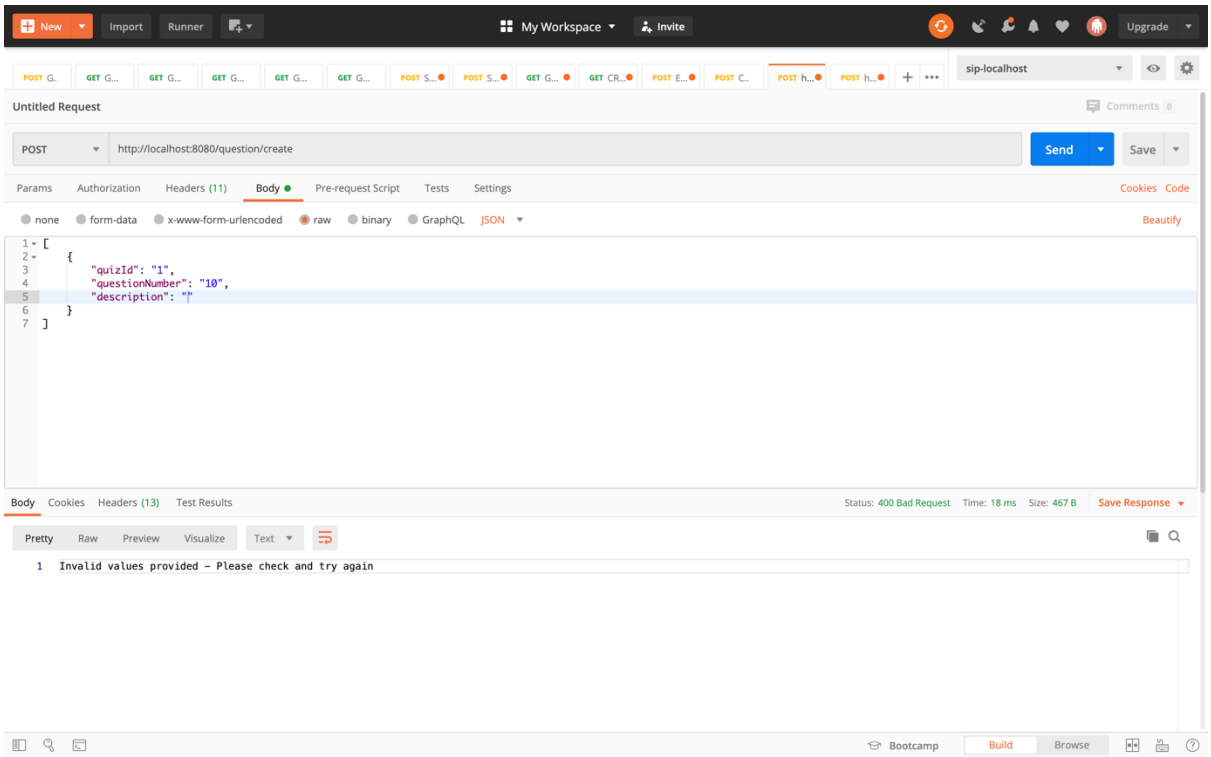
Status 204 returned when no content to return



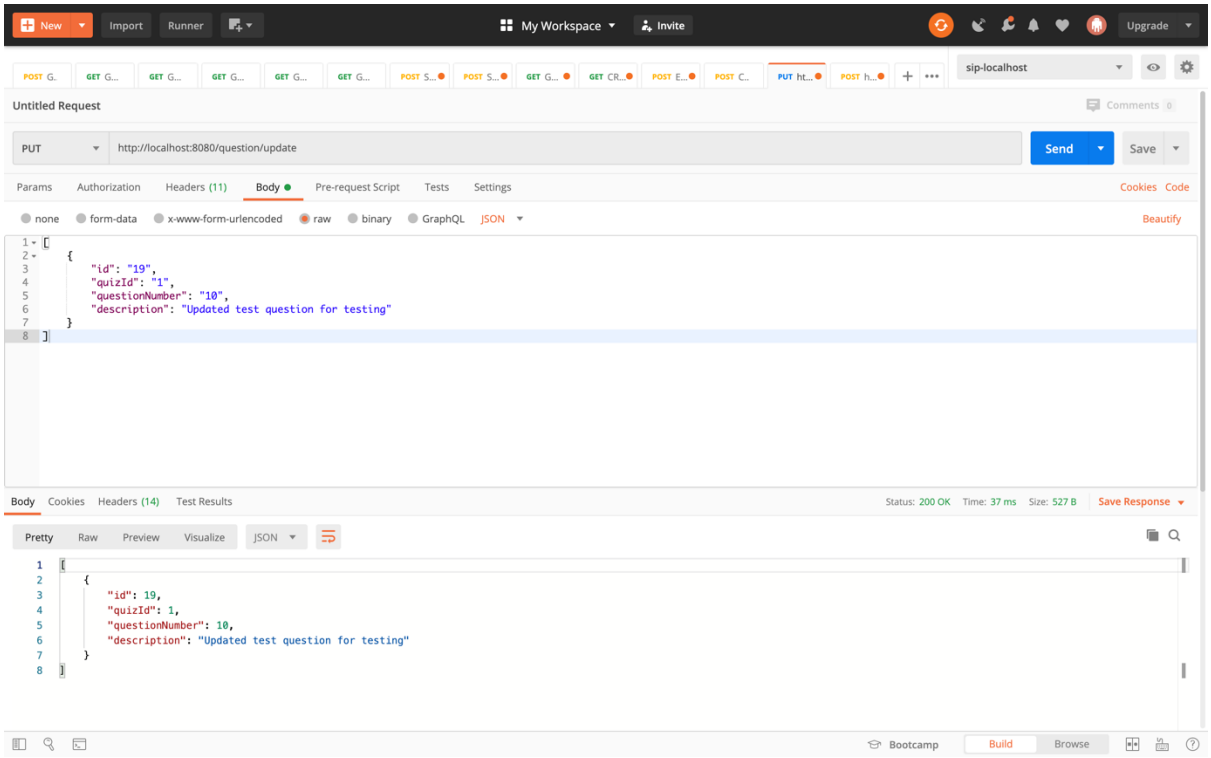
Question successfully created and question object returned within a list



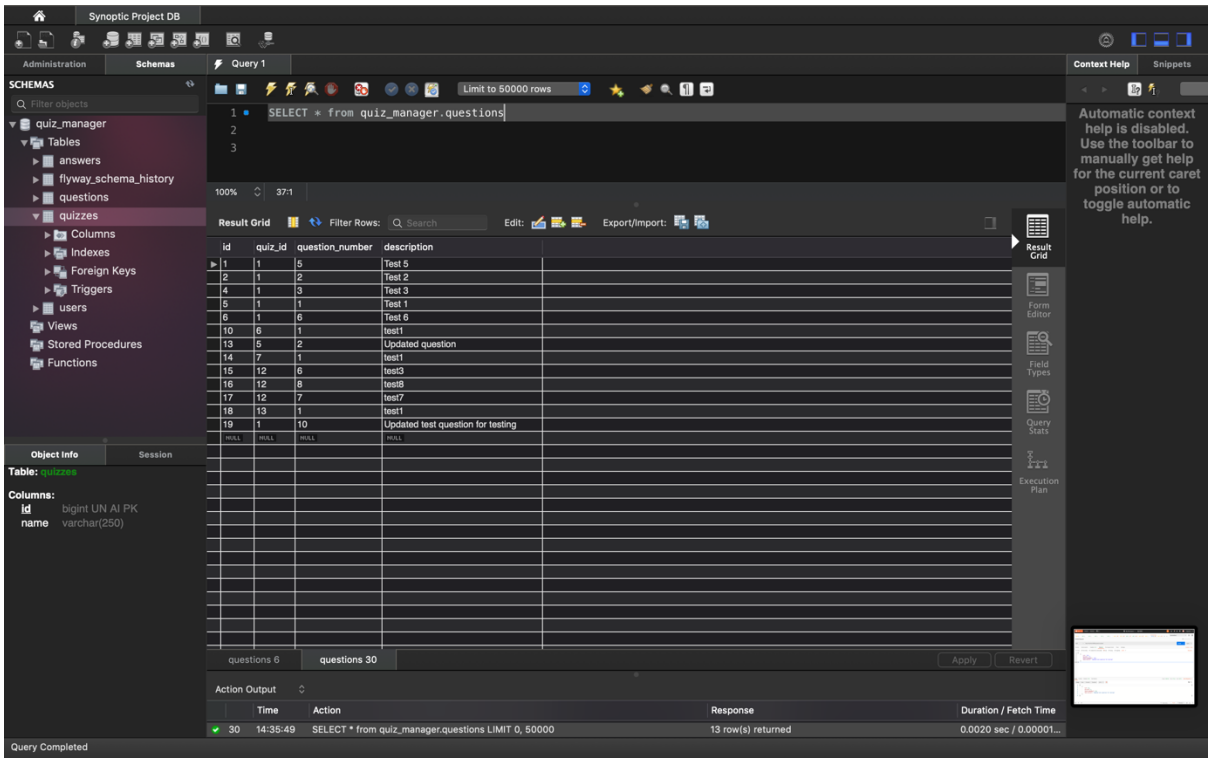
Question saved to database



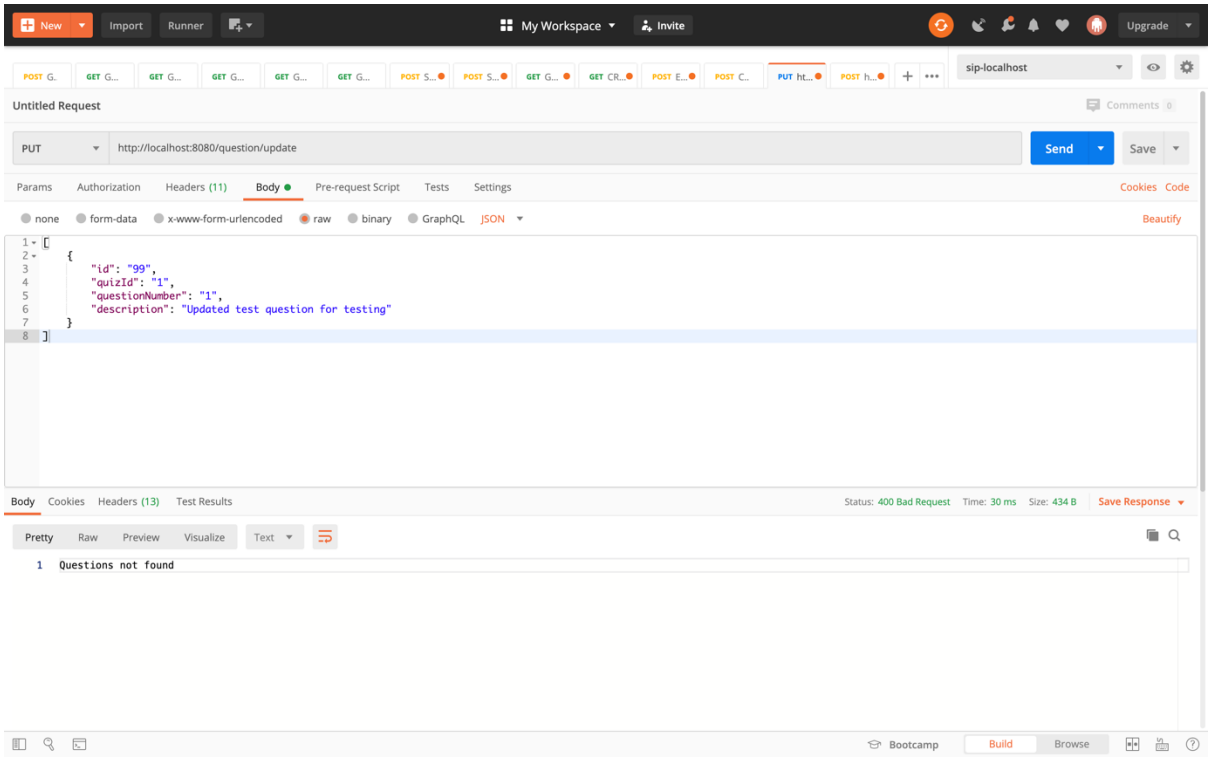
Invalid fields error message returned when fields provided are invalid



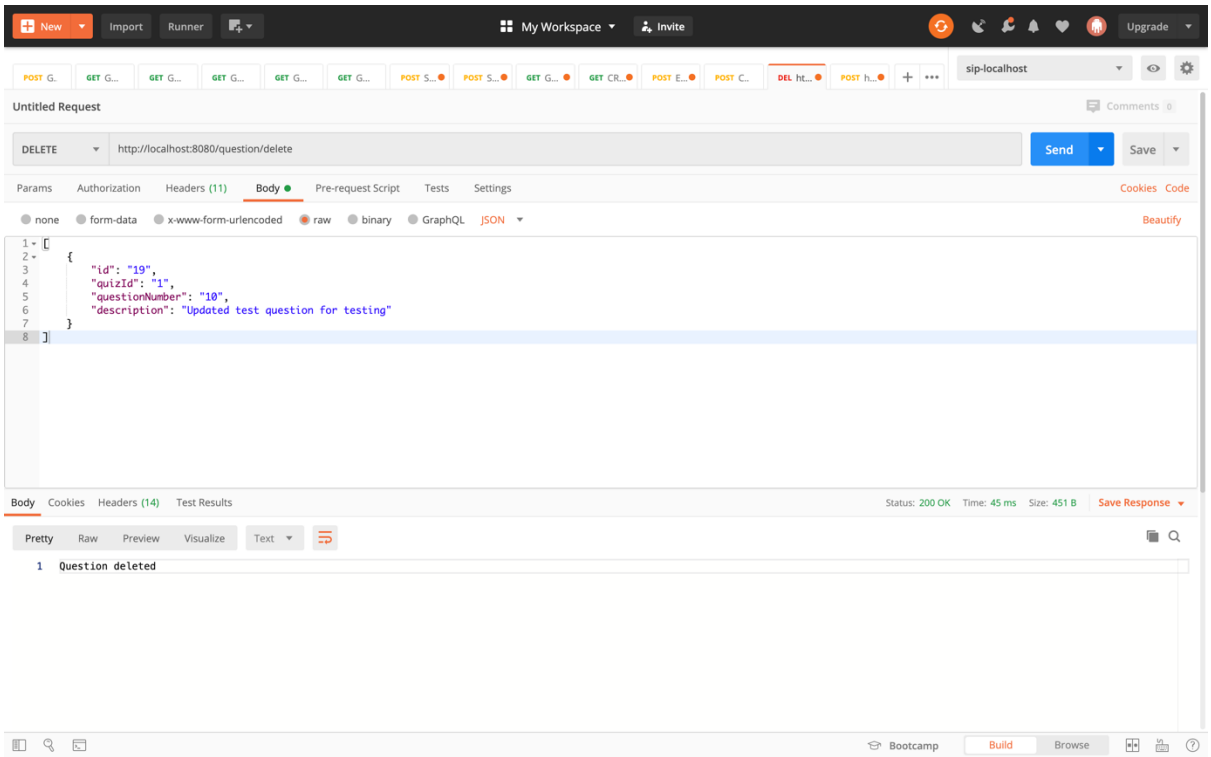
Question successfully updated and the updated question object is successfully returned as part of a list



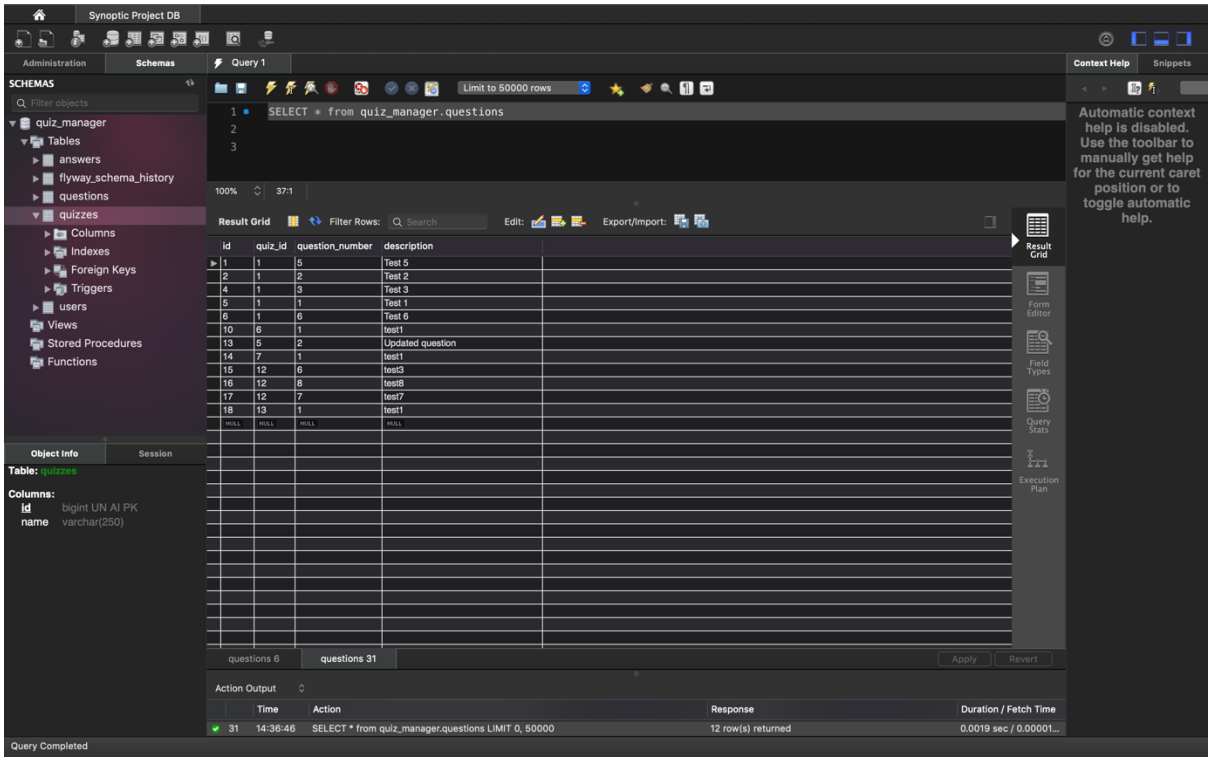
Question updated successfully in database



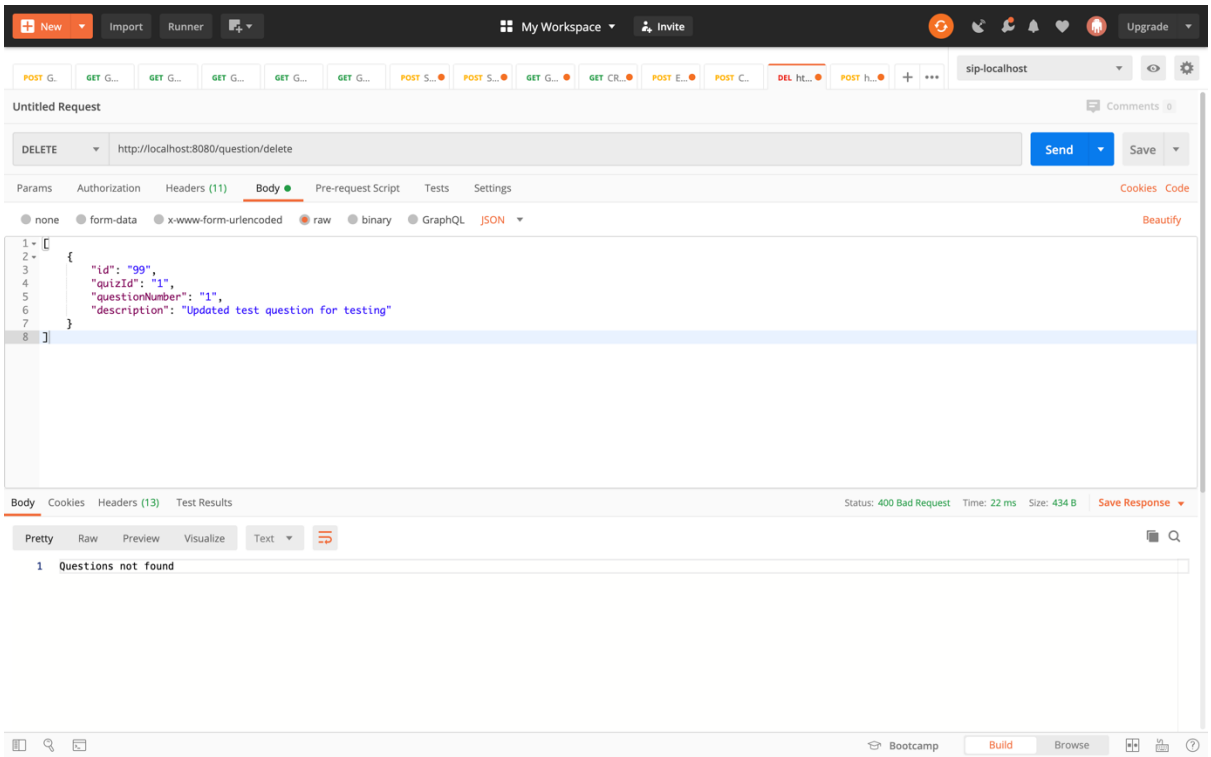
Error message returned when trying to update a quiz that does not exist



Quiz successfully deleted and success message returned in response



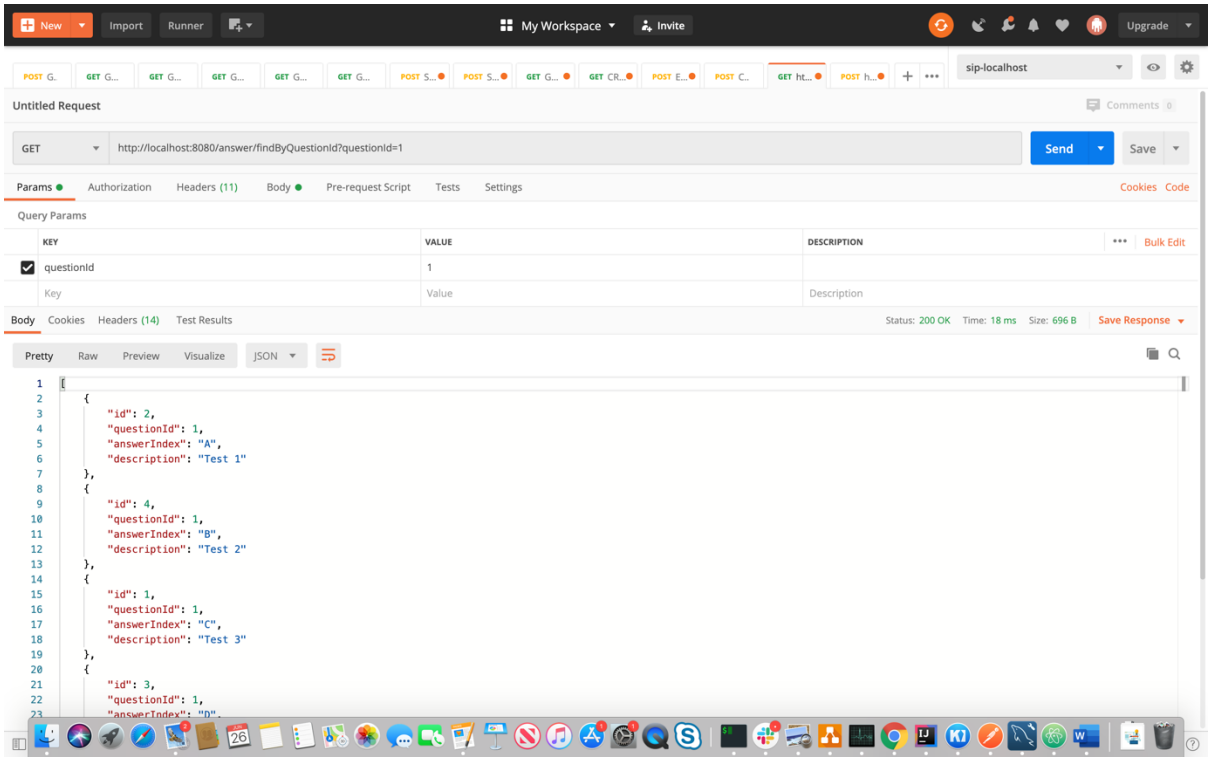
Quiz successfully deleted from the database



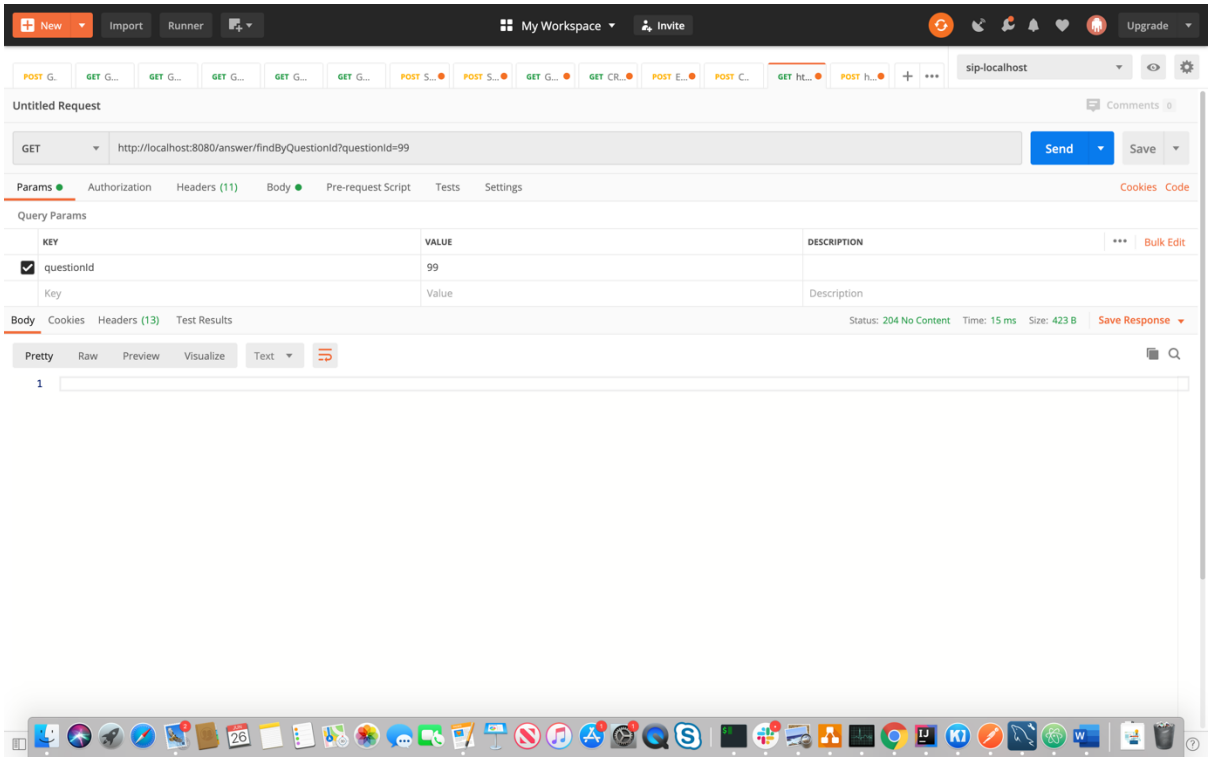
Error message returned when trying to delete a quiz that does not exist

ANSWER ENDPOINT

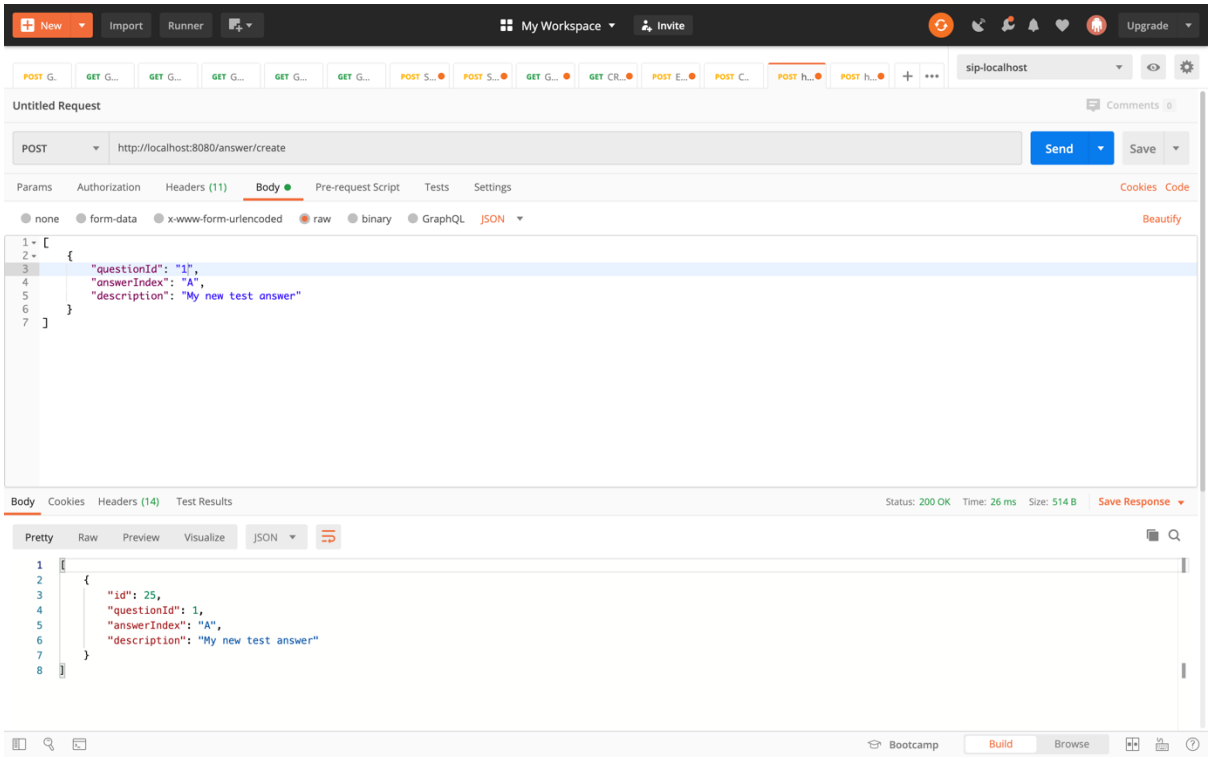
Endpoint	Test	Expected Outcome	Outcome	Pass/Fail
/answer/findByQuizId	Find answer using existing question id	Return list of answers	Returned list of answers	Pass
/answer/findByQuizId	Find answers using non-existent question id	Return "This answer currently has no questions"	Status: 204 no content	Pass
/answer/create	Create answer with valid fields	Return list of answers (one answer in this case)	Returned list of answers (one answer in this case)	Pass
/answer/create	Create answer with invalid fields	Return "Invalid values provided – Please try again"	Returned "Invalid values provided – Please try again"	Pass
/answer/update	Update answer with valid fields	Return list of answers (one answer in this case)	Returned list of answers (one answer in this case)	Pass
/answer/update	Update answer with invalid fields	Return "answers not found"	Status: 204 no content	Fail
/answer/update	Re-tested - Update answer with invalid fields	Return "answers not found"	Return "answers not found"	Pass – (After researching, changed the status to 400)
/answer/delete	Delete existing answer	Return "answer deleted"	Returned "answer deleted"	Pass
/answer/delete	Delete non-existent answer	Return "answers not found"	Status: 204 no content	Fail
/answer/delete	Re-tested - Delete non-existent answer	Return "answers not found"	Return "answers not found"	Pass – (After researching, changed the status to 400)



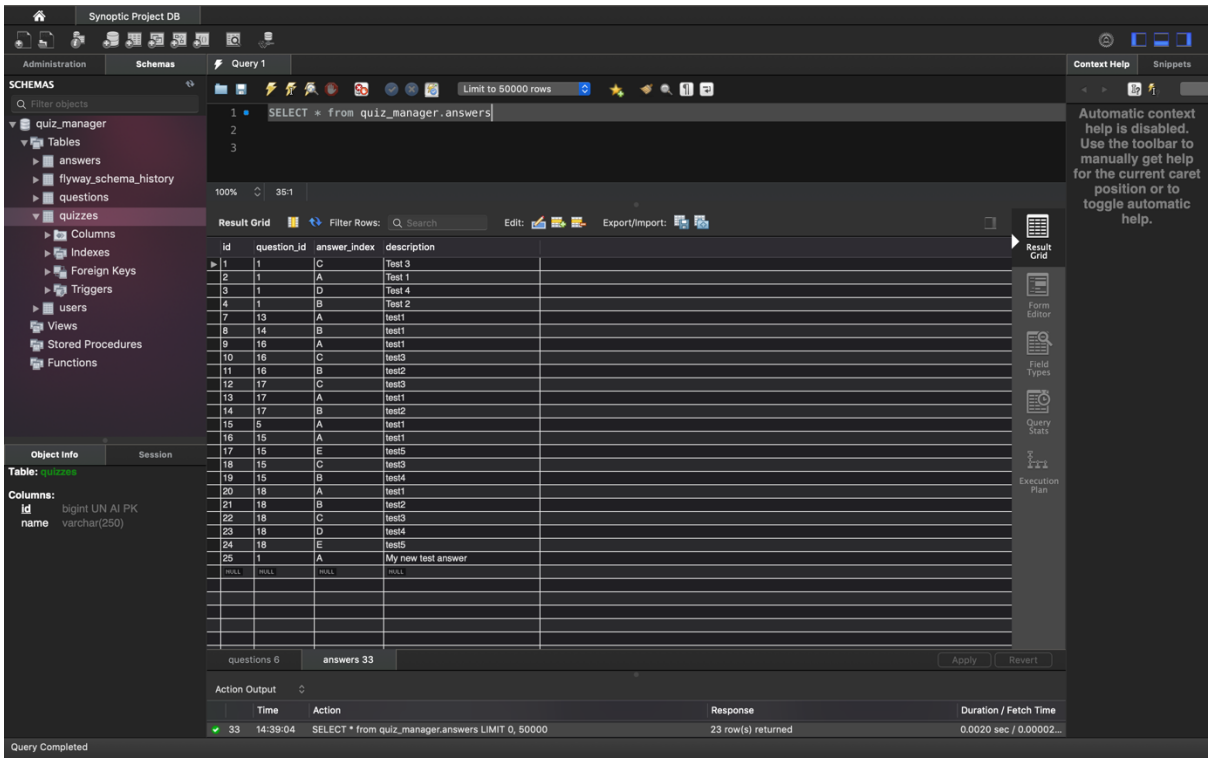
List of answer objects successfully found and returned after search request made



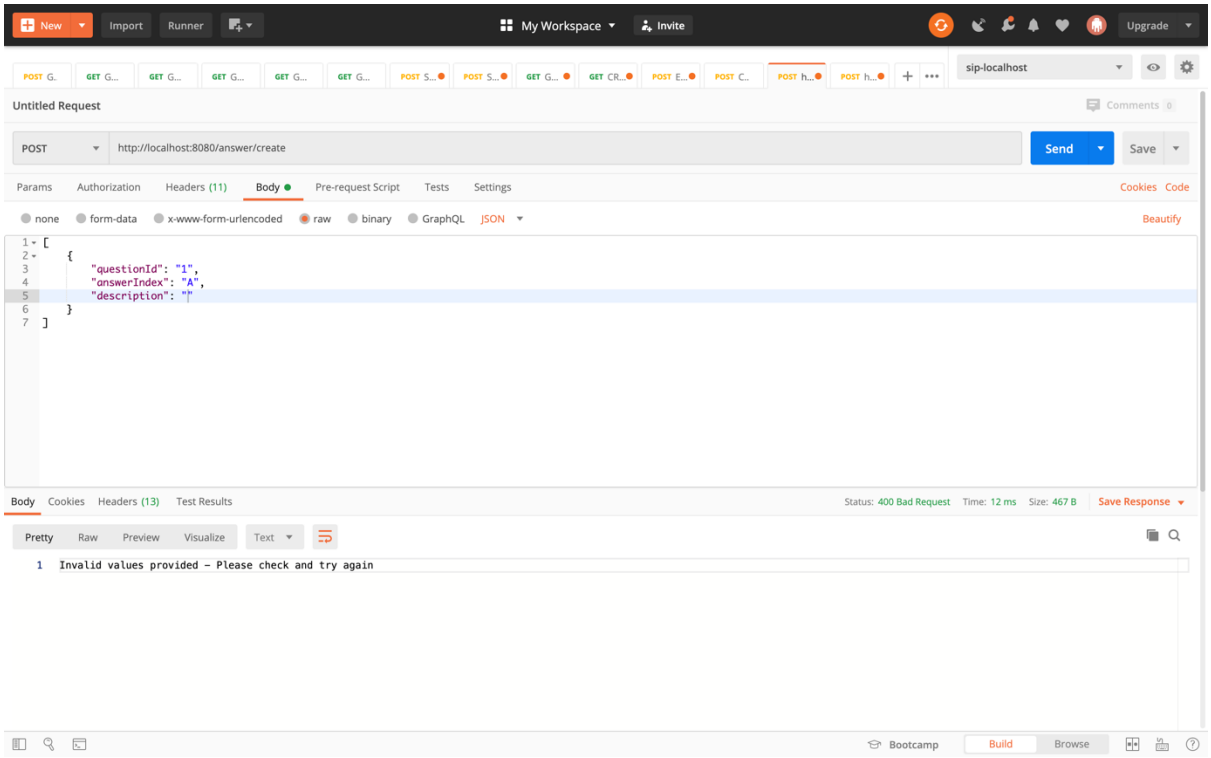
Status 204 received as response when there is no content to return following search



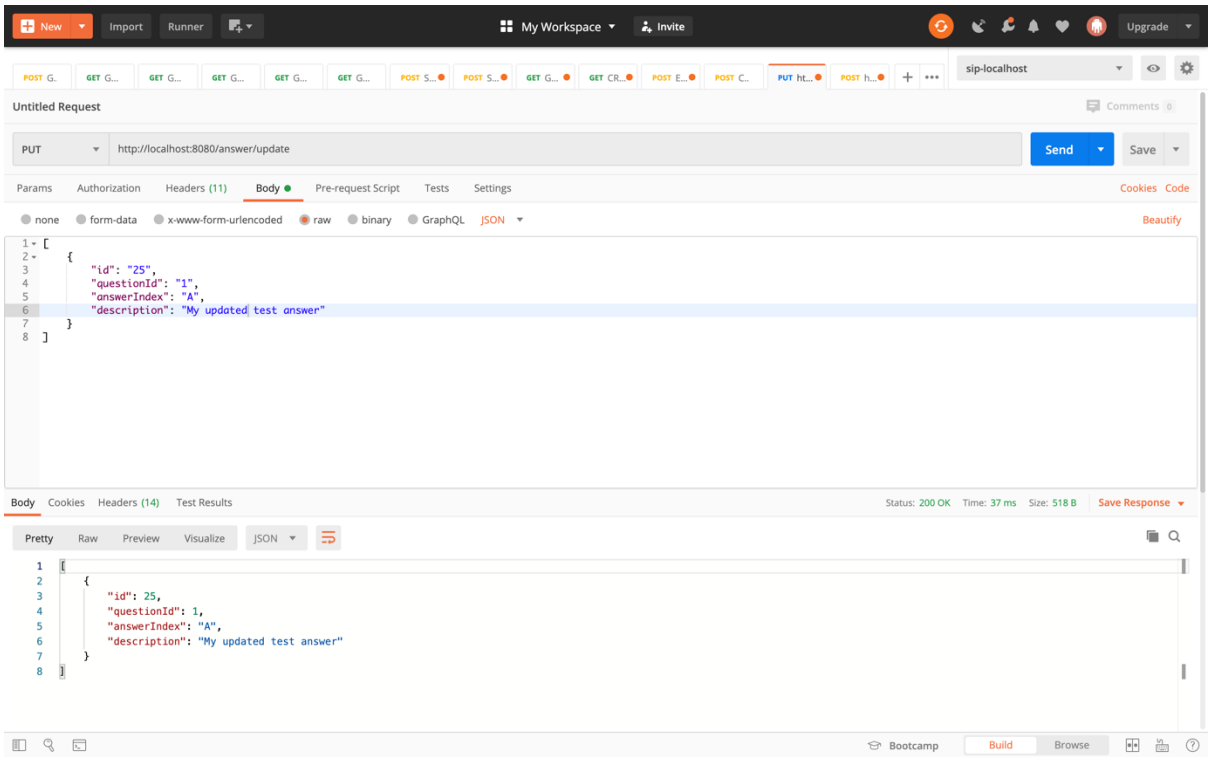
Answer object returned after successful ‘create answer’ request



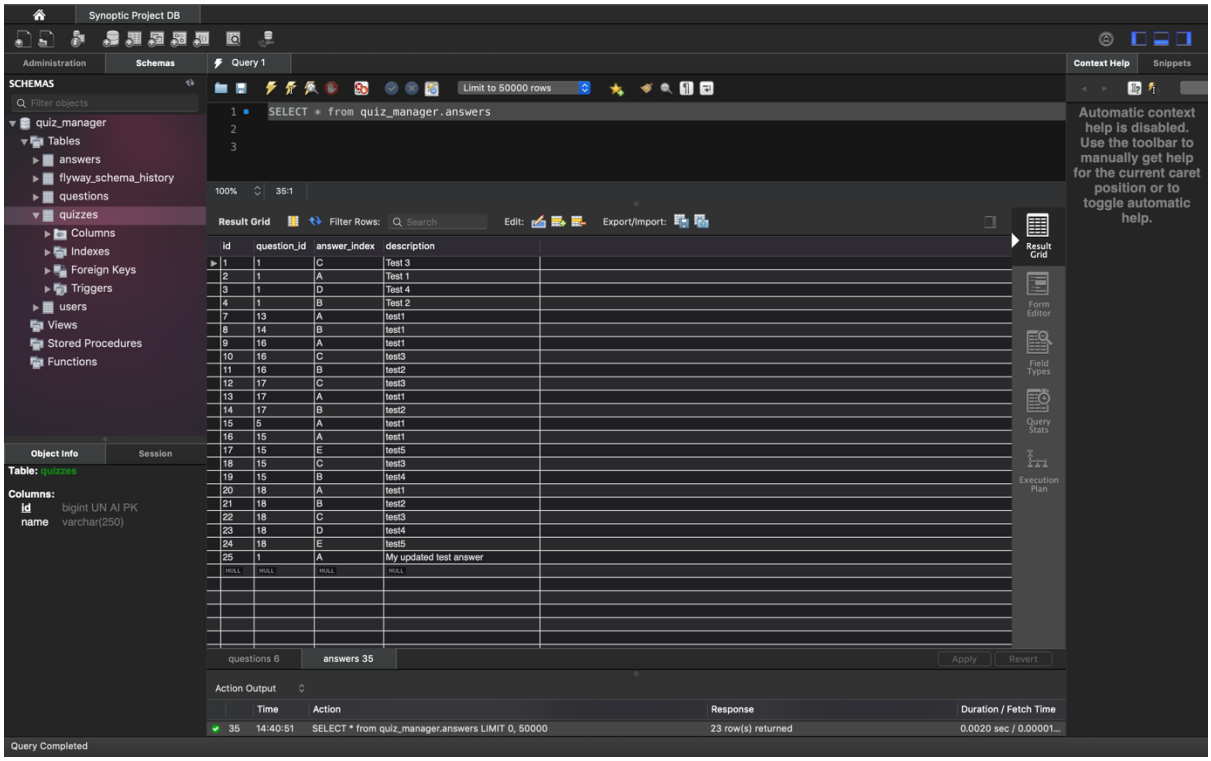
Answer saved to database



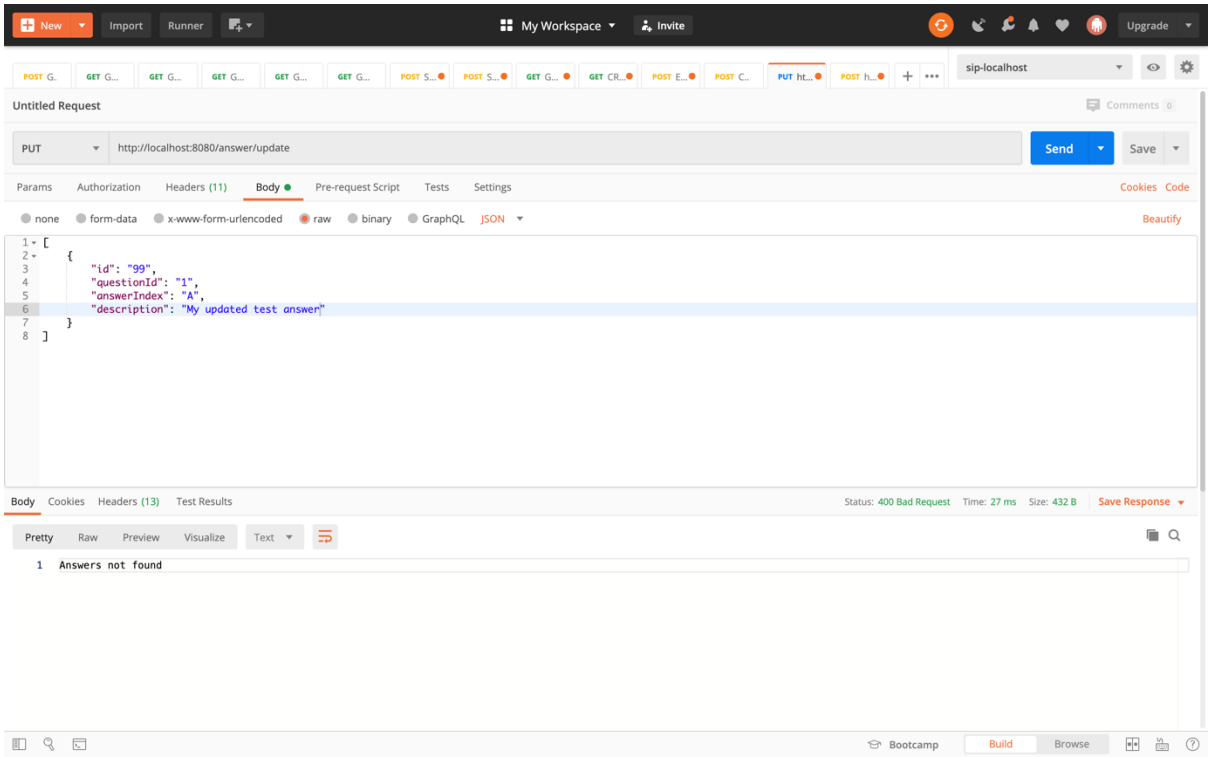
Invalid values error returned after invalid values received through create answer request



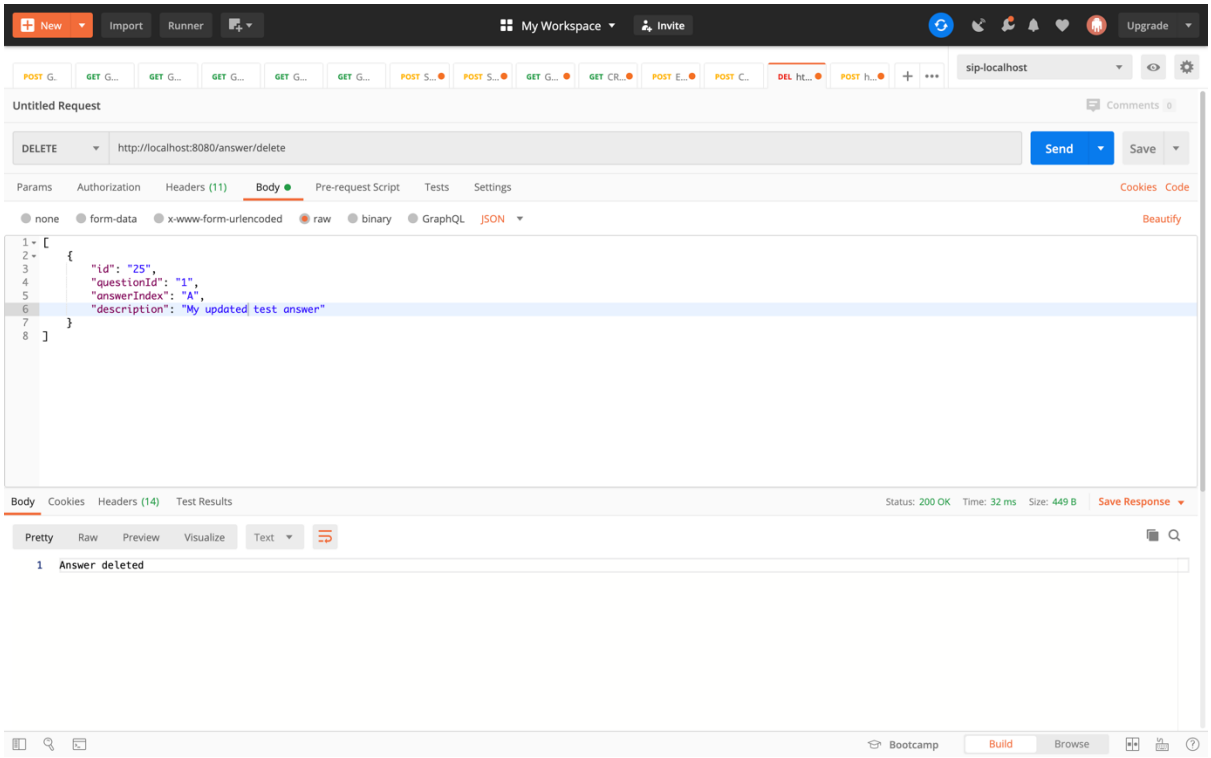
Updated answer object returned within a list following successful update request



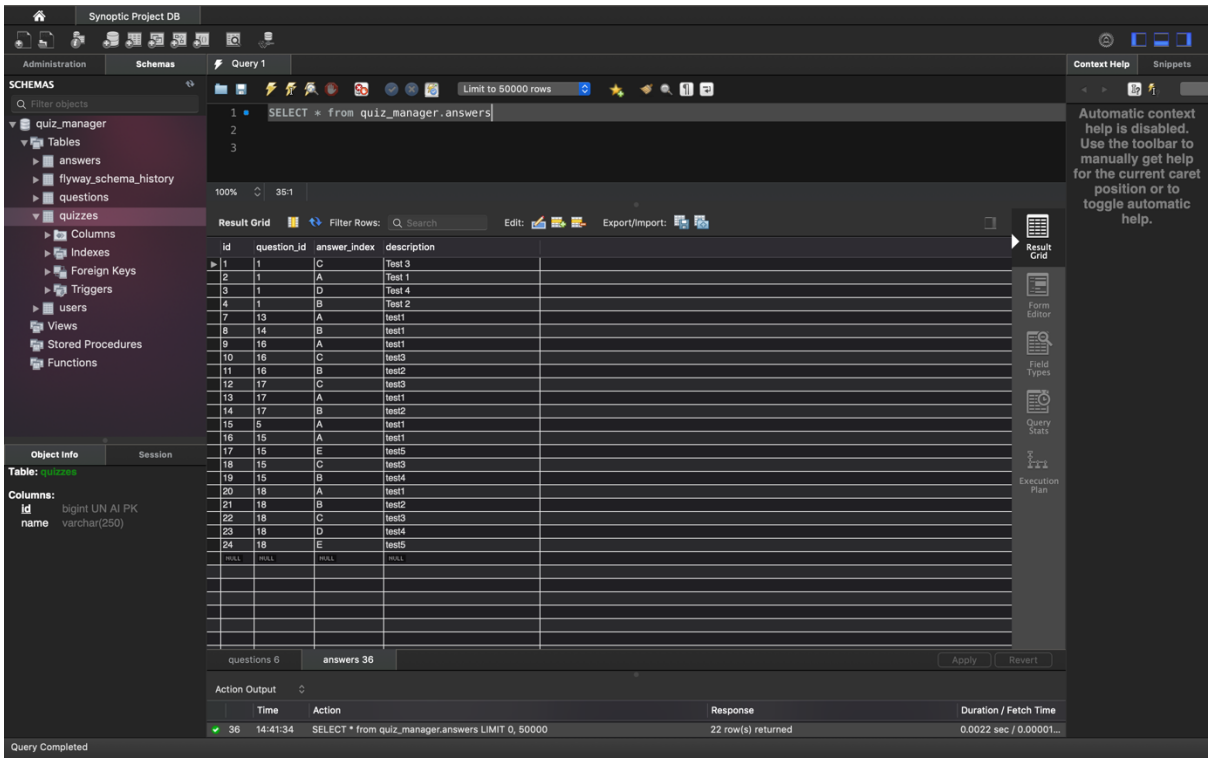
Answer updated in database



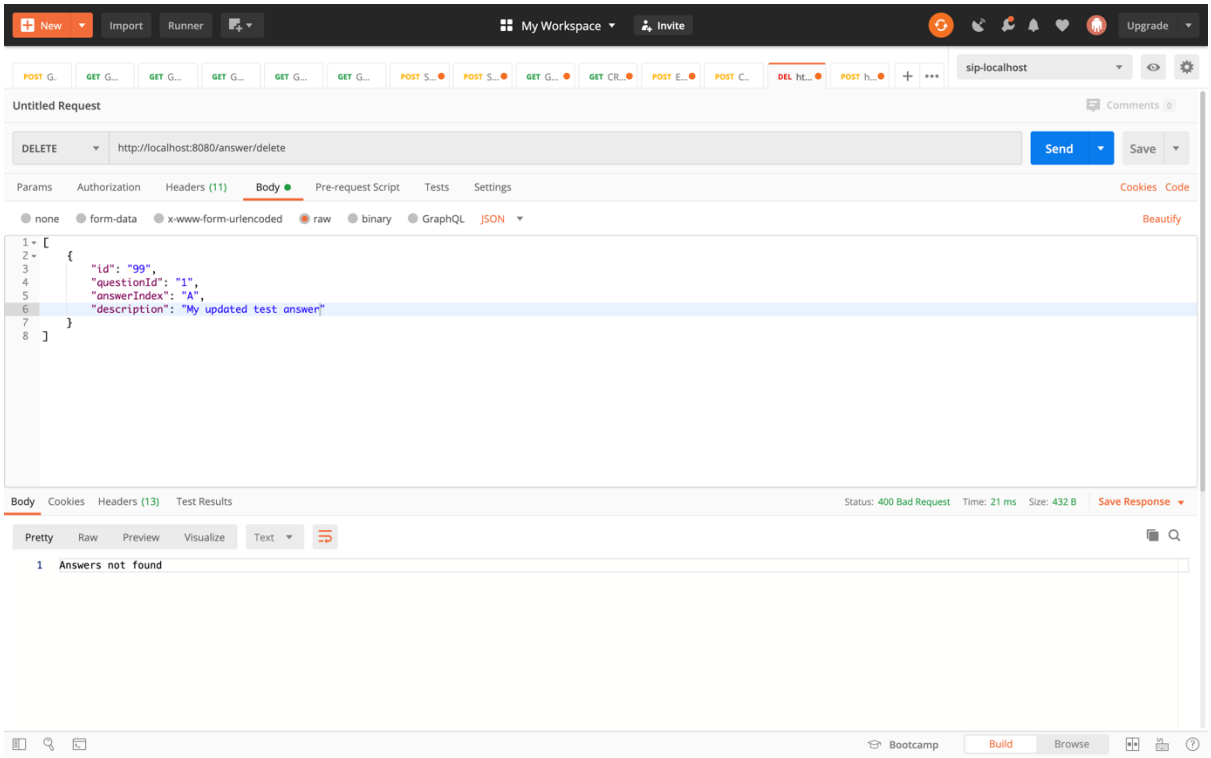
Error message received in response after attempting to update a non-existent answer



Success message received after successfully ‘delete answer’ request made



Answer successfully deleted from the database



Error message received in response after attempting to delete a non-existent answer

While testing I found that there were a number of cases where the test failed and I was not receiving an error message/ no content message as expected. After reading more online about using the status 204, it seemed that the status was definitely the right status to use for the search endpoints because 204 is a success message and where there is no content to return does not mean there has been an error with the search. I decided that this would be handled on the front end by doing a check for status 204 and then rendering a notification to handle this.

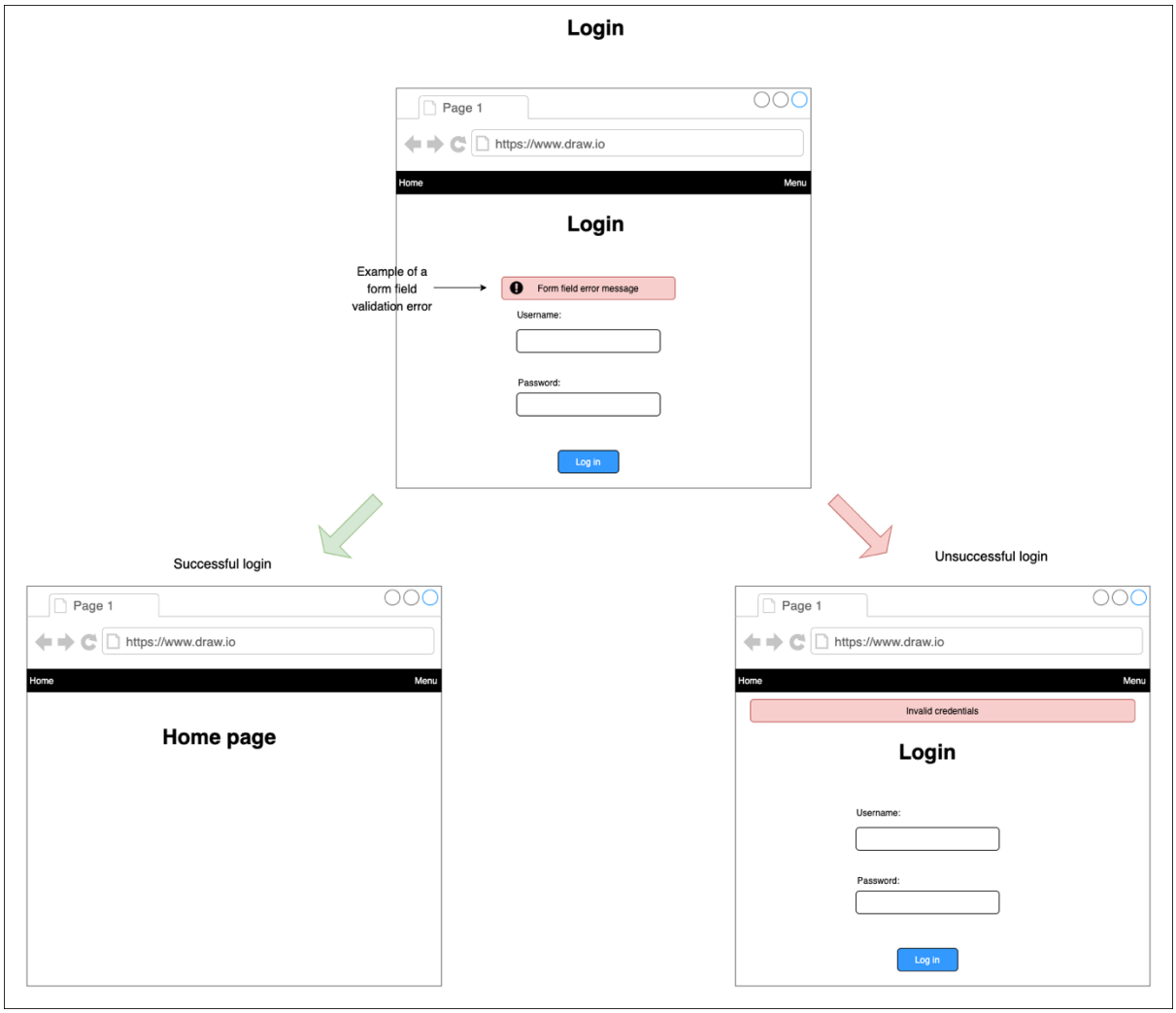
The other endpoints that were failing due to the status 204 response were the update and delete endpoints. In these cases, I decided that the 204 response was the wrong response to use because attempting to delete or update a non-existent quiz/question or answer should result in an error. I decided that the 400 bad request response was the best fit for these scenarios and they also allowed me to return a string message within the response which the front end could use to render a notification to the user.

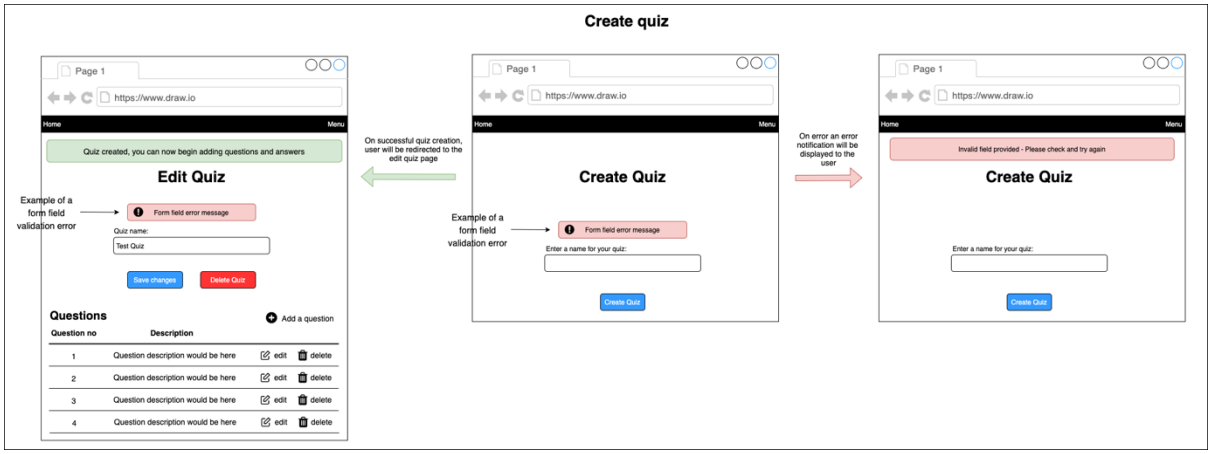
I retested the endpoints and noted that they now passed doe to the changes made. Unfortunately, I ran out of time to test the invalid field errors for some of the endpoints but if I had more time, I would ensure these were manually tested and tested within the integration tests.

DESIGN – FRONT END

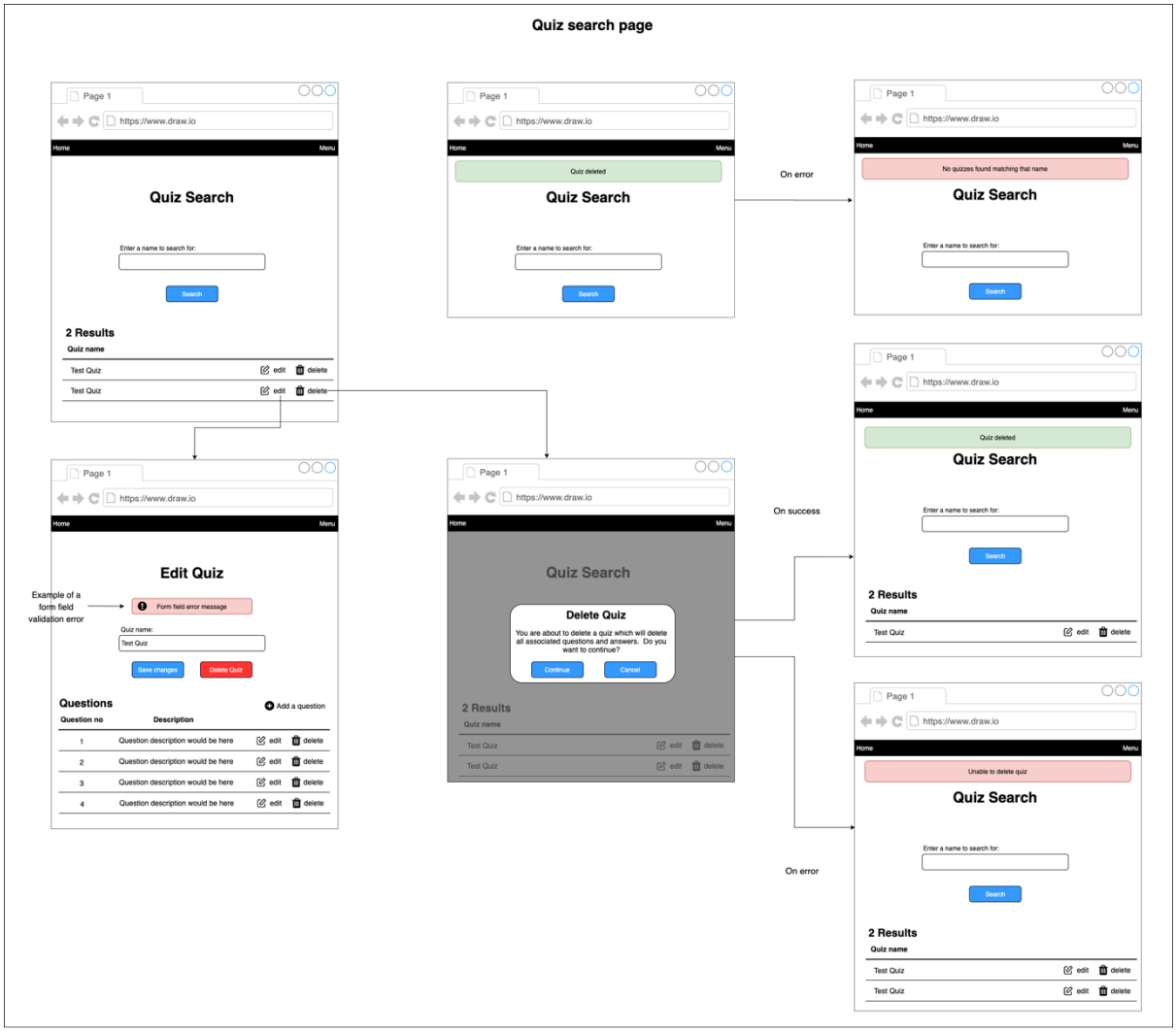
MOCKUPS

Below is the mock-up for the login page. On success the user will be redirected to the home page and a notification will render to let them know they logged in successfully. If the credentials are invalid, the user remains on the login page and a notification is rendered to let them know they credentials were invalid.

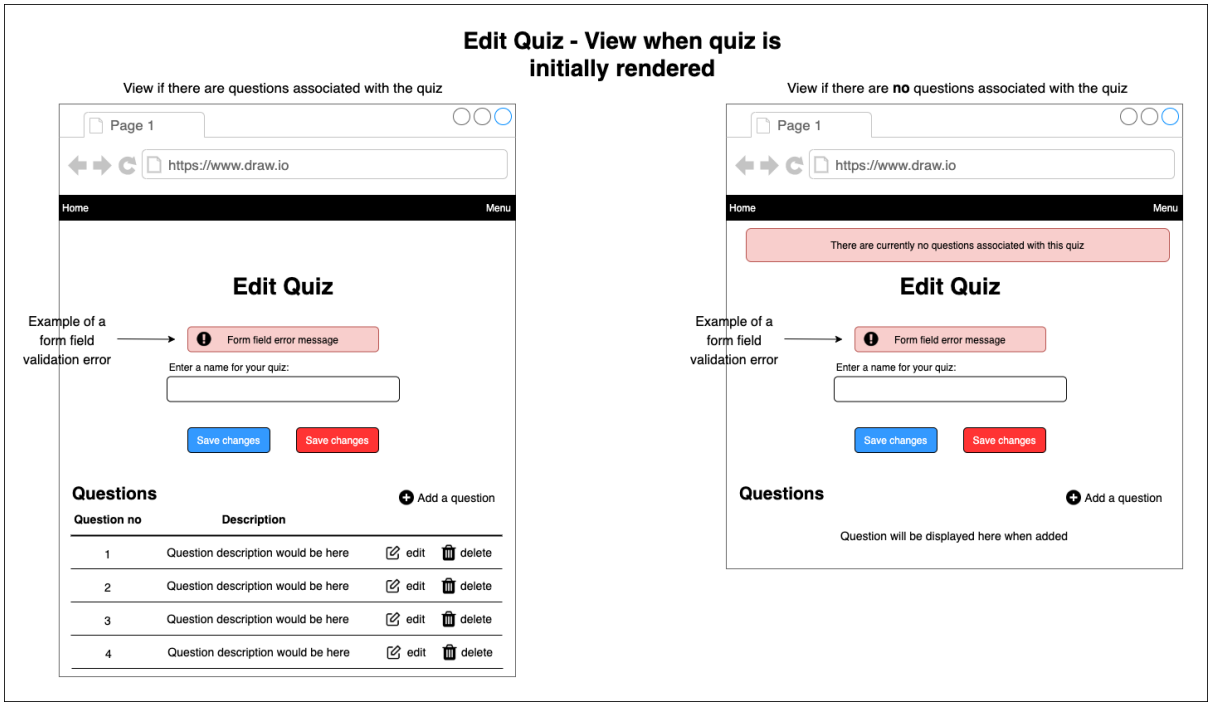




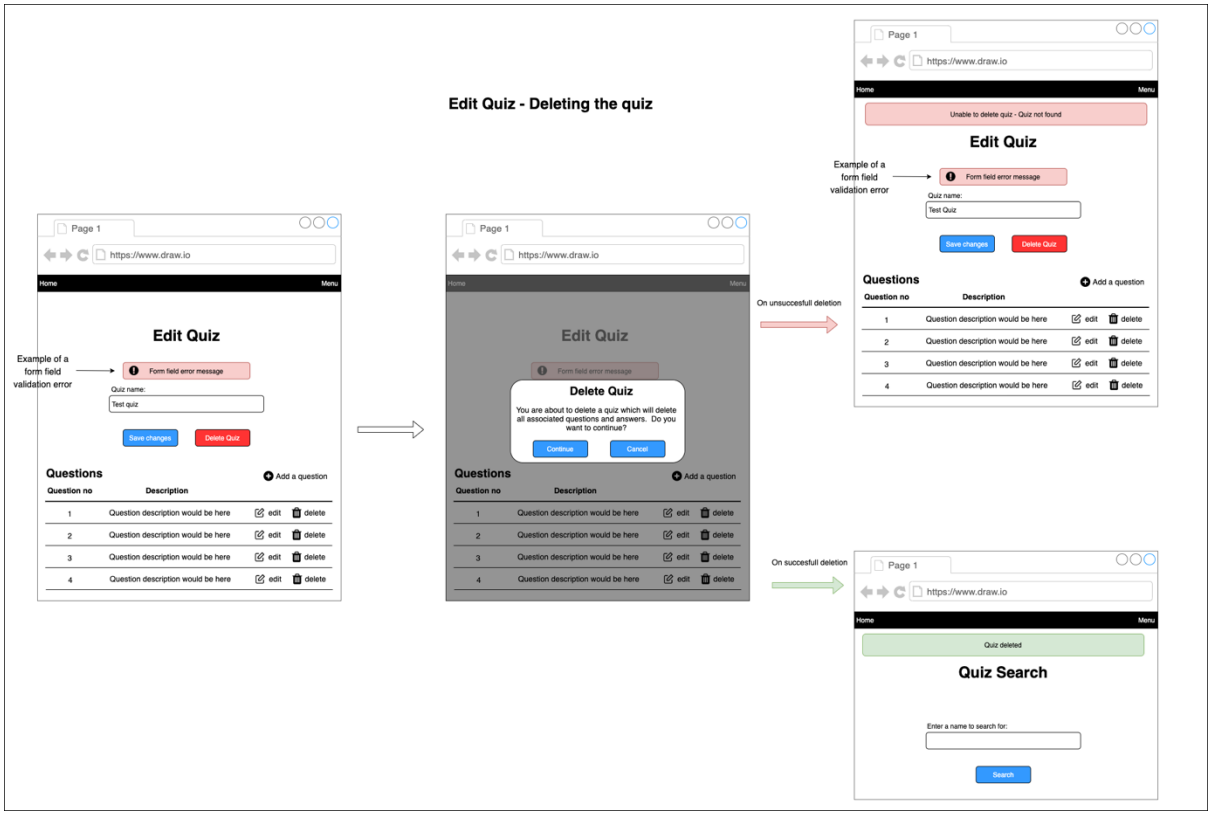
Above shows my design of the create quiz page. On error a notification is rendered and on success the user would be taken to the edit quiz page and the quiz they created would be loaded. N.B there would be no questions at that stage.



Above is the quiz search page. Here I have mocked up how the page would be rendered with results if the search is successful. Each result would have an edit and delete option and the diagrams show how they would be handled. The mock-ups also show how the errors would potentially be handled.

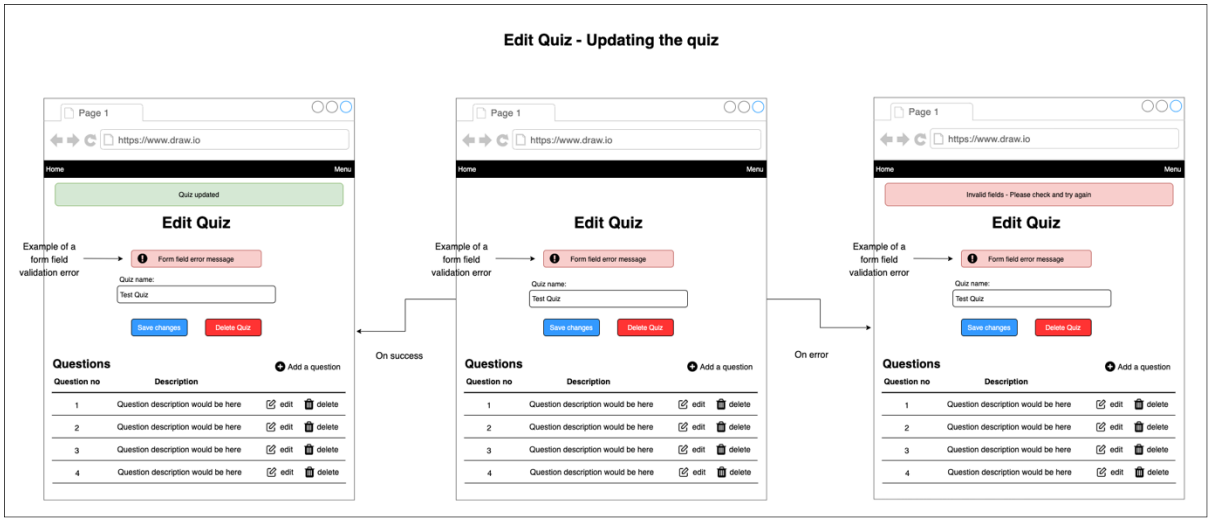


Above, the mock-ups show how the edit quiz page would be rendered when the quiz is initially made and when how it will look once some questions have been added.

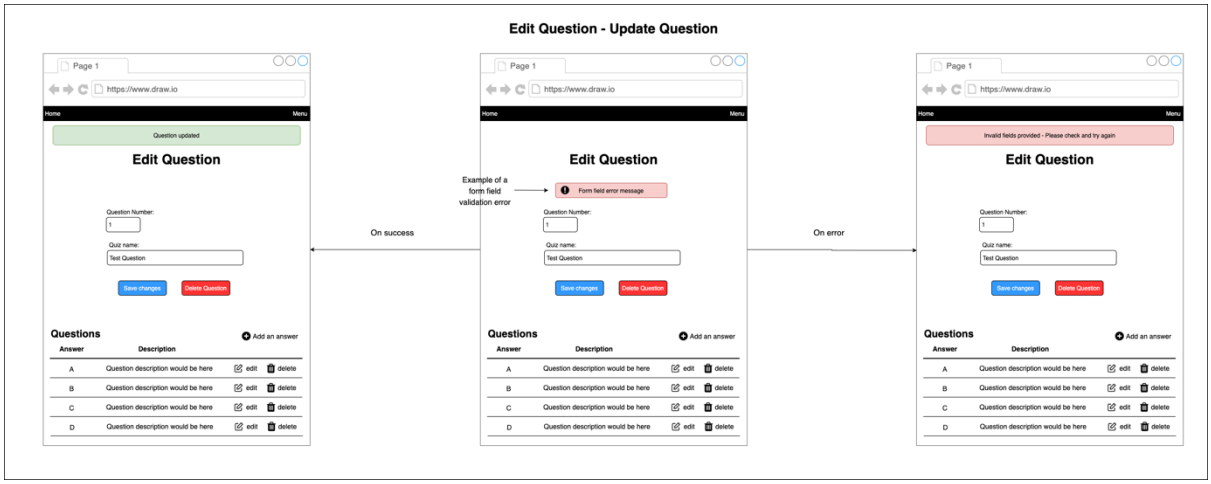


Above shows a breakdown of the process for deleting a quiz. I have mocked up the modal that I wanted to use as a way of preventing the users from accidentally deleting a quiz. The mock-ups also show how error and success would be handled.

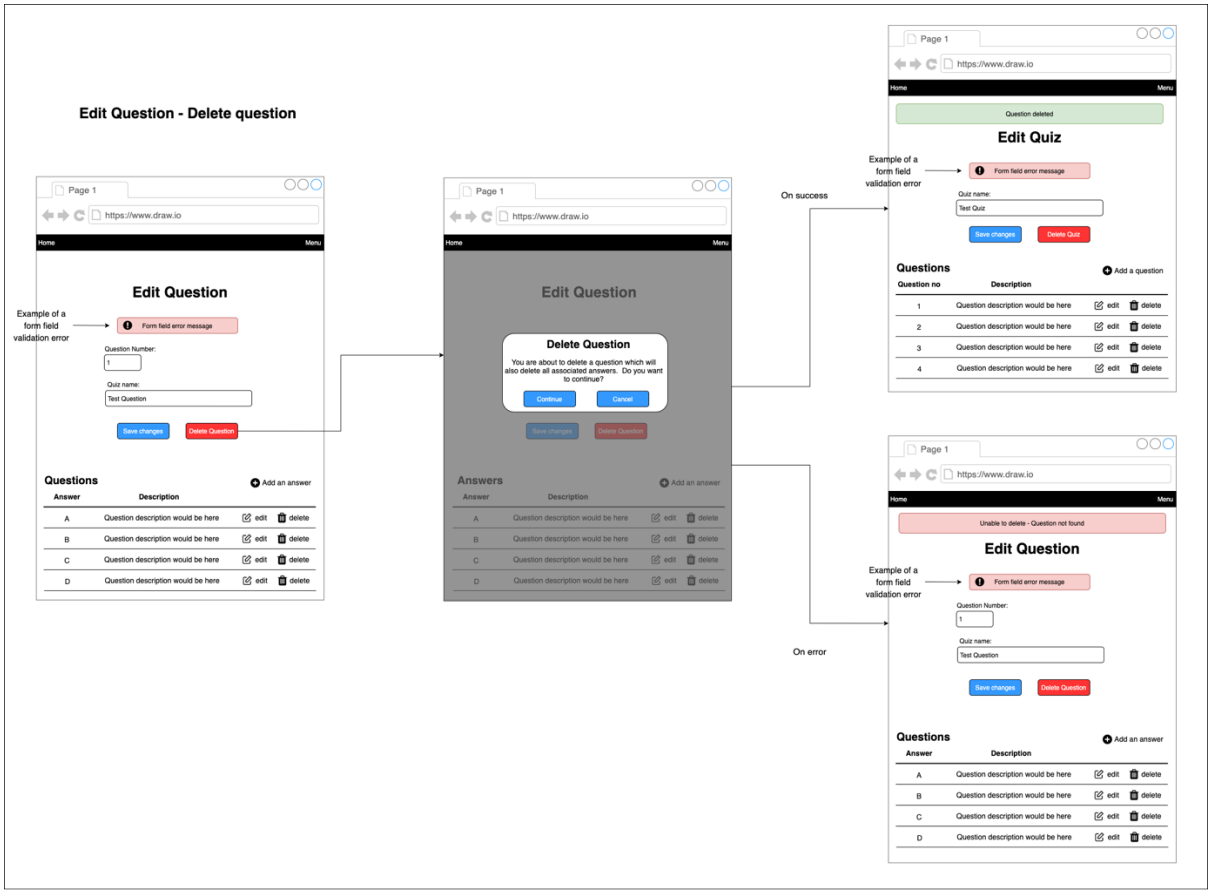
The mock-ups above show how the different options on the 'edit quiz' page would be handled. This includes the page for creating a new question, the page that would render when a question's 'edit' button is clicked. I have also tried to mock up the process that is carried out when a user clicks a question's delete option.



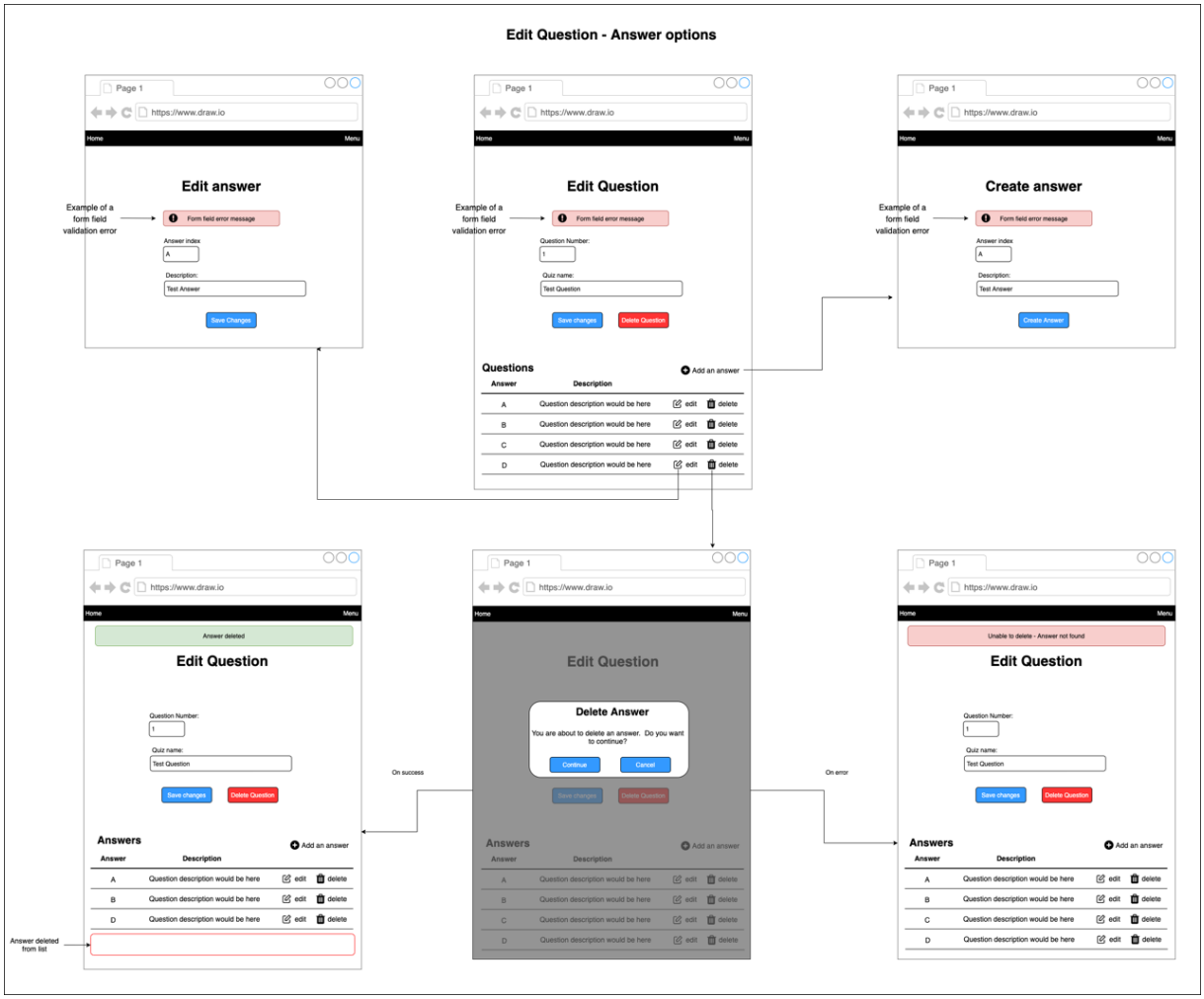
The above mock-ups show the notification and error messages used in the process of saving changes to a quiz.



The above mock-ups show the notification and error messages used in the process of saving changes to a question.



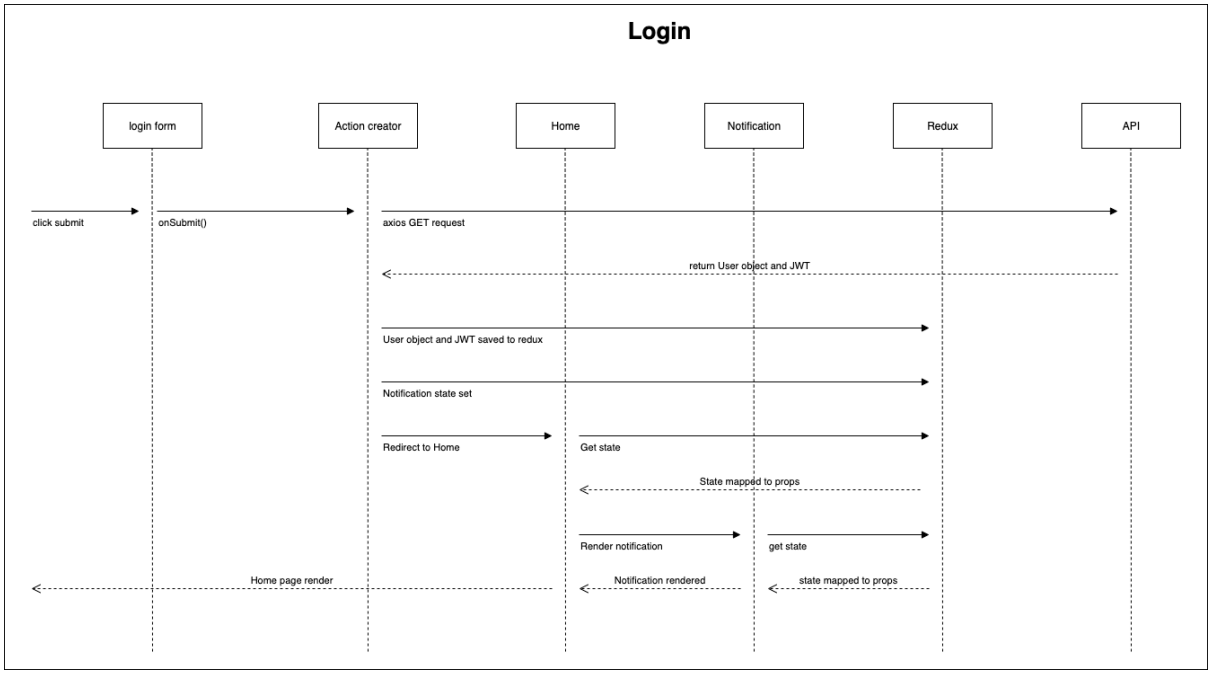
Above shows a breakdown of the process for deleting a question. I have mocked up the modal that I wanted to use as a way of preventing the users from accidentally deleting a quiz. The mock-ups also show how error and success would be handled.



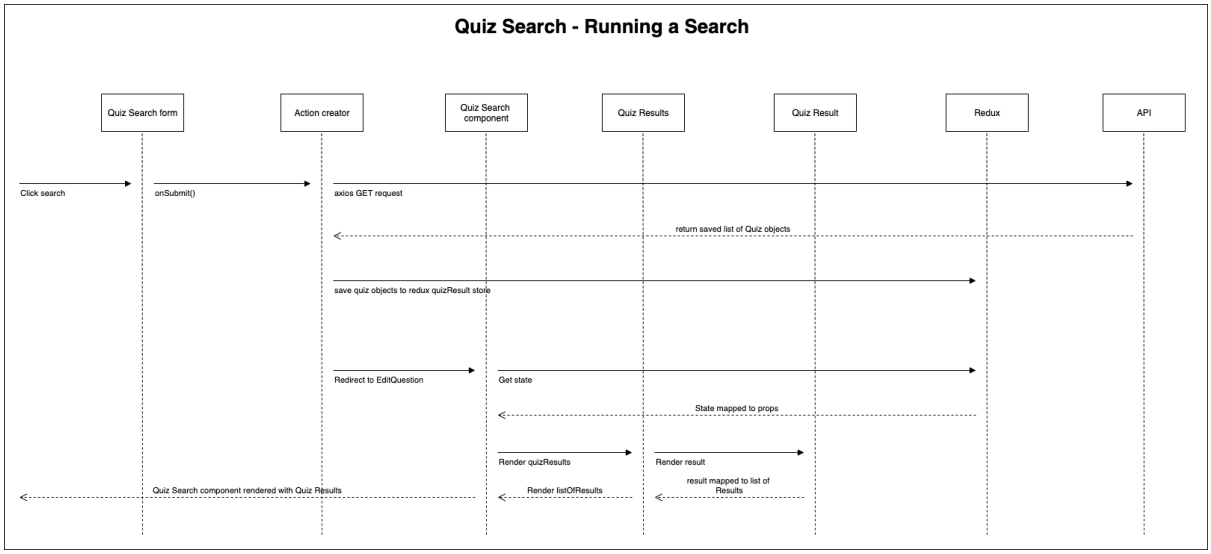
The mock-ups above show how the different options on the ‘edit question’ page would be handled. This includes the page for creating a new answer, the ‘edit answer’ page that would render when an answer’s ‘edit’ button is clicked. I have also tried to mock-up the process that is carried out when a user clicks a question’s delete option.

UML SEQUENCE DIAGRAMS

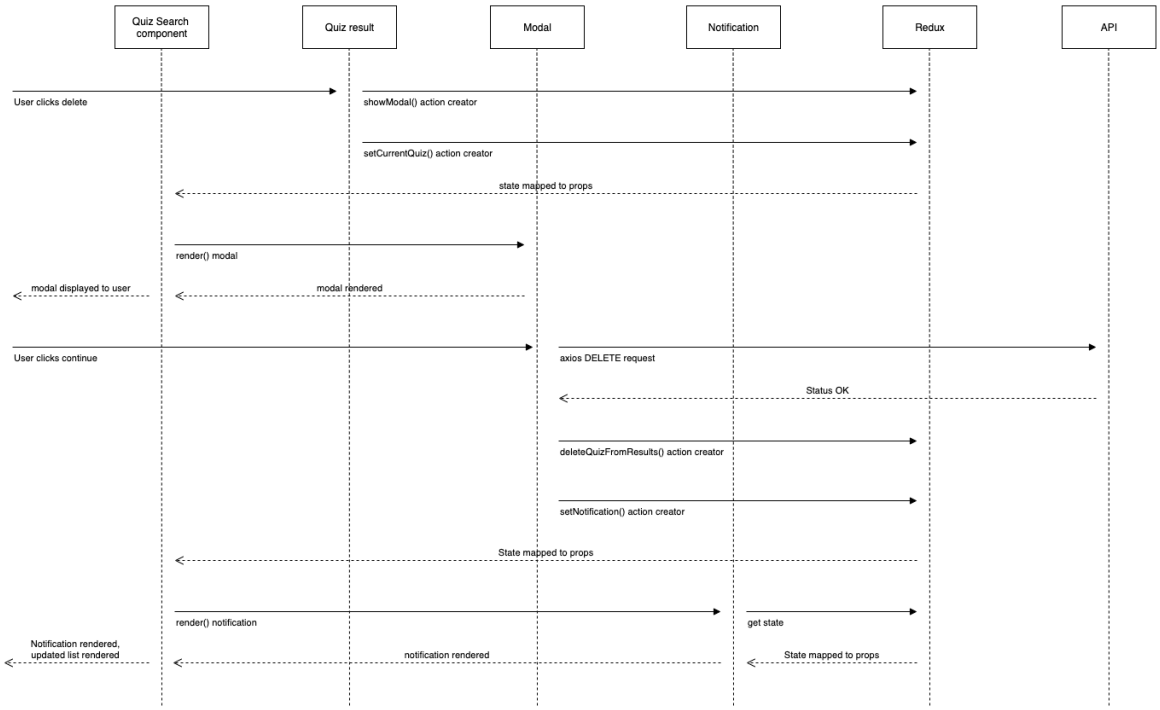
LOGIN



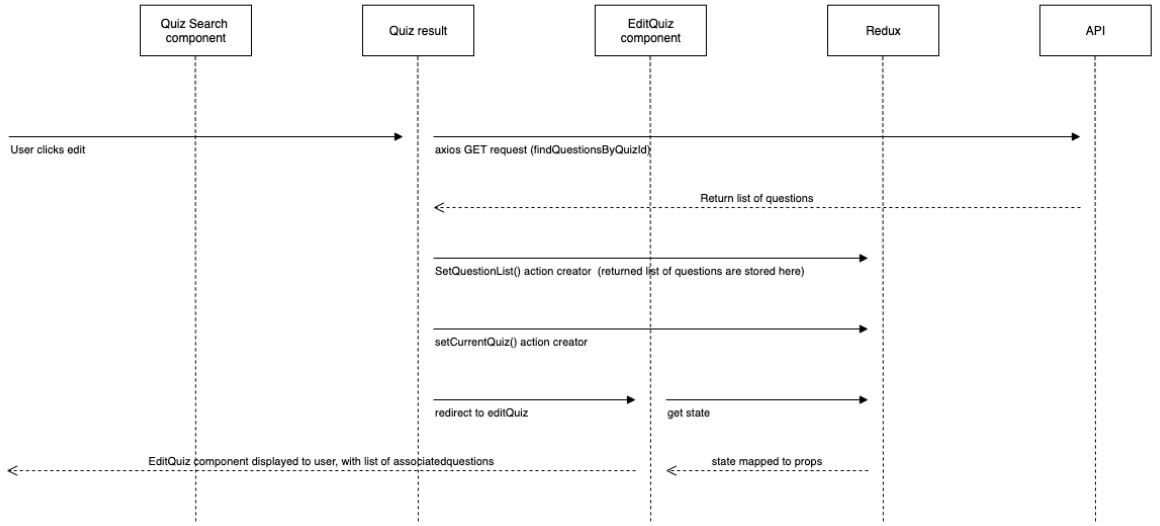
QUIZ SEARCH



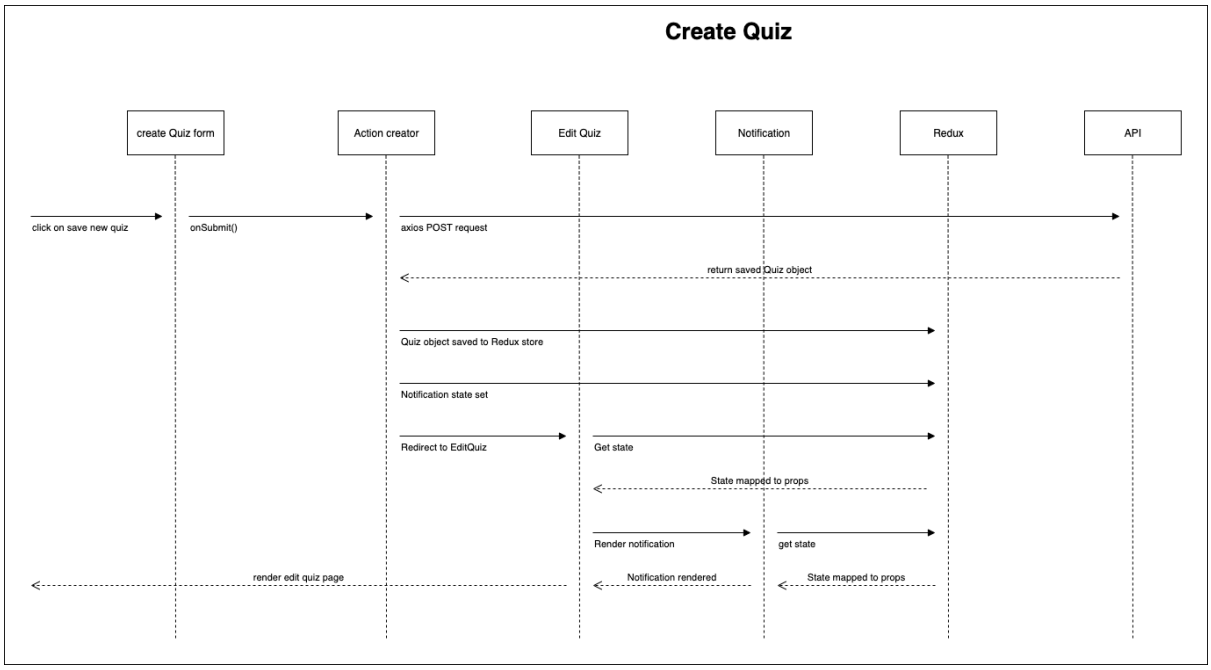
Quiz Search - Deleting a Quiz via search result option



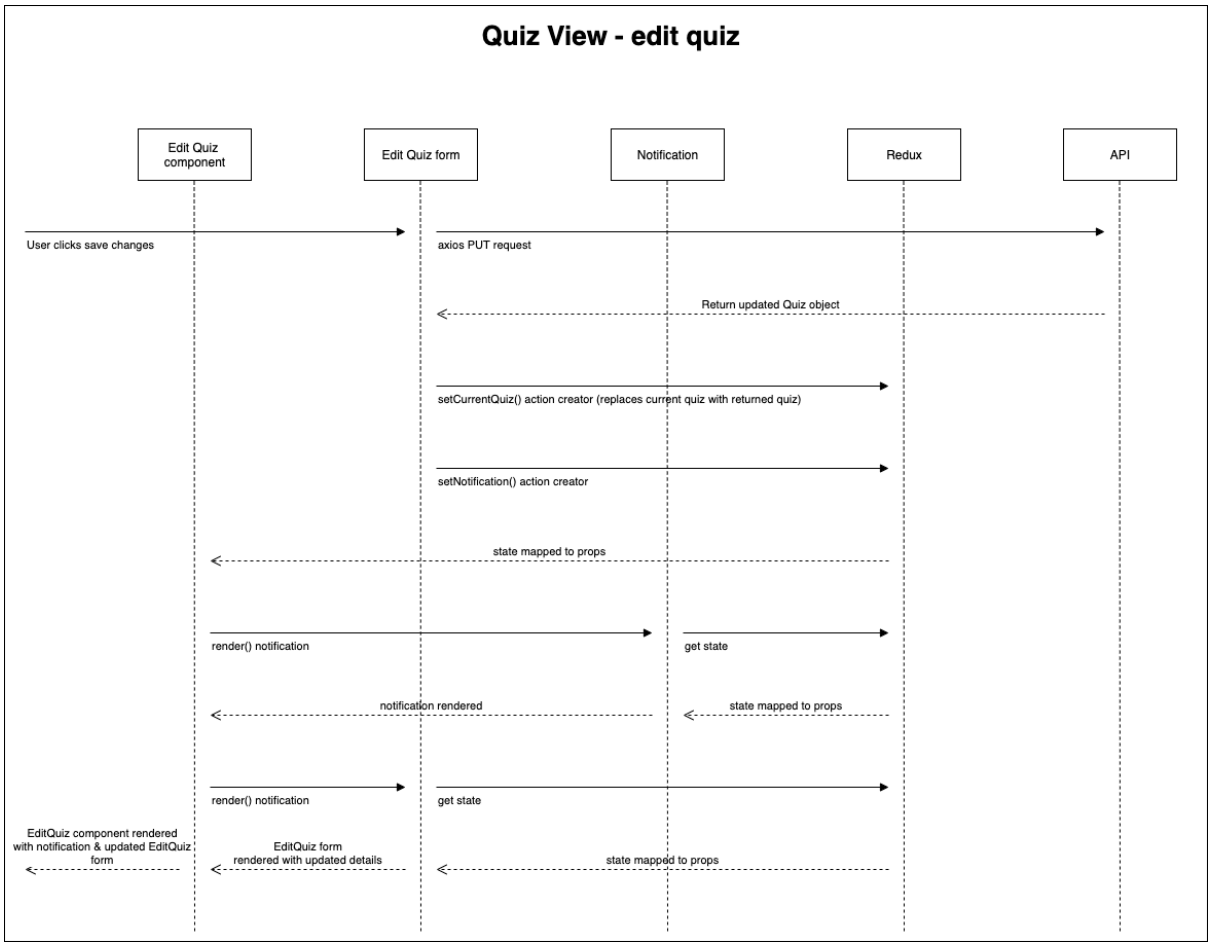
Quiz Search - Clicking a Quiz result's edit option



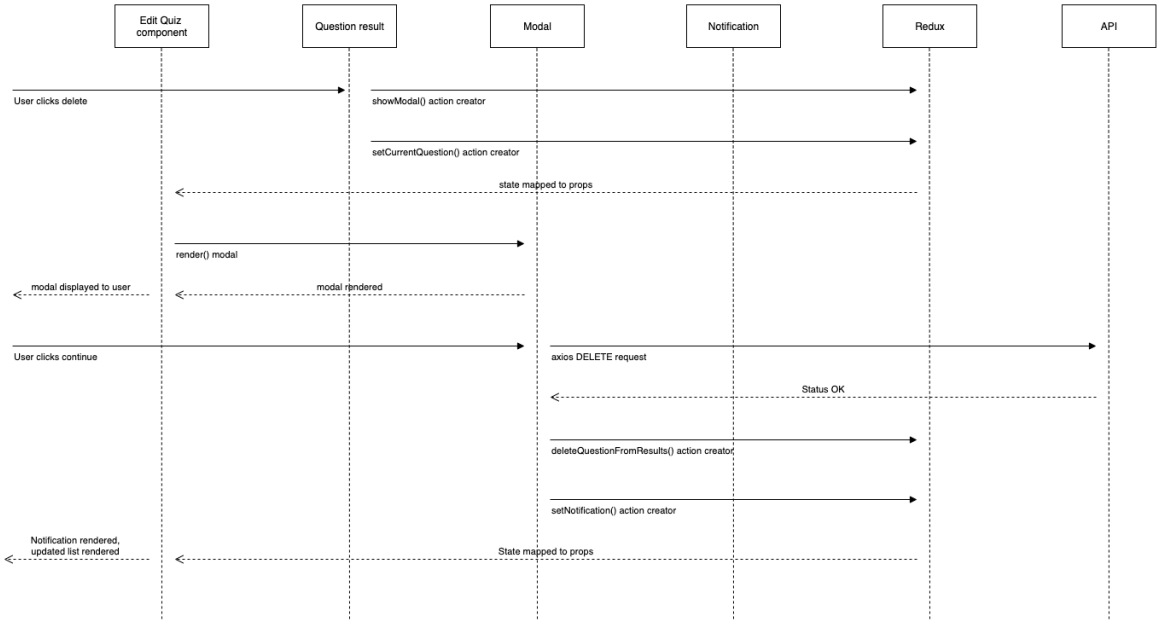
CREATE QUIZ



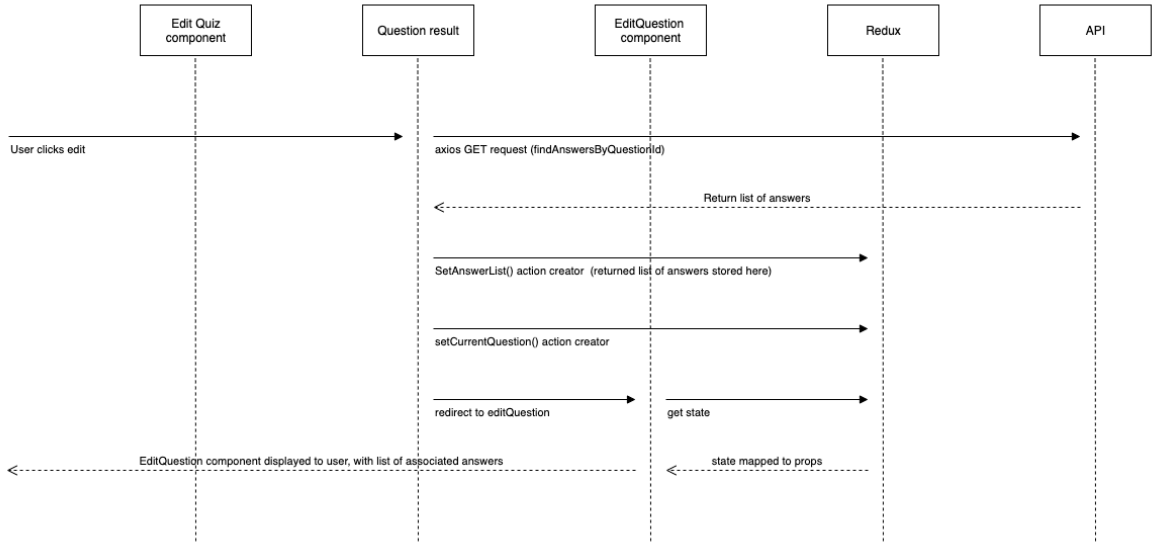
QUIZ VIEW

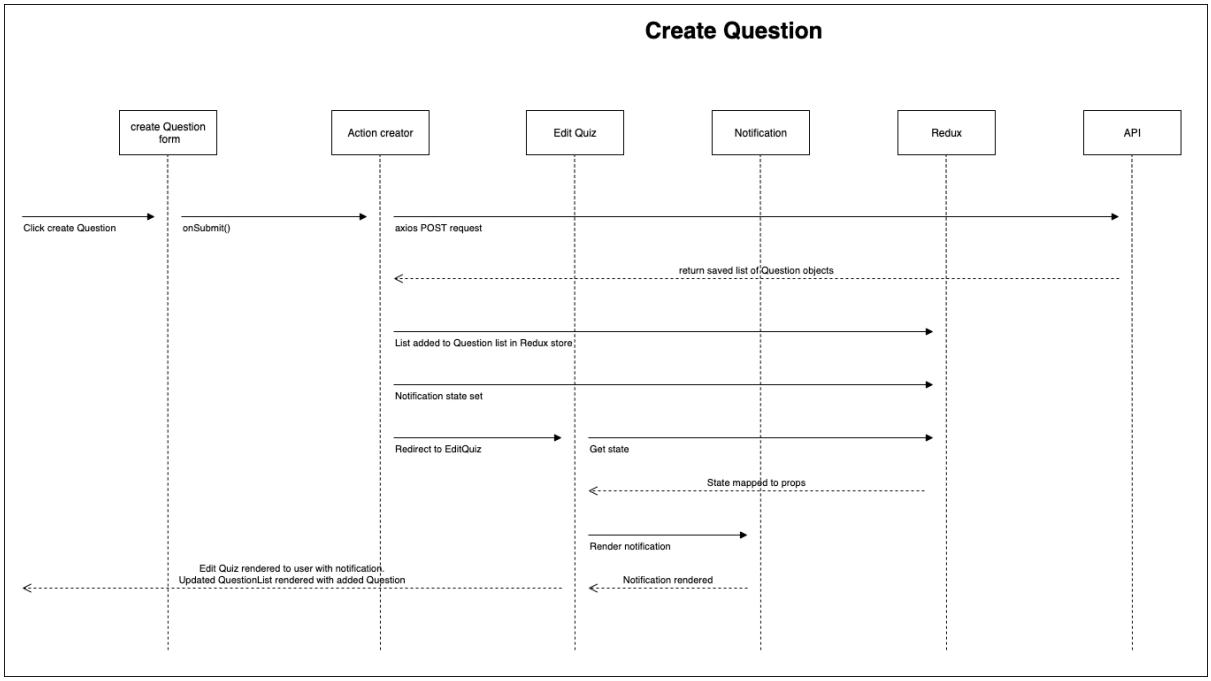


Edit Quiz - Deleting a Question via question result option

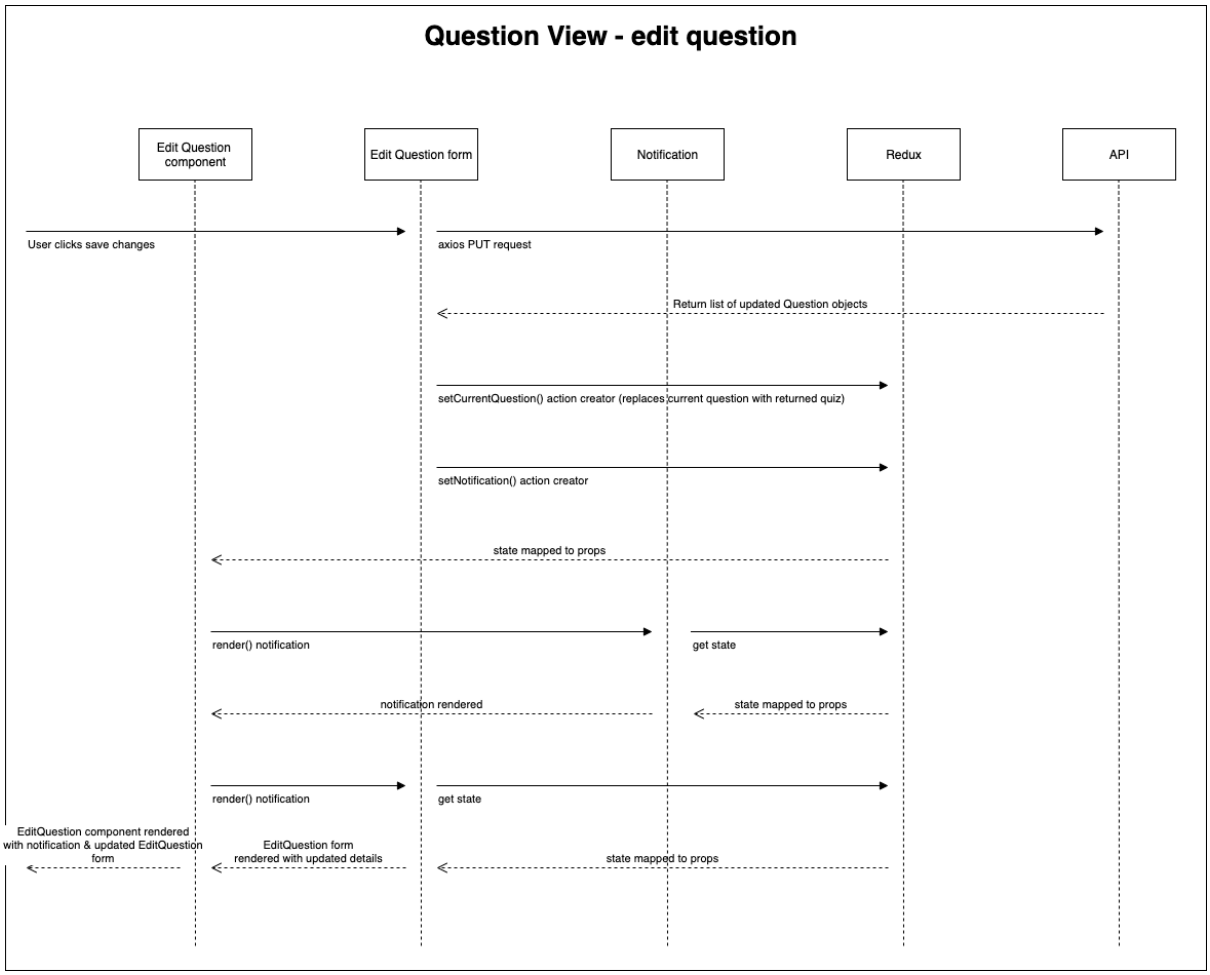


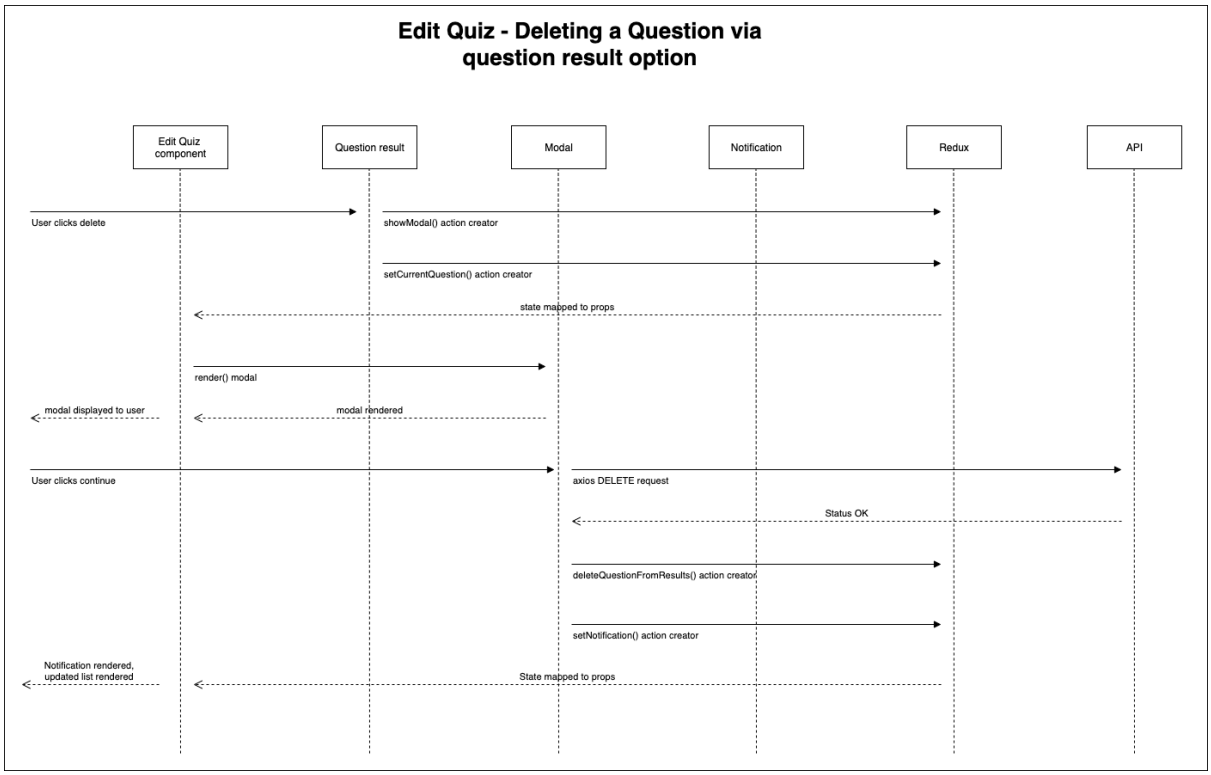
Edit Quiz - Clicking a Question result's edit option

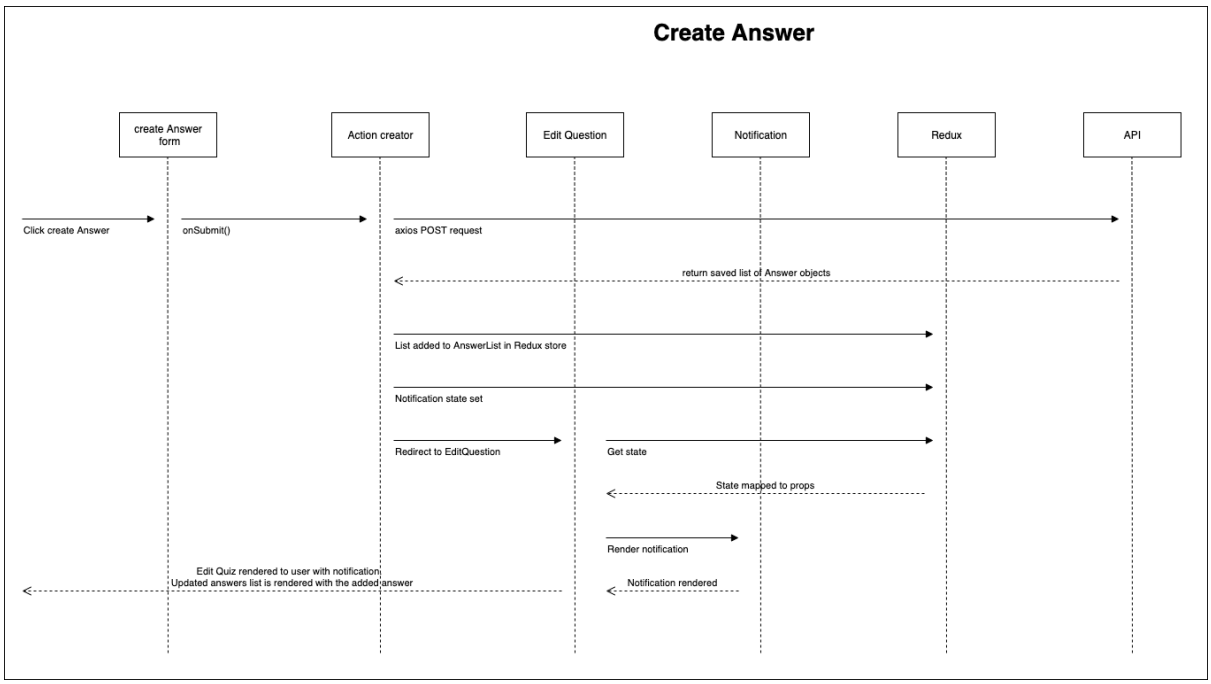


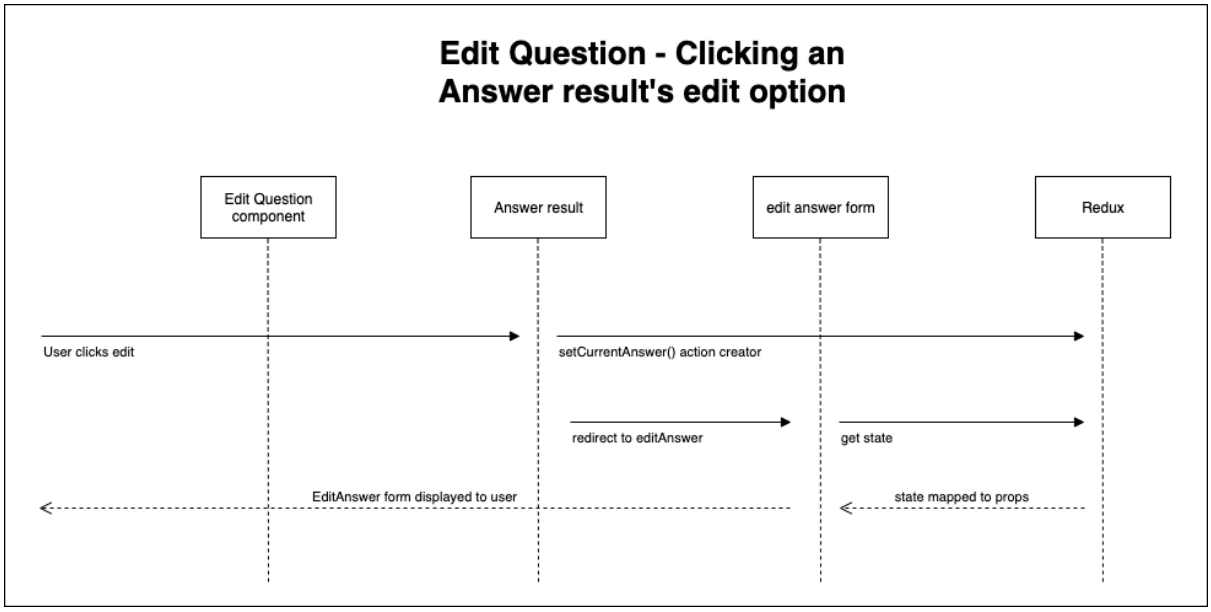


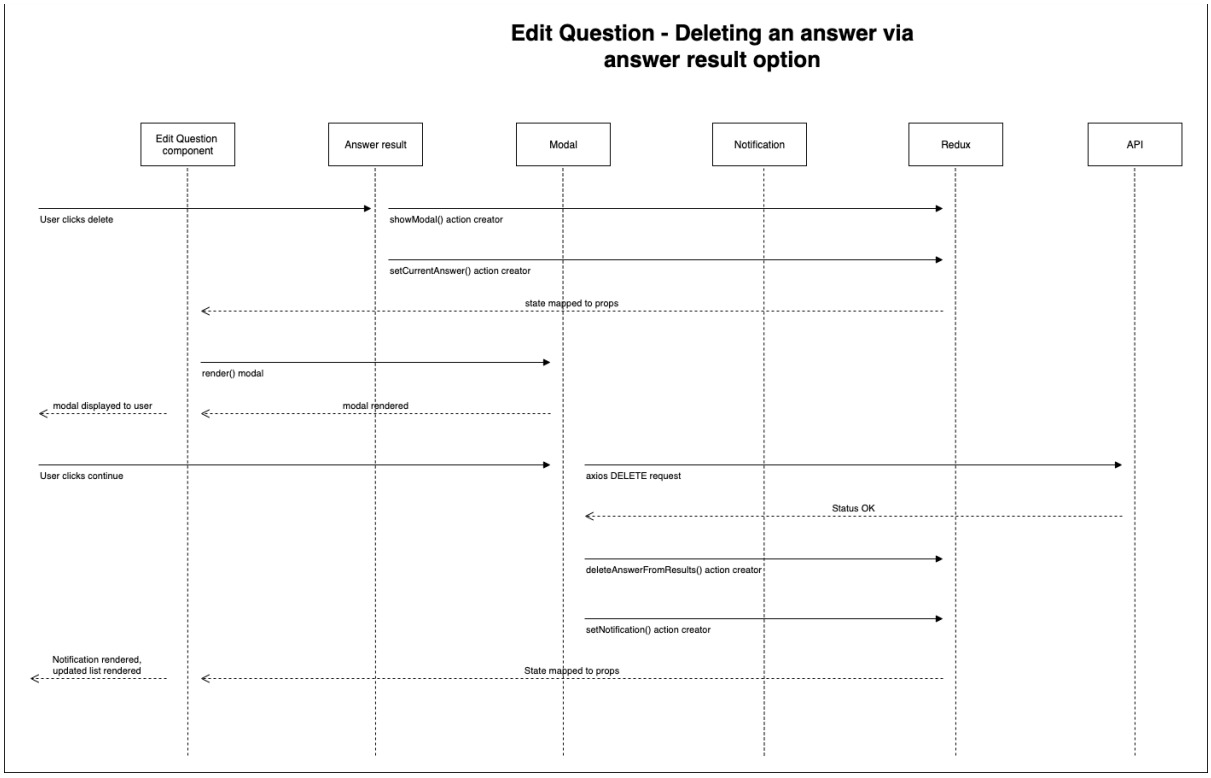
QUESTION VIEW




Edit Quiz - Deleting a Question via
question result option
Edit Quiz component
Question result
Modal
Notification
Redux
API
User clicks delete
showModal() action creator
setCurrentQuestion() action creator
state mapped to props
render() modal
modal displayed to user
modal rendered
User clicks continue
axios DELETE request
Status OK
deleteQuestionFromResults() action creator
setNotification() action creator
Notification rendered,
updated list rendered
State mapped to props


Create Answer
create Answer form
Action creator
Edit Question
Notification
Redux
API
Click create Answer
onSubmit()
axios POST request
return saved list of Answer objects
List added to AnswerList in Redux store
Notification state set
Redirect to EditQuestion
Get state
State mapped to props
Render notification
Edit Quiz rendered to user with notification
Updated answers list is rendered with the added answer
Notification rendered


Edit Question - Clicking an Answer result's edit option
Edit Question component
Answer result
edit answer form
Redux
User clicks edit
setCurrentAnswer() action creator
redirect to editAnswer
get state
EditAnswer form displayed to user
state mapped to props


Edit Question - Deleting an answer via answer result option
Edit Question component
Answer result
Modal
Notification
Redux
API
User clicks delete
showModal() action creator
setCurrentAnswer() action creator
state mapped to props
render() modal
modal displayed to user
modal rendered
User clicks continue
axios DELETE request
Status OK
deleteAnswerFromResults() action creator
setNotification() action creator
Notification rendered, updated list rendered
State mapped to props

IMPLEMENTATION – FRONT END

WHAT WENT WELL

I was able to complete the app within the given time and was able to implement all of the key features that had been outlined in the documentation.

PROBLEMS ENCOUNTERED

TESTING – FRONT END

JEST TESTING

MANUAL TESTING

I have conducted manual testing of the app by going through each of the acceptance criteria areas and checking whether or not the app meets the criteria. Due to time limitations I was unable to provide this in the same format as the back end; however I have outlined the manual tests below:

The tests are as follows:

ALL USERS

- Login check carried out
  - Passed as I was able to log in and received a success message
- Can a search be conducted for a quiz?
  - Yes, I was able to find an existing quiz and result was displayed
- Can a user view all quizzes?
  - Yes, I was able to click on the option and view a list of all available quizzes. This was checked against the data held in the database vie MySQL workbench

USERS WITH EDIT PERMISSION

- Does a user have the option to create a quiz?
  - Yes, I was able to successfully create a quiz. Check conducted using MySQL workbench to double check data was saved to DB

- Does a user have the option to create a question?
  - Yes, I was able to successfully create a question. Check conducted using MySQL workbench to double check data was saved to DB

- Does a user have the option to delete an answer?
  - Yes, I was able to successfully edit a quiz. Check conducted using MySQL workbench to double check data was saved to DB

- Does a user have the option to edit a quiz?
    - Yes, I was able to successfully edit a quiz. Check conducted using MySQL workbench to double check data was saved to DB

- Does a user have the option to edit a question?
    - Yes, I was able to successfully edit a question. Check conducted using MySQL workbench to double check data was saved to DB

- Does a user have the option to edit an answer?
    - Yes, I was able to successfully edit an answer. Check conducted using MySQL workbench to double check data was saved to DB

- Does a user have the option to delete a quiz?
    - Yes, I was able to successfully delete a quiz. Check conducted using MySQL workbench to double check data was saved to DB

- Does a user have the option to delete a question?
    - Yes, I was able to successfully delete a question. Check conducted using MySQL workbench to double check data was saved to DB
    -
- Does a user have the option to delete an answer?
    - Yes, I able to successfully delete an answer. Check conducted using MySQL workbench to double check data was saved to DB

USERS WITH VIEW PERMISSION

- Is the user able to view a quiz and the quizzes questions?
    - Yes, I was able to log in as a 'view' user, find a quiz and view its detail and questions
- Is the user able to view a question and its answers?
    - Yes, I was able to click on the view option next to a question and view its data as well as all of its answers. No options to edit or delete were visible due to permission level

USERS WITH RESTRICETED PERMISSION

- Is the user able to view a quiz and its questions, but has no option to go any further
    - Yes, I was able to log in as a restricted user and was able to view a quiz and its questions. There was no option to go any further.

OVERALL EVALUATION

AREAS FOR IMPROVEMENT

- If there was more time I would want to put more time into finishing the Jest testing of the front end so this is compete. In addition to this I would also want to be able to finish off the testing of the back end to improve the overall coverage.
- If I did this again I would leave more time to add more tests and fully finish the documentation.
- A possible improvement would be the addition of a field to note whether an answer is correct or not.
- Another possible improvement could be a field for quizzes to show their status e.g. draft, review or ready for use.
- A create account endpoint and front-end page would be useful for users to create accounts. This could be limited to users with 'Edit' access who could create accounts for others.
- An endpoint and front-end page for editing permission would be useful for users to edit permissions.

USER MANUAL

DEPENDENCIES

The back-end app uses Java 8 and Maven so these will have to be installed before the app can be run. In addition to this the database is run in a docker container and the app has been set up to make use of docker compose to quickly spin up a docker container with the database. Docker will need to be installed for this to work so below I have outlined how to install both Java 8 and Docker for Windows and Mac users.

To run the front end, users will have to have Node installed so that node commands can be executed to either run the front end or run the tests.

BACK-END SETUP - WINDOWS

JAVA 8 SET UP

1. Check if you have Java installed by opening a new command prompt window and typing: **javac** or **java** or **java -version.** If you don't have Java nothing will be returned
2. Head over to Oracle's website: https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html
3. You will need to sign up and log in
4. Next Click the java se 8 windows x64 version to download the .exe file
5. Once that's installed go to: programme files/java/jre-8/bin and copy the file path
6. Go back to 'This PC' and right click anywhere
7. A menu will pop up, click on properties
8. Click on 'Advanced system settings'
9. Then click on environment variables
10. In the system variables box, find and lick on 'Path'
11. Click edit
12. Click on 'new' and then paste the path you copied
13. If you have a terminal open still from checking your version of java, you will need to close this and reopen it.
14. Type java -version and you should see the details of the java version you've installed which will be: java version "1.8.0_xxx". This indicates that Java 8 is now installed.

DOCKER SET UP

1. Search for task manager and open it.
2. Click on the performance tab and check that virtualisation is enabled (by default this should be enabled). If it is not, you'll just need to follow the steps between 3 and 4 (**a-d** or **a** - **iii** if you are really unlucky). This will need to be done though before installing Docker as Docker makes use of this feature and won't work without it.
    a. If it is not enabled, do a search for "turn windows features on or off" and click to open
    b. Scroll down until you see the option **Hyper-V** and click the check box next to it
    c. Click OK and Windows will install the necessary files to install it.
    d. You will then need to do a restart and virtualization should now be running
        i. If this is still not running, you will need to enable virtualization via you pc/laptop's bios settings
        ii. Restart you pc/laptop and you will see a message telling you what key to press to go into the BIOS options.
        iii. The virtualization option will either be under configuration or advanced but this will differ between devices. When you find the option, turn it on and exit the BIOS. This will now definitely start it. (You may need to do a restart)
3. Now you are ready to install Docker. Go to https://hub.docker.com/editions/community/docker-ce-desktop-windows/
4. Click on 'Get Docker desktop form Windows (stable)'
5. This will download the Docker .exe file.
6. Navigate to your downloads and open the file, which will begin the installation wizard. Follow the install wizard to install.
7. Once this has been installed you may get a message asking if you want to enable Hyper-V containers. Click ok to this message.
8. If you get an error, it is likely you just need to restart your computer.
9. After this if you expand the hidden icons menu in the bottom right hand corner, you will see the docker icon. Click on this and at the top of the menu it should say Docker desktop is running.
10. To check, open a new command prompt window and type: **docker -v**
11. This should give you the version of docker. Next type: **docker run hello world**
12. If you receive a hello from docker confirming everything is running correctly then the process is complete

BACK-END API SETUP - MAC

JAVA 8 DOWNLOAD & SET UP

1. Check if you have Java installed by entering opening a new terminal window and typing: **java -version.** If you don't have Java nothing will be returned
2. If you don't have Java installed, head over to Oracle's website: https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html
3. You will have to sign up and Login
4. click the MacOS x64 version to download the .exe file
5. Once downloaded, go to your 'downloads' folder
6. Double click on the download file

7. Double click on the JDK 8 .pkg file
8. If you get a message that the file can't be opened because apple can't check for malicious software
9. Click ok to close the notification
10. Go to system preferences
11. Security & privacy
12. On the 'general' tab, you will see the file listed at the bottom.
13. Click 'open anyway'
14. The Java install window will now appear
15. Click continue
16. Click install
17. Your mac will ask you to enter your password to confirm you are legit and definitely want to continue and install this.
18. Once installed you can click move to trash as you won't need the installer any more
19. Now if you got to your terminal and type java -version you will see the details of the version of have you have installed. The first line should be: **java version "1.8.0_XXX"**
20. The last step is to configure the **JAVA_HOME** environment
21. First, type: **echo $JAVA_HOME**
22. Nothing should appear because you have shouldn't have Java set up
23. Next type: **ls -la**
24. Check if you have: **.bash_profile**
25. If you don't, you'll just need to create it by typing type: **touch .bash_profile**
26. Then type: **open -e .bash_profile**
27. In the window that opens, type: **export JAVA_HOME=$(/usr/libexec/java_home)**
28. Then **cmd+s** to save
29. close the window
30. Close your terminal (importantly make sure you also quit the terminal app)
31. Then reopen it and type**: echo $JAVA_HOME**

DOCKER DOWNLOAD & SET UP

1. Head over to https://hub.docker.com/editions/community/docker-ce-desktop-mac/
2. Click on get Docker desktop for mac (stable)
3. In the window that opens, after the download, drag the docker icon into applications.
4. This will finish off the installation.
5. Now all you have to do is go to your applications and double click on the docker icon to start it up.

FRONT-END SET UP WINDOWS/MAC

To run the front end you will need to have node installed on your computer.

1. To check if you have node installed try typing **node -v** to see if you get the version number. If this works try typing **npm -v** to see if you get the version number for npm. If both work, you can skip the steps below.

2. Head over to https://nodejs.org/en/
3. Download the version ending with LTS (this indicated a stable version that has long term support)
4. Double click on the downloaded file and follow the steps of the installation wizard to complete the installation process.
5. Once installed, try the commands in step one to double check the install was successful. You should get version numbers for both commands.

RUNNING THE APP

STEP 1 – START UP THE BACK-END API

1. First you will need to spin up a container by typing: **docker-compose up**
2. You should see the lines below once the container has been created and the database is usable:

```
quiz-manager-tests | 2020-06-28T10:10:44.463508Z 0 [System] [MY-010931] [Server] /usr/sbin/mysqld: ready for connections. Version: '8.0.19'  socket: '/var/run/mysqld/mysqld.sock'  port: 3306  MySQL Commun
ity Server - GPL.
quiz-manager    | 2020-06-28T10:10:44.487488Z 0 [System] [MY-010931] [Server] /usr/sbin/mysqld: ready for connections. Version: '8.0.19'  socket: '/var/run/mysqld/mysqld.sock'  port: 3306  MySQL Community
 Server - GPL.
quiz-manager-tests | 2020-06-28T10:10:44.596557Z 0 [System] [MY-011323] [Server] X Plugin ready for connections. Socket: '/var/run/mysqld/mysqlx.sock' bind-address: '::' port: 33060
quiz-manager    | 2020-06-28T10:10:44.609651Z 0 [System] [MY-011323] [Server] X Plugin ready for connections. Socket: '/var/run/mysqld/mysqlx.sock' bind-address: '::' port: 33060
```

3. Now, right click and select the option to open a new tab
4. To start the app, in the new tab, type: **mvn spring-boot:run**

```
docker-compose (docker-compose)  ⌘1   mvn (java)  ⌘2   npm (node)  ⌘3
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.3.0.RELEASE)

2020-06-29 01:46:55.360  WARN 44407 --- [           main] o.s.boot.StartupInfoLogger               : InetAddress.getLocalHost().getHostName() took 5000 milliseconds to respond. Please verify your network
configuration (macOS machines may need to add entries to /etc/hosts).
2020-06-29 01:47:00.366  INFO 44407 --- [           main] quizManager.api.ApiApplication           : Starting ApiApplication on HOD245.local with PID 44407 (/Users/ajaymungur/Documents/BCS - Synoptic Proj
ect work/api/target/classes started by ajaymungur in /Users/ajaymungur/Documents/BCS - Synoptic Project work/api)
2020-06-29 01:47:00.367  INFO 44407 --- [           main] quizManager.api.ApiApplication           : The following profiles are active: dev
2020-06-29 01:47:01.003  INFO 44407 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFERRED mode.
2020-06-29 01:47:01.053  INFO 44407 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 45ms. Found 4 JPA repository interfaces.
2020-06-29 01:47:01.568  INFO 44407 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8080 (http)
2020-06-29 01:47:01.574  INFO 44407 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2020-06-29 01:47:01.575  INFO 44407 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.35]
2020-06-29 01:47:01.642  INFO 44407 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2020-06-29 01:47:01.643  INFO 44407 --- [           main] o.s.web.context.ContextLoader            : Root WebApplicationContext: initialization completed in 1244 ms
2020-06-29 01:47:01.818  INFO 44407 --- [           main] o.f.c.internal.license.VersionPrinter    : Flyway Community Edition 6.4.1 by Redgate
2020-06-29 01:47:01.823  INFO 44407 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2020-06-29 01:47:02.139  INFO 44407 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2020-06-29 01:47:02.165  INFO 44407 --- [           main] o.f.c.internal.database.DatabaseFactory  : Database: jdbc:mysql://127.0.0.1:32860/quiz_manager (MySQL 8.0)
2020-06-29 01:47:02.258  INFO 44407 --- [           main] o.f.core.internal.command.DbValidate     : Successfully validated 6 migrations (execution time 00:00.039s)
2020-06-29 01:47:02.280  INFO 44407 --- [           main] o.f.core.internal.command.DbMigrate      : Current version of schema `quiz_manager`: 6
2020-06-29 01:47:02.282  INFO 44407 --- [           main] o.f.core.internal.command.DbMigrate      : Schema `quiz_manager` is up to date. No migration necessary.
2020-06-29 01:47:02.509  INFO 44407 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'applicationTaskExecutor'
2020-06-29 01:47:02.553  INFO 44407 --- [         task-1] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name: default]
2020-06-29 01:47:02.586  INFO 44407 --- [         task-1] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 5.4.15.Final
2020-06-29 01:47:02.616  WARN 44407 --- [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during view
 rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
2020-06-29 01:47:02.663  INFO 44407 --- [         task-1] o.hibernate.annotations.common.Version   : HCANN000001: Hibernate Commons Annotations {5.1.0.Final}
2020-06-29 01:47:02.730  INFO 44407 --- [         task-1] org.hibernate.dialect.Dialect            : HHH000400: Using dialect: org.hibernate.dialect.MySQL5Dialect
2020-06-29 01:47:03.110  INFO 44407 --- [         task-1] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.No
JtaPlatform]
2020-06-29 01:47:03.115  INFO 44407 --- [         task-1] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2020-06-29 01:47:03.662  INFO 44407 --- [           main] o.s.s.web.DefaultSecurityFilterChain     : Creating filter chain: any request, [org.springframework.security.web.context.request.async.WebAsyncMan
agerIntegrationFilter@68e2d03e, org.springframework.security.web.context.SecurityContextPersistenceFilter@36c07c75, org.springframework.security.web.header.HeaderWriterFilter@5ec9eefa, org.springframework
.web.filter.CorsFilter@120aa40b, org.springframework.security.web.authentication.logout.LogoutFilter@7a66c35a, quizManager.api.JwtFilter@5300cac, org.springframework.security.web.savedrequest.RequestCache
AwareFilter@5d850a25, org.springframework.security.web.servletapi.SecurityContextHolderAwareRequestFilter@3a3f96ab, org.springframework.security.web.authentication.AnonymousAuthenticationFilter@32ab408e,
org.springframework.security.web.session.SessionManagementFilter@4602f874, org.springframework.security.web.access.ExceptionTranslationFilter@231b35fb, org.springframework.security.web.access.intercept.Fi
lterSecurityInterceptor@29bbc63c]
2020-06-29 01:47:03.692  INFO 44407 --- [           main] pertySourcedRequestMappingHandlerMapping : Mapped URL path [/v2/api-docs] onto method [springfox.documentation.swagger2.web.Swagger2ControllerWebM
vc#getDocumentation(String, HttpServletRequest)]
2020-06-29 01:47:04.053  INFO 44407 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) with context path ''
2020-06-29 01:47:04.054  INFO 44407 --- [           main] d.s.w.p.DocumentationPluginsBootstrapper : Documentation plugins bootstrapped
2020-06-29 01:47:04.056  INFO 44407 --- [           main] d.s.w.p.DocumentationPluginsBootstrapper : Found 1 custom documentation plugin(s)
2020-06-29 01:47:04.084  INFO 44407 --- [           main] s.d.s.w.s.ApiListingReferenceScanner     : Scanning for api listing references
2020-06-29 01:47:04.178  INFO 44407 --- [           main] DeferredRepositoryInitializationListener : Triggering deferred initialization of Spring Data repositories…
2020-06-29 01:47:04.287  INFO 44407 --- [           main] DeferredRepositoryInitializationListener : Spring Data repositories initialized!
2020-06-29 01:47:04.295  INFO 44407 --- [           main] quizManager.api.ApiApplication           : Started ApiApplication in 19.217 seconds (JVM running for 19.492)
```

You will see the above in your terminal indicating that the app is now running

5. To stop the app press **ctrl+c**
6. To stop the docker container, click on the tab where you ran docker from. Press **ctrl+c** to stop

Tests

To run the tests, make sure the app is not running. Follow **step 1** and then type: **mvn test**

STEP 2 – RUN THE FRONT END

1. Once the back end is up and running, open up a new terminal tab as in step 1 (so you will have three tabs open).
2. Type: **npm install** (all of the required dependencies will be downloaded and installed)
3. Once the dependencies have been installed, type: **npm start**
   This will start the app and open a browser window where the app will be usable. Once the app is up and running you will see the message below in the terminal.

```
Compiled successfully!

You can now view quiz-manager in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.0.15:3000

Note that the development build is not optimized.
To create a production build, use npm run build.
```

4. To stop the app, press **ctrl+c**

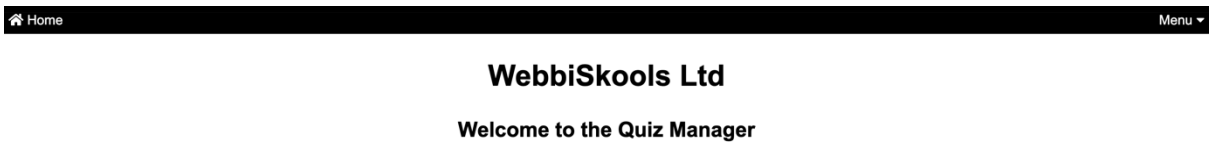
The front end server should fire up and a browser window should open. If the browser window does not open, simply open a new browser window and type: http://localhost:3000/ and you will see the home page. You wil then be able to begin using the app.

Tests

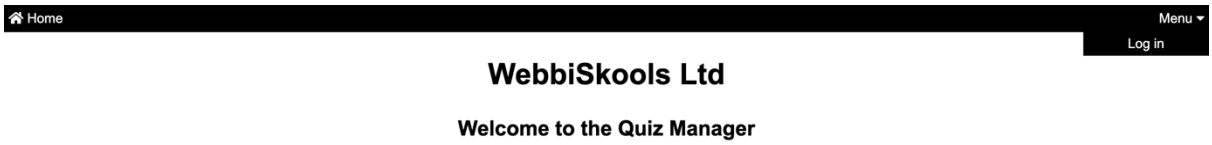
- To run the tests, make sure the front end has been stopped. Then type: **npm test**
- To exit out of the tests press **ctrl+c**

HOME PAGE & NAVIGATION

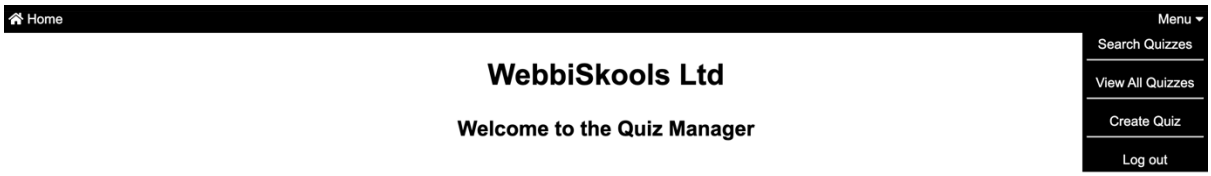
1. When you start the app you will see the view below of the home page



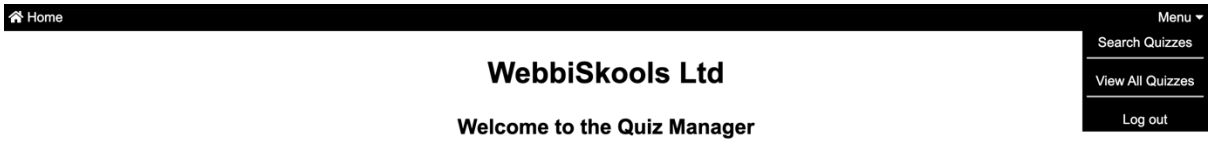
2. Pre-login, the only option available via the drop down is to log in



3. Post login you will see the following options if you have – EDIT

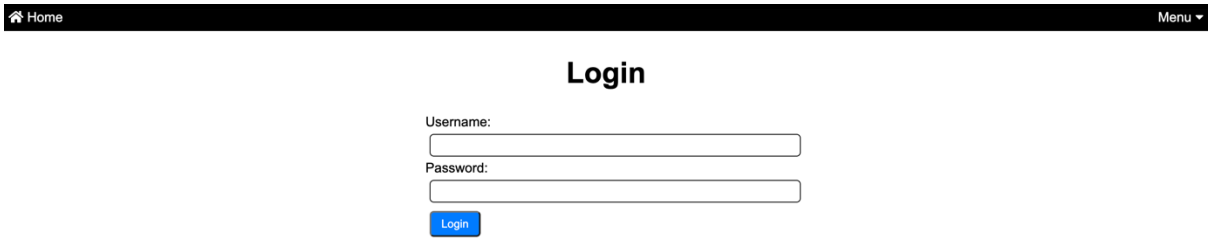


4. Post login you will see the following options if you have n – RESTRICTED & VIEW permission

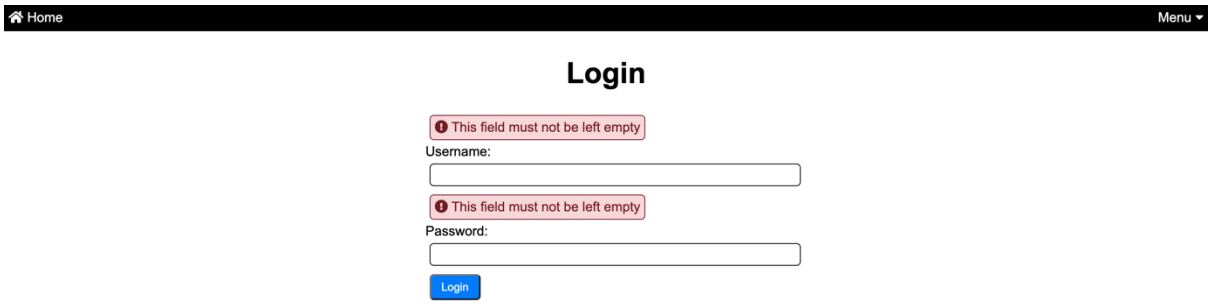


LOGIN

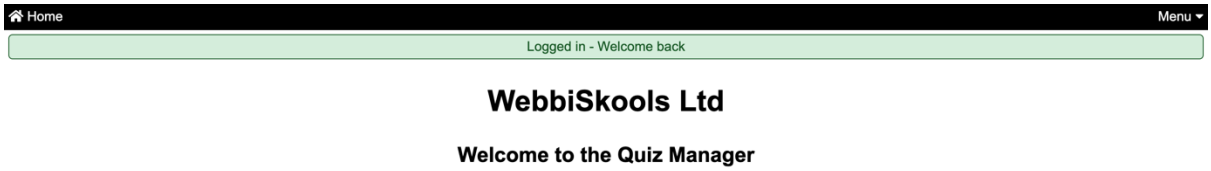
1. After clicking login you will be taken to the page below



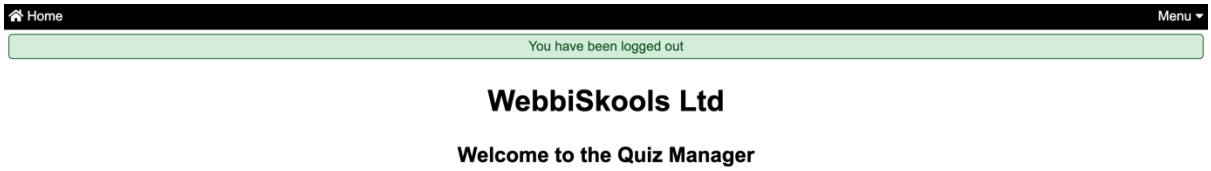
2. Each field has validations to prevent attempts to log in without providing any values



3. On successful login, you will be redirected to the home page and a notification will confirm login

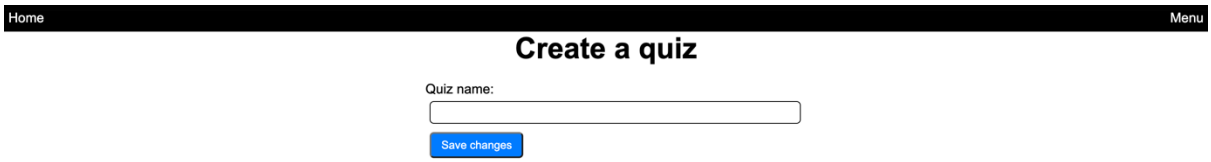


4. When you log out, you will be redirected to the home page and a notification will confirm you have been logged out

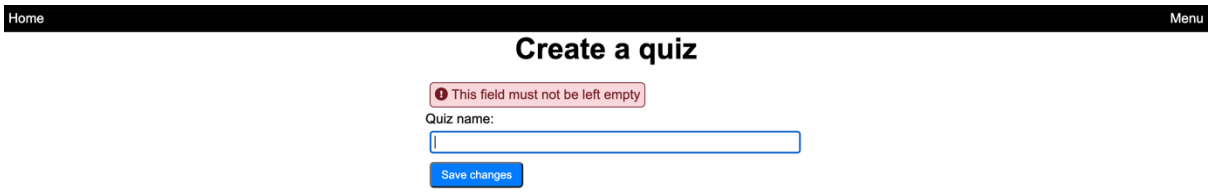


CREATING A QUIZ

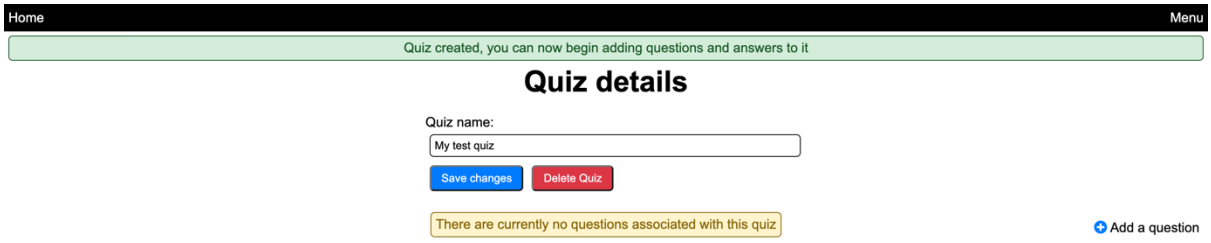
1. After clicking the create quiz option you will be taken to the page below



2. The name field has validation to prevent you submitting the form with no value entered

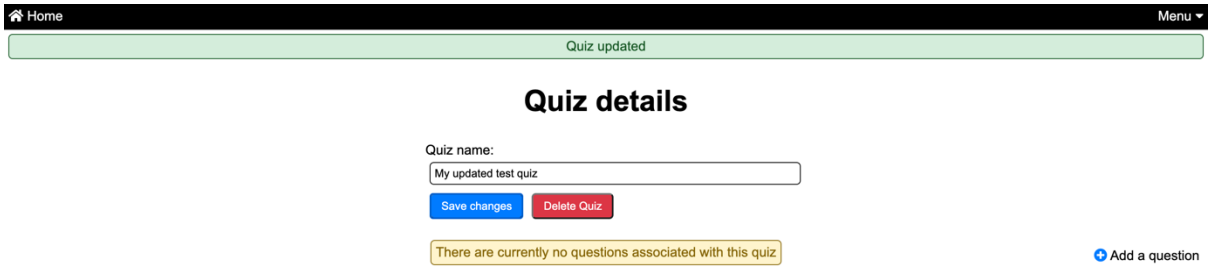


3. Upon successful quiz creation you are redirect to the quiz view page

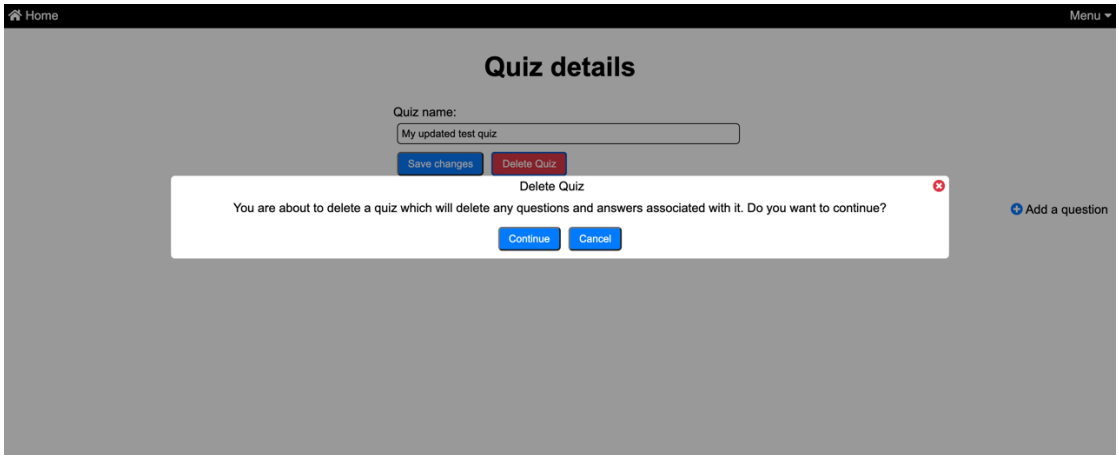


QUIZ VIEW PAGE

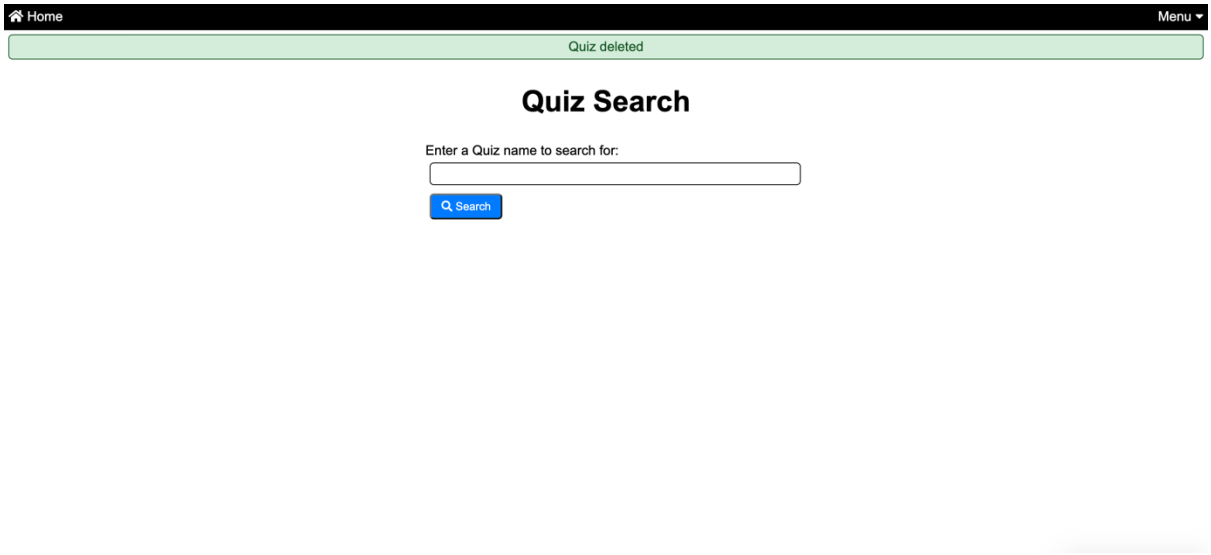
1. Continued from the previous section when accessing the quiz view page you will either have a form rendered as below if you have EDIT permission, otherwise only the details will render with no options.



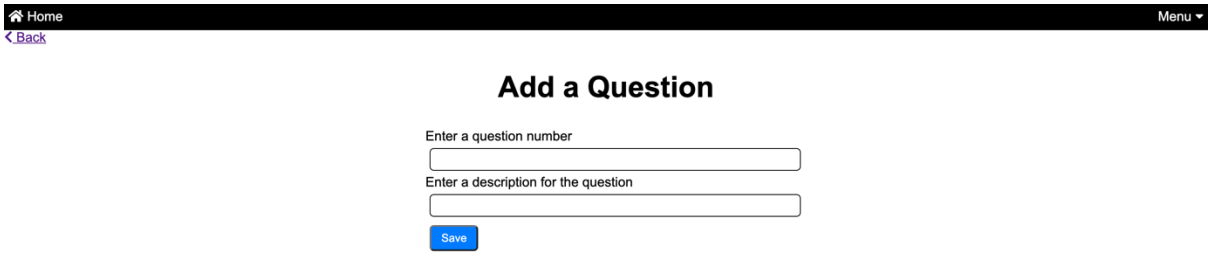
2. When you click on delete quiz, a modal will render as below.



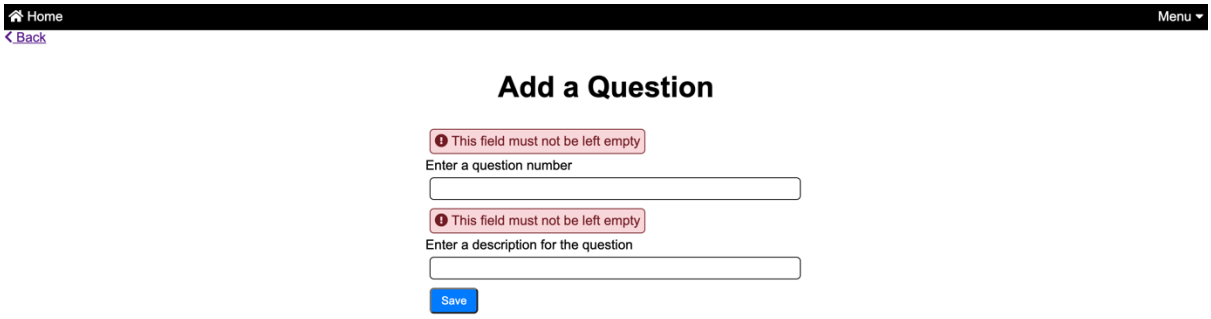
If you click cancel, no action will take place and the modal will disappear. If you click continue, you will be redirected to the quiz search page and a notification will confirm the quiz was deleted.



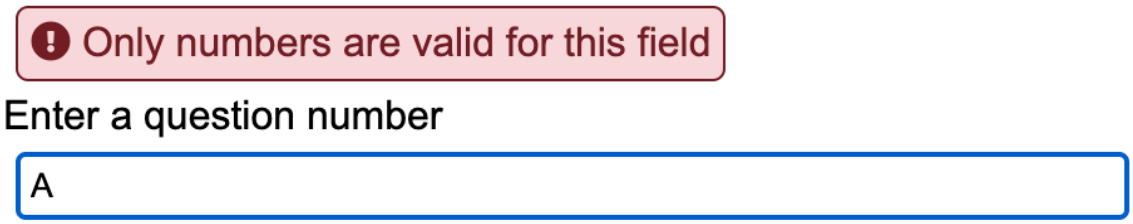
3. After clicking `Add a question` you will be taken to the page below

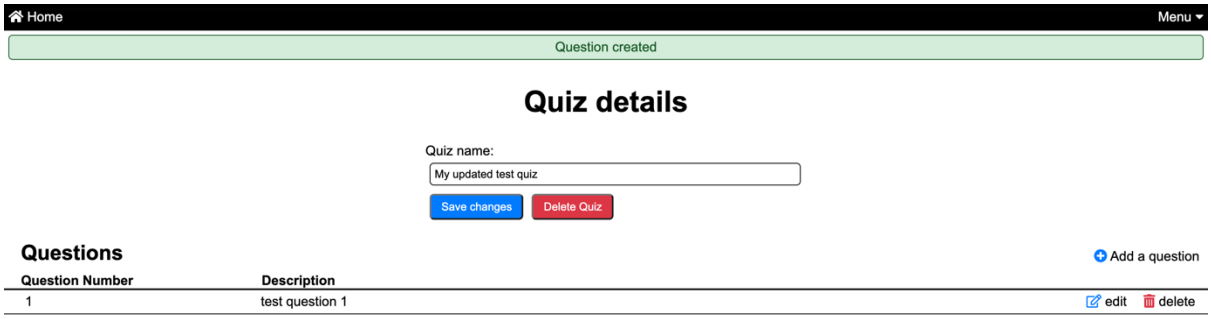


As with all of the other forms, there are field validations to prevent fields being left blank

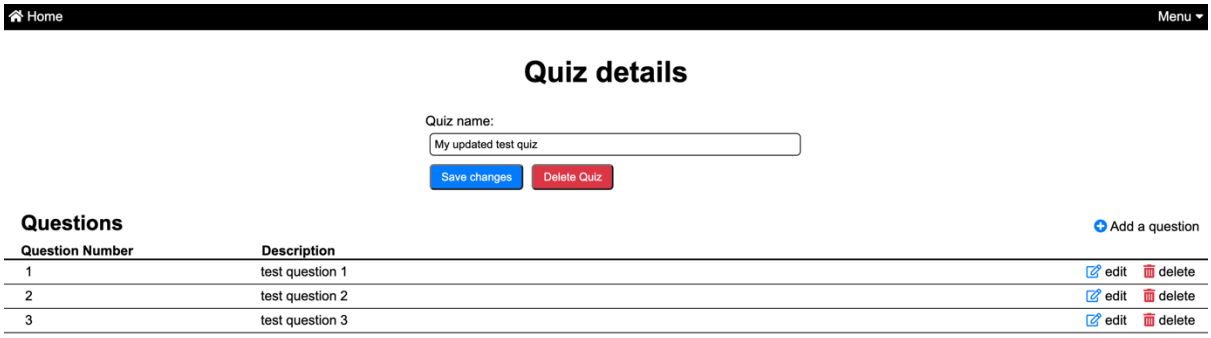


There is a further layer of validation to ensure only numbers are entered for the question number.

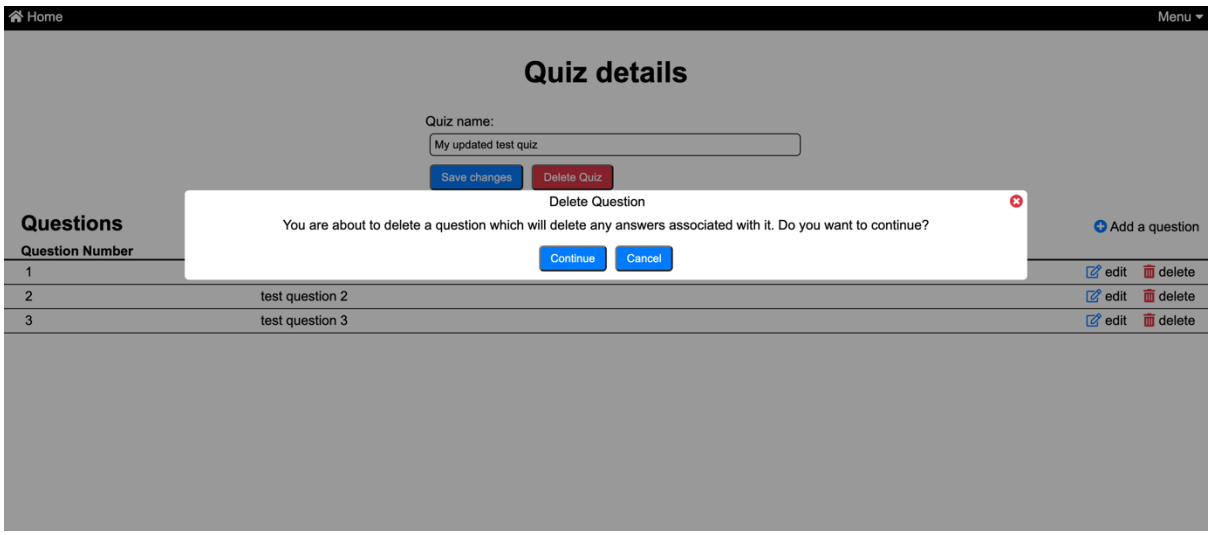


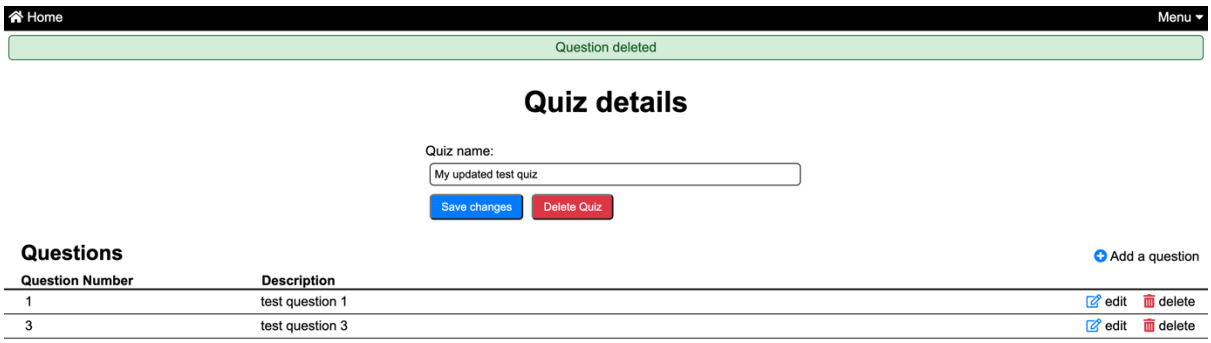


4. The question list – always in order

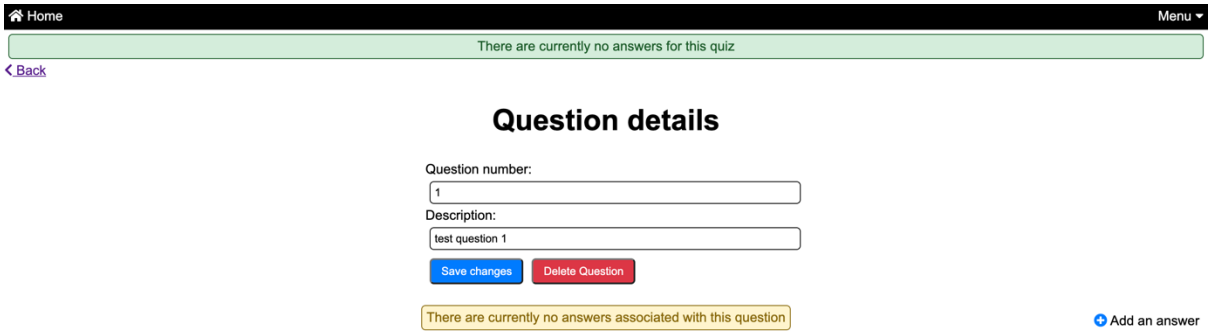


5. Deleting a question



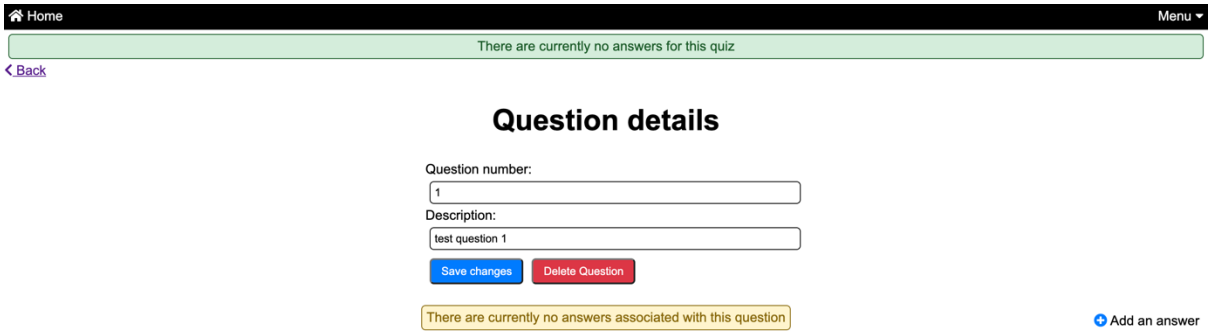


6. Editing a question

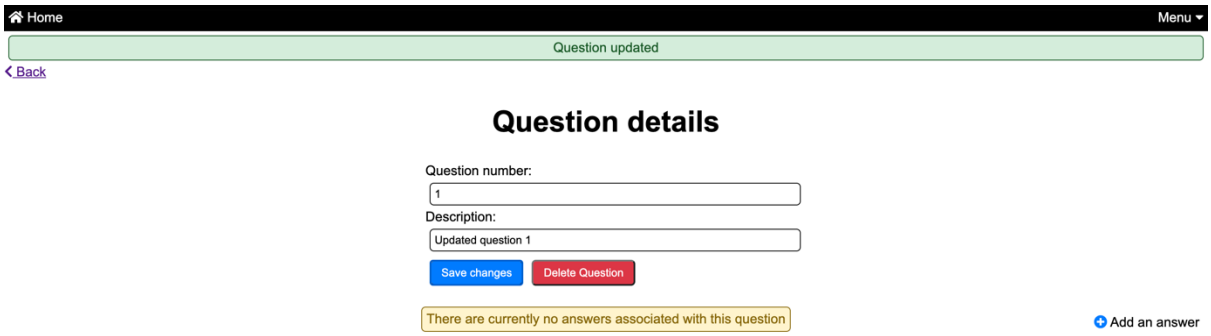


QUESTION VIEW PAGE

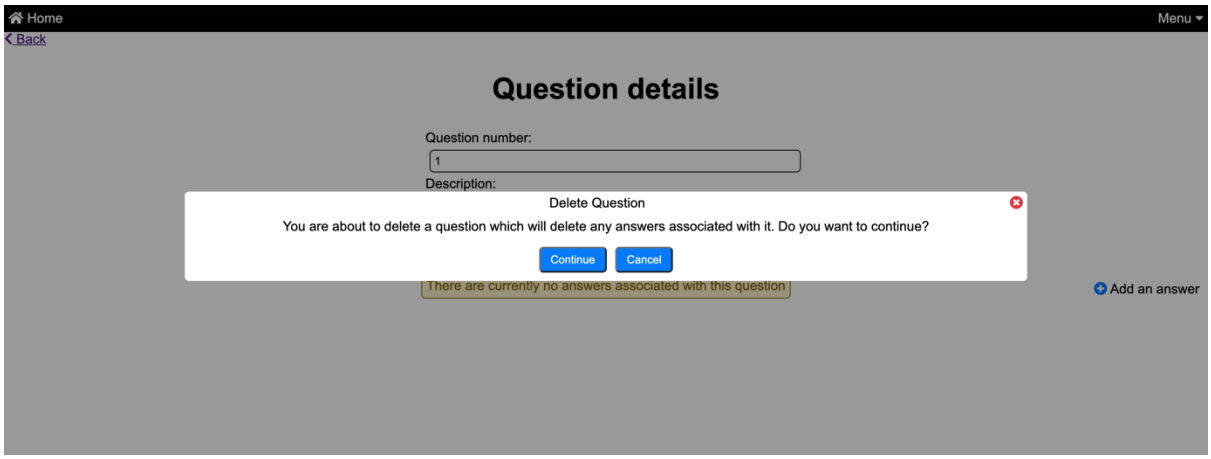
1. Details or Form

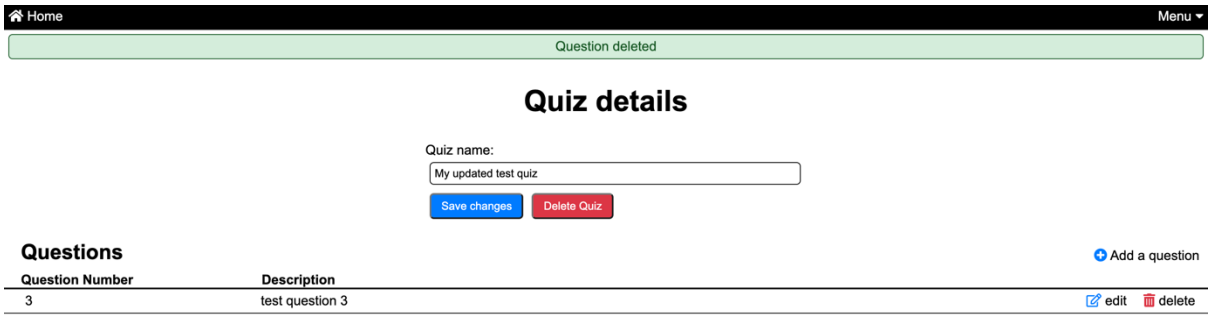


2. Update

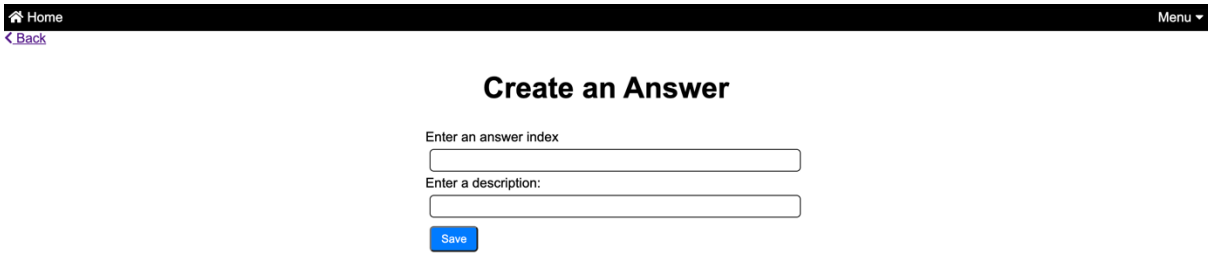


3. Delete question





4. Adding an answer



Home Menu

Back

Create an Answer

This field must not be left empty

Enter an answer index

This field must not be left empty

Enter a description:

Save

This should be a single letter e.g. A, B, C, D or E

Enter an answer index

1

Home Menu

Answer created

Back

Question details

Question number:

1

Description:

test question 1

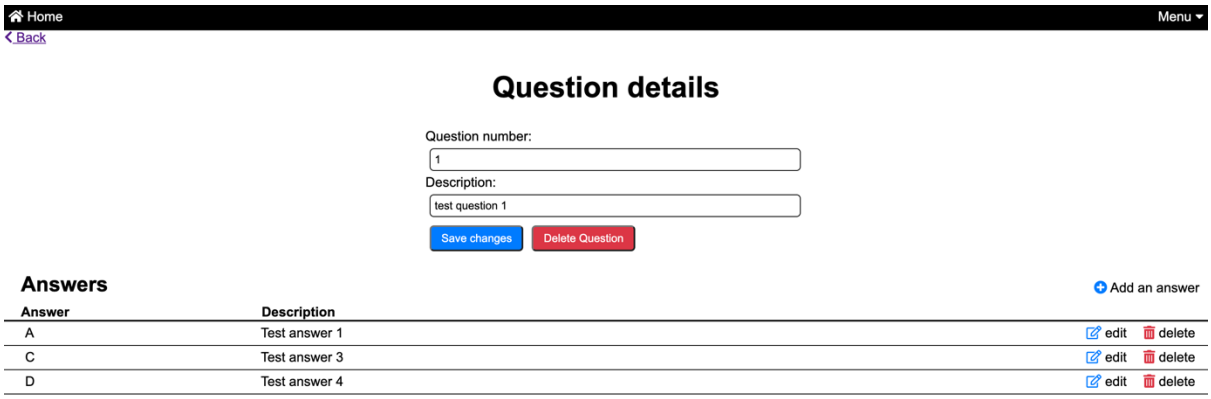
Save changes Delete Question

Answers

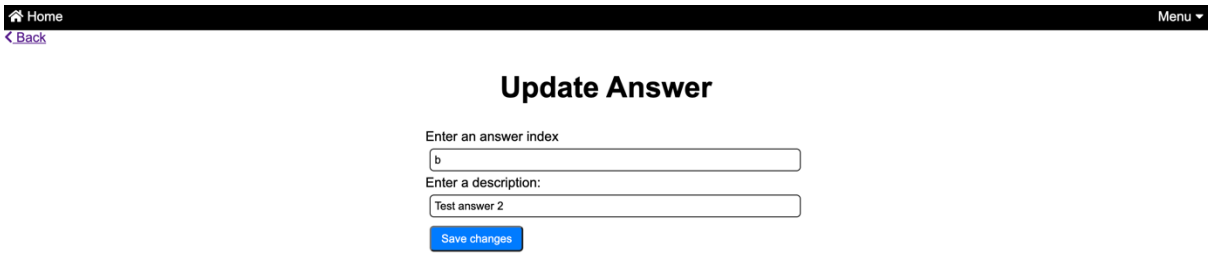
Add an answer

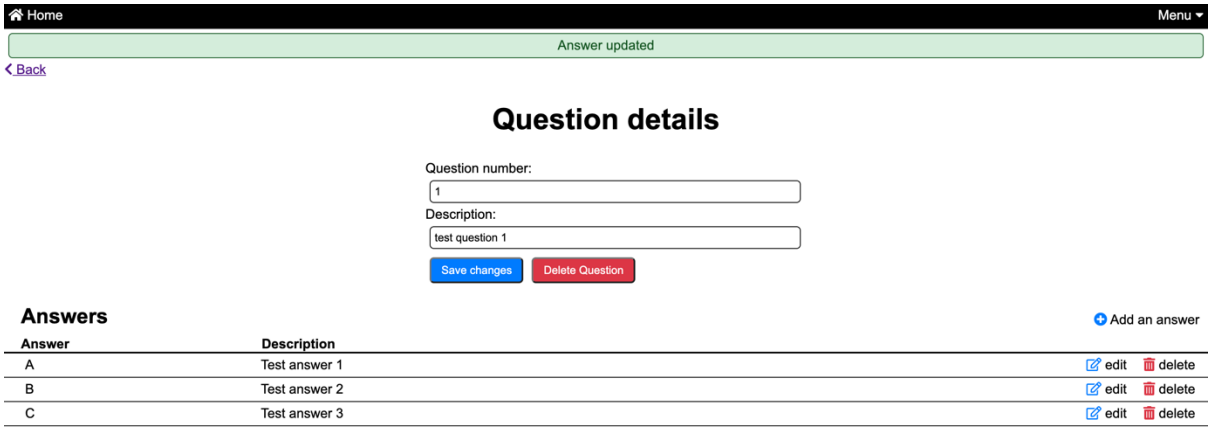
Answer	Description	
A	Test answer 1	edit delete

5. The answer list

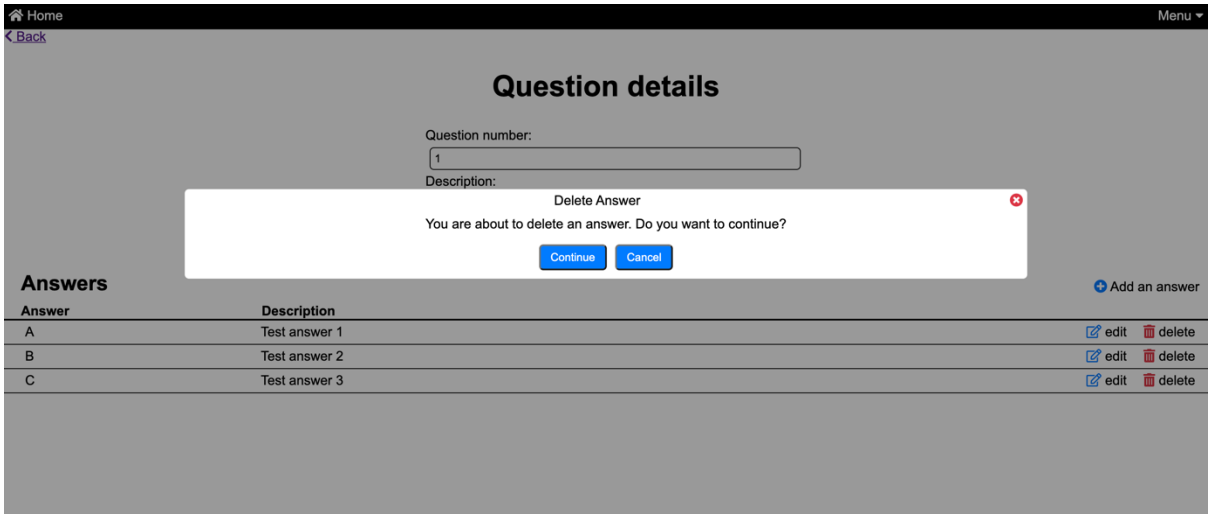


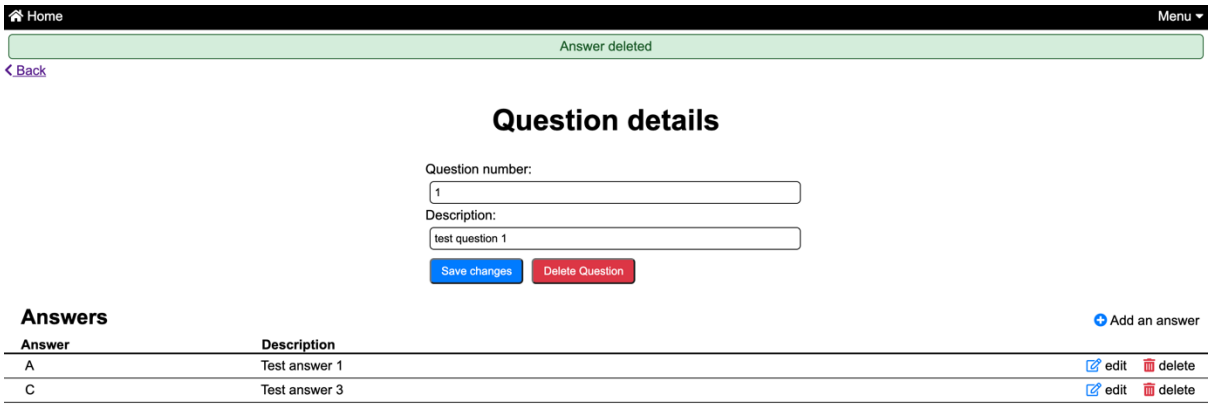
6. Editing an answer





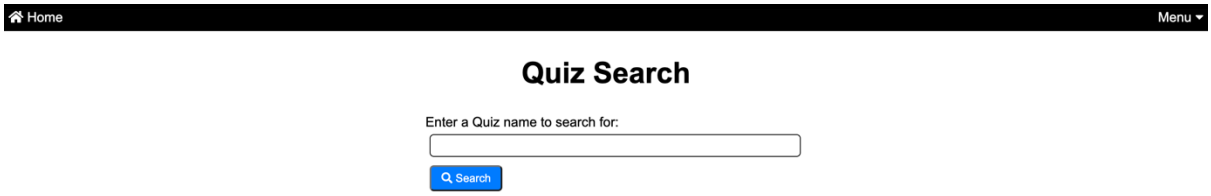
7. Deleting an answer



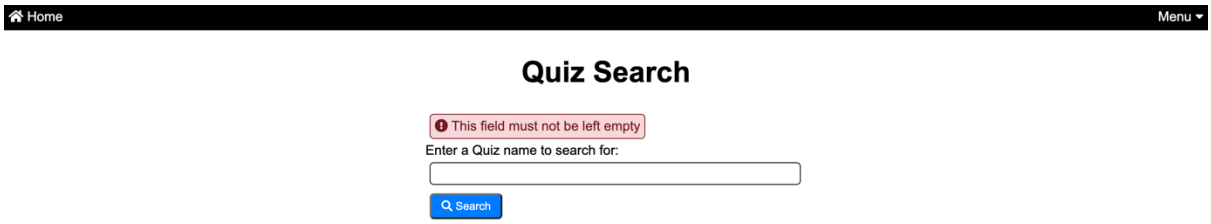


SEARCHING

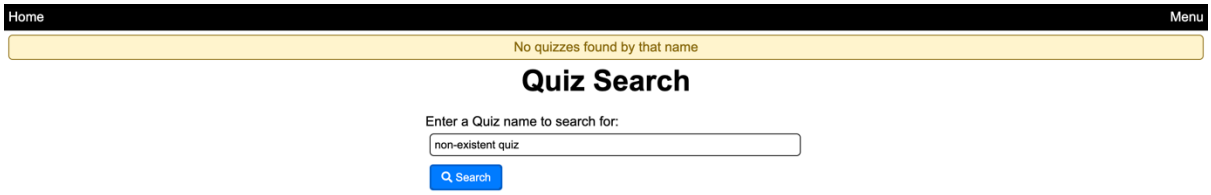
1. Search page

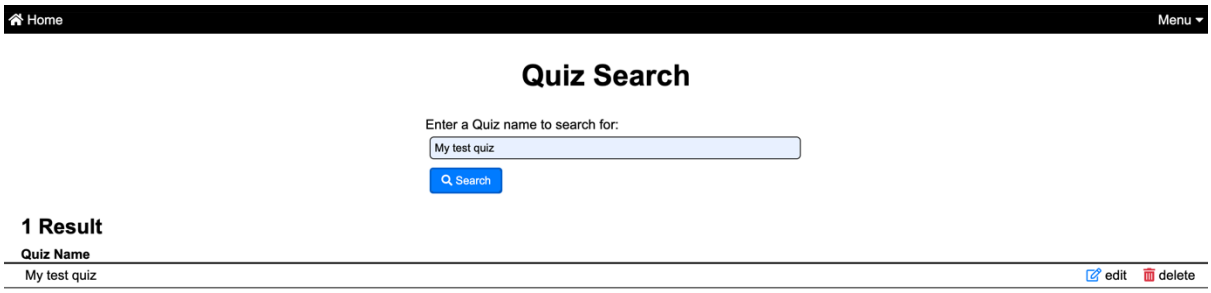


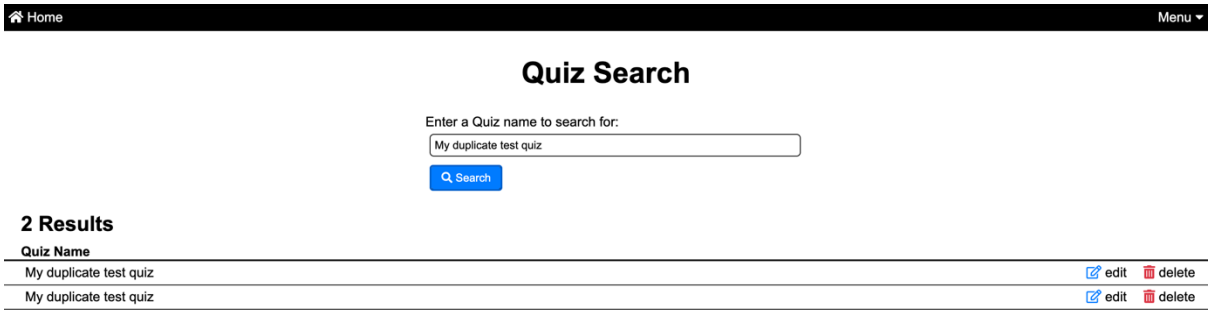
2. Validation



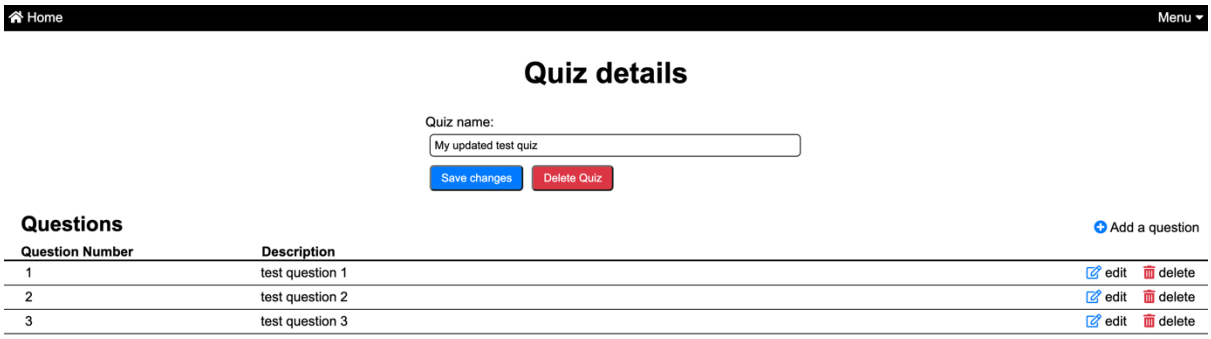
3. Results



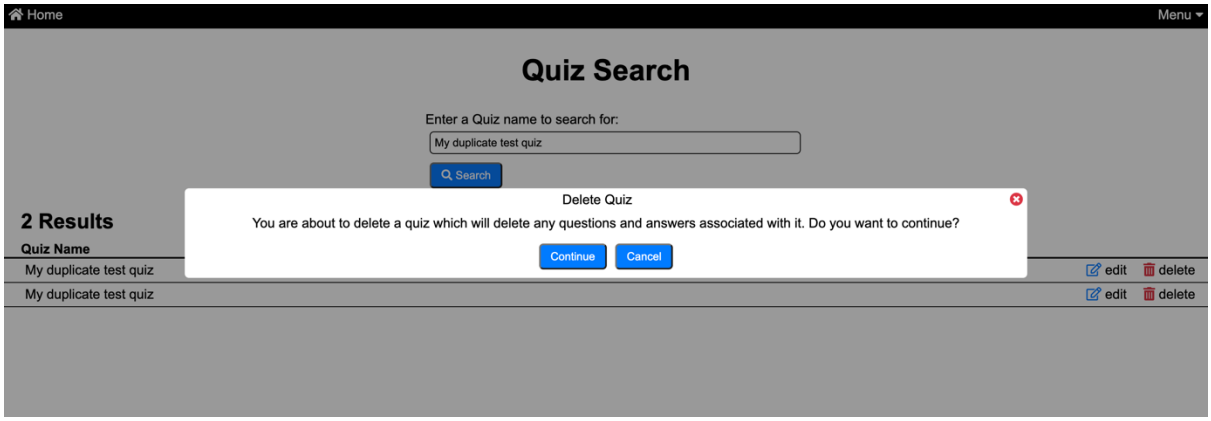


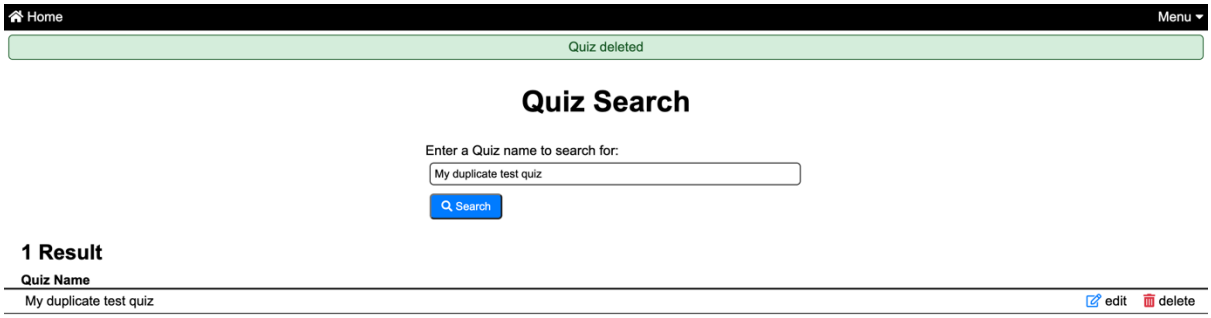


4. Edit



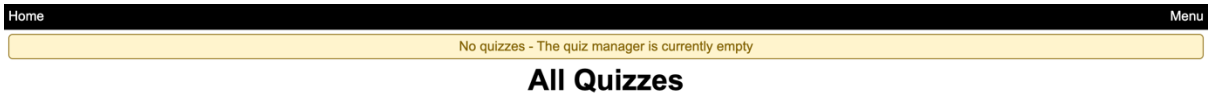
5. Delete

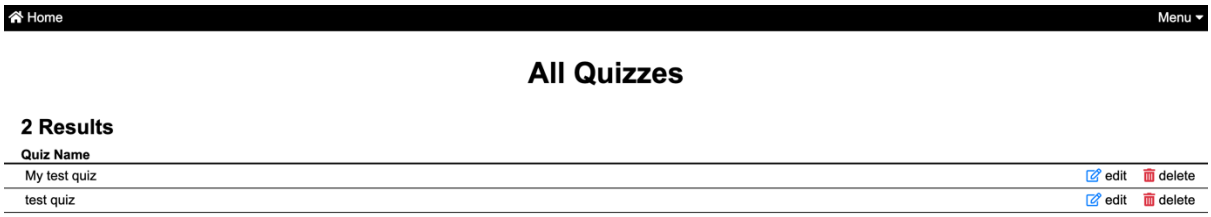




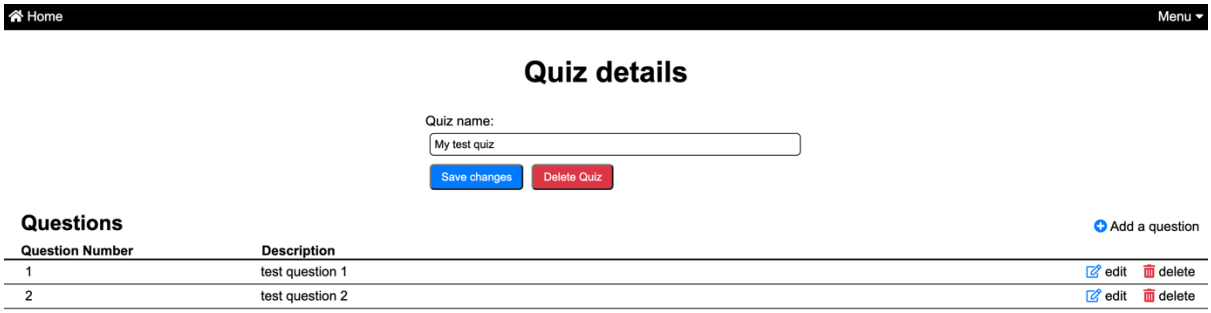
VIEW ALL QUIZZES

1. Results

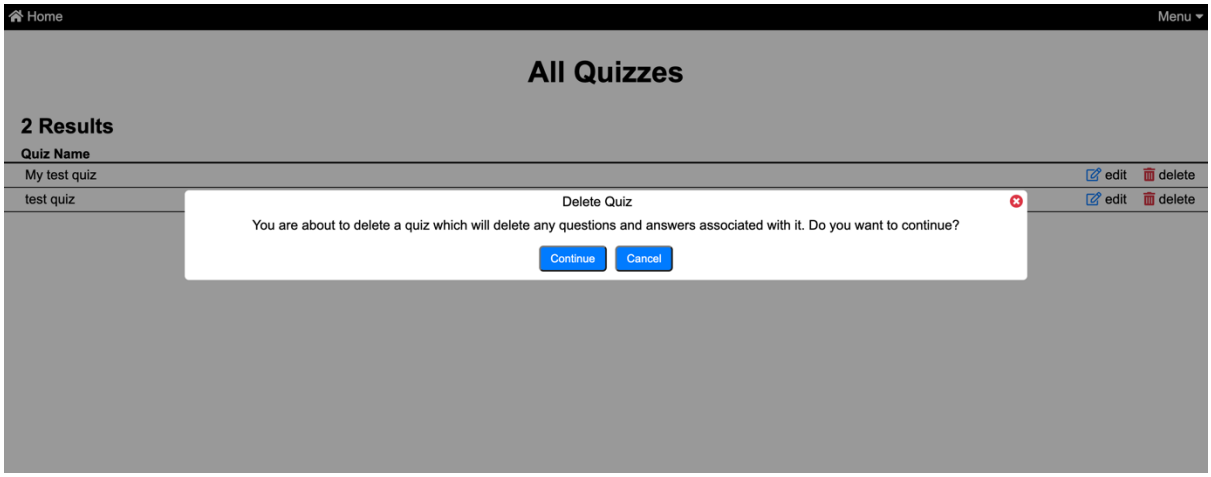


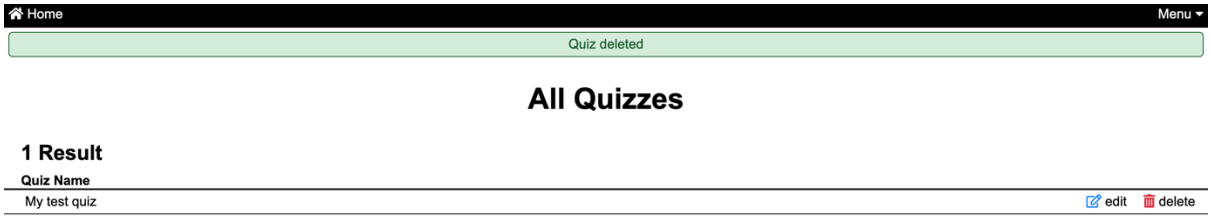


2. Edit


Home
Menu
Quiz details
Quiz name:
My test quiz
Save changes
Delete Quiz
Questions
Add a question
Question Number
Description
1
test question 1
edit
delete
2
test question 2
edit
delete

3. Delete

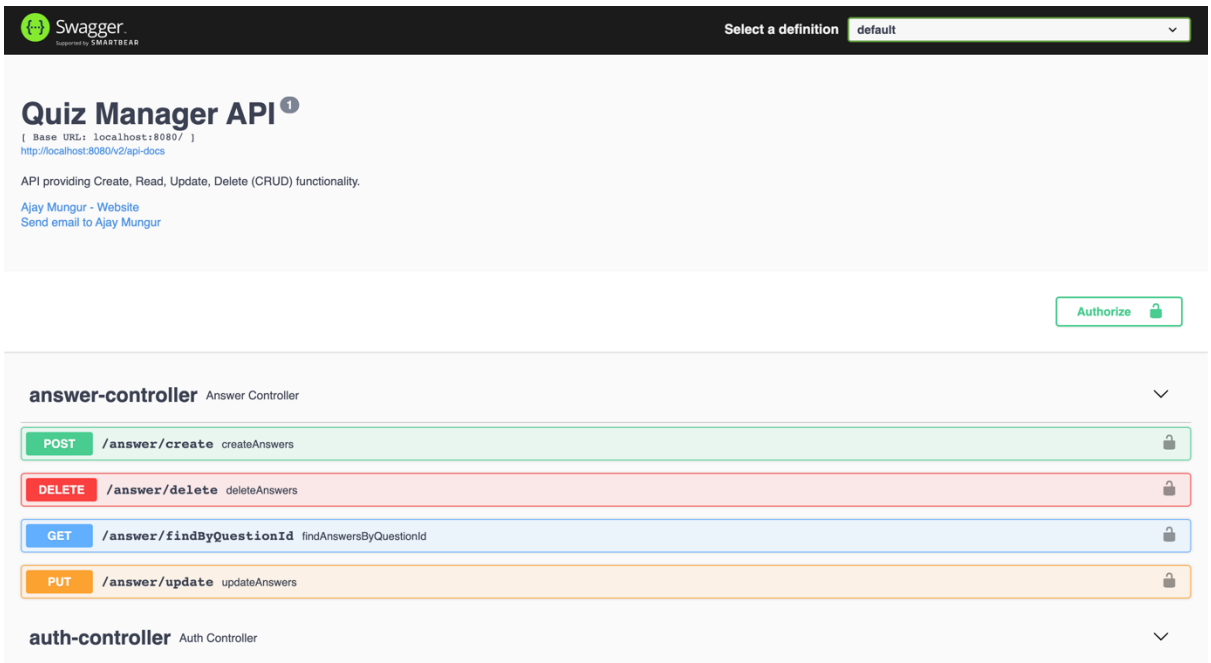

Home
Menu
All Quizzes
2 Results
Quiz Name
My test quiz
edit
delete
test quiz
edit
delete
Delete Quiz
You are about to delete a quiz which will delete any questions and answers associated with it. Do you want to continue?
Continue
Cancel


Home
Menu
Quiz deleted
All Quizzes
1 Result
Quiz Name
My test quiz
edit
delete

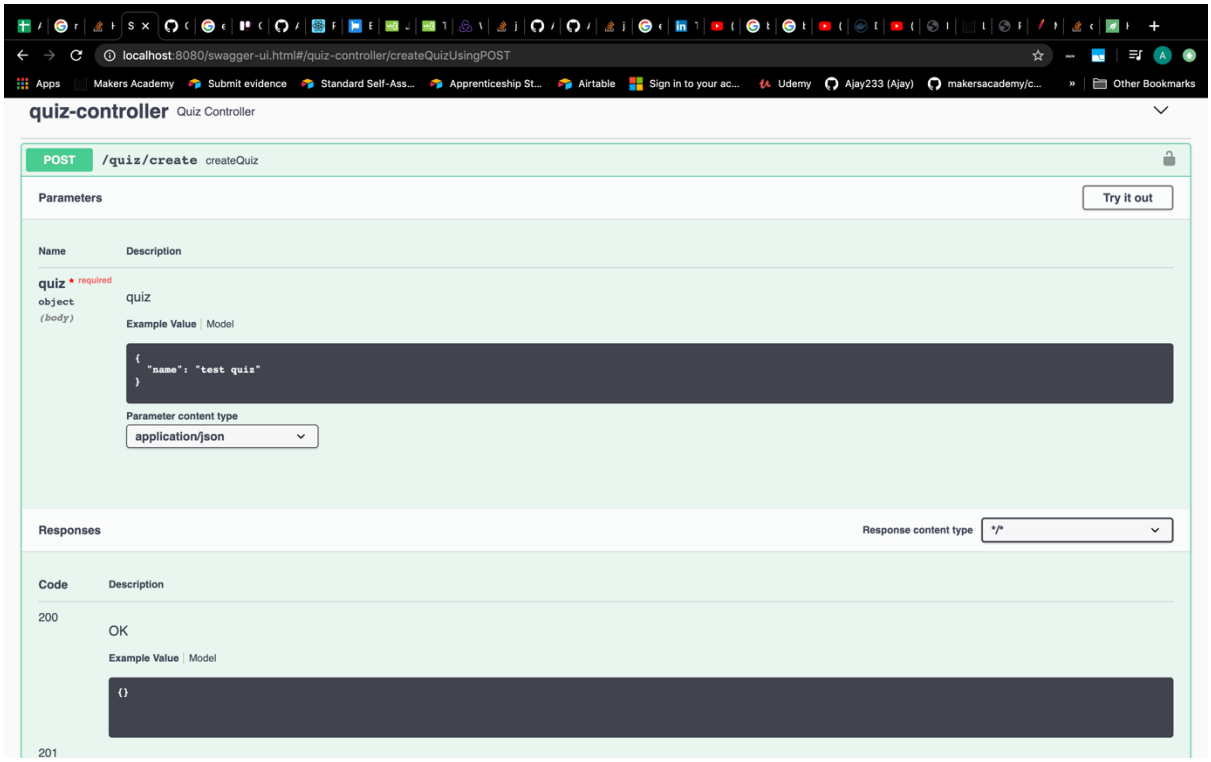
VIEWING THE BACK-END API SWAGGER DOCUMENTATION

The back-end API is only viewable via the Swagger documentation which shows all of the available endpoints. To access the Swagger documentation copy and paste: http://localhost:8080/swagger-ui.html#/ into your browser.

1. View of swagger (what it shows)

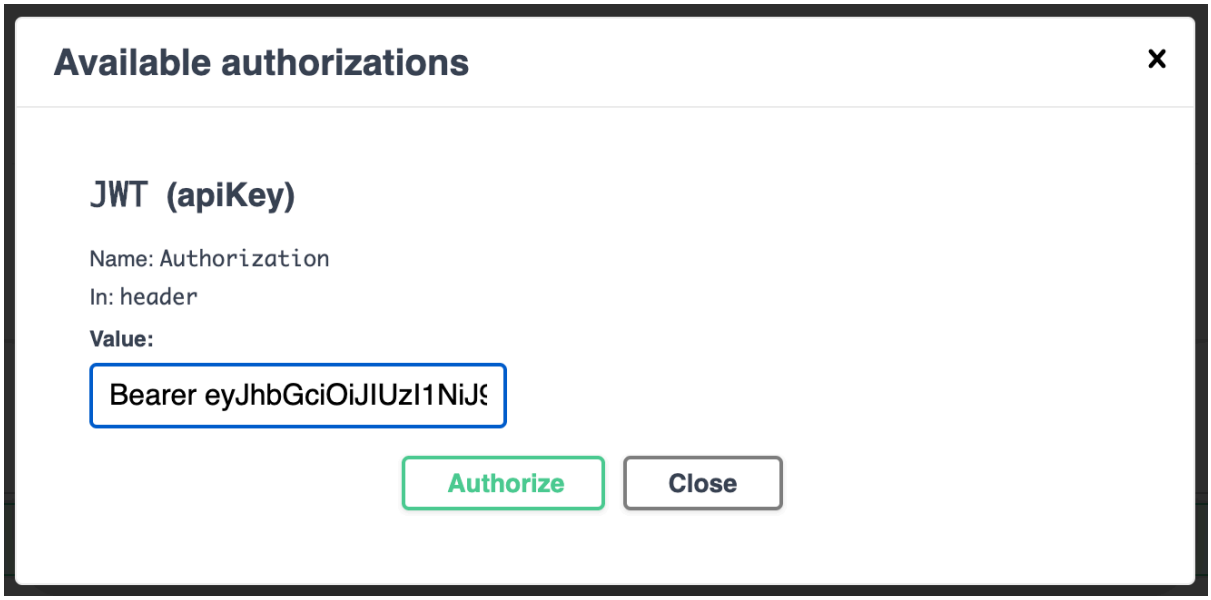


You can try out Swagger by clicking on an endpoint and clicking 'try out'

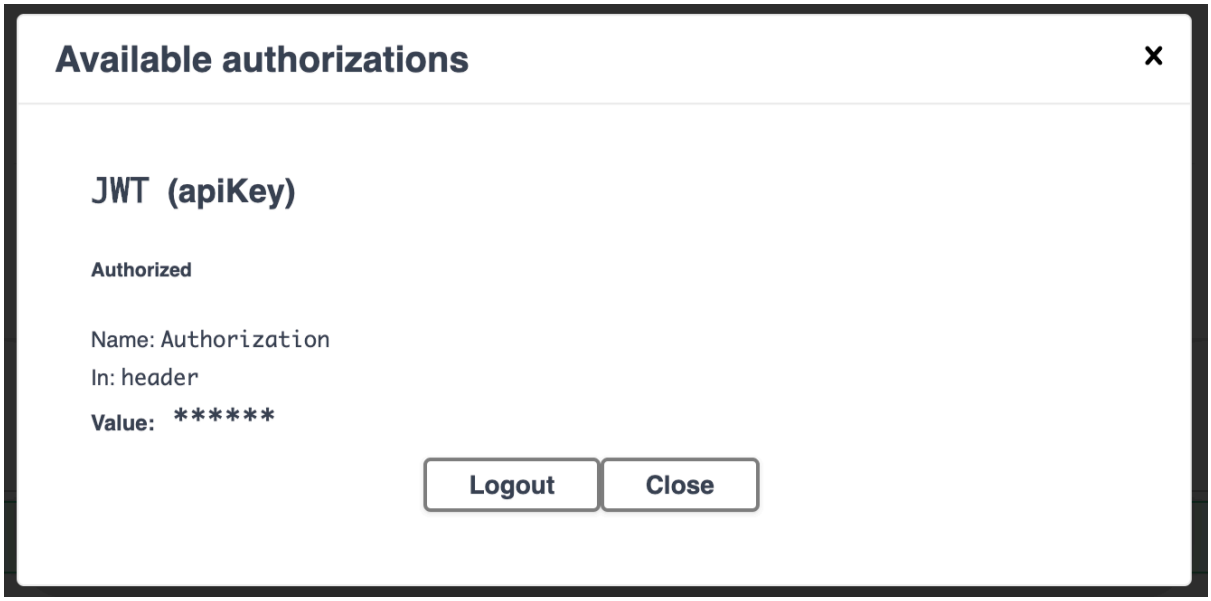


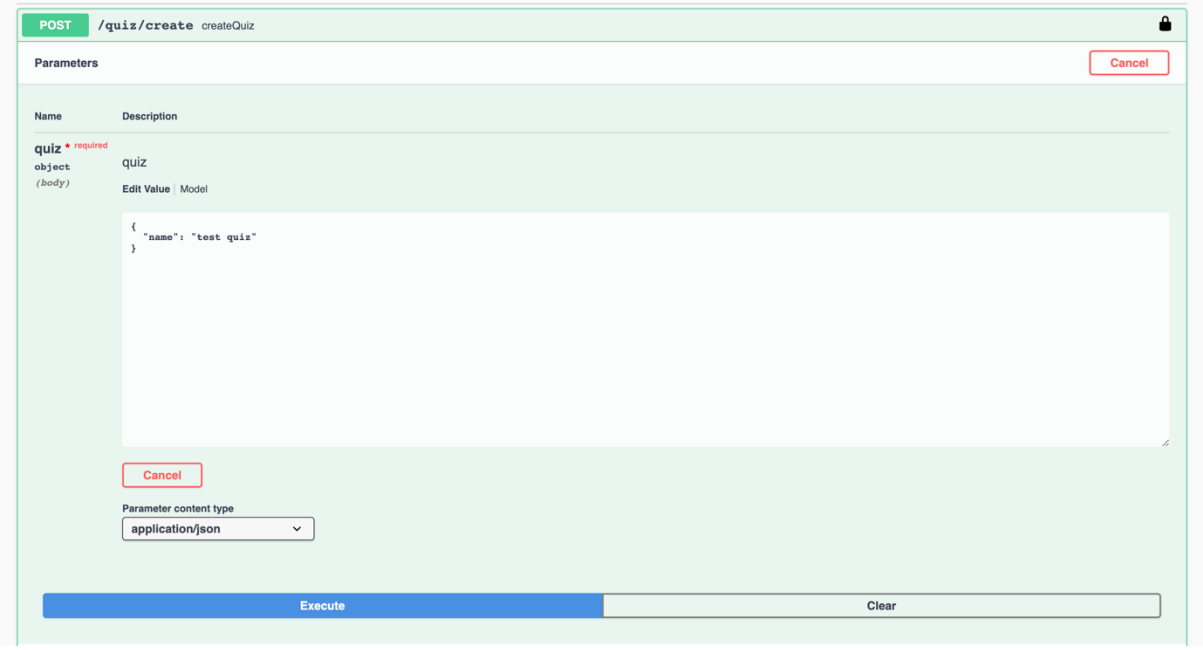
2. You will need to set up the JWT Authorization header to use any endpoints other than login.
3. To do this click login using the login endpoint.
4. Copy the JWT
5. Click the Authorize button

Then enter Bearer, followed by the token



Then click authorize and this will now set up you up to be able to use the endpiints





DEFAULT USERS

Default users have been included with the app as standard. There are 2 user accounts for each type of user.

PERMISSION LEVEL: EDIT

- **User 1**:
  - Username: edit-user1
  - Password: kRnUrWtYw3

- **User 2**:
  - Username: Edit-user2
  - Password: jQnTqVsYv2

PERMISSION LEVEL: VIEW

- **User 1**:
  - Username: view-user1
  - Password: kRnTrWtYv3

- **User 2**:
  - Username: view-user2
  - Password: 4z6B8EbGdJ

PERMISSION LEVEL: RESTRICTED

- **User 1**:
  - Username: restricted-user1
  - Password: QmTqVsXv2x

- **User 2**:
  - Username: restricted-user2
  - Password: z6B8DbGdJf

DATABASE PASSWORD

The database username and password have been pre-set in the within the 'application-dev.properties' file and can be anything you want. The current password and username would need to be changed to a more difficult username and password before the app is put into production.

KEY FOR JWT

The key for the JWT is set within the 'application-dev.properties' file and can be anything you want. It is already set up but can be changed.