QUESTION BOX

A web application used to perform create, read, update, delete (CRUD) operations on a database and also utilize modern security features to ensure data integrity.

Github Repo: https://github.com/TarunGopinath6/https---github.com-TarunGopinath6-Question-Box

TECHNOLOGIES USED:

Backend Development:

- Node.js: An asynchronous event-driven javascript runtime, designed to build scalable network applications
- Express.js: A fast and minimalist web application framework for Node.js used for building the backend server.
- MongoDB: A NoSQL database used for storing question data in a flexible, schema-less format.

Frontend Development:

- React.js: A popular JavaScript library used for building interactive user interfaces.
- Axios: A promise-based HTTP client used for making API requests and handling responses between the frontend and backend.

Additional Technologies:

- HTML/CSS: Used for structuring and styling the user interface of the frontend application.
- JavaScript: The primary programming language used for implementing the application logic and functionality.
- JWT (JSON Web Tokens): A secure and compact way of transmitting information between parties as a JSON object.

API ENDPOINTS:

Base URL: http://localhost:3000

Authentication: Specific endpoints require an Authorization header carrying the accessToken which will be validated using an internal method. This is implemented for the endpoints which need to have data integrity after login.

axiosInstance is an axios request, configured with required interceptors to ensure smooth authentication and error handling dataflow. This is mainly responsible for fetching new accessToken using the refreshToken if expiration occurs. Apart from this, it also handles other known and unknown errors. Find more information on this down below on Page 6.

- **Login endpoint:**

  URL : '/check_user'
  Method : POST
  Description : Validates credentials, authenticates and generates JWT, email for subsequent API requests
  Request Body : email – email id of the user
  password – password for the email id
  Response : 200 – Successful login, returns JWT tokens and email
  404 – User not found
  401 – Invalid password
  500 – Error in database connectivity

  Example:

  Request:

  ```
  POST            /check_user
  Content-Type: application/json
  {
            "email" : tarungopinath6@gmail.com
            "password": 123123

  }
  ```

  Response:

  ```
  Status: 200 OK
  Content-Type: application/json
  {

            "accessToken": <JWT Access Token>
            "refreshToken": <JWT Refresh Token>
            "email"           : <Email ID sent in request body>

  }
  ```

- **Create User endpoint:**

  URL : '/new_user'
  Method : POST
  Description : Takes email and password, hashes the same and stores it in the backend for user authentication
  Request Body : email – email id of the user
  password – password for the email id
  Response : 200 – Successful, user created
  500 – Error in database connectivity

  Example:

  Request:

```
POST            /new_user
Content-Type: application/json
{
        "email" : tarungopinath6@gmail.com
        "password": 123123

}
```

Response:

```
Status: 200 OK
Content-Type: application/json
User created successfully
```

- **Insert Question endpoint:**

URL	: '/insert_question'
Method	: POST
Authentication	: requireToken – Authenticates the JWT in the header
Description	: Inserts new question into database
Request Body	: question – the question to be asked option1 – option 1 for the question option2 – option 2 for the question option3 – option 3 for the question option4 – option 4 for the question
Response	: 200 – Successful, question inserted 500 – Error in database connectivity

Example:

Request:

```
POST            /insert_question
Authorization: <JWT Access Token>
Content-Type: application/json
{
        "question": <ANY QUESTION>
        "option1": <OPTION>
        "option2": <OPTION>
        "option3": <OPTION>
        "option4": <OPTION>
        "answer": <ANSWER>

}
```

Response:

```
Status: 200 OK
Content-Type: application/json
Inserted successfully
```

- **Get Questions endpoint:**

URL : '/get_questions'
Method : POST
Authentication : requireToken – Authenticates the JWT in the header
Description : Gets all the questions from database
Request Body : NIL
Response : 200 – JSON object of questions returned
500 – get_questionsERROR

Example:

```
Request:
        POST            /get_questions
        Authorization: <JWT Access Token>
        Content-Type: application/json

Response:
        Status: 200 OK
        Content-Type: application/json
        <RECORDS as JSON Object>
```

- **Update Question endpoint:**

URL : '/update_question'
Method : POST
Authentication : requireToken – Authenticates the JWT in the header
Description : Updates question with specific _id in the database
Request Body : _id - _id pre-defined from the database
question – the question to be asked
option1 – option 1 for the question
option2 – option 2 for the question
option3 – option 3 for the question
option4 – option 4 for the question
Response : 200 – Update successful
500 – UpdateERROR: Question not updated

Example:

```
Request:
        POST            /update_question
        Authorization: <JWT Access Token>
        Content-Type: application/json
        {
                "_id" : <_id of QUESTION>
                "question": <QUESTION>
                "option1": <OPTION>
                "option2": <OPTION>
                "option3": <OPTION>
                "option4": <OPTION>
                "answer": <ANSWER>
        }

    Response:
        Status: 200 OK
        Content-Type: application/json
        Update successful
```

- **Delete Question endpoint:**

URL	: '/delete_question'
Method	: POST
Authentication	: requireToken – Authenticates the JWT in the header
Description	: Deletes question with specific _id in the database
Request Body	: _id - _id pre-defined from the database
Response	: 200 – Question deleted successfully 500 – Delete error

Example:

```
    Request:
        POST            /delete_question
        Authorization: <JWT Access Token>
        Content-Type: application/json
        {
                "_id" : <_id of QUESTION>

        }

    Response:
        Status: 200 OK
        Content-Type: application/json
        Question deleted successfully
```

- **Refresh Token endpoint:**

URL	: '/refresh_token
Method	: POST
Authentication	: NIL
Description	: Checks the authenticity of the refreshToken sent and returns a new accessToken
Request Body	: refresh_token: refreshToken stored in localStorage email: email ID stored in localStorage
Response	: 200 – New access token returned 500 – Failed to verify refresh token

Example:

```
    Request:
        POST            /refresh_token
        Authorization: NIL
        Content-Type: application/json
```

```
{
        “refresh_token” : <REFRESH TOKEN>
        “email”: <EMAIL>
}
```

Response:

    Status: 200 OK
    Content-Type: application/json
    <NEW ACCESS TOKEN>

- **requireToken Authentication middleware function:**

  Called as a middleware function in all the endpoints where authentication is required. It decodes the token received as header using jwt.decode(token, secretKey) and returns user if successful, else returns 401

- **axiosInstance axios object with interceptor:**

  response:
  returns response as per usual

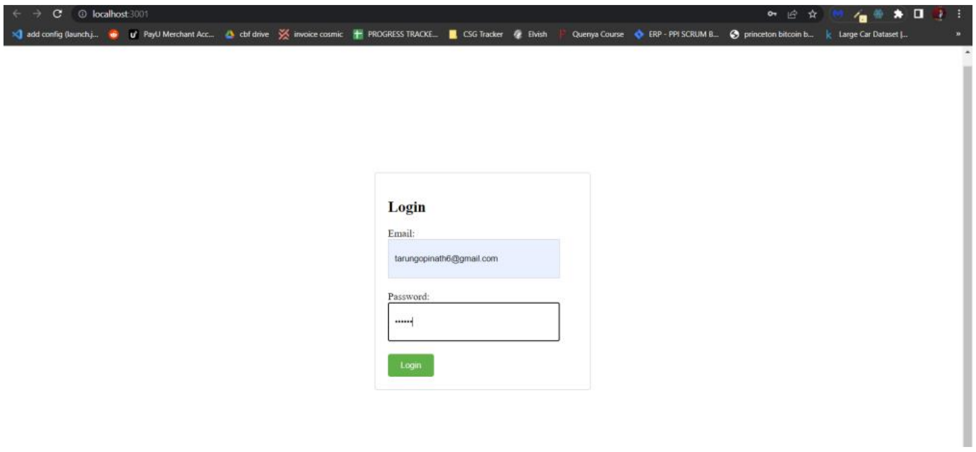
  error.response is undefined:
  Checks for timeout message in error.message, and returns “Please try again”
  If no timeout message is there, returns “SERVER ERROR – CORS/AXIOS”

  error.response.status is 401: (Token authentication error)
  Gets refreshToken, accessToken and email from locaStorage, checks for the nullity of either or all of them, if so, returns “SERVER ERROR – Tokens”
  If they exist, checks for the validity of the refreshToken, if invalid sends to login, if valid, it creates a request to “/refresh_token” and then updates the tokens in localStorage, executes the original request with the newly updated tokens.

  Unknown error: If none of the above conditions are satisfied, returns “UNKNOWN” ERROR”

Emphasis was not laid on frontend as the main objective here is to show the backend functionality, so just a skeletal frontend structure was used for testing.

Login page, before login



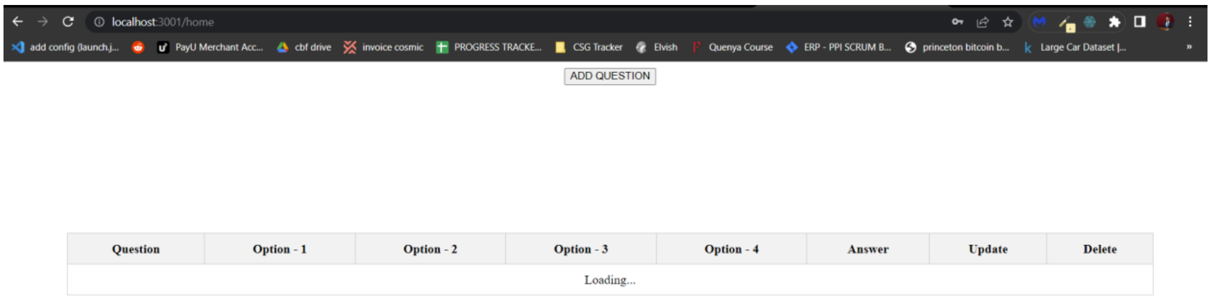
Login

Email: tarungopinath6@gmail.com

Password: ••••••

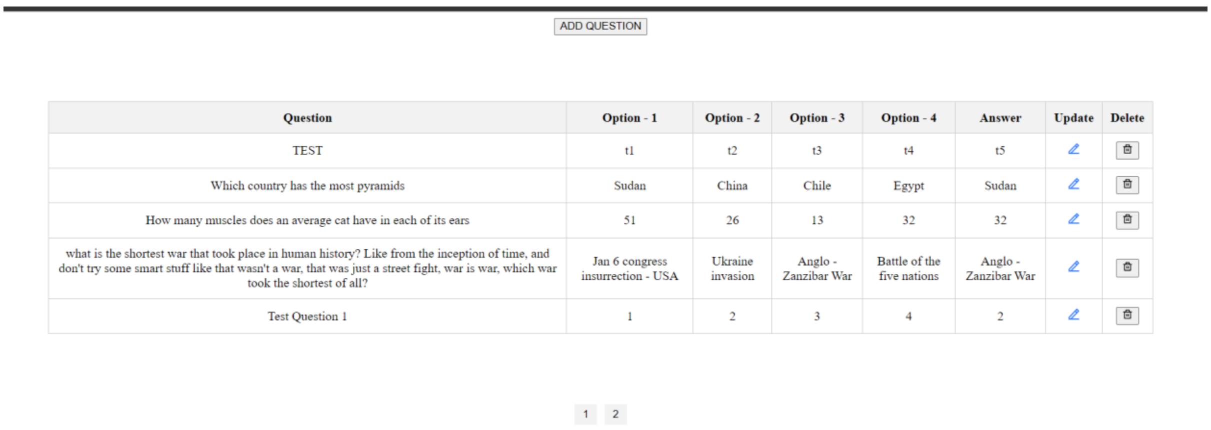
Login

After Login, redirected to home page, waiting for questions to load:



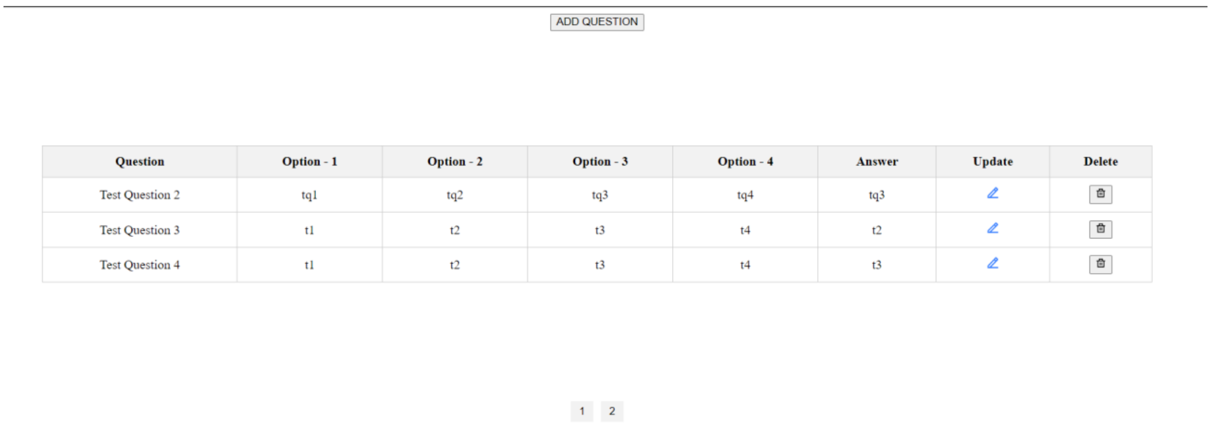
Question	Option - 1	Option - 2	Option - 3	Option - 4	Answer	Update	Delete
Loading...							

Home page, after questions have loaded:



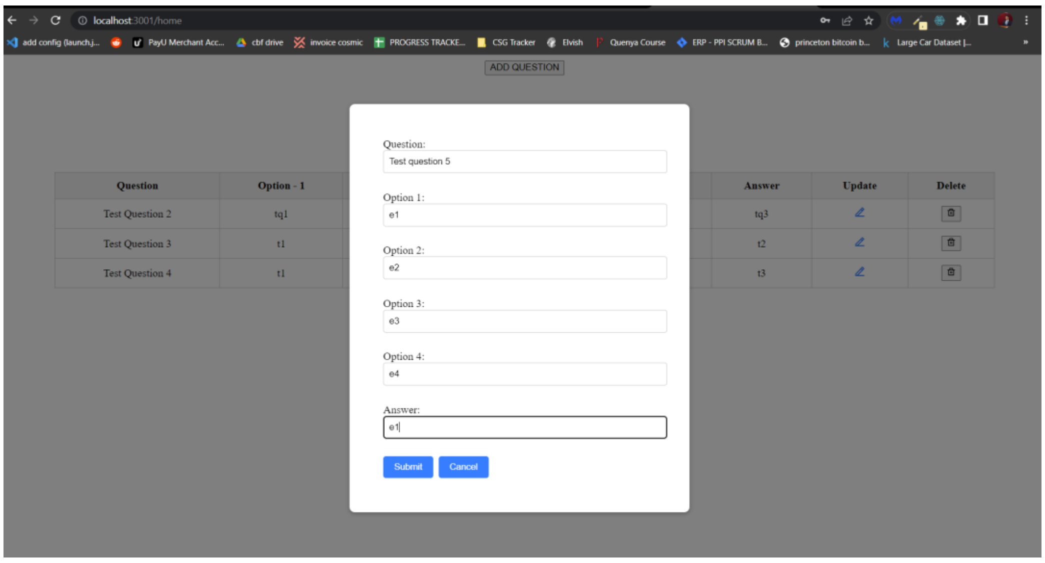
Question	Option - 1	Option - 2	Option - 3	Option - 4	Answer	Update	Delete
TEST	t1	t2	t3	t4	t5	✎	🗑
Which country has the most pyramids	Sudan	China	Chile	Egypt	Sudan	✎	🗑
How many muscles does an average cat have in each of its ears	51	26	13	32	32	✎	🗑
what is the shortest war that took place in human history? Like from the inception of time, and don't try some smart stuff like that wasn't a war, that was just a street fight, war is war, which war took the shortest of all?	Jan 6 congress insurrection - USA	Ukraine invasion	Anglo - Zanzibar War	Battle of the five nations	Anglo - Zanzibar War	✎	🗑
Test Question 1	1	2	3	4	2	✎	🗑

Pagination implemented (page 2 below):



Question	Option - 1	Option - 2	Option - 3	Option - 4	Answer	Update	Delete
Test Question 2	tq1	tq2	tq3	tq4	tq3	✎	🗑
Test Question 3	t1	t2	t3	t4	t2	✎	🗑
Test Question 4	t1	t2	t3	t4	t3	✎	🗑

Add question modal:



Question:
Test question 5

Option 1:
e1

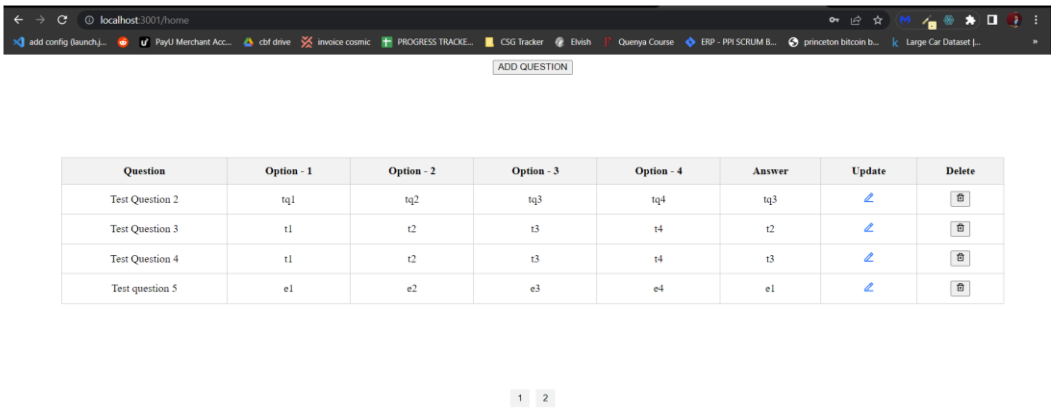
Option 2:
e2

Option 3:
e3

Option 4:
e4

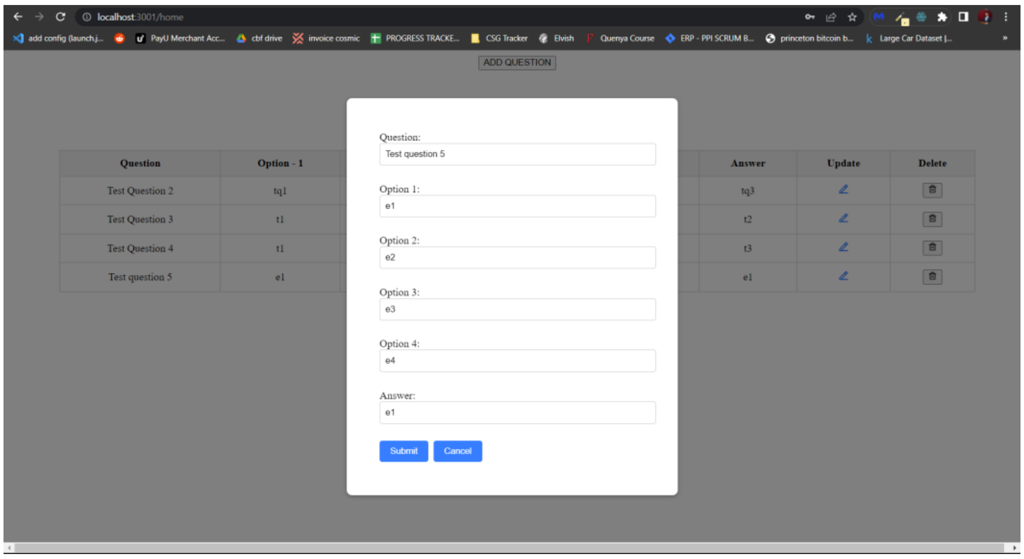
Answer:
e1

Automatic re-render of table after new question submitted:



Question	Option - 1	Option - 2	Option - 3	Option - 4	Answer	Update	Delete
Test Question 2	tq1	tq2	tq3	tq4	tq3	✎	🗑
Test Question 3	t1	t2	t3	t4	t2	✎	🗑
Test Question 4	t1	t2	t3	t4	t3	✎	🗑
Test question 5	e1	e2	e3	e4	e1	✎	🗑

Update question modal, with the existing data pre-filled:



Question: Test question 5

Option 1: e1

Option 2: e2

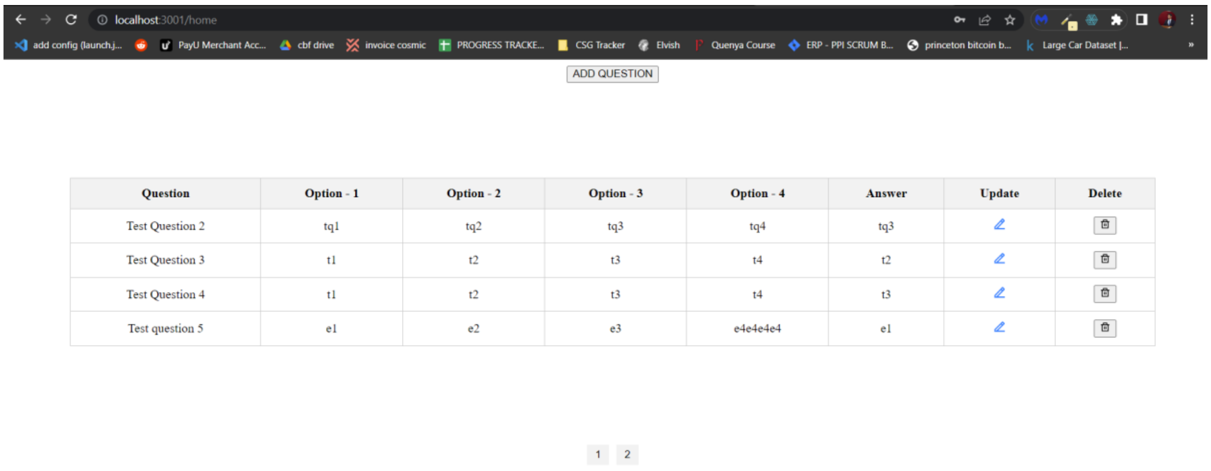
Option 3: e3

Option 4: e4

Answer: e1

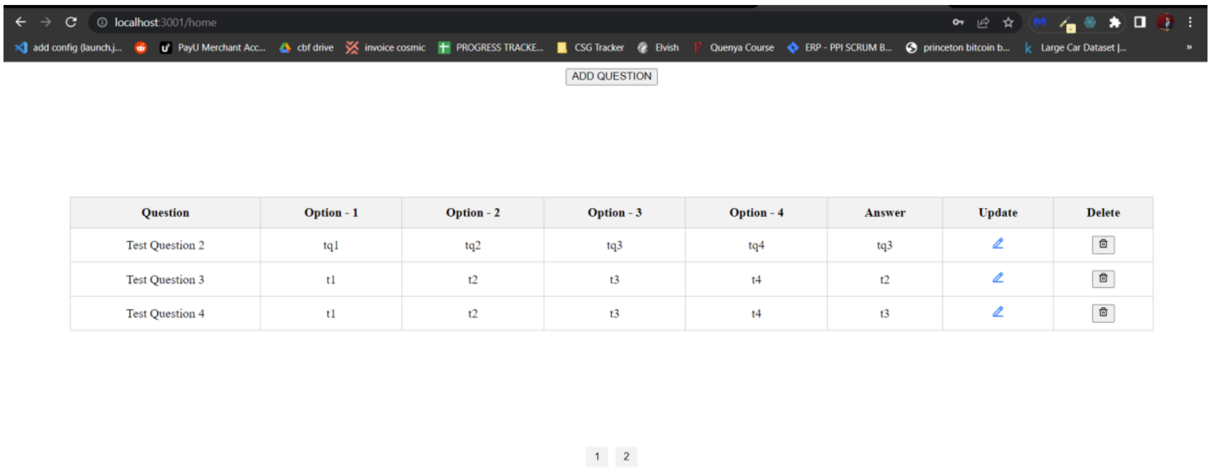
Question	Option - 1	Answer	Update	Delete
Test Question 2	tq1	tq3	🖉	🗑
Test Question 3	t1	t2	🖉	🗑
Test Question 4	t1	t3	🖉	🗑
Test question 5	e1	e1	🖉	🗑

Update question modal, after new data changes made and submitted, home page re-render:



Question	Option - 1	Option - 2	Option - 3	Option - 4	Answer	Update	Delete
Test Question 2	tq1	tq2	tq3	tq4	tq3	🖉	🗑
Test Question 3	t1	t2	t3	t4	t2	🖉	🗑
Test Question 4	t1	t2	t3	t4	t3	🖉	🗑
Test question 5	e1	e2	e3	e4e4e4e4	e1	🖉	🗑

Home page re-rendered after delete button clicked:



Question	Option - 1	Option - 2	Option - 3	Option - 4	Answer	Update	Delete
Test Question 2	tq1	tq2	tq3	tq4	tq3	🖉	🗑
Test Question 3	t1	t2	t3	t4	t2	🖉	🗑
Test Question 4	t1	t2	t3	t4	t3	🖉	🗑