Assignment 3: Quiz app - Admin

***IMPORTANT**: This assignment must be done individually.*

Read Section A to understand the programming requirements, Section B to understand the programming tasks that you need to carry out, Section C to know what you need to submit as the result.

A. Description

In this assignment, you will complete the first sprint to create a **login-less** admin space for Quiz app. In details, CRUD operations to manage questions include:

✓ - ~~List all questions~~
✓ - ~~Get details of a question~~
✓ - ~~Create a new question~~
✓ - ~~Update a question~~
✓ - ~~Delete a question~~

You have 2 options:

- **(1) Server-side rendering using Handlebars**

  You need to implement yourself some JS tasks to **Add answer, Remove answer…** OR **hint**: just fixed with 4 answers for all questions (some score deduced only)

  OR

- [✓ **(2) SPA with ReactJS + backend APIs**]

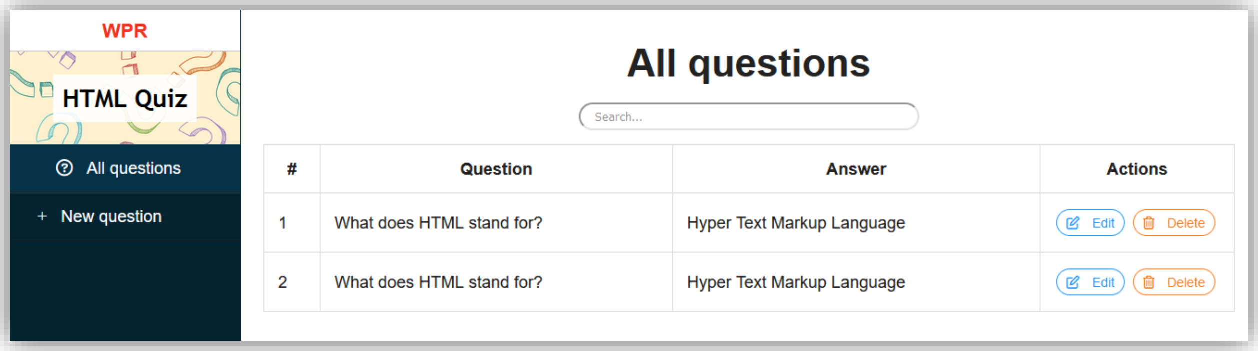
  You need to implement the backend APIs (described in *section A.2*)

Hint: ReactJS helps interactive tasks much easier to implement

1. Frontend

~~All questions~~

Retrieve & display a table of all questions from the database (see `index.html`)



All questions

Search...

#	Question	Answer	Actions
1	What does HTML stand for?	Hyper Text Markup Language	Edit Delete
2	What does HTML stand for?	Hyper Text Markup Language	Edit Delete

✓ - ~~Menu active: *All questions*~~
- Use-cases:
  - ✓ ~~**Search questions**:~~
    - ✓ ~~**User**: types *keyword* into search box~~,
    - ✓ ~~**System**: updates the table to display only questions having text contains *keyword*~~

    ✓ ~~**IMPORTANT**: case insensitive~~

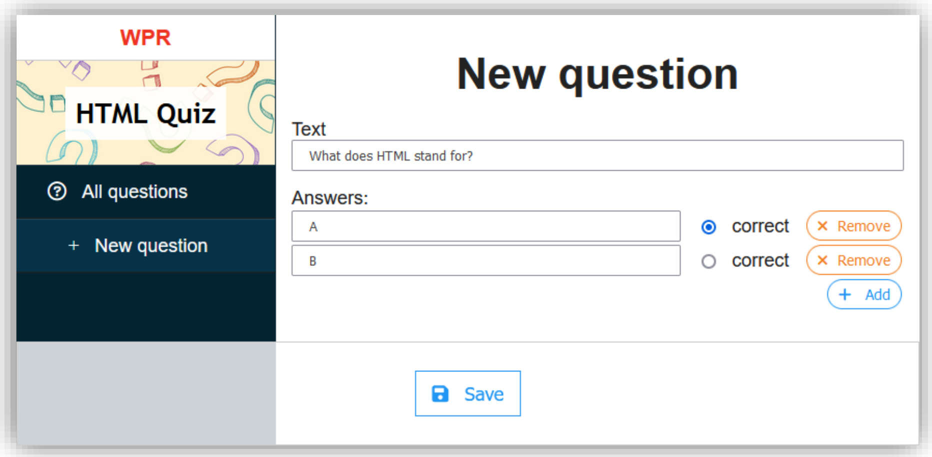
    Hint: (~~*for server-side rendering*) you may use a form for keyword, when user press "Enter", just refresh the page with filtered data~~

  - ✓ ~~**Edit question**:~~
    - ~~**User**: clicks "Edit" to Update the selected~~ question
    - ~~**System**: displays form with data of selected question for editing~~
  - ✓ ~~**Delete question**:~~
    - ~~**User**: clicks "Delete" to Delete the selected question~~
    - ✓ ~~**System**: asks for confirmation from user. If user confirmed, delete the~~ selected ~~question from database & update table to reflect the changes.~~

    ✓ **Hint**: see [`window.confirm()`]

Create a new question

Display a form for adding new question (see `add.html`)



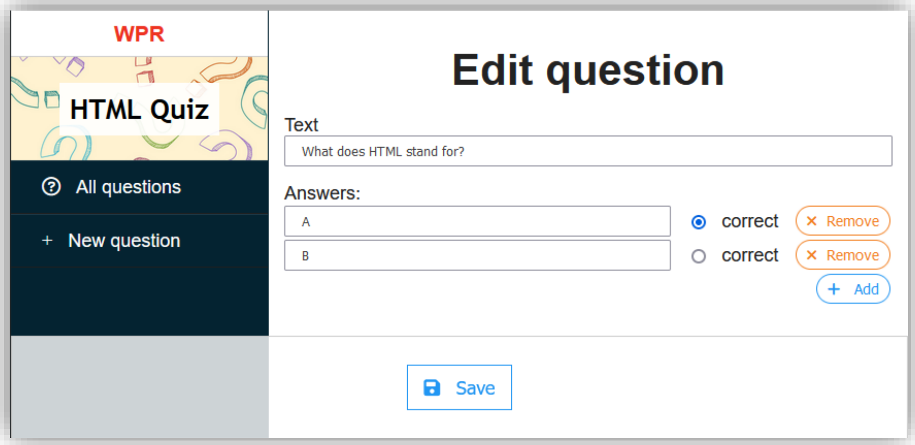
- ~~Menu active: *New question*~~
- Use-cases:
  - ~~**Add answer**~~:
    - ~~**User**: clicks "Add" to add more answer~~
    - ~~**System**: adds a new row for entering another answer~~
  - ~~**Remove answer**~~:
    - ~~**User**: clicks "Remove" to remove~~ answer
    - ~~**System**: removes the selected answer~~
  - ~~**Mark answer correct**~~:
    - **User**: checks ~~radio box corresponding to the correct answer~~
  - **Save question**:
    - ~~**User**: clicks "Save" to save question~~
    - **System**: checks & alerts user if data entered properly (te~~xt~~, correct answer, empty answer…). ~~If data is valid, system inserts question into the database & redirects user to table of "*All questions*"~~

?? . .

Hint: se~~t `correctAnswer` as -1 if no correct answer selected~~. "~~Remove answer" which is correct also update this value.~~

Update a question

Display form for editing selected question (see `edit.html`)



✓ - ~~Menu active: *none*~~
- Use-cases:
  - ✓ ~~**Add answer**~~**,** ~~**Remove answer**~~**,** ~~**Mark answer correct**~~: similar to "*Create a new question*"
  - ~~**Save question**~~:
    - ~~**User**: clicks "Save" to save question~~
    - **System**: checks & alerts user if data entered properly (te~~xt, correct answer~~, empty answer…). If ~~data is valid~~, ~~system updates the selected question with updated data into the database & redirects user to table of "All questions"~~

✓ 2. API end-points (in case you choose option (2) – SPA with ReactJS)

Note about Question format

The data about question follows the structure below:

- `_id`: question id (auto generated value by the Mongo DBMS)
- `text:` question content
- `answers:` array of options for user to choose
- `correctAnswer:` index of correct answer in the array of `answers`

For example,

```
{
    "text": "Who is making the Web standards?"
    "answers": [
        "Mozilla",
        "Microsoft",
        "Google",
        "The World Wide Web Consortium"
    ],
    "correctAnswer": 3
}
```

1 ~~**Question: Get all questions**~~

✓ ~~**GET:** /questions~~

- ~~**Request:**~~
  - **Data:** none
- **Response:**
  - **Status:** 200 (OK)
  - ~~**Data:**~~ array of all questions

~~*Use-case*: show a table of all questions~~

Client: sends request to get all questions

Server: gets all questions from database & returns

✓ 2 ~~**Question: Create a new question**~~

POST: /questions

- **Request:**
  - **JSON body:** ~~an object of required fields~~ to create a new question, containing text, array of answers & index of the correct answer
- ✓ **Response:**
  - **Status:** 201 (CREATED)
  - **Data:** just created question

Use-case: user fills in the form for required information then click "Save"

Client: send request to create a new question

Server: create a new **Question** with data from user → save into the database & returns

✓ 3 Edit -> ~~**Question: Get details of a question**~~

✓ **GET:** /questions/:id

- **Request:**
  - **Route param:**
    - :id id of the question
- ✓ **Response:**
  - **Status:** 200 (OK)
  - **Data:** an object of question info
- ✓ **Extension:**
  - In case of invalid :id, return
    - **Status:** 404 (NOT FOUND)

Use-case: user selects to edit an existing question

~~**Client:** sends request to get details of an existing question specified by~~ :id

~~**Server:** query for question specified by given :id from database & returns.~~

✓ 4 ~~**Question: Update a question**~~

PUT: /questions/:id

- ~~`Request:`~~
  - ~~`Route param:`~~
    - `:id id of the question`
  - ~~`JSON body: an object of question info to update`~~
- ✓ `Response:`
  - `Status: 200 (OK)`
  - `Data: The updated question`
- ✓ `Extension:`
  - ✓ `In case of invalid id`
    - ~~`Status: 404 (NOT FOUND)`~~
  - ? ~~`In case of invalid data`~~ `for question`
    - `Status: 400 (INVALID REQUEST)`

Use-case: user fills in the form with updated information then click “Save”

Client: sends request ~~with data to update the `Question`~~ specified by `:id`

Server:

- ✓ Get the question with specified id → if not exist, return 404 NOT FOUND
- Validate data for question → if invalid, return 400 INVALID REQUEST
- ✓ Update the specified question with data & return

5 ✓ ~~Question: Delete a question~~

`DELETE: /questions/:id`

- `Request:`
  - `Route param:`
    - `:id id of the question`
- `Response:`
  - `Status: 200 (OK)`

✓ *Use-case*: user fills in the form with updated information then click “Save”

Client: sends request to delete the `Question` specified by `:id`

Server: deletes the question specified by `:id` & return 200 OK (even if `:id` is valid)

Data Model

- **Questions:** A question contains the text content, an array of options for user to select, also the correct answer for this question (identified by its index in the options array)

NOTE: In this assignment, you HAVE TO

1. ✓ Name your db as **wpr-quiz**. Import & use the provided ***questions*** collection (with sample data for testing). We will use this file for marking also.
2. ✓ Serve ExpressJS server at port **3001**
3. ✓ ReactJS app run at default port **3000**

✓4. Use MongoDB with default configuration (default port **27017**, **no username/password** required)

B. Task requirements

In this assignment, you are free to organize your app

1. Option (1) server-side rendering

a. *Structure your web application*

Note: `package.json` must be included.

Name your folder as

- server-side-rendering

b. *Server-side rendering*

- Create routes to serve functions as mentioned in *(Section A)*

[✓ 2.] Option (2) fullstack SPA with React

a. ~~*Structure your web application*~~

✓ **Note**: `package.json` must be included.

✓ Name your folders as

- ✓ frontend (for reactjs)
- ✓ backend (for express apis)

✓ b. *Frontend (ReactJS)*

- Create components to serve functions as mentioned in *(Section A)*

✓ c. *Backend (ExpressJS + MongoDB)*

- Create API routes to serve functions as mentioned in *(Section A)*

3. Weekly plan

You have to schedule yourself a weekly plan to complete your tasks, an example is given below.

Week	Functionalities
1 **Check point 1**	- Shared between functions: Layout/ Routing ○ active menu - Function: All questions with search - Other functions: empty pages/ empty components
2 **Final submission**	- Function: Create a new question - Function: Update a question - Function: Delete a question

C. Submission

You must submit a single zip file containing the application to the portal by the due date. The zip file name must be of the form a3_Sid.zip, where *Sid* is your student identifier (the remaining bits of the file name must not be changed!). For example, if your student id is 1801040001 then your zip file must be named a3_1801040001.zip.

Note: Do not include `node_modules` folder into your submission

IMPORTANT: failure to name the file as shown will result in no marks being given!

NO PLAGIARISM: if plagiarism is detected, 0 mark will be given!