CSE 316: Fundamentals of Software Development
Fall, 2023
Programming Assignment 03 – node.js, express, mongoDB

Assigned: Wednesday, 10/18/2023
Due: Wednesday, 11/08/2023, 11:59 PM.

Learning Outcomes:
After completing this homework assignment, you will have

1. learned server-side programming.
2. understand the principles of NoSQL Databases.
3. gained experience with full-stack development.

Introduction:
In this assignment, we will augment the fake stackoverflow.com web application that we have been building with a server. This server will host all the static and dynamic resources required by the application. The server will run on the Node platform, which is convenient since we can now write JavaScript code for both the client and the server. We will also use MongoDB as the back-end database to store data that will persist across user sessions. We will still use React as the front end to render content in the browser.

Getting Started
We will use MongoDB as the NoSQL database to store data related to this application. Follow the instructions in the official MongoDB documentation to install the free community edition. On Windows, you should unselect the option *"MongoDB as a Service"* to complete the installation. After you install it, follow the instructions to start MongoDB as a **background service**.

When you install MongoDB, the Mongo Shell (**mongosh**) should also have been installed. (If it wasn't then, follow the instructions here.) The Mongo Shell provides a command line interface that is used to interact with the databases in MongoDB. If *mongosh* is successfully installed then the command **mongosh** should connect to the local instance MongoDB on your machine and open an interpreter where we can type commands to interact with MongoDB. Try the command `show dbs` and you should see a list of existing databases in your local instance. **Note that by default, the MongoDB service will run on *127.0.0.1* (localhost), port *27017*. It is recommended that you do not change these settings for grading convenience.**

Install Node.js. We will use this to manage React and the packages needed to run our server. When you install Node.js, it will come with the **npm** package manager, which will also get installed. We will use **npm** to install dependencies and also to start the react application.

Download/clone your personal GitHub repository. The repository has a *server* and *client* directory. Each directory has the `package.json` and `package-lock.json` files which list the dependencies of the *server* and *client* applications respectively. In each of the directories run **`npm install`** to install the necessary dependencies. The following paragraphs list the dependencies that will be used.

We will use the express framework to write server-side code. Install express in the server directory using the command **`npm install express`**, if not already installed. If you don't yet understand how to use express, look at the worksheet examples posted in Brightspace. For more detailed guidance look at the official Express documentation.

We will use the mongoose data modeling library. Mongoose will help us connect with a MongoDB instance and define operations to manage and manipulate the data according to the needs of our application. Install it in the server directory using **`npm install mongoose`**, if not already installed.

We will use the nodemon process manager so we don't have to restart the server every time we save changes to our server during the development process. Install it in the server directory using **`npm install nodemon`**, if not already installed. Alternatively, if the local install does not work, you can install nodemon globally using the command **`npm install -g nodemon`**. This is a good option for nodemon since it can be used across multiple node projects. To run the server using nodemon use the command **`nodemon server/server.js`** instead of **`node server/server.js`**.

We will use the axios library to send HTTP requests to our server from our client application. Refer to the code examples in Brightspace, which demonstrate how axios is used in conjunction with React and Express. Install it in the client directory using **`npm install axios`**, if not already installed.

We will use the `cors` middleware to enable `CORS` for seamless connection between the client and the server during the development process. This is typically removed when the application is deployed in production to prevent `CORS` attacks. However, since we are assuming a development environment in this homework, we will keep the middleware. Read more about `CORS` https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS.

Grading

We will clone your repository and test your code in the Chrome web browser. You will get points for each functionality implemented. **Make sure you test your code in Chrome**. The rubric we will use is shown below:

1. Home Page: 20 pts.
2. Post a New question: 10 pts.
3. Searching by text: 10 pts.
4. Searching by tags: 10 pts.
5. Answers Page: 10 pts.
6. Post a new answer: 10 pts.
7. All Tags page: 20 pts.
8. Questions of a tag: 10 pts.
9. MongoDB Schema: 20 pts.
10. Modularity: 5 pts.
    Modularity means that your code should be divided into components defined in different files. Related components should be in one file. If a component needs functionality that belongs to other files, then they must be imported and exported appropriately.
11. Code Quality: 5 pts.
    Your code should comply with standard best practices followed in the React and JavaScript communities.

Total: 130 pts.

IMPORTANT NOTES:

1. The application must be made using **node**, **mongodb**, and **react**. You will not receive credit if you use anything else or do not use any of the above technologies. You will be penalized if bad coding practices are found in your codebase.
2. You are free to choose a git workflow. You can use the same workflow as in homework 2 or whatever is convenient for you.
3. Remember to write down the contribution of each team member in the README.
4. We will use the rules defined in ESLint https://eslint.org/docs/latest/rules/ to test code quality. To run ESLint on your code base type the following commands:

```
$ cd </path/to/your/repo>
$ npx eslint src # run eslint on all files in src/
$ npx eslint src/<file>.js # run eslint on individual files
```

Client/Server Application Architecture

The homework repository is structured as a client/server application. It has 2 directories – **server** and **client**. The **server** directory contains the data model in the **models** directory in files

answers.js, *questions.js*, and *tags.js*. These files are empty. You should fill them up with the schema definition for each corresponding document in our MongoDB collection as defined in the **Data Model** section. The **server** directory has a test script called **`populate_db.js`**. You should run this script to verify that your document schema is defined correctly. Read the instructions at the top of the script to see how to run it. If this script throws an error, you will know that something is wrong with the schema definition and you have to fix it. **You must not change this test script. If you do, you will not get any credit**.

The **`server/server.js`** file is the main server script. We will run this script in Node to start your server using the command **`node server/server.js`**. On running the script, a server should start in **https://locahost:8000**. Further, the server should connect to a running instance of MongoDB on server launch. The MongoDB instance should run with default settings, that is, **`mongodb://127.0.0.1:27017/`**. **The database name must be `fake_so`**.
When the server is terminated (using CTRL+C), the database should also be disconnected and the message **"Server closed. Database instance disconnected"** should be displayed.

The **client** directory has the same structure as that of the React application you made in homework 2. You can reuse all of the code that you wrote for homework 2 here. You should use Axios in the client application to send HTTP method requests to the server.

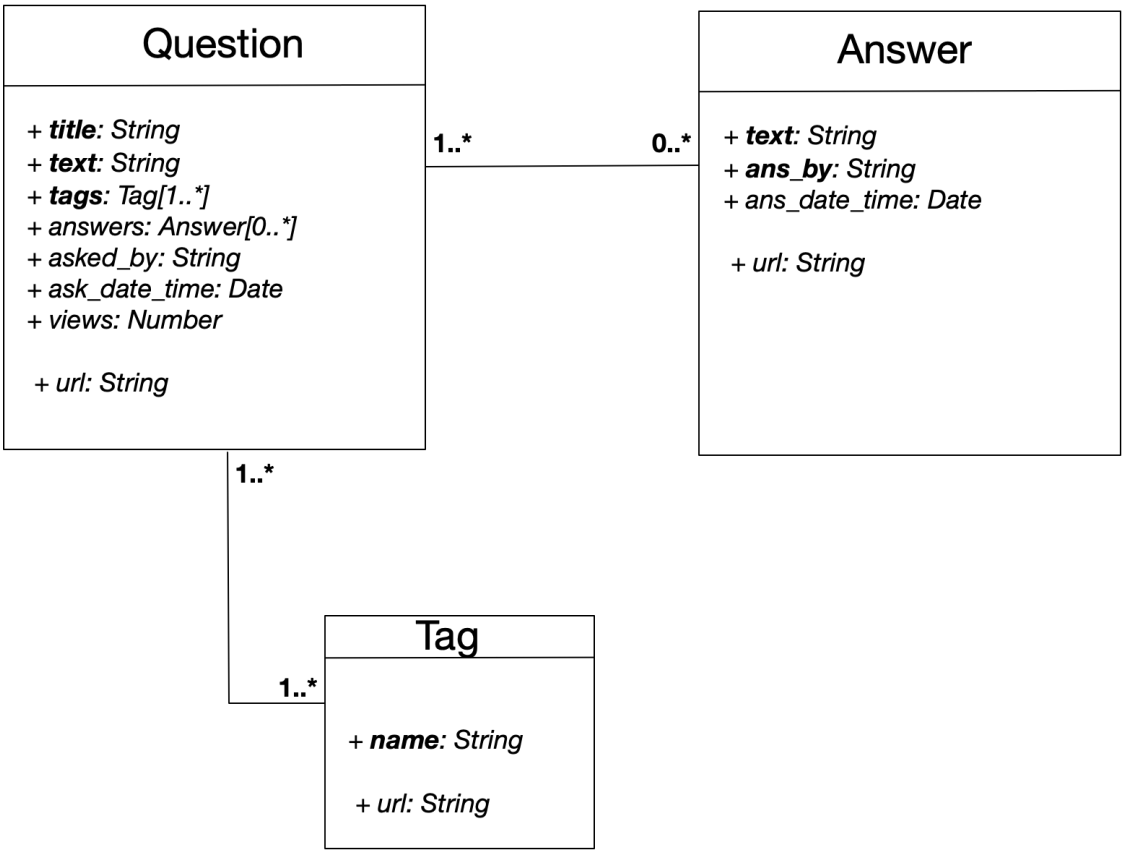
Summary of the default host/port settings:

Client Instance	**`https://localhost:3000`**
Server Instance	**`https://localhost:8000`**
Database Instance	**`mongodb://127.0.0.1:27017/fake_so`**

Assignment Description Continues On Next Page

Data Model

The primary data elements we need to store for this application are questions, tags, and answers. To this end, we will use a UML association notation to describe the data model. Read more about UML class diagrams here.



The UML model describes the schema for all documents in our MongoDB database. All elements in a document, except `url`, are fields, The `url` element is a read-only (*get*) virtual method that returns the URL for the document. So, they should return the following strings:

- `Question.url` returns *`posts/question/_id`*.
- `Answer.url` returns *`posts/answer/_id`*.
- `Tag.url` returns *`posts/tag/_id`*.

The elements highlighted in bold are **required fields**.

Additional Constraints

- *Question.title* must not be more than 100 characters.
- *Question.ask_date_time* must have the current timestamp as the default value.
- *Question.views* must have 0 as the default value.
- *Question.asked_by* must have 'Anonymous' as the default value.
- *Question.ask_date_time* and *Answer.ask_date_time* must have the current timestamp as the default value.

Note the UML association diagram shows edges between each document model. These edges have multiplicity associated with them as specified on the endpoints. For example, `0..*`, on the endpoint of the edge to *Answer* shows that a question can have 0 or more answers. Similarly, an answer is associated with 1 or more questions. Use the association information to select the appropriate data structures.

Application Behavior and Layout

The front-end functional requirements for this application are the same as homework 1 and 2. For convenience, the requirements have been listed below again. You can reuse the React code and CSS you created for those homeworks. Although you will have to make some changes to your front-end code so you can send HTTP requests to your node server.

Assignment Description Continues On Next Page

Home Page

When a user loads the application in the browser for the first time, the home page should be displayed as shown in figure 1. The home page has two parts, the *banner* and the *main* body which will display the content. The *banner* should be displayed at the top of the page and it should contain the following

- The title of the application **Fake Stack Overflow**
- A search bar where users can do textual searches.

The *main body* has two parts. The left side is a menu and the right side displays all questions asked in the forum.

The menu has two links – *Questions* and *Tags*. Clicking on the *Questions* link always displays the home page, i.e, the page being currently described. Clicking on the *Tags* page will display the Tags page (described later). If the user is currently on the page that shows all questions, the *Questions* link should be highlighted with a gray background color. If the user is on the Tags page, the *Tags* link should be highlighted with gray background color. Further, the right side of the *main body* of the *home* page should be displayed as shown in figure 1 with the following elements:

- A header which displays the text **All Questions** and a *button* with the label **Ask Question**.
- The total number of questions currently in the model.
- Three buttons – *Newest*, *Active,* and *Unanswered.*
    - Clicking the *Newest* button should display all questions in the model sorted by the date they were posted. The most recently posted questions should appear first.
    - Clicking the *Active* button should display all questions in the model sorted by answer activity. The most recently answered questions must appear first.
    - The *Unanswered* button should display only the questions that have no answers associated with them.
- Each question should be displayed as shown in figure 1 with the following elements:
    - The no. of answers and the no. of times a question has been viewed. Every time a user clicks on a question should increase the no. of views by 1.
    - Question title.
    - Question metadata, which includes the username of the user who posted the questions and the date the question was posted. The metadata has a particular format. If a question was posted on day X, for the entirety of day X, the question date should appear in seconds (if posted 0 mins. ago), minutes (if posted 0 hours ago), or hours (if posted less than 24 hrs ago). On the other hand, if we were viewing the page 24 hrs after the question was posted then the metadata should be

displayed as *<username> asked <Month><day> at <hh:min>*. Further, if the question is viewed a year after the posted date then the metadata should be displayed as *<username> asked <Month><day>,<year> at <hh:min>*. Here are a few examples:

- question posted on Feb 9th, 2023, 09:20:22 and viewed on the same day at 11:30:21, the metadata should be displayed as *<username> asked 2 hours ago*.
- question posted on Feb 9th, 2023, 09:20:22 and viewed on the same day at 09:25:58, the metadata should be displayed as *<username> asked 5 minutes ago*.
- question posted on Feb 9th, 2023, *09:20:22* and viewed on the same day at 09:20:58, the metadata should be displayed as *<username> asked 36 seconds ago*.
- question posted on Feb 9th, 2023, *09:20:22* and viewed on Mar 31, 2023, 09:20:58, the metadata should be displayed as *<username> asked Feb 9 at 09:20*.
- question posted on Feb 9th, 2022, *09:20:22* and viewed on Mar 31, 2023, 09:20:58, the metadata should be displayed as *<username> asked Feb 9, 2022 at 09:20*.

- All questions should be displayed in *Newest* order by default.
- There should be a dotted line to divide each question entry.
- If the total no. of questions is more than the page can hold, add a scroll bar.

- Make sure that all fonts and content are clearly legible. They don't have to be the exact same as the fonts in figure 1.

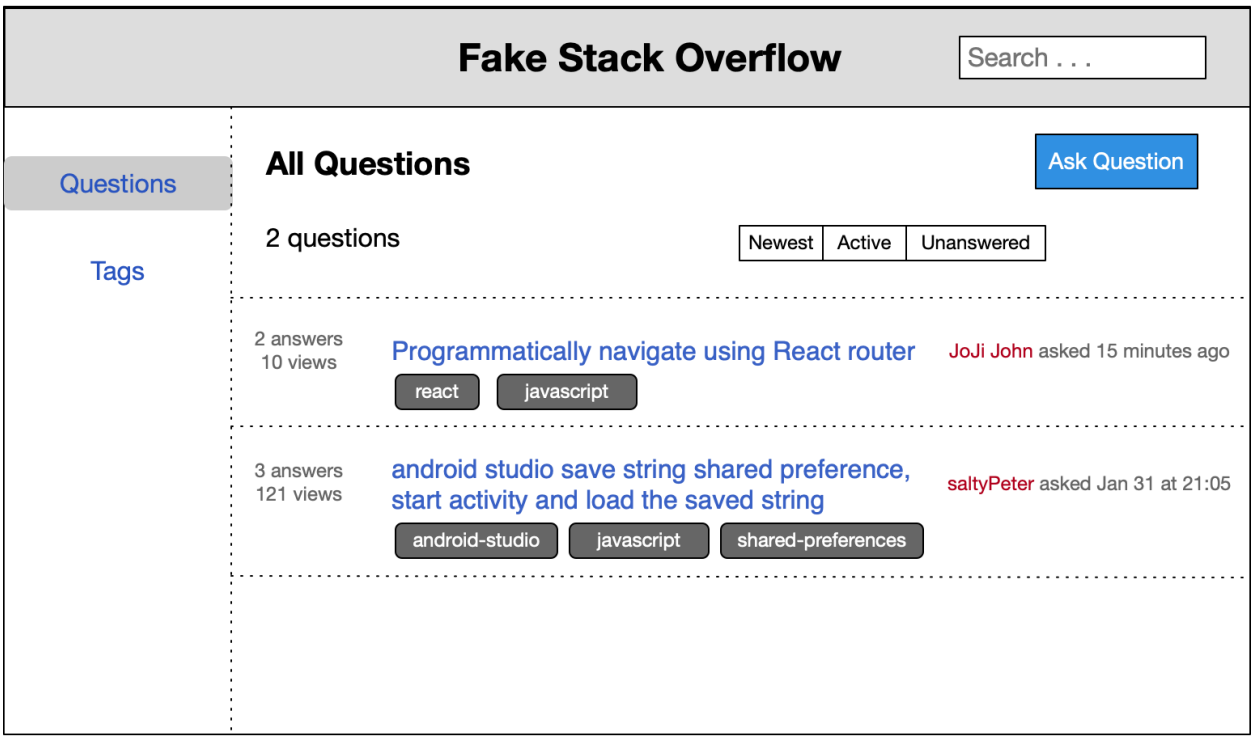


Figure 1

New Question Page

When a user clicks on the **Ask Question** button, the *main body* section of the page should display a form as shown in figure 2 with the following elements:

- A text box for question title. The title should not be more than 100 characters and should not be empty.
- A text box for question text. Should not be empty. No restriction on max length of characters.
- A user is allowed to add hyperlinks in question text. Hyperlinks are identified in the text if a user encloses the name of the hyperlink in [] and follows it up with the actual link in (). For example, in Figure 2, the question text has a hyperlink “web scripting”. The target of a hyperlink, that is, the stuff within () cannot be empty and must begin with “https://” or “http://”. Warn the user with an error message if this constraint is violated.
- A text box for a list of tags that should be associated with the question. This is a whitespace-separated list. Should not be more than 5 tags. Each tag is one word, hyphenated words are considered one word. The length of a tag cannot be more than 10 characters.
- A text box for the username of the user asking the question. The username should not be empty. Display an appropriate error message for invalid user input below the text box.

- A button with the label *Post Question*.
- Each input element in the form should have an appropriate hint to help the user enter the appropriate data as shown in Figure 2.
- Display an appropriate error message for invalid inputs below the respective input element.

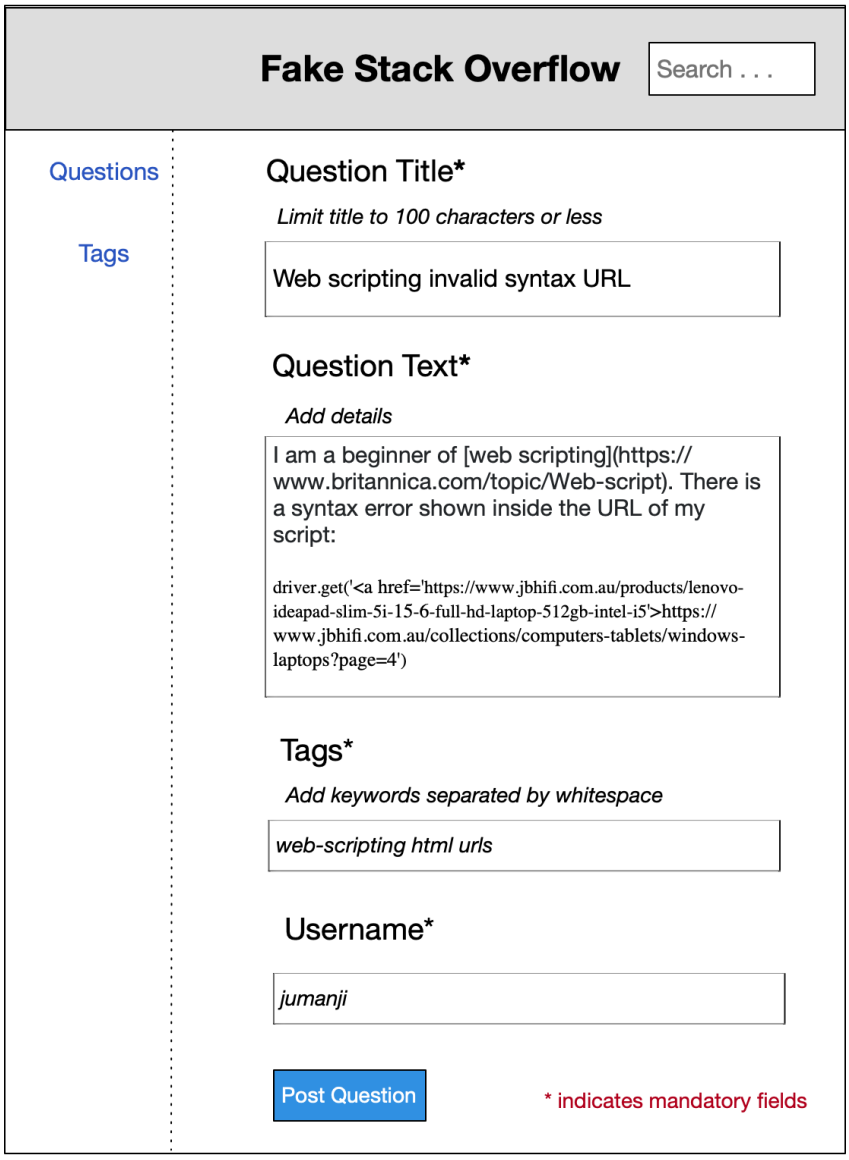


Figure 2

When the *Post Question* button is pressed, the question should be added to the *data object* in *model.js*. If the question is added successfully then the user should be taken to the home page where the *main body* section should display all the questions including the question currently added. Further, the page should also display the total no. of questions, which should have

incremented by 1. Figure 3 shows an example where the first question displayed was most recently asked by the user *jumanji* using the inputs on the new question form.

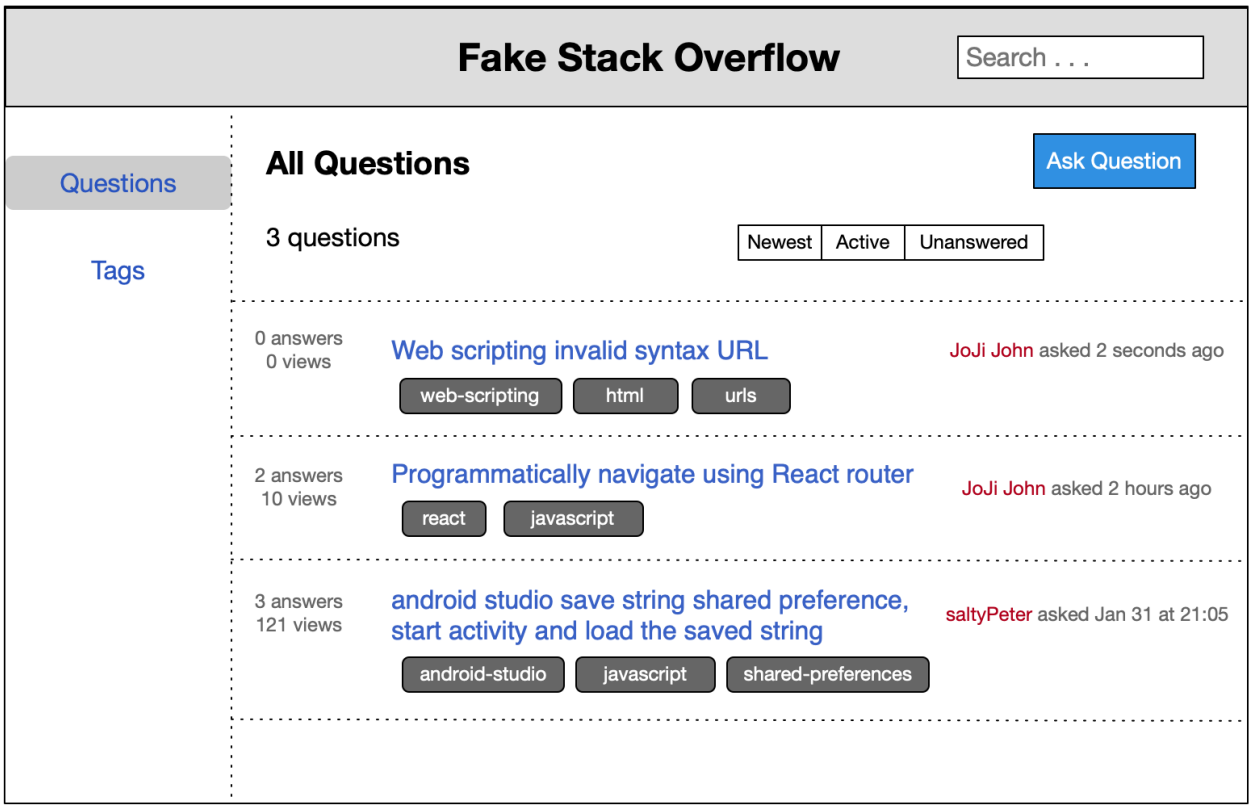


Figure 3

Searching

A user can search for certain questions based on words occurring in the question text or title. On pressing the ENTER key, The search should return *all questions in the default (newest) order for which their title or text contains at least one word in the search string*. For example, in figure 4, there is only one question in our data model with title or text that matches the search string 'Shared Preference'.

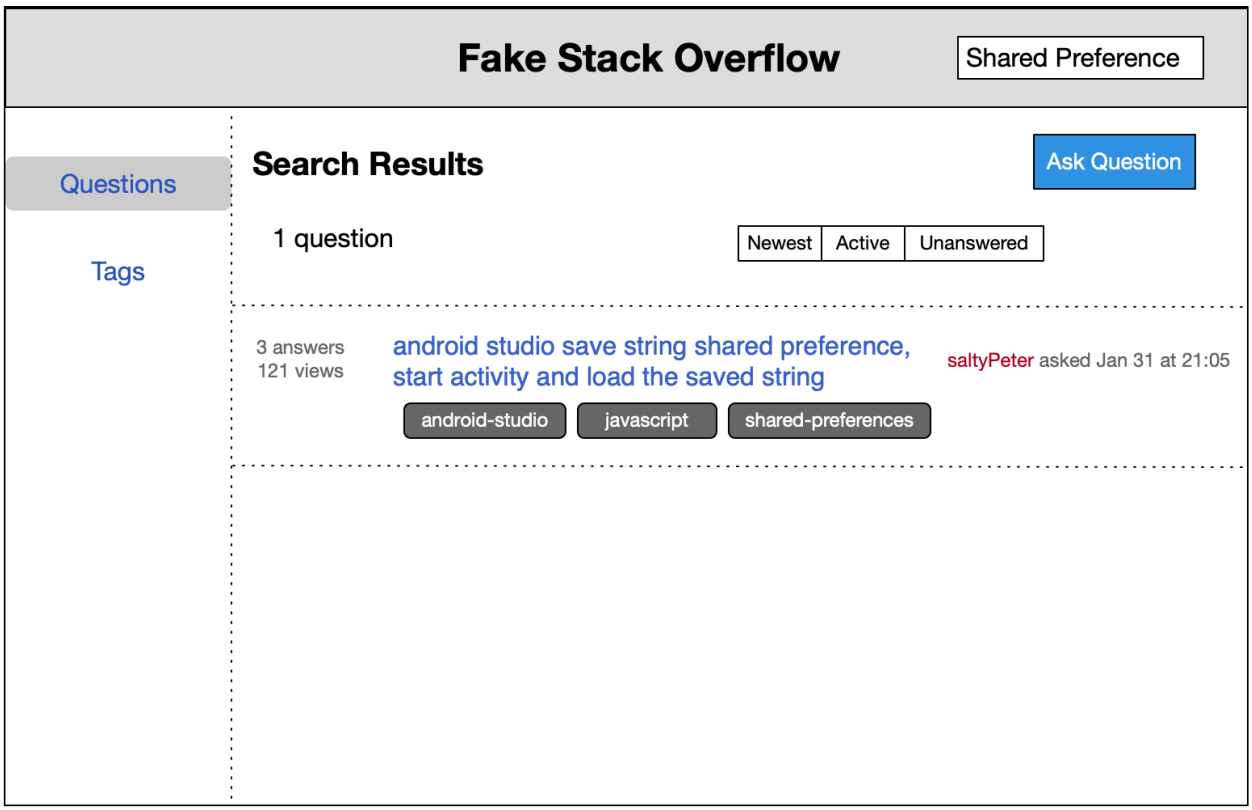


Figure 4

Furthermore, if a user surrounds individual words with [] then all questions with a tagname in [] should be displayed. *The search results should be displayed when the user presses the **ENTER** key*. See figure 5.

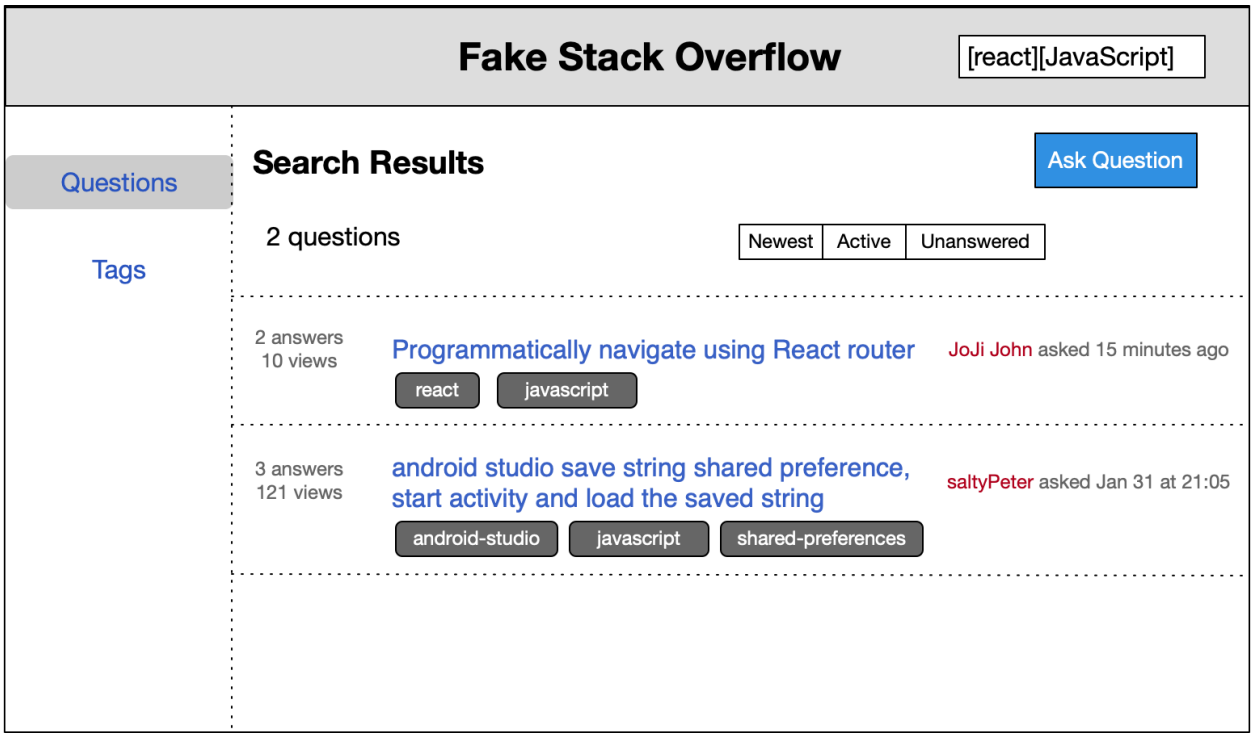


Figure 5

Note the searching is case-insensitive. Also, a search string can contain a combination of [tagnames] and non-tag words, that is, not surrounded with []. In this scenario, all questions tagged with at least one tag in the search string or text/title containing at least one of the non-tag words should be displayed. For example, if the search string is [react][android] javascript then all questions tagged with *react* or *android* or both should be considered. Also, questions with the non-tag word *javascript* in their text/title should be considered.

If the search string does not match any question or tag names then display the **No Questions Found**. The total number of questions displayed should be 0 and the page title and the button to ask a question should remain.

Answers Page

Clicking on a question link should increment by 1 the no. of views associated with the question and load the answers for that question in the *main* section of the home page. Note the banner should still remain at the top of the page. The answers should be displayed as in figure 6 with the following elements:

- The text **N answers**, where N is the total no. of answers given for the question. The title of the question. A *button* with the label **Ask Question**. You are free to add other style

constraints to the elements other than what has already been shown. However, make sure that they are clearly legible.

- The text **N views** indicating the no. of times the question has been viewed (including this one).
- The question text.
- The metadata for the question **<user> asked <date>**. This metadata format is the same as the one described in the page that displays all questions.
- The answers to the question. An answer has 2 parts as shown in figure 6
    - the answer itself,
    - the answer metadata in the format **<user> answered <date>**. The date format requirements in the metadata are exactly the same as the requirements for questions (see home page described before).
- If no. of answers do not fit on the page, then add a scroll bar.
- The answers should be displayed in ascending order of the day and time they were posted. In other words, display the answers that were posted most recently first.
- If the question text or an answer has hyperlinks, they should be displayed as such. Clicking on a hyperlink should open the target in a new tab or window. For example, in Figure 6, the first answer has two hyperlinks.
- A *button* with the label **Answer Question** at the end of all answers. Make sure that the button and the label are clearly visible. You are free to add a different style from the one shown.
- Answers must be divided by a dotted line as shown in figure 6.

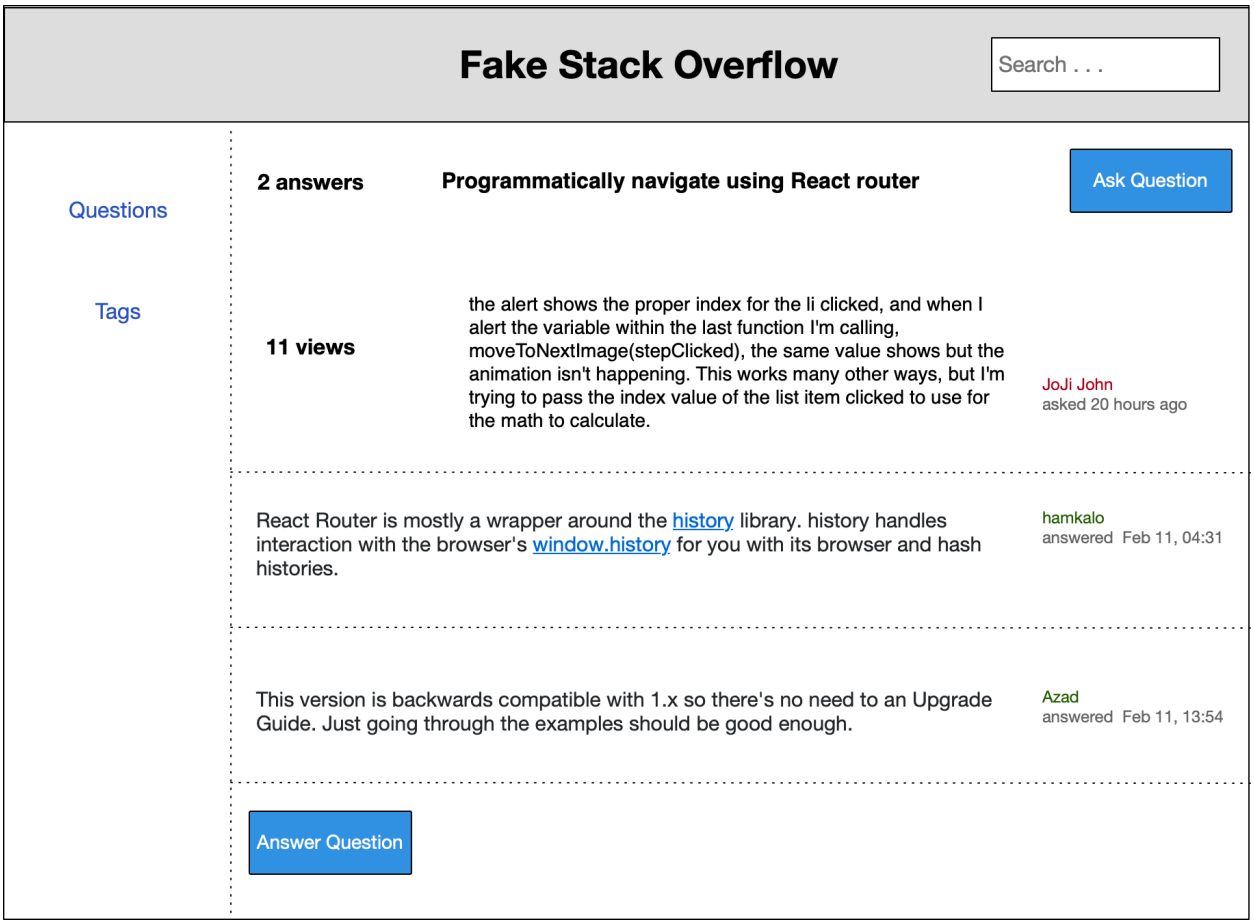


Figure 6

Pressing the *Ask A Question* button on this page will render elements as described in the **New Question Page** section.

New Answer Page

Pressing the *Answer Question* button will display a page with input elements to enter the new answer text and username. Note the menu should remain on the left of the page as shown in figure 7.

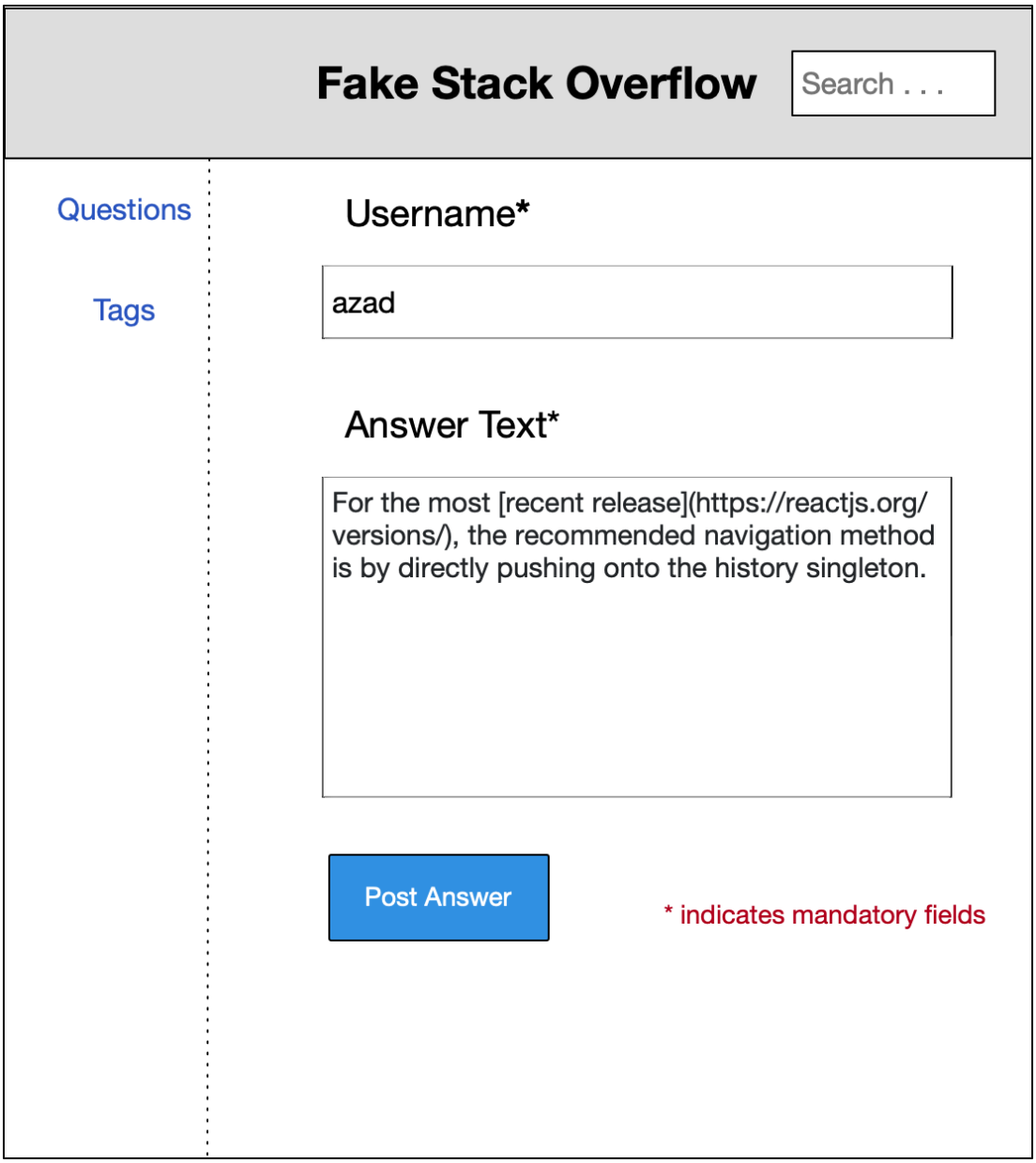


Figure 7

Pressing the *Post Answer* button, should capture the answer text and the username and update the data model. If the inputs have no errors, then all the answers are displayed as shown on the **Answers Page.** Note there should now be 3 answers for this question since a new answer was posted and this answer must be the first one shown in the list.

If the answer text or username is empty, then display appropriate error messages below the respective input fields.

A user is allowed to add hyperlinks in answer text. Hyperlinks are identified in the text if a user encloses the name of the hyperlink in [] and follows it up with the actual link in (). For example, in Figure 7, the answer text has a hyperlink “recent release”. The target of a hyperlink, that is, the stuff within () cannot be empty and must begin with “https://” or “http://”. Warn the user with an error message if this constraint is violated.

Tags Page

Clicking on the *Tags* link in the menu should display the list of all tags in the model. Tag names are case-insensitive so the tag name ‘React’ and ‘react’ should be considered the same for all practical purposes. Additionally, the *Tags* link in the menu should be highlighted with a gray background since the *Tags* page is being displayed currently. The page should render the following elements as shown in figure 8:

1. The text **N Tags**, where **N** is the total number of tags.
2. The text **All Tags**.
3. A button with the label **Ask Question**. This is the same button that was described in the *Questions* and *Answers* pages.
4. Tag names in groups of 3, that is, each row should have at most 3 tags. Each tag should be displayed in a box with dotted borders. The block should display the tag name as a link and the no. of questions associated with the tag in a new line in the same block.

Figure 8 shows an example of the page.

Upon clicking a tag link, all questions associated with the tag should be displayed. For example, if the *javascript* tag is clicked then the page should show all questions with the *javascript tag*. The style is similar to the home page.

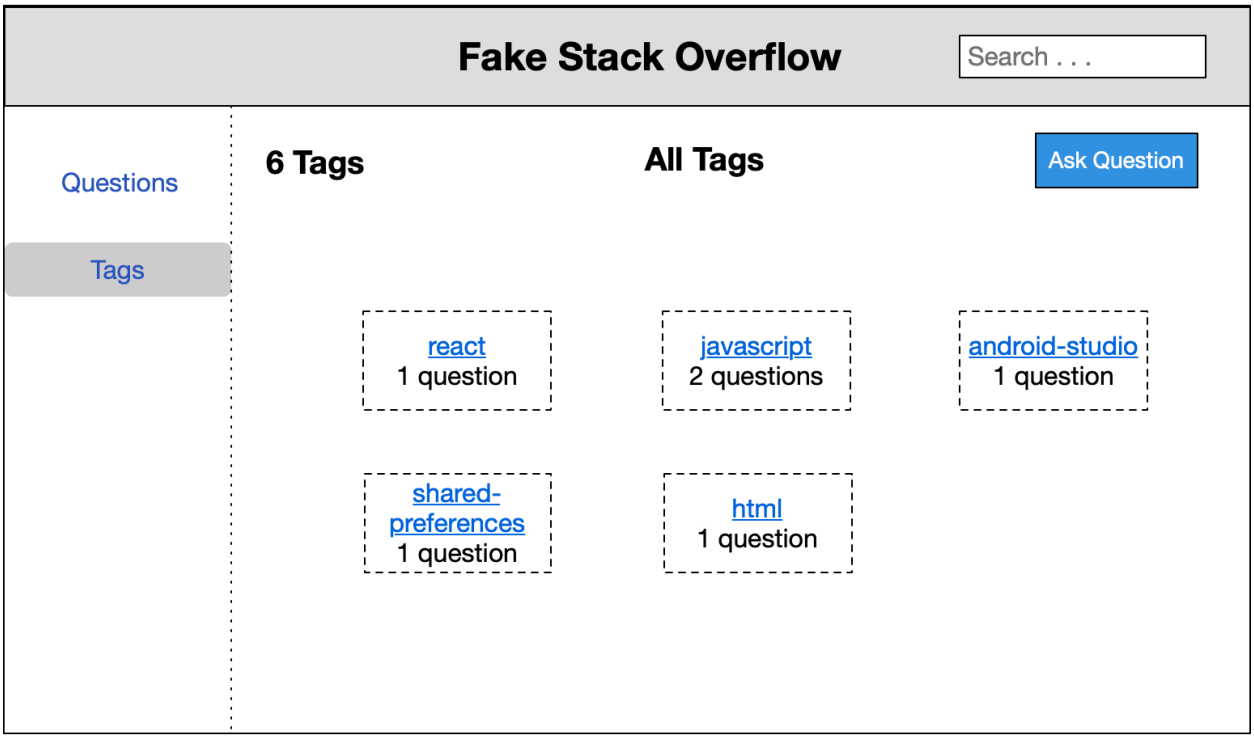

Fake Stack Overflow
Search . . .
Questions
Tags
6 Tags
All Tags
Ask Question
react
1 question
javascript
2 questions
android-studio
1 question
shared-preferences
1 question
html
1 question

Figure 8