Homework #7

ANONYMOUS Q&AIT (AJAX)

Overview

Repository Creation

👀 Click on this link: https://classroom.github.com/a/lLsxMHlY (https://classroom.github.com/a/lLsxMHlY) to accept this assignment in GitHub classroom.

- This will create your homework repository
- Clone it once you've created it

Submission Process

Submit **homework through gradescope** by uploading your GitHub repository:

- gradescope can be accessed through Brightspace
- gradescope will only accept submissions via GitHub

You will be given access to a private repository on GitHub, and it will contain some initial files (such as a linter configuration and starter files).

Push your code to the homework repository on GitHub so that your latest code is present.

Submit your repo through gradescope. The gradescope assignment will close, so make sure you submit before the deadline.

Note that if you make changes to your repo after you submitted through gradescope, you must resubmit through gradescope (gradescope does not sync changes).

Goals / Topics Covered

You'll be using the following concepts:

- fetch
- sending back json from Express

Description

Create a question-and-answer site where users post questions and answers to these questions anonymously. You will create this "single page app" using AJAX calls instead of regular page rendering and form submission.

You will:

1. Implement routes to create an API for retrieving questions and adding new ones
2. Use JavaScript to trigger background requests to the API from the form submit buttons

Use the following resources as reference:

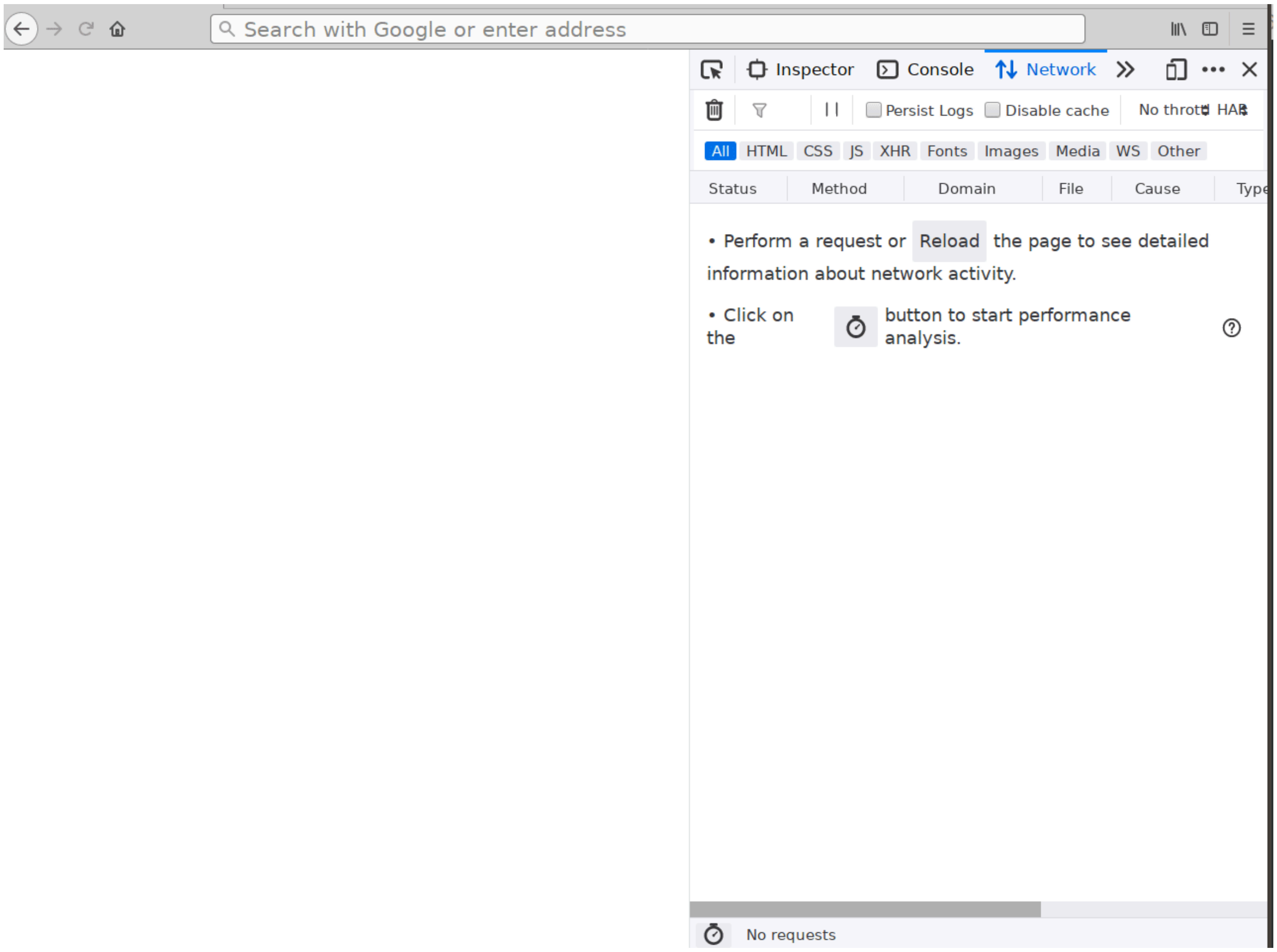
1. Slides on AJAX Part 1 (../slides/20/ajax.html)
2. Slides on AJAX Part 2 (../slides/21/ajax-express.html)
3. AJAX POST (from Part 2) (../slides/21/ajax-express.html#/47)

Check out the animation below to see how the site should work. Pay attention to:

- the button presses and the corresponding changes in the ui
- …as well as the network tab
- (notice that there are no page refreshes!)

The animation below shows:

1. loading the initial page / data
2. asking a question
3. adding answers to a question (optional)



• Perform a request or Reload the page to see detailed information about network activity.

• Click on the button to start performance analysis.

No requests

Submission Process

You will be given access to a private repository on GitHub. The repository will have a partially built Express application. The final version of your assignment should be in GitHub:

- **Push** your changes to the homework repository on GitHub.

No Commit Minimum / eslint Requirements

- There is no minimum number of commits required
- Linting is not required (and not configured)

Code Structure:

You should first create an express application that will be served on port 3000

The structure of the directory you're given looks like this:

```
├── .env # you must create this on your own
├── .gitignore
├── README.md
├── package-lock.json
├── package.json
├── public
│   ├── index.html
│   ├── javascripts
│   │   └── index.js
│   └── stylesheets
│       └── style.css
└── src
    ├── app.mjs
    ├── config.mjs
    └── db.mjs
```

Note that this will be implemented **as a single page web app**. This means that to implement these features:

1. **create routes that send back JSON** (essentially create an API)
2. utilize the API by writing client side JavaScript that:
    - constructs an http request by retrieving the values of form elements
    - requests data from url constructed in the background (AJAX)
    - parses the result of the background request
    - modifies the DOM appropriately

Some of This Project is Already Built for You!

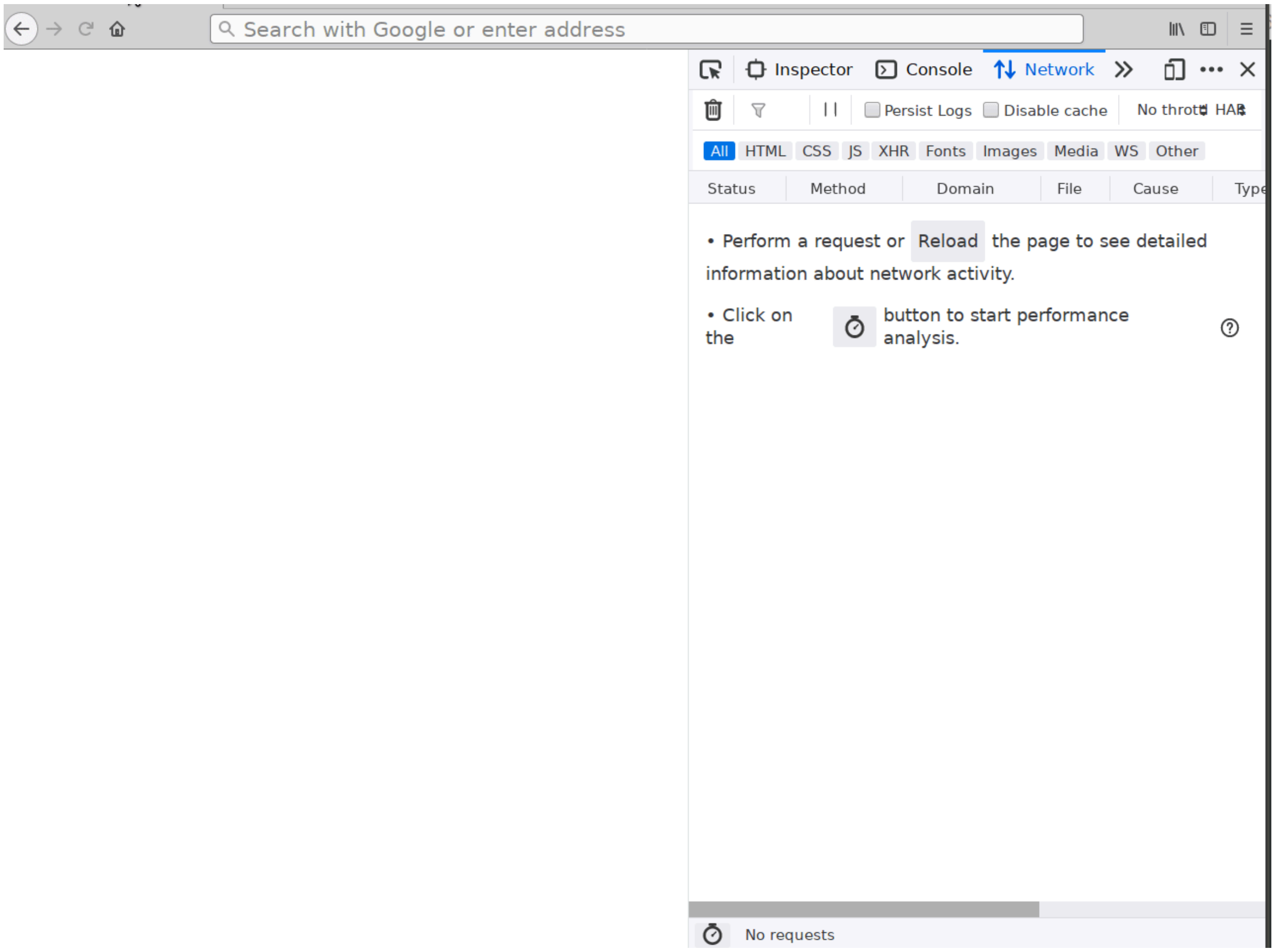
1. Server side code
    - database setup (a mongoose model, database connection) is provided through `db.mjs`
    - configuration with `.env`
    - a partially implemented Express application is in `app.mjs`
        - it has basic setup for json body parsing, static files, etc.
        - it has stubs for route handlers, but the implementations are left out
2. Client side code
    - there are static files present in `public`
    - the html is already present in `index.html`, and **you can add markup if you like**
    - the client side JavaScript is mostly left unimplemented (`index.js`), so you'll have to write the majority of it

Getting Existing Questions (Reading Data)

1. Create a `.env` file with a DSN variable (for example `DSN=mongodb://localhost/qanda`)
2. Check out the schema in `db.mjs` to familiarize yourself with the "data model"
3. Note that an `_id` field is automatically generated even if it is not specified (so each question will have an `_id`)
4. Add some questions to the database (use the commandline client to do the following)… here's an example (feel free to add your own questions/answers)

    ```
    db.questions.insertOne({
      "question": "What's the most delicious dessert?",
      "answers": ["ice cream on a pizza", "choco taco"]
    })
    ```

5. In the server side code (`app.mjs`), fill in the route for `GET /questions/` so that it gives back all of the questions (and their answers) from the database as json (you can use `res.json` or `res.send` with a JavaScript object… and express will stringify the object and set the appropriate headers)
6. Test this route in your browser; you should get back a json document
7. In the client side JavaScript code (`index.js`), once the DOM has been loaded, make a background request using `fetch` to get all of the questions
    - if an error occurs with the request (a 404, or an event listener for error get triggered), minimally use `console.log` to output the error
    - however, feel free to add more robust error handling, such as displaying a friendly error message in the DOM
8. For every question in the JSON response from the server, create elements for each question and answer, along with a button to add an answer
    - append the elements to the `main` element in `index.html` (do this using client-side JavaScript)
    - do not use any libraries to do this (no jQuery, React, vue, etc.)
    - the exact elements to create up to you; the reference solution uses a heading for the question, an unordered list for the answers, and an input button for the submit button, but you can mark up this part any way you like
9. Opening up `localhost:3000/index.html` in your browser, with the network tab open, should:
    - show a page with all answers and questions
    - along with an extra request in the network tab going to `localhost:3000/questions` (this is the background request)
10. It should look something like this (note that your tab will likely show fetch rather than xhr):


Search with Google or enter address
Inspector
Console
Network
Persist Logs
Disable cache
No thrott
HAR
All
HTML
CSS
JS
XHR
Fonts
Images
Media
WS
Other
Status
Method
Domain
File
Cause
Type
• Perform a request or Reload the page to see detailed information about network activity.
• Click on the button to start performance analysis.
No requests

Modals

You can manually create modal dialog boxes. They can be implemented by creating an element with a z-index higher than other elements. You can have it stretch out to the width and height of the window, and fix it to the upper left-hand corner. There are native modal dialog boxes, however, described below.

Some example markup and css is given for this… but you can style modals any way you like (and the styling doesn't have to match).

As for the JavaScript implementation, you can write your own manually (see alternative below):

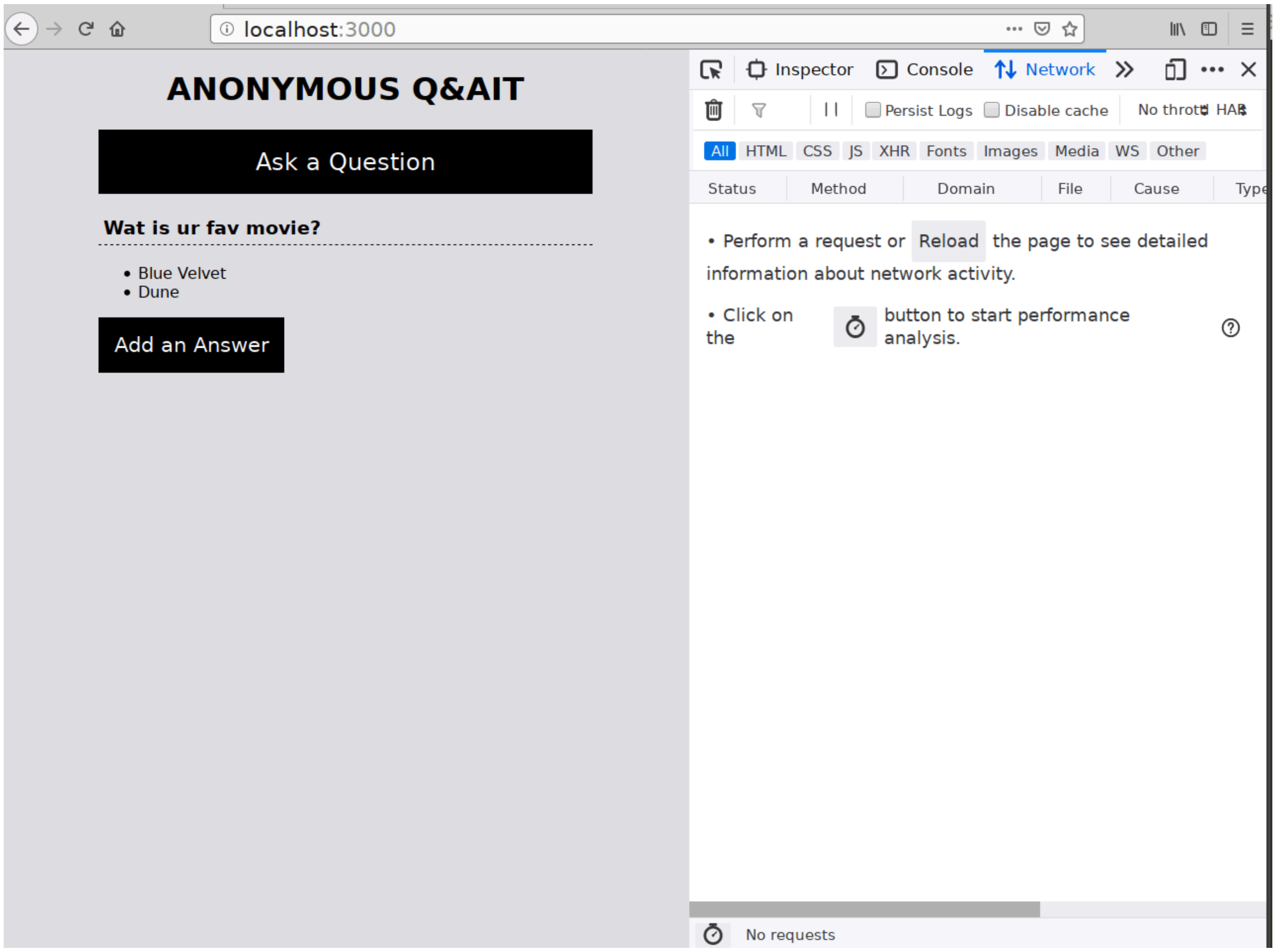
1. start off with all modals not visible (the starter css initializes them to `display: none`)
2. when the ask button is clicked (or any of the question's corresponding answer button is clicked if you're doing the optional part) … make the modal visible
3. sooo… add some click event listeners to the button(s)
    - these should either modify the `style` attribute of the appropriate modal
    - … or apply css classes (either your own, or the ones supplied as hints) Browsers support "native" modal dialogs (https://developer.mozilla.org/en-US/docs/Web/API/HTMLDialogElement), but

👀 Or… if you'd prefer, there are native modals available (see the docs) (https://developer.mozilla.org/en-US/docs/Web/API/HTMLDialogElement/showModal)…. if you're taking this route, you must minimally remove `display: none` from the stylesheets (and of course, you can just work with your own styles entirely).

Asking New Questions (Adding Data)

1. In the route, `POST /questions/`, create a new question in the database
    - it should give back a JSON response
    - if a new question is successfully created in the database, it should send back the object inserted as the response
    - otherwise, send back an object with a key called error… with a value containing an error message
    - note again that the `_id` field is auto generated
2. On the client side, add a on click handler to the `Ask a Question` button (if you already haven't done so)
    - it should show the provided HTML modal with a form for submitting questions (again, do this if you already haven't done so from the previous instructions)
    - when the button in the modal's form is pressed….
    - collect the form data (just the question text)
    - use an AJAX POST to send the question text to the server
    - (of course, use the `POST /questions/` as the url for your AJAX post; this will cause the new question to be saved in the database if the previous step, 1, were implemented correctly)
    - in the AJAX callback
        - on a successful result (that is no error key in the JSON response, and an id present for the boject)
            - use the returned JSON containing the saved question to….
            - add the `question` text to the page
            - use the `_id` attribute as an `id` attribute on a new html element that will contain this questions answers (to be used in the next part)
            - add a button, `Add an Answer`, to pop up the modal for answering a question (which will be describe in the next section) * close the modal and clear all fields
        - On failure

        - it's adequate to just log out the error message to the console
        - …or you can add a message in the DOM (perhaps in the modal)
3. When testing this…
    - keep the network tab open to see the requests being made
    - it's also helpful to look for errors in the JavaScript console on your browser
4. Here's an example of how this may work (pay attention to the network tab)

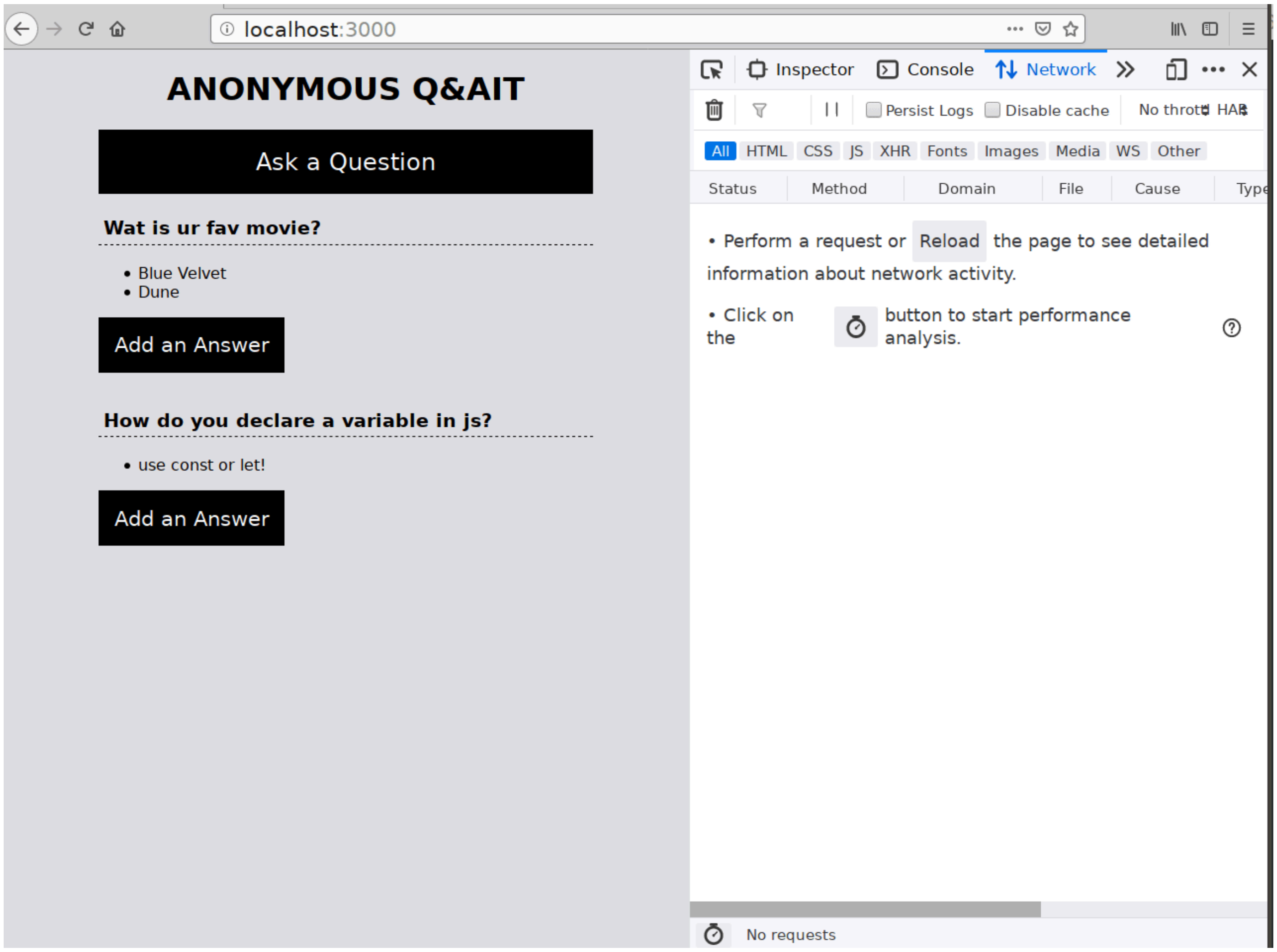


Adding New Answers (Optional)

1. The route in your server side express app, `POST /questions/:id/answers/` , will add an answer to an existing question
2. On the client side, add an on click handler to the button `Add Your Answer`
    - it should show the provided modal with a form for submitting answers
    - it should also set the form's hidden field for id to the `_id` of the question that is being answered
        - this part may be tricky… as you'll have to have the id of the question available as you're adding an event listener to the modal's button for submission
        - so it must be done at the time that the question is added back to the DOM
        - which means that you may have to go back to the previous sections on Asking New Questions or Reading Existing Questions to modify your code for adding questions to the DOM
    - when the button to submit the form is pressed, collect the `object_id` (the hidden input with id `question-id`) and answer text from the form
    - send an AJAX POST to the server with this data (using a url constructed from the question's id)

- again, you can get the `object_id` by
- using the hidden input (whose value can be filled in by the click callback)
- in the AJAX callback
  - On success
    - the user's answer is added beneath the question * close the modal and clear all fields
  - On failure
    - minimally, log out the error

3. Here's an example of how it may work:



localhost:3000

ANONYMOUS Q&AIT

Ask a Question

Wat is ur fav movie?

- Blue Velvet
- Dune

Add an Answer

How do you declare a variable in js?

- use const or let!

Add an Answer

• Perform a request or Reload the page to see detailed information about network activity.

• Click on the button to start performance analysis.

No requests