Write JavaScript for the Web

10 hours · Medium

Last updated on 4/2/20

Use asynchronous programming

Nice! You passed this exercise!

Evaluated skills

- ✓ handle errors
- ✓ use async/await
- ✓ handle AJAX response data
- ✓ build and send AJAX requests

Question 1

Which of the following correctly handles a promise?

( )

```javascript
myPromise.try(
    (data) => {
        console.log(data);
    }
).catch(
    (error) => {
        console.log(error);
    }
);
```

✓ (•)

```javascript
myPromise.then(
    (data) => {
        console.log(data);
    }
).catch(
    (error) => {
        console.log(error);
    }
);
```

( )

```javascript
myPromise.then(
    (data) => {
        console.log(data);
    }
).error(
    (error) => {
        console.log(error);
    }
);
```

( )

```javascript
myPromise.resolve(
    (data) => {
        console.log(data);
    }
).reject(
    (error) => {
        console.log(error);
    }
);
```

Using .then() and .catch allows us to react correctly to a promise when it resolves or rejects respectively.

Question 2

Which of the following describes the three states of a promise?

( )
- Promise is started
- Promise is pending
- Promise is resolved

( )
- Promise is implemented
- Promise is resolved
- Promise is complete

✓ (•)
- Promise is pending
- Promise is resolved
- Promise is rejected

( )
- Promise is resolved
- Promise is rejected
- Promise is complete

*While the code within a promise is executing (API calls, timeouts etc), it is **pending**. If the code completes correctly, the promise is **resolved**, and if it fails, it is **rejected**.*

Question 3

Which of the following correctly creates a function which returns a promise?

✓ (•)

```javascript
function makeMeAPromise() {
    return new Promise((resolve, reject) => {
        // Do some stuff here
        if (success) {
            resolve(successData);
        }
        else {
            reject(errorData);
        }
    });
}
```

( )

```javascript
function makeMeAPromise() {
    return new Promise(resolve, reject) => {
        // Do some stuff here
        if (success) {
            resolve(successData);
        }
        else {
            reject(errorData);
        }
    };
}
```

( )

```javascript
function makeMeAPromise() {
    return new Promise((resolve, reject) => {
        // Do some stuff here
        success.then(data).catch(error);
    });
}
```

( )

```javascript
function makeMeAPromise() {
    return new Promise((resolve, reject) => {
        // Do some stuff here
        if (success) {
            return successData
        }
        else {
            return error;
        }
    });
}
```

The promise constructor takes an executor function with arguments `resolve` and `reject` as an argument.

Question 4

Which of the following correctly submits JSON data in a POST request to a server? Assume that the server will respond with status 200 or 201 when the request is successful.

( )

```javascript
let postRequest = new XMLHttpRequest();
postRequest.open('POST', myData, 'http://mybackend.api');
postRequest.onreadystatechange = () => {
    if (postRequest.readyState === 4) {
        if (postRequest.status === 200 || postRequest.status === 201) {
            console.log('Data successfully sent to server!');
        }
        else {
            console.log('Error status: ' + postRequest.status);
        }
    }
};
postRequest.setRequestHeader('Content-Type', 'application/json');
postRequest.send();
```

( )

```javascript
let postRequest = new XMLHttpRequest();
postRequest.open('POST', 'http://mybackend.api');
postRequest.onreadystatechange = () => {
    if (postRequest.readyState === 4) {
        if (postRequest.status === 200 || postRequest.status === 201) {
            console.log('Data successfully sent to server!');
        }
        else {
            console.log('Error status: ' + postRequest.status);
        }
    }
};
postRequest.setRequestHeader('Content-Type', 'application/json');
postRequest.send(myData);
```

✓ (•)

```javascript
let postRequest = new XMLHttpRequest();
postRequest.open('POST', 'http://mybackend.api');
postRequest.onreadystatechange = () => {
    if (postRequest.readyState === 4) {
        if (postRequest.status === 200 || postRequest.status === 201) {
            console.log('Data successfully sent to server!');
        }
        else {
            console.log('Error status: ' + postRequest.status);
        }
    }
};
postRequest.setRequestHeader('Content-Type', 'application/json');
postRequest.send(JSON.stringify(myData));
```

( )

```javascript
let postRequest = new XMLHttpRequest();
postRequest.open('POST', 'http://mybackend.api');
postRequest.onreadystatechange = () => {
    if (postRequest.readyState === 4) {
        if (postRequest.status === 200 || postRequest.status === 201) {
            console.log('Data successfully sent to server!');
        }
        else {
            console.log('Error status: ' + postRequest.status);
        }
    }
};
postRequest.send(JSON.stringify(myData));
```

To send a POST request containing JSON data:

1. *Create a new XMLHttpRequest object and use its `open()` method to make it a POST request and give it a URL*
2. *Set the `onreadystatechange` function to react to any data or errors returned*
3. *Set the request's `Content-Type` header to `application/json`*
4. *Send the "stringified" data with the `send()` method*

Question 5

Consider the following function:

```javascript
// makes AJAX call of type 'verb' to 'url'
// 'data' is an optional JavaScript object for POST calls
// returns a Promise
function makeARequest(verb, url, data) {
    return new Promise((resolve, reject) => {

        // build request object with function arguments
        let request = new XMLHttpRequest;
        request.open(verb, url);

        // if server responds with error, reject with error
        // otherwise, respond with response as JSON object
        request.onreadystatechange = () => {
            if (request.readyState === 4) {
                if (request.status !== 200) {
                    reject(request.status);
                }
                else {
                    resolve(JSON.parse(request.response));
                }
            }
        }

        // if request is POST, send request with data
        // otherwise, just send request
        if (verb === 'POST') {
            request.setRequestHeader('Content-Type', 'application/json');
            request.send(JSON.stringify(data));
        }
        else {
            request.send();
        }
    });
}
```

Which two of the following options are correct ways to use this function to send the response from a GET request in a second POST request?

Careful, there are several correct answers.

✓ [x]

```javascript
makeARequest('GET', 'http://backend.api').then(
    (data) => {
        makeARequest('POST', 'http://backend.api', data).then(
            () => {
                console.log('Data posted successfully!');
            }
        ).catch(
            (error) => {
                console.log('An error occurred in the POST request!');
                console.log(error);
            }
        );
    }
).catch(
    (error) => {
        console.log('An error occurred in the GET request!');
        console.log(error);
    }
);
```

[ ]

```javascript
let data = makeARequest('GET', 'http://backend.api');
makeARequest('POST', 'http://backend.api', data);
```

[ ]

```javascript
let data = makeARequest('GET', 'http://backend.api');
if (data) {
    makeARequest('POST', 'http://backend.api', data);
}
```

✓ [x]

```javascript
async function doubleRequest() {
  try {
    let data = await makeARequest('GET', 'http://backend.api');
    try {
      await makeARequest('POST', 'http://backend.api', data);
      console.log('Requests complete!');
    }
    catch(error) {
      console.log('Error: server responded with status: ' + error);
    }
  }
  catch(error) {
    console.log('Error: server responded with status: ' + error);
  }
}
```

Because each request takes time, we need to work asynchronously, either using `.then()` and `.catch()`, or using `async/await`.

Question 6

What is missing from this code block?

```javascript
returnsAPromise(myData).then(
    (data) => {
        console.log('Hey I got my data!');
        console.log(data);
    }
);
```

- ( ) A set of curly braces
- ( ) A semicolon
- ( ) A set of parentheses
- ✓ (•) A `.catch()` block

Of course, this block is missing a `.catch()` block! What will happen if the promise returned by `returnsAPromise()` is rejected?!

Question 7

Typically, what is the value of an expression placed after the `await` keyword?

- ( ) A JavaScript object
- ✓ (•) A promise
- ( ) A boolean
- ( ) A function

*The `await` keyword pauses the execution of an `async` function until the **promise** placed after it resolves or rejects. If anything other than a promise is placed after `await`, it is converted to a resolved promise.*

< SUMMARY

MANAGE YOUR DEPENDENCIES >

Teacher

Will Alexander

Scottish developer, teacher and musician based in Paris.