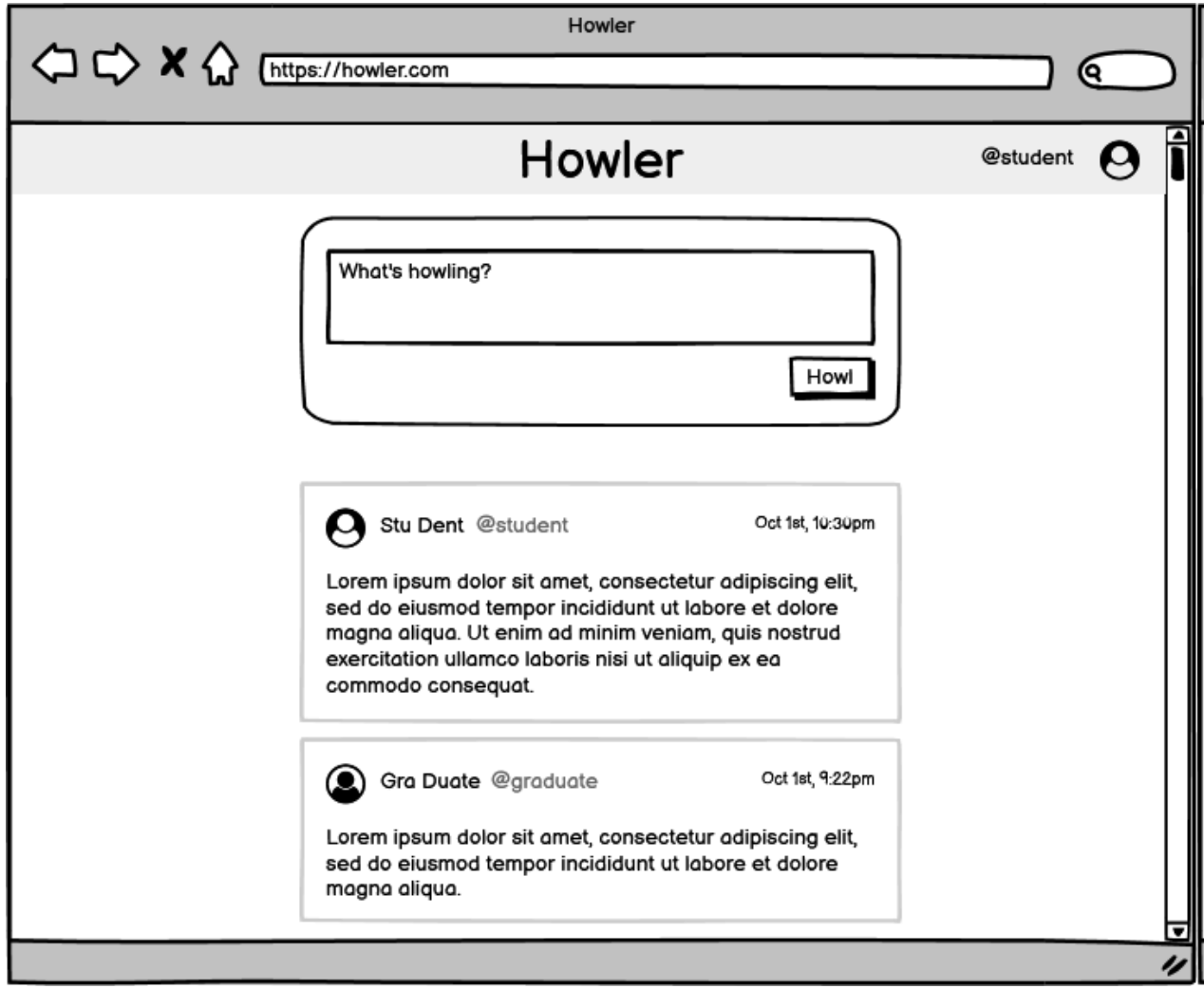
CSC 342: Applied Web-based Client-Server Computing

Prof. Domínguez – Fall 2023

Administrivia

- Homework 3 feedback is in progress
- Project Milestone 1
  - Due Friday 10/20
- Homework 4
  - Due Friday 10/27
  - **You should have already started this assignment!**

Homework 4: Howler



Lecture 15: Fetch API

CSC 342
Applied Web-based Client-Server Computing

Fall 2023

Client-side Rendering with a REST API

- Let's take a quick look at our client-side rendered app from last class
- Modify our CSR app so that it serves the REST API from our REST lecture
    1. Use the JSON data and API router in our backend
    2. Attach the API router to our CSR server
- We'll also restructure the code to make it more organized

Where is the Data?

- Normally, the data we need is not directly available in the client
  - We need to retrieve it from a data source
    - Our server
    - Some other server

- **Q:** But in client-side rendering, how do we get data via JavaScript?

- **A:** Fetch API

- **Alternatively:** an older API called `XMLHttpRequest`

Fetch API

Fetch API

- Provides the ability to make HTTP requests via JavaScript
  - Can send and receive data
  - No need to navigate away from the current page

- Fetch API components:
  - `fetch()` method
  - `Headers` interface
  - `Request` interface
  - `Response` interface

```
fetch(url)
fetch(url, options)
fetch(request)
```

`fetch()` Method

```
fetch(url)
fetch(url, options)
fetch(request)
```

- First parameter is required
  - Can be a URL or a `Request` object
- The second parameter `options` configures the request (optional)
- Returns a promise
  - Resolves with any HTTP response, **even if it is an HTTP error**
  - Rejects when there is a ***network*** error
    - The user is offline
    - DNS does not resolve
    - The server is unreachable

```
fetch('/resource')
.then(res => {
  console.log('Response:', res);
})
.catch(error => {
  console.log('Error:', error);
});
```

The `Response` Object

- A fetch request that receives an HTTP response resolves with a `Response` object

- Some useful properties of a `Response` `res`:
  - `res.body` is a stream resource (stream of byte data)
  - `res.headers` contains the response headers as a `Headers` object
  - `res.status` is the HTTP status code (e.g., 404)
  - `res.statusText` is the status message (e.g., "Not Found")
  - `res.ok` is a Boolean that indicates if the HTTP status is successful (2XX status)

Successful `Responses`

- We can use `res.status`, `res.statusText`, and `res.ok` to verify if a request was successful

```
fetch('http://localhost:3000/path')
.then(res => {
  if(!res.ok) {
    throw new Error('This request was not successful: ' + res.statusText);
  }
  return res;
})
.then(res => {
  console.log('Response:', res);
})
.catch(error => {
  console.log('Error:', error);
});
```

Reading the `Response` Body

- There are built-in functions to convert `res.body` from a byte stream
- These **return a promise** that resolves with data in a specific format
- The two most used:
  - `res.text()` returns the body as a string
  - `res.json()` parses the body as a JSON string and returns the resulting object

```
fetch('/resource')
.then(res => {
  return res.text();
})
.then(text => {
  console.log('Response:', text);
});
```

```
fetch('/json_resource')
.then(res => {
  return res.json();
})
.then(obj => {
  console.log('Response:', obj);
});
```

Updating Our App to Use `fetch()`

- Let's update our client-side rendered app

- Retrieve data from the JSON endpoints we created earlier
  1. Modify `APIClient_mock.js` to use `fetch()`
     - It will be an actual API client now
  2. Verify that the response is successful
  3. Make sure to parse JSON responses into JS objects

Configuring Our `fetch()` Call

- By default, `fetch()` makes a **GET** request
- We can customize the request via the second parameter `options`
- Some useful properties:
  - `method` (e.g., `"GET"`, `"PUT"`, etc.)
  - `headers`: object literal or `Headers`
  - `body`: request body

```
fetch('/resource', {
  method: "POST",
  headers: {
    'Accept': 'application/json',
    'Content-Type': 'application/json'
  },
  body: JSON.stringify(jsObject)
})
.then( ...
```

See you next class!