# React Native

HTTP/Fetch

Sending data

# Sending data to web server

- Two methods
  - **GET** requests include all required data in the URL.
  - **POST** requests supply additional data from the client (browser) to the server in the message body.

- Difference:
  - https://www.diffen.com/difference/GET-vs-POST-HTTP-Requests

# Supplying request options

```
// Example POST method implementation:


postData(`http://example.com/answer`, {answer: 42})

  .then(data => console.log(JSON.stringify(data))) //
JSON-string from `response.json()` call

  .catch(error => console.error(error));
```

```
function postData(url = ``, data = {}) {
  // Default options are marked with *
    return fetch(url, {
      method: "POST", // *GET, POST, PUT, DELETE, etc.
      mode: "cors", // no-cors, cors, *same-origin
      cache: "no-cache", // *default, no-cache, reload,
                                  //force-cache, only-if-cached
      credentials: "same-origin", // include, same-origin, *omit
      headers: {
          "Content-Type": "application/json; charset=utf-8",
          // "Content-Type": "application/x-www-form-urlencoded",
      },
      redirect: "follow", // manual, *follow, error
      referrer: "no-referrer", // no-referrer, *client
      body: JSON.stringify(data), // body data type must match
                                          // "Content-Type" header
  })
  .then(response => response.json()); // parses response to JSON
}
```

A fetch returns a promise.

Headers are a web protocol. See: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers

POST has a body

# Using Fetch to send data

- See

https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

# Uploading data: GET example

```
componentDidMount(){
  const data = {foo:"John", bar:2};
  var myHeaders = new Headers();
  myHeaders.append("Content-Type", "text/plain");
  return fetch(`http://cs-ithaca.eastus.cloudapp.azure.com/~barr/testGet.php?foo=${encodeURIComponent(data.foo)}&bar=${encodeURIComponent(data.bar)}`,
  {
  method: "GET",
  headers: myHeaders,
  }
)
```

Data sent in plain text

Must use backward single qote

This encodes a string in the proper format for sending via HTTP

This is example http-GET on the class web site.

1. Try with the name “George”, then “Sally”
2. Add a TextInput component and get the name from that.

# Uploading data: GET example

```
  .then((response) => response.json())
  .then((responseJson) => {
    this.setState({
      isLoading: false,
      dataSource: responseJson,
    }, function(){
    });
  })
  .catch((error) =>{
    console.error(error);
  });
}
```

This is example http-GET on the class web site.
Try with the name “George”, then “Sally”
Add a TextInput component and get the name from that.

# Uploading data: Post example

```
componentDidMount(){
  var myHeaders = new Headers();
  myHeaders.append("Content-Type", "application/json");
  var url = 'http://cs-ithaca.eastus.cloudapp.azure.com/~barr/testPost.php’
  var data = {foo: 'George', bar: 22};
```

Will send JSON encoded data

# Uploading data: Post example

```
fetch(url, {
  method: 'POST', // or 'PUT'
  body: JSON.stringify(data), // data can be `string` or {object}!
  headers: myHeaders
}).then(res => res.json())
.then(responseJson => {  this.setState({
        isLoading: false,
        dataSource: responseJson,
      });   })
.catch(error => Alert.alert('Error:'+ error));
}
```

Using “Post” with JSON encoded data

Encode data as JSON

# Uploading a file

```
var formData = new FormData();
var fileField = document.querySelector("input[type='file']");

formData.append('username', 'abc123');
formData.append('avatar', fileField.files[0]);

fetch('https://example.com/profile/avatar', {
  method: 'PUT',
  body: formData
})
.then(response => response.json())
.catch(error => console.error('Error:', error))
.then(response => console.log('Success:', JSON.stringify(response)));
```

# Supplying your own request object

- Instead of passing a path to the resource you want to request into the fetch() call, you can create a request object using the Request() constructor, and pass that in as a fetch() method argument (next slide)

```
var myHeaders = new Headers();

var myInit = { method: 'GET',
               headers: myHeaders,
               mode: 'cors',
               cache: 'default' };

var myRequest = new Request('flowers.jpg', myInit);

fetch(myRequest).then(function(response) {
  return response.blob();
}).then(function(myBlob) {
  var objectURL = URL.createObjectURL(myBlob);
  myImage.src = objectURL;
});
```

Change the http-GET example on the class web site to use a request object.

# Copying the request object

- Request() accepts exactly the same parameters as the fetch() method. You can even pass in an existing request object to create a copy of it:

```
var anotherRequest = new Request(myRequest, myInit);
```

- request and response bodies are one use only.
  - Making a copy like this allows you to make use of the request/response again,
  - Can varying the init options if desired.
  - The copy must be made before the body is read, and reading the body in the copy will also mark it as read in the original request.

# Headers

- HTTP headers allow the client and the server to pass additional information with the request or the response.
- An HTTP header consists of
  - its case-insensitive name followed by a colon ':’,
  - then by its value (without line breaks).
  - Leading white space before the value is ignored.
- Reference:

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers

# Headers

- Headers can be grouped according to their contexts:
  - General header: Headers applying to both requests and responses but with no relation to the data eventually transmitted in the body.
  - Request header: Headers containing more information about the resource to be fetched or about the client itself.
  - Response header: Headers with additional information about the response, like its location or about the server itself (name and version etc.).
  - Entity header: Headers containing more information about the body of the entity, like its content length or its MIME-type.

# Headers

- The Headers interface allows you to create your own headers object via the Headers() constructor.
- A headers object is a simple multi-map of names to values:

```
var content = "Hello World";
var myHeaders = new Headers();
myHeaders.append("Content-Type", "text/plain");
myHeaders.append("Content-Length", content.length.toString());
myHeaders.append("X-Custom-Header", "ProcessThisImmediately");
```

# Headers

- The same can be achieved by passing an array of arrays or an object literal to the constructor:

```
myHeaders = new Headers({
  "Content-Type": "text/plain",
  "Content-Length": content.length.toString(),
  "X-Custom-Header": "ProcessThisImmediately",
});
```

# Headers:  errors

- All of the Headers methods throw a TypeError if a header name is used that is not a valid HTTP Header name.
- The mutation operations will throw a TypeError if there is an immutable guard (see below).
  - Otherwise they fail silently. For example:

```
var myResponse = Response.error();
try {
  myResponse.headers.set("Origin", "http://mybank.com");
} catch(e) {
  console.log("Cannot pretend to be a bank!");
}
```

# Headers:  errors

- A good use case for headers is checking whether the content type is correct before you process it further. For example:

```
fetch(myRequest).then(function(response) {
  var contentType = response.headers.get("content-type");
  if(contentType && contentType.includes("application/json")) {
    return response.json();
  }
  throw new TypeError("Oops, we haven't got JSON!");
})
.then(function(json) { /* process your JSON further */ })
.catch(function(error) { console.log(error); });
```

# Body

- Both requests and responses may contain body data. A body is an instance of any of the following types:
  - ArrayBuffer
  - ArrayBufferView (Uint8Array and friends)
  - Blob/File
  - string
  - URLSearchParams
  - FormData

# Body

- The Body mixin defines the following methods to extract a body (implemented by both Request and Response).
- These all return a promise that is eventually resolved with the actual content.
  - arrayBuffer()
  - blob()
  - json()
  - text()
  - formData()