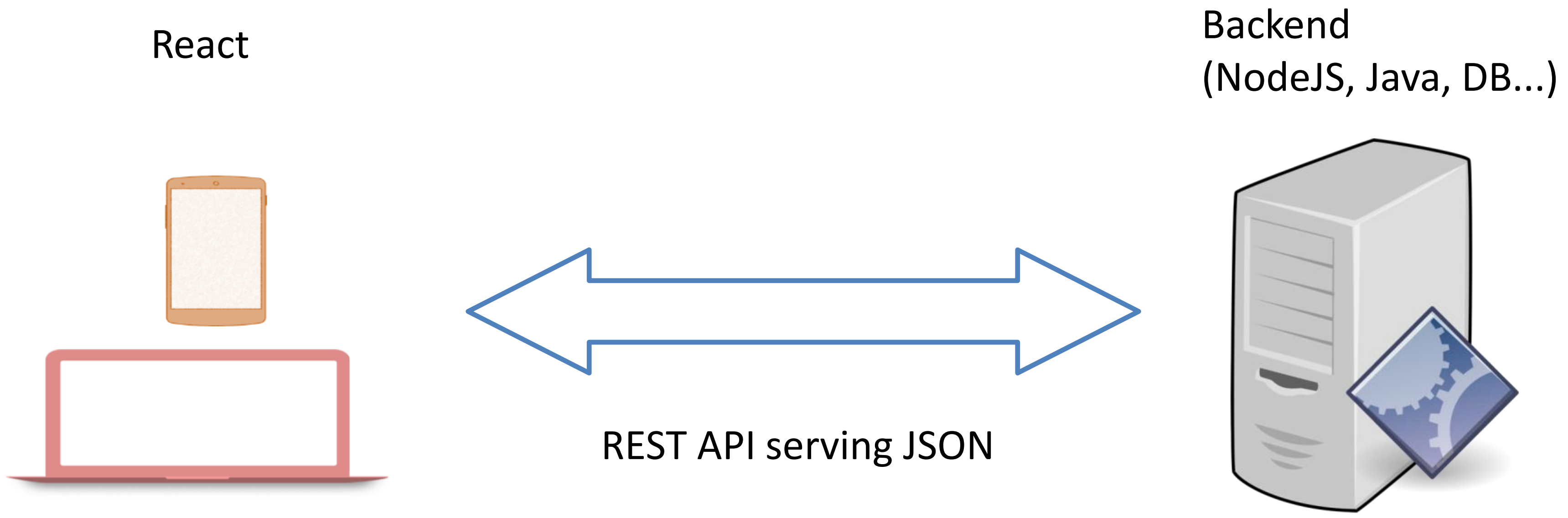
Fetch data from Server

RUJUAN

Modern WEB Architecture



Intro

A very common task in modern websites and applications is retrieving individual data items from the server to update sections of a webpage without having to load an entire new page. This seemingly small detail has had a huge impact on the performance and behavior of sites, so in today’s lecture, we'll explain the concept and look at technologies that make it possible:

- XMLHttpRequest - 2001
  - Ajax - 2005
  - Axios - ?
- Fetch - 2015

XMLHttpRequest API

The XMLHttpRequest API enables web apps to make HTTP requests to web servers and receive the responses programmatically using JavaScript. This in turn enables a website to update just part of a page with data from the server, rather than having to navigate to a whole new page.

```
const xhttpr = new XMLHttpRequest();
xhttpr.open('GET', 'Api_address', true);

xhttpr.send();

xhttpr.onload = ()=> {
if (xhttpr.status === 200) {
        const response = JSON.parse(xhttpr.response);
        // Process the response data here
} else {
        // Handle error
}
};
```

Ajax

AJAX stands for Asynchronous JavaScript and XML. jQuery is a library used to make JavaScript programming simple and if you are using it then with the help of the $.ajax() method you can make asynchronous HTTP requests to get data.

```
$.ajax({
        url: 'APIURL',
        method: 'GET',
        success: function(response) {
                const parsedData = JSON.parse(response);
                // Process the parsed data here
        },
        error: function(xhr, status, error) {
                // Handle any errors
        }
});
```

Fetch

The Fetch API provides a JavaScript interface for accessing and manipulating parts of the HTTP pipeline, such as requests and responses. It also provides a **global** fetch() method that provides an easy, logical way to fetch resources **asynchronously** across the network.

This kind of functionality was previously achieved using `XMLHttpRequest`.

Syntax:

```
   let promise = fetch(url, [options]);
```

- `url` – the URL to access.
- `options` – optional parameters: method, headers etc.
- Without `options`, this is a simple `GET` request, downloading the contents of the url.

Fetch - GET

```
let promise = fetch('http://localhost:3000/products')
```

The browser starts the request right away and returns a promise that the calling code should use to get the result.

Getting a response is usually a two-stage process.

- First, the promise, returned by fetch, resolves with an object of the built-in Response class as soon as the server responds with headers.
  - `promise.then(response => console.log(response.ok, response.status));`
- Second, to get the response body, we need to use an additional method call.
  - response.text() – read the response and return as text,
  - response.json() – parse the response as JSON,
  - …

```
promise.then(response => response.json())
       .then(result => console.log(result));
```

Fetch async & await

```
async function fetchProductById(id) {
    let response = await fetch("http://localhost:3000/products/" + id
);
    if (response.ok) {
        let json = await response.json();
        console.log(json);
    } else {
        console.log("HTTP-Error: " + response.status);
    }
}

fetchProductById(1);
```

Axios

Axios

Axios is a promise-based HTTP Client for node.js and the browser. It is isomorphic (= it can run in the browser and nodejs with the same codebase). On the server-side it uses the native node.js http module, while on the client (browser) it uses XMLHttpRequests.

It's similar to JavaScript Fetch API.

Features:

- Make XMLHttpRequests from the browser
- Make http requests from node.js
- Supports the Promise API
- Intercept request and response
- Transform request and response data
- …

- To use Axios with React, install first: `npm install axios`

Why Axios?

1. It has good defaults to work with JSON data. Unlike alternatives such as the Fetch API, you often don't need to set your headers. Or perform tedious tasks like converting your request body to a JSON string.
2. Axios has function names that match any HTTP methods. To perform a GET request, you use the `.get()` method.
3. Axios does more with less code. Unlike the Fetch API, you only need one `.then()` callback to access your requested JSON data.
4. Axios has better error handling. Axios throws 400 and 500 range errors for you. Unlike the Fetch API, where you have to check the status code and throw the error yourself.
5. Axios can be used on the server as well as the client. If you are writing a Node.js application, be aware that Axios can also be used in an environment separate from the browser.

Request method aliases

axios.request(config)

axios.get(url[, config])

axios.delete(url[, config])

axios.post(url[, data[, config]])

axios.put(url[, data[, config]])

Axios methods

The Axios methods have 3 parameters:

1. URL (you can set a default URL)
2. Data (optional) – It is available for POST, PUT, PATCH. It is not available in GET, DELETE.
3. Options (optional). You can pass query parameters and headers to the server. In header, you pass authorization tokens if the API is secure.

Query params/URL params vs Body

Query params and body have one thing in common. Pass data to server that is required for processing the request

Query params/URL params	Body
Used for HTTP requests	Use for **POST/PUT** requests
Developers use query params in the back-end when the input is not complex. For example, you only need text inputs. Query params **don't support** complex object like **nested JSON**.	**You can pass** complex object such **nested JSON**.
URL is too verbose. That doesn't look clean.	URL is so much clean. Just domain and resource.
There is a size limit.	You can pass more data than query params.
Data is passed explicitly in the URL.	Data is passed implicitly in the body.
Values are string	Values are JSON object

Response Schema

The response for a request contains the following information.

```
{
    // `data` is the response that was provided by the server
    data: {},
    status: 200,
    statusText: 'OK',
    headers: {},

    // `config` is the config that was provided to `axios` for the request
    config: {},

    // `request` is the request that generated this response
    // It is the last ClientRequest instance in node.js (in redirects)
    // and an XMLHttpRequest instance in the browser
    request: {}
}
```

```
axios.get('/users/12345')
    .then(function (response) {
        console.log(response.data);
        console.log(response.status);
        console.log(response.statusText);
        console.log(response.headers);
        console.log(response.config);
    });
```

Demos

```
try{
    const response = await
axios.post('http://localhost:8080/products’, data);
    setProducts(response.data);
} catch(err){
    setError(err);
}
```

```
useEffect(() => {
    axios.get('http://localhost:8080/products')
        .then(response => setProducts(response.data))
}, []);
```

The Axios Instance

- You can create custom Axios instances with specific configurations, base URLs, and interceptors.
- Helpful for managing different parts of your application with distinct settings.

```
axios.create([config])
```

```
export default axios.create({
    baseURL: "http://localhost:8000",
    headers: {
        "Content-type": "application/json"
    }
});
```

Request Config

These are the available config options for making requests. Only the `url` is required. Requests will default to `GET` if method is not specified.

https://axios-http.com/docs/req_config

```
{
   // `url` is the server URL that will be used for the request
   url: '/user',
   method: 'get', // default
   // `baseURL` will be prepended to `url` unless `url` is
absolute.
   // It can be convenient to set `baseURL` for an instance of
axios to pass relative URLs
   // to methods of that instance.
   baseURL: 'https://some-domain.com/api',
   headers: {
         'X-Requested-With': 'XMLHttpRequest'
   },
   params: { ID: 12345 },

   data: {firstName: 'Fred'},

   data: 'Country=Brasil&City=Belo Horizonte',

   timeout: 1000, // default is `0` (no timeout)

   responseType: 'json'// default
}
```

Refactor Code

```
import axios from "axios";

export default axios.create({
    baseURL: "http://localhost:8000",
    headers: {
        "Content-type": "application/json"
    }
});
```

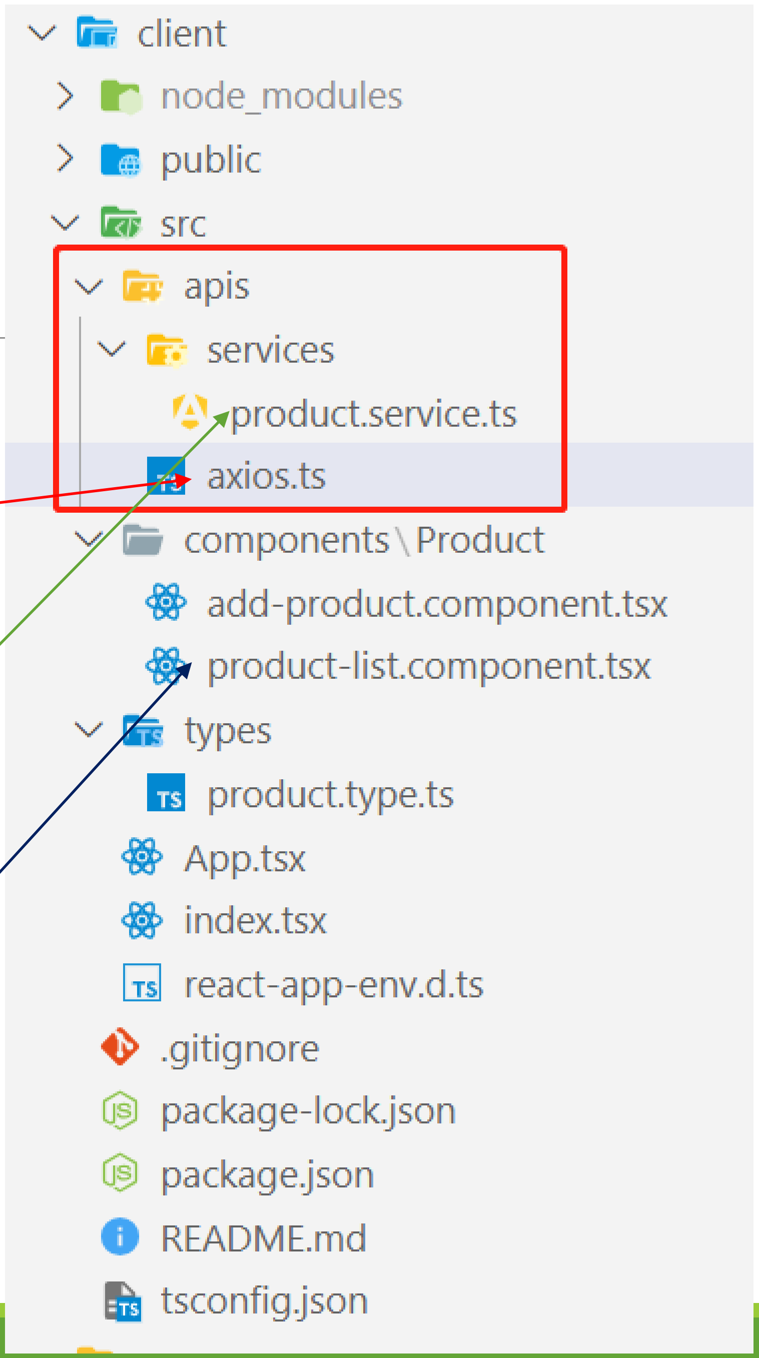
```
import http from '../axios';
import IProduct from '../../types/product.type';

const getAll = () => {
    return http.get('/products');
}

export default {getAll}
```

```
  useEffect(() => {
    async function retrieveProducts() {
      const response = await productService.getAll();
      setProducts(response.data);
    }

    retrieveProducts();
  }, []);
```



```
client
  node_modules
  public
  src
    apis
      services
        product.service.ts
      axios.ts
    components\Product
      add-product.component.tsx
      product-list.component.tsx
    types
      product.type.ts
    App.tsx
    index.tsx
    react-app-env.d.ts
  .gitignore
  package-lock.json
  package.json
  README.md
  tsconfig.json
```