upGrad

*#RahoAmbitious*

# Full Stack Software Development

upGrad

**Course:** Advanced Frontend Development Using React

**Lecture on:** Backend Integration

# In the last class, we discussed

- Forms in HTML handle data in the DOM, whereas React forms manage data within the internal state of the form component itself
- Such components are called ‘controlled components’
- Material-UI provides a component library for a common set of out-of-the-box but customisable components that can be used to build forms
- Material-UI Form Validator library can be used to plug in data validation into the forms created using Material-UI

# Poll 1

How do you use React forms to monitor the modifications?

A. By adding the event handlers
B. They are monitored automatically.
C. Both of the above
D. None of the above

# Poll 1 (Answer)

How do you use React forms to monitor the modifications?

**A. By adding the event handlers**

B. They are monitored automatically.

C. Both of the above

D. None of the above

# Poll 2

State whether the following statement is True or False:

We can have more than one validator for a single field.

A. True

B. False

# Poll 2 (Answer)

State whether the following statement is True or False:

We can have more than one validator for a single field.

**A. True**

B. False

# Today's Agenda

1. Introduction to REST APIs
    - CRUD operations
    - Response codes
2. Session and Authentication
3. Using Fetch

# Backend Integration

upGrad

So far in our application, we have the static data on which we are working. In the real world, this is not the way in which data is handled.

In a real-life scenario, we have the database at the backend, which stores our data, and we can use, modify and operate on it using the APIs exposed by the backend server.

Data cannot be handled in a static manner because we may have to perform the CRUD (Create, Replace, Update, Delete) operations on the data. Therefore, we need to integrate the application with the back end to have more dynamicity in the application.

# Introduction to REST APIs

# Introduction to REST APIs

upGrad

## What is RESTful?

- REST stands for ‘REpresentational State Transfer’
- It defines a set of architectural constraints, but it is not a protocol or a standard
- REST APIs are stateless, that is, the response of one request does not depend on a previous request in any way
- When a RESTful API is called, it transfers a representation of the state of an entity or resource to the endpoint or requestor
- This communication occurs over HTTP methods, such as GET, POST, PUT, PATCH and DELETE
- JSON is the most commonly used data format

## CRUD Operations

APIs are a way of exchanging data between the server and the client.

Most data exchanges are in the form of resources or entities. For example, a product or an employee record. One can perform the following operations on an entity:

- **C**reate
- **R**ead
- **U**pdate
- **D**elete

## CRUD in REST

REST uses the following HTTP methods corresponding to CRUD operations:

- **C**reate - POST
- **R**ead - GET
- **U**pdate - PUT/PATCH
- **D**elete - DELETE

## Response Codes

Since REST uses the HTTP protocol, the response for an API call is also mapped to HTTP response codes, such as:

- **200 - ‘OK’** (universal successful response)
- **201 - ‘Created’** (using POST)
- **400 - ‘Bad Request’** (invalid request data was provided to an API)
- **401 - ‘Unauthorized’** (user not authenticated)
- **403 - ‘Forbidden’** (user is authenticated but not authorised to perform the operation)
- **404 - ‘Not Found’** (the requested resource or entity does not exist)

# Poll 3

Which of the following are true for REST APIs?

(**Note:** More than option may be correct.)

A. REST APIs use the SOAP protocol for data transfer.
B. REST APIs are based on the HTTP protocol.
C. REST APIs preserve state and context on the server side between calls.
D. JSON is the most commonly used data format in REST APIs.

# Poll 3 (Answer)

Which of the following are true for REST APIs?

(**Note:** More than option may be correct.)

A. REST APIs use the SOAP protocol for data transfer.

**B. REST APIs are based on the HTTP protocol.**

C. REST APIs preserve state and context on the server side between calls.

**D. JSON is the most commonly used data format in REST APIs.**

# Poll 4

What will be the status code of the response when we pass a non-numeric value to a parameter that expects a numeric value in a REST API?

A. 201
B. 302
C. 400
D. 404

# Poll 4 (Answer)

What will be the status code of the response when we pass a non-numeric value to a parameter that expects a numeric value in a REST API?

A. 201
B. 302
**C. 400**
D. 404

# Session and Authentication

## How Does an API Know If You Have Access to the Resource?

Some common ways that prove to the API that you have the right to access a resource are as follows:

- **Session cookies**
  The server sets a cookie after you log in and this cookie is sent across with each subsequent request, which the server can use to check if you are authenticated
- **Auth tokens**
  Similar to a cookie, the server might also issue you a token that needs to be sent as part of an HTTP header for all requests. The server can then read this cookie to check if you are authenticated

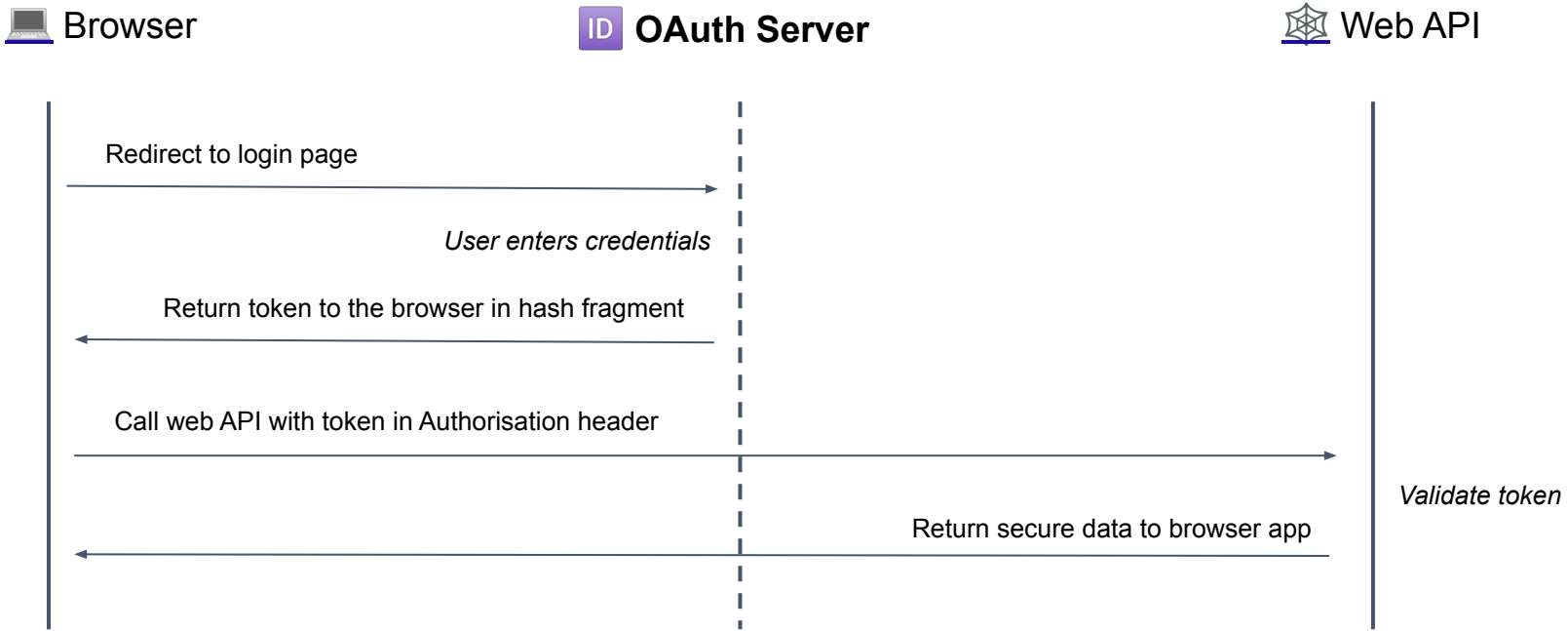
upGrad

## Auth Mechanisms

- Basic Authentication using HTTP
- API Keys
- OAuth 2.0
- JSON Web Tokens (JWT)

upGrad

## OAuth 2.0 Implicit Flow



**Note:** The auth token is returned in the hash fragment of the URL since any request to a URL containing a hash fragment does not send the hash fragment over the network. Hence, it is not possible for an attacker to get the auth token by sniffing the network.

# Poll 5

Which of the following is/are valid mechanism(s) for authentication in Web APIs?
(**Note:** More than option may be correct.)

A. JWT
B. Local storage
C. Session cookie
D. GET request

# Poll 5 (Answer)

Which of the following is/are valid mechanism(s) for authentication in Web APIs?
(**Note:** More than option may be correct.)

**A. JWT**

B. Local storage

**C. Session cookie**

D. GET request

# Poll 6

How does the OAuth 2.0 server return the auth token to the client app in the implicit flow?

A. In the response body
B. As a URL parameter
C. By setting a cookie
D. In the hash fragment of the redirect URL

# Poll 6 (Answer)

How does the OAuth 2.0 server return the auth token to the client app in the implicit flow?

A. In the response body
B. As a URL parameter
C. By setting a cookie
**D. In the hash fragment of the redirect URL**

# Using Fetch

# Fetching Data From Server

upGrad

The data can be sent or received from an API server through the following ways:

- **XMLHttpRequest** (XHR)
  Commonly also referred to as AJAX
- Using a library, such as **Axios**
- Using the browser JavaScript **Fetch** API

# Fetching Data From Server

upGrad

## XMLHttpRequest

Native browser object:

```
var xhr = new XMLHttpRequest();
xhr.onload = function () {
    if (xhr.readyState === xhr.DONE) {
        if (xhr.status === 200) {
            console.log(xhr.response);
            // responseText contains the response data
            console.log(xhr.responseText);
        }
    }
};
// Request for a product with the id 1234
xhr.open("GET", "/product/1234");
xhr.send(null);  // You can send some data instead of null to a POST API
```

# Fetching Data From Server

upGrad

## Using Axios

Promise-based HTTP client for NodeJS and browser:

```
import * as axios from 'axios';

// Make a request for a product with a given ID
axios.get('/product/1234')
  .then(function (response) {
    // do something with the data
    console.log(response);
  })
  .catch(function (error) {
    // handle error
    console.log(error);
  });
```

# Fetching Data From Server

upGrad

## Using Fetch API

```
// Make a request for a product with a given ID
fetch('/product/1234')
  .then(function (response) {
    // parse JSON response
    return response.json();
  })
  .then(function (json) {
    // do something with the data
    console.log(response);
  })
  .catch(function (error) {
    // handle error
    console.log(error);
  });
```

# Fetching Data From Server

upGrad

## POST Request Using Fetch

```
// Create a request to add a new product
fetch('/product', {
  method: 'POST', // 'GET', 'PUT', 'PATCH', 'DELETE' all work here
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify(requestData),
)
  .then(function (response) {
    // parse JSON response
    return response.json();
  })
  .then(function (json) {
    // do something with the data
    console.log(response);
  })
  .catch(function (error) {
    // handle error
    console.log(error);
  });
```

# Fetching Data From Server

upGrad

## Integrating React Components With Data

- **Reading data from server**
  Example: browse product, contact list
- **Writing data to server**
  Example: create new user form

# Fetching Data From Server

upGrad

## Presenting Data

Load data right after the component mounts:

```
function ContactList() {
  const [data, setData] = React.useState(null);
  React.useEffect(() => {
    fetch('/contacts')
      .then(response => response.json())
      .then(json => setData(json));
  });

  if (!data) return <div>Loading. Please wait...</div>
  return (
    <div>
      Contacts
      {data.contacts.map((contact) => { ... })}
    <div>
  );
}
```

# Fetching Data From Server

upGrad

## Writing Data

Write data on form submission or in event handler:

```
function AddContact() {
  const addContact= () => {
    fetch('/contact', {
      method: 'POST',
       headers: {
         'Content-Type': 'application/json',
       },
       body: JSON.stringify(requestData),
    );
  }

  return (
    <form onSubmit={addContact}>
      ...
    </form>
  );
}
```

# Fetching Data From Server

upGrad

## Phone Directory

Now, let's integrate the phone directory app with the data from the app. Basically, this will be a three-step process:

1. Define a state variable (initially empty) to contain the data

```
const [subscribersList, setSubscribersList] = useState([]);
```

1. Define a function loadData() to call the API and update the state variable's value to the response from the API

```
async function loadData() {
  const rawResponse = await  fetch("http://localhost:7081/api/contacts");
  const data = await rawResponse.json();
  setSubscribersList(data);
});
```

# Fetching Data From Server

upGrad

## Phone Directory

3. Call the loadData() function when the component mounts

```
useEffect(()=>{
  loadData();
},[]);
```

Code Reference

# Poll 7

Which of the following return a Promise when an API call is made?
(**Note:** More than option may be correct.)

A. XMLHttpRequest
B. Axios
C. Fetch
D. ActiveXObject

# Poll 7 (Answer)

Which of the following return a Promise when an API call is made?
(**Note:** More than option may be correct.)

A. XMLHttpRequest
**B. Axios**
**C. Fetch**
D. ActiveXObject

# Poll 8

Where in the lifecycle of a React component is it best to make an API call to fetch the data to be used for rendering the component?

A. componentDidMount
B. componentWillReceiveProps
C. componentWillUnmount
D. render

# Poll 8 (Answer)

Where in the lifecycle of a React component is it best to make an API call to fetch the data to be used for rendering the component?

**A. componentDidMount**

B. componentWillReceiveProps

C. componentWillUnmount

D. render

# Hands-on Exercise

upGrad

Build a React app that displays your IP address by calling a public API:

***https://api.ipify.org/?format=json***

The stub code is provided here.

The solution code is provided here.

All the code used in today's session can be found in the link provided below:

https://github.com/upgrad-edu/react-hooks/tree/Session10

# Doubt Clearance (5 minutes)

upGrad

# Important Questions

# Important Questions

upGrad

1. What are the different ways to work with async data in React?
2. How do we handle server errors?
3. What are the best practices around error handling?
4. What are the common error response codes?
5. What are the different HTTP methods used in backend of web applications?
6. What is the difference between PUT and POST?

# Key Takeaways

upGrad

- REST APIs provide a way of communicating data between the client and the server. They are stateless and support CRUD operations based on the HTTP methods, such as GET, POST, PUT, PATCH and DELETE
- Session cookies and auth tokens are commonly used ways to handle authentication
- OAuth 2.0 is a widely accepted protocol used to handle API authentication
- You can use XMLHttpRequest, a library like Axios or the JavaScript Fetch API to send/receive data asynchronously from the client web app

# These tasks are to be completed after today's session:

| MCQs |
| --- |
| Coding Questions |
| Course Project (Part B) - MongoDB Collections and API Routes Checkpoint 4 |

# In the next class, we will discuss...

1. Introduction to Redux
    - The Problems with React's State Management
    - The Principles of Redux
2. Redux Flow
    - Redux Pattern
    - Redux Terminology
3. Using Redux with React

upGrad

*#RahoAmbitious*

# Thank You!