SWINBURNE UNIVERSITY OF TECHNOLOGY

Week 4 – React.js & Database

Acknowledgement of Country

We respectfully acknowledge the Wurundjeri People of the Kulin Nation, who are the Traditional Owners of the land on which Swinburne's Australian campuses are located in Melbourne's east and outer-east, and pay our respect to their Elders past, present and emerging.

We are honoured to recognise our connection to Wurundjeri Country, history, culture, and spirituality through these locations, and strive to ensure that we operate in a manner that respects and honours the Elders and Ancestors of these lands.

We also respectfully acknowledge Swinburne's Aboriginal and Torres Strait Islander staff, students, alumni, partners and visitors.

We also acknowledge and respect the Traditional Owners of lands across Australia, their Elders, Ancestors, cultures, and heritage, and recognise the continuing sovereignties of all Aboriginal and Torres Strait Islander Nations.

Props in React.js

SWINBURNE UNIVERSITY OF TECHNOLOGY

What is Props (Properties):

Props serve as a communication mechanism between components and allow parent components to **pass data** to their child components. They are immutable, meaning child components **cannot modify** them. Through props, you can pass various types of data to child components, including strings, numbers, objects, arrays, and more.

In the parent component, you can pass props to child components in a manner similar to adding attributes to HTML elements. In the child component, you access these passed props data using **this.props**. This empowers parent components to convey dynamic data and configuration to child components, enabling child components to render in diverse ways.

Props in React.js

SWINBURNE UNIVERSITY OF TECHNOLOGY

```
import React from 'react';
import Children from './Children';

class Parent extends React.Component {
  render() {
    const msg = "this is parent element";

    return (
      <div>
        <h2>Parent Component</h2>
        <Children msg={msg} />
      </div>
    );
  }
}
export default Parent;
```

```
import React from 'react';

class Children extends React.Component {
  render() {
    return (
      <div>
        <h2>Hi, this is Children</h2>
        <p>Hi, here is the value of Parent {this.props.msg}!</p>
      </div>
    );
  }
}
export default Children;
```

State in React.js

SWINBURNE UNIVERSITY OF TECHNOLOGY

What is State :

State represents internal data or the state of a component that can be changed as needed. Unlike props, state is mutable, and you can update it using React's provided **setState** method. When state changes, React automatically re-renders the component to reflect the most recent state.

In general, state is used to store data that changes over time, such as user input, interactions, and dynamic changes. For instance, a counter component might use state to store the current count value. When a user clicks a button, you can use setState to update the count value and trigger a re-render.

State in React.js

SWINBURNE UNIVERSITY OF TECHNOLOGY

```
class Counter extends React.Component {
  constructor() {
    super();
    this.state = { count: 0 };
  }

  increment = () => {
    this.setState({ count: this.state.count + 1 });
  };

  render() {
    return (
      <div>
        <p>Count: {this.state.count}</p>
        <button onClick={this.increment}>Increment</button>
      </div>
    );
  }
}
```

*Sets the initial value of **count** to 0 in **constructor()***

State in React.js

SWINBURNE UNIVERSITY OF TECHNOLOGY

```
class Counter extends React.Component {
  constructor() {
    super();
    this.state = { count: 0 };
  }

  increment = () => {
    this.setState({ count: this.state.count + 1 });
  };

  render() {
    return (
      <div>
        <p>Count: {this.state.count}</p>
        <button onClick={this.increment}>Increment</button>
      </div>
    );
  }
}
```

Define the function ***increment()*** to set the value of **count** by this.***setState()***

State in React.js

SWINBURNE UNIVERSITY OF TECHNOLOGY

```
class Counter extends React.Component {
  constructor() {
    super();
    this.state = { count: 0 };
  }

  increment = () => {
    this.setState({ count: this.state.count + 1 });
  };

  render() {
    return (
      <div>
        <p>Count: {this.state.count}</p>
        <button onClick={this.increment}>Increment</button>
      </div>
    );
  }
}
```

Display the value of **count** in the <p> tag

Use the ***onClick()*** function to trigger the ***increment()*** function

Introduction of D3.js

SWINBURNE UNIVERSITY OF TECHNOLOGY

Introduction of D3.js

SWINBURNE UNIVERSITY OF TECHNOLOGY

D3.js (Data-Driven Documents) is a JavaScript library for creating data visualizations. It enables developers to manipulate data using HTML, SVG, and CSS to generate interactive and visual charts, graphs, and visual elements. D3.js emphasizes a data-driven approach, allowing developers to dynamically generate, update, and render visual content based on data.

Introduction of D3.js

SWINBURNE UNIVERSITY OF TECHNOLOGY

D3.js provides powerful features for data manipulation, data binding, DOM manipulation, and animations, enabling developers to create highly customized and interactive data visualizations. As D3.js doesn't offer advanced chart or graph encapsulation, developers have detailed control and customization over data and presentation, often requiring some programming and design experience.

Introduction of D3.js

SWINBURNE UNIVERSITY OF TECHNOLOGY

Key features of D3.js include:

- **Data-Driven**: D3.js encourages a data-centric mindset, binding data to Document Object Model (DOM) elements to achieve dynamic visualization.
- **Flexibility**: D3.js offers foundational building blocks, enabling developers to create various custom visual charts and graphs.
- **DOM Manipulation**: D3.js allows manipulation of the Document Object Model (DOM) to create, update, and remove elements in response to data changes.

Install D3.js with: `npm install d3`

Introduction of D3.js

SWINBURNE UNIVERSITY OF TECHNOLOGY

Key features of D3.js include:

- **Data Transformation**: D3.js provides robust data transformation tools for processing and converting data to fit different visualization needs.
- **Animation Effects**: D3.js permits adding transitions and animation effects to visualizations, making data changes smoother and more engaging.
- **Interactivity**: D3.js enables developers to add interactivity to visualizations, such as mouse hover, click, and drag interactions.

Install D3.js with: `npm install d3`

Usage of D3.js

SWINBURNE UNIVERSITY OF TECHNOLOGY

D3.js : Bar Chart

```
import React, {Component} from 'react';
// import your d3 library here
// to install ds by : npm install d3
import * as d3 from "d3";

// This is the basic template for the d3 chart
class SampleChart extends Component {

}

export default SampleChart;
```

Usage of D3.js

```
import React, { Component } from 'react'
import * as d3 from 'd3'

class sampleChart extends Component {
    componentDidMount() {
        this.generateChart();
    }
    generateChart() {
        const data = [10, 5, 7, 8, 2, 8];

        const svg = d3.select("body")
                    .append("svg")
                    .attr("width", 1200)
                    .attr("height", 400);

        svg.selectAll("rect")
            .data(data)
            .enter()
            .append("rect")
            .attr("x", (d, i) => i * 70)
            .attr("y", (d, i) => 300 - 10 * d)
            .attr("width", 65)
            .attr("height", (d, i) => d * 10)
            .attr("fill", "green");
    }
    render() {
        return <div ></div>
    }
}
export default sampleChart;
```

In order to display the bar chart when the sampleChart component is mounted to the DOM, we will utilize the ComponentDidMount lifecycle.

In D3.js, loading the visualization logic within the **ComponentDidMount** lifecycle method is a common practice in React applications.

SWINBURNE UNIVERSITY OF TECHNOLOGY

Usage of D3.js

```
import React, { Component } from 'react'
import * as d3 from 'd3'

class sampleChart extends Component {
    componentDidMount() {
        this.generateChart();
    }
    generateChart() {
        const data = [10, 5, 7, 8, 2, 8];

        const svg = d3.select("body")
                      .append("svg")
                      .attr("width", 1200)
                      .attr("height", 400);

        svg.selectAll("rect")
            .data(data)
            .enter()
            .append("rect")
            .attr("x", (d, i) => i * 70)
            .attr("y", (d, i) => 300 - 10 * d)
            .attr("width", 65)
            .attr("height", (d, i) => d * 10)
            .attr("fill", "green");
    }
    render() {
        return <div ></div>
    }
}
export default sampleChart;
```

generateChart() is the method we use to create D3.js charts. In React, this step is crucial as it ensures that the chart is only displayed when the component is mounted to the DOM.

Firstly, we define a parameter which contains the data for the chart we want to visualise. Next, we define an image in **SVG** format using the D3.js method. SVG is used because it is scalable and the data will not appear pixelated regardless of how the screen size is scaled or broadened.

SWINBURNE UNIVERSITY OF TECHNOLOGY

Usage of D3.js

```
import React, { Component } from 'react'
import * as d3 from 'd3'

class sampleChart extends Component {
    componentDidMount() {
        this.generateChart();
    }
    generateChart() {
        const data = [10, 5, 7, 8, 2, 8];

        const svg = d3.select("body")
                      .append("svg")
                      .attr("width", 1200)
                      .attr("height", 400);

        svg.selectAll("rect")
            .data(data)
            .enter()
            .append("rect")
            .attr("x", (d, i) => i * 70)
            .attr("y", (d, i) => 300 - 10 * d)
            .attr("width", 65)
            .attr("height", (d, i) => d * 10)
            .attr("fill", "green");
    }
    render() {
        return <div ></div>
    }
}
export default sampleChart;
```

The ***d3.select()*** method selects an HTML element. It selects the first element that matches the passed parameter and creates a node for it. Here we have passed the body element.

The ***attr()*** method is responsible for adding attributes to the element, which can be any attribute of the HTML element, such as class, height, width, etc.

SWINBURNE UNIVERSITY OF TECHNOLOGY

Usage of D3.js

```
import React, { Component } from 'react'
import * as d3 from 'd3'

class sampleChart extends Component {
    componentDidMount() {
        this.generateChart();
    }
    generateChart() {
        const data = [10, 5, 7, 8, 2, 8];

        const svg = d3.select("body")
                      .append("svg")
                      .attr("width", 1200)
                      .attr("height", 400);

        svg.selectAll("rect")
            .data(data)
            .enter()
            .append("rect")
            .attr("x", (d, i) => i * 70)
            .attr("y", (d, i) => 300 - 10 * d)
            .attr("width", 65)
            .attr("height", (d, i) => d * 10)
            .attr("fill", "green");
    }
    render() {
        return <div ></div>
    }
}
export default sampleChart;
```

.selectAll("rect") selects all rectangular elements in the SVG.

.data(data) binds the data to the selected rectangle element.

.attr("x", (d, i) => i * 70): here the x-coordinate of the rectangles is set, and the x-coordinate interval of each rectangle is set to 70, according to the index i of the data in the array

```
▼<svg width="1200" height="400">
    <rect x="0" y="200" width="65" height="100" fill="green"></rect> =
    <rect x="70" y="250" width="65" height="50" fill="green"></rect>
    <rect x="140" y="230" width="65" height="70" fill="green"></rect>
    <rect x="210" y="220" width="65" height="80" fill="green"></rect>
    <rect x="280" y="280" width="65" height="20" fill="green"></rect>
    <rect x="350" y="220" width="65" height="80" fill="green"></rect>
</svg>
```

Usage of D3.js

```
import React, { Component } from 'react'
import * as d3 from 'd3'

class sampleChart extends Component {
    componentDidMount() {
        this.generateChart();
    }
    generateChart() {
        const data = [10, 5, 7, 8, 2, 8];

        const svg = d3.select("body")
                      .append("svg")
                      .attr("width", 1200)
                      .attr("height", 400);

        svg.selectAll("rect")
            .data(data)
            .enter()
            .append("rect")
            .attr("x", (d, i) => i * 70)
            .attr("y", (d, i) => 300 - 10 * d)
            .attr("width", 65)
            .attr("height", (d, i) => d * 10)
            .attr("fill", "green");
    }
    render() {
        return <div ></div>
    }
}
export default sampleChart;
```

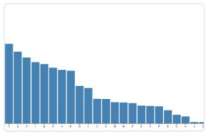
.attr("width", 65): sets the width of the rectangle to a fixed value 65

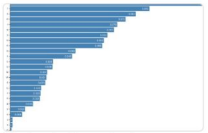
.attr("height", (d, i) => d * 10): set the height of the rectangle, according to the data d, set the height of the rectangle to 10 times the data value d

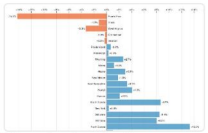
.attr("fill", "green"): set the fill colour of the rectangle to green

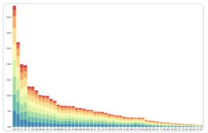
```
▼<svg width="1200" height="400">
    <rect x="0" y="200" width="65" height="100" fill="green"></rect> =
    <rect x="70" y="250" width="65" height="50" fill="green"></rect>
    <rect x="140" y="230" width="65" height="70" fill="green"></rect>
    <rect x="210" y="220" width="65" height="80" fill="green"></rect>
    <rect x="280" y="280" width="65" height="20" fill="green"></rect>
    <rect x="350" y="220" width="65" height="80" fill="green"></rect>
</svg>
```

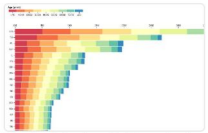
More Charts of D3.js

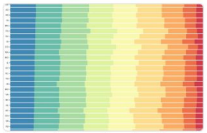

Bar chart

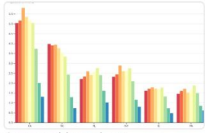

Horizontal bar chart

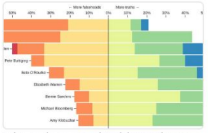

Diverging bar chart


Stacked bar chart

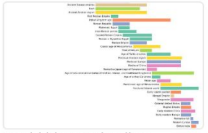

Stacked horizontal bar chart

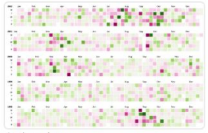

Stacked normalized horizon...

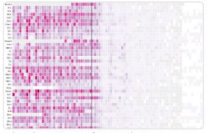

Grouped bar chart

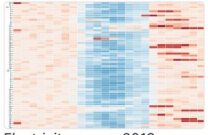

Diverging stacked bar chart

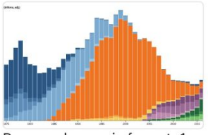

Marimekko chart


World history timeline


Calendar


The impact of vaccines

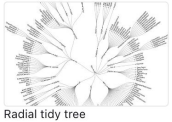

Electricity usage, 2019

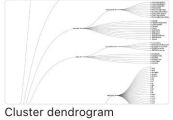

Revenue by music format, 1...

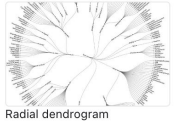

Treemap

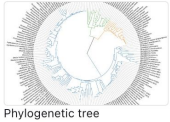

Cascaded treemap


Nested treemap

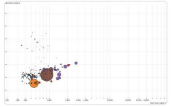

Radial tidy tree

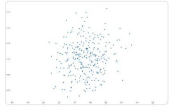

Cluster dendrogram

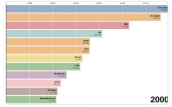

Radial dendrogram


Phylogenetic tree


Force-directed tree

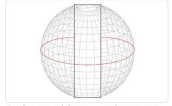

The wealth & health of natio...


Scatterplot tour

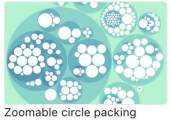


Bar chart race

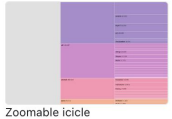

Zoom to bounding box


Orthographic to equirectang...


World tour


Zoomable circle packing


Collapsible tree

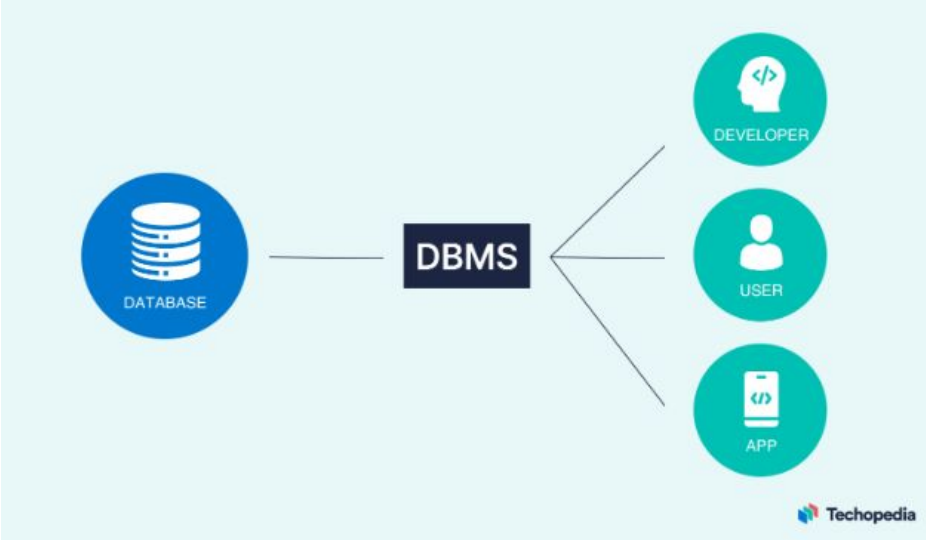

Zoomable icicle

How to store your data ?

Introduction of Database

- Databases play a crucial role in web systems and are considered a core component that significantly impacts the functionality, performance, and reliability of the system. Here's the importance of databases in a web system:

Data Storage and Management: Databases store and manage all the data required by a web application, including user information, product details, order records, log data, etc. Databases provide structured data storage, enabling efficient organization, retrieval, and updates of data.



Introduction of Database

Databases play a crucial role in web systems and are considered a core component that significantly impacts the functionality, performance, and reliability of the system. Here's the importance of databases in a web system:

Data Consistency and Integrity: Relational databases ensure data integrity and consistency through constraints such as primary keys, foreign keys, and uniqueness constraints. This is vital for maintaining data accuracy, especially when dealing with related data across multiple tables.

Data Querying and Analysis: Databases offer query languages like SQL that allow developers to retrieve and analyze data efficiently. This is crucial for generating reports, analyzing trends, making business decisions, and more.

Relational Databases (RDBMS)

SWINBURNE UNIVERSITY OF TECHNOLOGY

A relational database (RDBMS) is a database management system based on the relational model. In a relational database, data is organized and stored in tables (also known as relations), where each table consists of multiple rows and columns. Rows represent data records, and columns represent data fields. Relational databases use Structured Query Language (SQL) for querying, inserting, updating, and deleting data.

Data Structure: Data is stored in tables, and each table has defined columns and data types. Relationships between tables can be established, such as through primary keys and foreign keys.

Data Consistency and Integrity: Through constraints like primary keys, foreign keys, and uniqueness constraints, relational databases ensure data integrity and consistency. This means that data maintains logical associations between tables without contradictions or inconsistencies.

Relational Databases (RDBMS)

SWINBURNE UNIVERSITY OF TECHNOLOGY

- **A relational database (RDBMS)** is a database management system based on the relational model. In a relational database, data is organized and stored in tables (also known as relations), where each table consists of multiple rows and columns. Rows represent data records, and columns represent data fields. Relational databases use Structured Query Language (SQL) for querying, inserting, updating, and deleting data.

Query Language: SQL is used for various data operations, including selection, projection, joins, aggregation, and more, enabling flexible and powerful data retrieval and manipulation.

Relational Databases (RDBMS)

ROLL_NO	NAME	ADDRESS	PHONE	Age
1	Ram	Delhi	xxxxxxxxxx	18
2	RAMESH	GURGAON	xxxxxxxxxx	18
3	SUJIT	ROHTAK	xxxxxxxxxx	20
4	SURESH	Delhi	xxxxxxxxxx	18
3	SUJIT	ROHTAK	xxxxxxxxxx	20
2	RAMESH	GURGAON	xxxxxxxxxx	18

Non-Relational Databases (NoSQL)

SWINBURNE UNIVERSITY OF TECHNOLOGY

A non-relational database, often referred to as a **NoSQL database,** is a database management system that diverges from the traditional relational model. Non-relational databases are designed to handle various types of unstructured, semi-structured, or structured data and offer different data models and storage mechanisms compared to traditional relational databases.

Data Models: Non-relational databases support diverse data models such as document, key-value, column-family, and graph. Each data model is optimized for specific use cases and data structures.

Scalability: Many NoSQL databases are designed for horizontal scalability, allowing them to handle massive amounts of data and high levels of traffic by distributing data across multiple nodes or servers.

Non-Relational Databases (NoSQL)

Common types of non-relational databases include:

- **Document Stores**: Store data in JSON or BSON documents. Examples include MongoDB and Couchbase.
- **Key-Value Stores**: Store data as key-value pairs. Examples include Redis and Amazon DynamoDB.
- **Column-Family Stores**: Organize data into column families and columns. Examples include Apache Cassandra and HBase.
- **Graph Databases**: Store data in nodes and edges to represent complex relationships. Examples include Neo4j and Amazon Neptune.

NoSQL: Neo4j

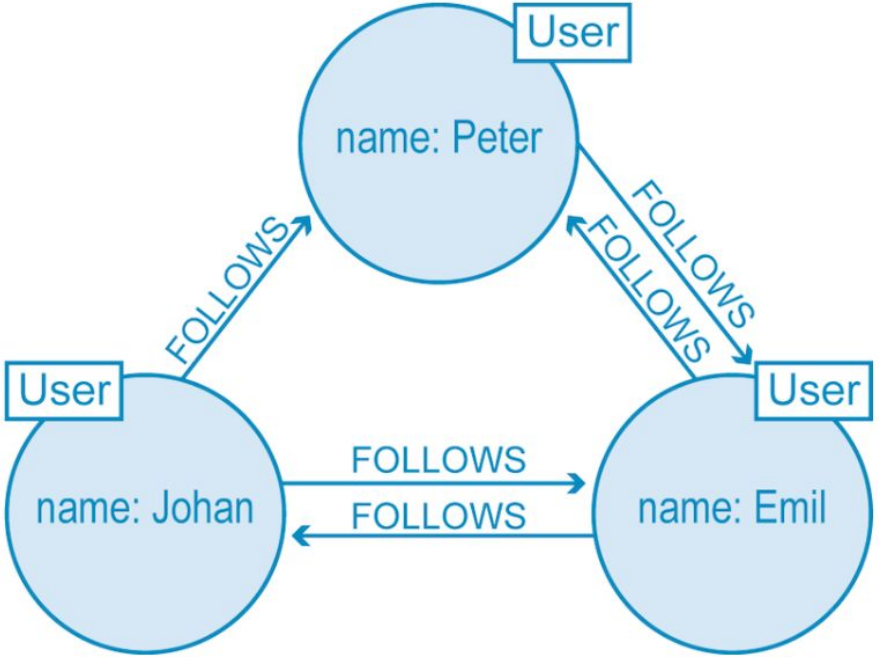
Neo4j is a high-performance graph database management system designed for storing, querying, and analyzing graph data. Graph databases employ a graph data model where data is stored in the form of nodes and relationships, making it well-suited for representing and dealing with complex relationships and connections. Here are some key features and an overview of Neo4j:

- **Graph Data Model:** Neo4j uses a data model comprising nodes and relationships to represent data. Nodes can have properties, while relationships can have direction and properties, allowing for precise representation of entity relationships and attributes.

- **High-Performance Queries**: Neo4j provides the Cypher query language, designed specifically for querying graph data. Cypher enables users to express complex graph queries, including matching nodes and traversing paths.

NoSQL: Neo4j

- **Complex Relationship Representation:** Neo4j excels in applications that require representation and analysis of intricate relationships, networks, and connections, such as social network analysis, recommendation systems, and knowledge graphs.
- **Application Areas:** Neo4j finds widespread application in fields such as social network analysis, recommendation systems, risk assessment, and biomedical research.



Learning Resources

MUI Documentation
https://mui.com/material-ui/getting-started/
D3.js Documentation
https://d3js.org/
Database Management System
https://www.techopedia.com/definition/24361/database-management-systems-dbms
What is MySQL
https://dev.mysql.com/doc/workbench/en/wb-develop-object-management-inspector.html
Data visualization in React using React D3
https://www.youtube.com/watch?v=YKDIsXA4OAc&ab_channel=LogRocket
What is neo4j
https://neo4j.com/