**TEORIA**

1. LWC Lifecycle Hook è un insieme di metodi che invocati durante il ciclo di vita di un componente Lightning Web Component (connectedCallback(),disconnectedCallback, renderedCallback(), constructor()).

2. Shadow DOM è una tecnica che permette di isolare il DOM di un componente dai suoi elementi esterni

3. Per Deploy di un Lightning Component si intende il processo di rendere un componente Lightning disponibile in un ambiente di Salesforce.

4. I principali decorators per gli LWC sono:

-@wire: collega una proprietà o un metodo a un flusso di dati proveniente da Apex o da una sorgente esterna.

-@api: rende pubblici una proprietà o un metodo, rendendoli accessibile da altri componenti.

-@track: monitora le modifiche apportate ad una proprietà e ne forza l'esecuzione del componente.

5. Lightning App Builder è uno strumento di Salesforce che permette di creare applicazioni personalizzate.

6. Il Controller lato Client (effettua operazioni locali) è un file JavaScript che gestisce la logica di interazione con l'utente, mentre il Controller lato Server (ha interazioni con Salesforce lato back end) è un file Apex che gestisce la logica e le operazioni sul server.

7. Per abilitare l'accesso al metodo di un controller lato server in un LWC, si utilizza la decorazione @wire o si utilizza una funzione Apex annotata con @AuraEnabled rendendo questa ultima accessibile al client.

8. Per eseguire una funzione prima del rendering di un componente si utilizza il lifecycle hook constructor() o connectedCallback().

9. Lightning Components sono basati su un'architettura Aura, mentre gli LWC sono basati su tecnologie web standard (come HTML, CSS e JavaScript). Dunque la principale differenza è che LWC è più performante e sfrutta l'uso di web standards.

10. Il Salesforce Lightning Design System è una libreria di componenti di interfaccia utente e di linee guida progettate al fine di creare applicazioni che rispettano le best practices di Design in Salesforce.

11. Gli attributi di un componente in Salesforce Lightning sono variabili utilizzate per memorizzare dati all'interno del componente stesso e che possono essere modificati nel codice JS.

12.

13. É una direttiva di markup dei componenti Aura che condiziona la visualizzazione di una sezione di codice. Ha un funzionamento simile  all' if nei linguaggi di programmazione infatti permette di rendere visualizzabile o meno una parte di un componente in base a una condizione booleana espressa nell'aura:if stesso.

14. Gli Helper in Salesforce Lightning sono funzioni JavaScript utilizzate per separare la logica di business dalla logica di rendering del componente che rendono possibile scrivere porzioni di codice riutilizzabili che possono essere chiamate sia da altri metodi del componente  sia dal controller .

15. È utilizzata per impostare i parametri da passare ad un controller Apex quando si esegue una chiamata asincrona (definisce i valori dei parametri da inviare al server)

16.  Definisce una funzione di callback che sarà eseguita una volta che un'azione  come una chiamata Apex avrà completato l'esecuzione.

17. È un file o una raccolta di file che vengono caricati nel sistema e resi disponibili per le applicazioni Salesforce.

18.  Un componente "child" è un componente che è "figlio" di  un altro componente, detto "parent". In Salesforce Lightning, il componente figlio viene implementato nel componente genitore utilizzando la sintassi di markup <c:childComponent />

**ESERCIZIO UNO**

APP

<aura:application extends="force:slds">

<c:CompleannoComp />

</aura:application>

COMPONENT

<aura:component>

<aura:attribute name="compleanno" type="Date"></aura:attribute>

<aura:attribute name="giorniRimanenti" type="Integer" default="0"></aura:attribute>

<aura:attribute name="sceltaRegalo" type="String"></aura:attribute>

<aura:attribute name="messagio" type="String"></aura:attribute>

<lightning:card>

<div>

<p>="Inserisci la data del tuo compleanno"></p>

<lightning:input name="compleanno" label="Data del tuo compleanno" type="date" value="{!v.compleanno}" onchange="{!c.calcolaGiorni}"></lightning:input>

<p>Mancano {!v.giorniRimanenti} giorni al tuo compleanno"</p>

<p>Cosa ti piacerebbe ricevere come regalo?</p>

<lightning:select name="sceltaRegalo" label="seleziona il regalo che preferisci" value="{!v.sceltaRegalo}" onchange="{!c.aggiornaMessaggio}">

<option value="">Seleziona regalo</option>

<option value="Viaggio">Viaggio</option>

<option value="Sorpresa">Sorpresa</option>

<option value="Regalo">Regalo</option>

</lightning:select>

<aura:if isTrue="{!not(empty(v.sceltaRegalo))}">

<p>

{!v.sceltaRegalo == 'Viaggio' ? 'Bel viaggio!!!!' : v.sceltaRegalo == 'Sorpresa' ? 'Bella sorpresa!!!!' : 'Bel regalo!!!!'}

</p>

</aura:if>

</div>

</lightning:card>

</aura:component>

CONTROLLER (NON FUNZIONA IL CALCOLO DEI GIORNI CHE MANCANO)

({

calcolaGiorni : function(component, event, helper) {

var dataSelezionata = component.get("v.dataSelezionata");

if (dataSelezionata) {

var oggi = new Date();

var compleanno = new Date(dataSelezionata);

var differenzaDate = compleanno - oggi;

var giorniMancanti = Math.ceil(differenzaDate / (1000 \* 3600 \* 24));

if (giorniMancanti < 0) {

compleanno.setFullyear(oggi.FullYear() + 1);

differenzaDate = compleanno - oggi;

giorniMancanti= Math.ceil(differenzaDate / (1000 \* 3600 \* 24));

}

component.set("v.giorniMancanti",giorniMancanti);

}

}

})

**ESERCIZIO DUE**

META

<?xml version="1.0" encoding="UTF-8"?>

<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">

    <apiVersion>62.0</apiVersion>

    <isExposed>true</isExposed>

    <targets>

        <target>lightning\_\_AppPage</target>

        <target>lightning\_\_HomePage</target>

        <target>lightning\_\_RecordPage</target>

    </targets>

</LightningComponentBundle>

HTML

<template>

    <div class="quiz">

        <form onsubmit={buttonSubmitt}>

            <div class="question">

                <label> Quale dei seguenti non è un loop di modelli?</label><br/>

                <lightning-radio-group options={opzioniDomanda1} value={rispostaDomanda} onchange={cambiaD1} required></lightning-radio-group>

            </div>

            <div class="question">

                <label> Quale dei seguenti file non è presente in un LWC?</label><br/>

                <lightning-radio-group options={opzioniDomanda2} value={rispostaDomanda2} onchange={cambiaD2} required></lightning-radio-group>

            </div>

            <div class="question">

                <label> Quale dei seguenti non è una direttiva?</label><br/>

                <lightning-radio-group options={opzioniDomanda3} value={rispostaDomanda3} onchange={cambiaD3} required></lightning-radio-group>

            </div>

            <lightning-button type="reset" label="Reset" onclick={resetRisposte}></lightning-button>

            <lightning-button type="submit" label="Submit" disabled={buttonDisabilitato}></lightning-button>

        </form>

        <template if:true={messaggioRis}>

            <div class={classRis}>{messaggioRis}</div>

        </template>

    </div>

</template>

JS

import { LightningElement, track } from 'lwc';

export default class QuizComponent extends LightningElement {

    @track rispostaDomanda1;

    @track rispostaDomanda2;

    @track rispostaDomanda3;

    @track messaggioRis;

    @track classRis;

    get opzioniDomanda1() {

        return [

            { label: 'for:each', value: 'for:each' },

            { label: 'iterator', value: 'iterator' },

            { label: 'map loop', value: 'map loop' },

        ];

    }

    get opzioniDomanda2() {

        return [

            { label: '.svg', value: '.svg' },

            { label: '.apex', value: '.apex' },

            { label: '.js', value: '.js' },

        ];

    }

    get opzioniDomanda3() {

        return [

            { label: '@track', value: '@track' },

            { label: 'if:true', value: 'if:true' },

            { label: 'for:each', value: 'for:each' },

        ];

    }

    get buttonDisabilitato() {

        return !this.rispostaDomanda1 || !this.rispostaDomanda2 || !this.rispostaDomanda3;

    }

    cambiaD1(event) {

        this.rispostaDomanda1 = event.detail.value;

    }

    cambiaD2(event) {

        this.rispostaDomanda2 = event.detail.value;

    }

    cambiaD3(event) {

        this.rispostaDomanda3 = event.detail.value;

    }

    resetRisposte() {

        this.rispostaDomanda1 = undefined;

        this.rispostaDomanda2 = undefined;

        this.rispostaDomanda3 = undefined;

        this.messaggioRis = null;

        this.classRis = '';

    }

    buttonSubmitt(event) {

        event.preventDefault();

        const risposteCorrette = [

            'iterator',     //

            '.apex',

            '@track'

        ];

        const risposteUser = [

            this.rispostaDomanda1,

            this.rispostaDomanda2,

            this.rispostaDomanda3

        ];

        const score = risposteUser.filter((risposte, index) => risposte === risposteCorrette[index]).length;

        if (score === 3) {

            this.messaggioRis = 'Tutte le risposte sono corrette!';

            this.classRis = 'successo';

        } else {

            this.messaggioRis = `${3 - score} risposta/e sbagliata/e. Riprova!`;

            this.classRis = 'errore';

        }

    }

}

CSS

.errore {

    color: red;

}

.successo {

    color: green;

}