Parekh Shaishav Deepeshkumar

23276453

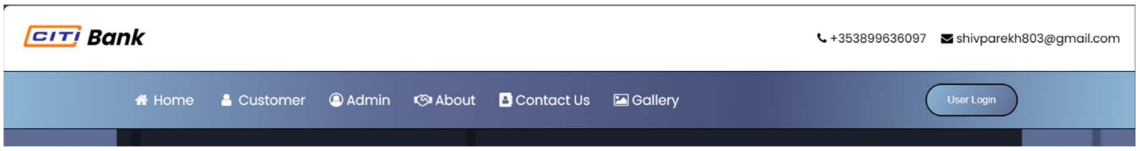
Assignment 2

Introduction

In response to the requirements outlined in the second assignment, I have implemented several new features in the Citi Bank web application to enhance user experience and functionality. Key features introduced are: The Dynamic Menu Bar, form validations, storage and retrieval of data, and event handling using javascript which enhances navigation and accessibility throughout the website.

Dynamic Menu Bar

The Dynamic Menu Bar I have used is a horizontal navigation bar designed to improve user navigation and accessibility. The key functionalities of the Dynamic Menu Bar include:



1. User Interface of Nav Bar: I have designed the navigation bar with a great CSS such that it eye catches the user and also used icons along with the navigation links to guide the user on the different links of the website. I have also used the hover functionality which displays the hidden dropdown menu of the customer.

2. Sticky Navigation: The navigation bar becomes sticky when the user scrolls over the website. This means that as the user scrolls down the page, the navigation bar remains fixed at the top of the viewport, always ensuring easy access to navigation options.

3. Implementation: To implement the sticky navigation feature, JavaScript is utilized to dynamically attach CSS attributes to the navigation bar based on the user's scrolling behavior. Specifically, the offset position of the navigation bar is obtained, and a

function is called to apply CSS attributes that fix the navigation bar in place while scrolling

```
// Sticky Navigation Bar
window.onscroll = function () { myFunction() };

var navbar = document.getElementById("navbar"); // Get the navbar
var sticky = navbar.offsetTop;

// Sticky navigation bar functionality
function myFunction() {
    if (window.pageYOffset >= sticky) {
        navbar.classList.add("sticky")
    } else {
        navbar.classList.remove("sticky");
    }
}
```

4. Enhanced User Experience: By giving users continuous access to navigation options and removing the need for them to scroll back to the top of the page in order to access the menu, the sticky navigation bar dramatically improves the user experience.

The navigation bar as the same in assignment 1 but now it is designed using JavaScript instead of HTML and CSS.

Form Validation Implementation

I have put in place extensive form validation features to guarantee data integrity and improve user experience in response to the requirement for strong form validation throughout the website. The form validation procedures cover a number of topics, such as handling null values, identifying inputs that are not necessary, and informing users of mistakes clearly and understandably through JavaScript alerts. The main method used for this is to highlight mistake spots and ask users to make adjustments by using JavaScript events, specifically the `onfocus()` event.

```
    //Validating Password
    if (pwd.length === 0) {
        alert("You must enter a valid password");
        document.getElementById('pwd').style.borderColor = "red"
        password.focus();
        return false;
    }
    else {
        document.getElementById('pwd').style.borderColor = "green"
    }

    if (pwd.length < 6) {
        alert("Length of Password must be greater than 6 characters");
        document.getElementById('pwd').style.borderColor = "red"
        password.focus();
        return false;
    }
    else {
        document.getElementById('pwd').style.borderColor = "green"
    }
```

Implementation Details

1. Form Validation Coverage:

Form validation has been implemented for all fields across different forms available on the website. These include but are not limited to:

- Name fields
- Radio buttons
- Email addresses
- Passwords
- Date fields
- Addresses
- Postal codes
- Individual form fields on various pages

2. Utilization of JavaScript Events:

- The `onfocus()` event is utilized to trigger validation functions when users interact with form fields. This event ensures that error messages are displayed in real-time as users navigate through the form.

3. Centralized Validation for User Signup:

- The `**formValidations.js**` file houses all of the form validations for the user signup page. This method facilitates easy updates and alterations to validation logic by encouraging code organisation and maintainability.

4. Embedded Validations for Other Pages:

- For pages with fewer form fields, validation scripts are embedded within the `<head>` tag of respective HTML documents. This approach minimizes overhead while ensuring that all necessary validations are in place.

```
<head>
    <script>
        function check_email() {
            alert("Please enter a valid email");
            document.getElementById("eid").style.borderBottomColor = 'red';
            //document.removeChild(iconElement)
        }

        else {

            document.getElementById("eid").style.borderBottomColor = 'green';
            var span = document.createElement('span')
            //span.style.marginLeft = "20%";
            // Create the <i> element
            // Create the <i> element
            var iconElement = document.createElement('i');
            // Set the class attribute
            iconElement.setAttribute('class', 'fa fa-check-circle-o');
            // Set the aria-hidden attribute
            iconElement.setAttribute('aria-hidden', 'true');
            // Apply styles using JavaScript
            iconElement.style.border = '2px solid white';
            iconElement.style.borderRadius = '100%';
            iconElement.style.background = 'white';
            iconElement.style.color = 'green';
            iconElement.style.fontSize = '30px';

            // Append the icon element to the document body, or any other desired parent element
            document.body.appendChild(iconElement);

            // Append the <i> element to the document body or any desired parent element
            span.appendChild(iconElement);
            var eid = document.getElementById('eid');
            eid.appendChild(span);
            storeUserSignInData()
        }
    }
</script>
```


5. Error Alerting and Highlighting:

- Null values and unrequired inputs are promptly alerted back to the user for correction.
- The location of errors is highlighted to users, facilitating quick identification and correction of erroneous inputs.

```
if (pwd.length === 0) {
    alert("You must enter a valid password");
    document.getElementById('pwd').style.borderColor = "red"
    password.focus();
    return false;
}
```

Implementing thorough form validation provides numerous advantages to both the website and its visitors through:

- Data Integrity: Form validation mechanisms enhance data integrity by preventing the submission of incomplete or erroneous data.

- User Experience: Clear error messages and highlighted error locations improve user experience by reducing frustration and streamlining the correction process.
- Efficiency: Utilizing JavaScript events for validation ensures real-time feedback to users, promoting efficient data entry and validation processes.

HTML Session Storage Implementation

The website stores data temporarily using HTML session storage, which allows for the creation of dynamic content and data access across various website areas. Maintaining data continuity across a user's browsing session is the primary purpose of session storage, which ensures that data is saved and accessible for the duration of the user's visit to the website.

Implementation Details

1. Session Storage Usage:

   - HTML session storage is employed to store data temporarily within the user's browser session. This ensures that the data is available and accessible across different parts of the website without the need for server-side storage or permanent data retention.

```
if (typeof (Storage) !== "undefined") {
    if (sessionStorage.clickcount) {
        sessionStorage.clickcount = Number(sessionStorage.clickcount) + 1;
        localStorage.setItem("user(" + sessionStorage.clickcount + ")", JSON.stringify(userArray));
        var user = JSON.parse(localStorage.getItem('user(' + sessionStorage.clickcount + ')'));
        var fname = user.name;
        var email = user.email;
        var phone = user.phone;
        var password = user.password;
        var address = user.address;
        var date = user.date;
        var accnt = user.accnt;
        var gender = user.gender;
        var post = user.post;

        createTable(fname, phone, email, address, post, date, gender, accnt, password);
```

2. Data Storage Mechanism:

   - Data is stored incrementally in session storage, with each data entry assigned a unique identifier, typically in the form of an incremental counter. This approach allows for the differentiation of multiple data entries and facilitates efficient retrieval of stored data.

3. Code Organization:

- A `**localStorage.js**` file contains all the code related to session storage, including functions for storing and retrieving data from session storage. This centralized approach to code organization promotes maintainability and ease of future updates.

4. Integration with Form Validation:

- Session storage functions are called upon successful validation of form fields, ensuring that only valid data is stored in session storage. This integration enhances data integrity and reliability by preventing the storage of invalid or unrequired data.

5. Data Visualization in Table Format:

- Retrieved data from session storage is passed to a `**table.js**` file, where it is visualized in tabular format using HTML tables. This visualization enables users to easily view and interact with stored data without the need for page reloads or additional server requests.

```
function createTable(fname, phone, email, address,post, date,gender, accnt, password) {
    // Create table body if it doesn't exist
    var tbl = document.getElementById("kt-datatable");
    var tbody = tbl.querySelector("tbody");
    if (!tbody) {
        tbody = document.createElement("tbody");
        tbl.appendChild(tbody);
    }

    // Create row
    const row = document.createElement("tr");

    // Create cells and text nodes
    const cellsData = [fname, phone, email, address,post, date, gender,accnt, password];
    cellsData.forEach(data => {
        const cell = document.createElement("td");
        const cellText = document.createTextNode(data);
        cell.appendChild(cellText);
        row.appendChild(cell);
    });

    // Append row to the table body
    tbody.appendChild(row);

    // Get all the rows of the table
    var tableRows = document.querySelectorAll('#kt-datatable tr');

    // Add event listeners to each row
    tableRows.forEach(function (row) {
        // Event listener for mouse entering the row
        row.addEventListener('mouseenter', function () {
            row.style.backgroundColor = 'green'; // Change background color on hover
        });

        // Event listener for mouse leaving the row
        row.addEventListener('mouseleave', function () {
            row.style.backgroundColor = 'lightcyan'; // Reset background color when mouse leaves
        });
    });
```

6. User Feedback:

- Upon successful storage of data, a popup box is displayed using jquery swal to notify the user that their data has been saved. This feedback mechanism enhances user experience by providing clear and immediate feedback regarding the outcome of their actions. This swal box is initialized in the **formValidation.js** file

```
var timeout = setTimeout(function () {
    console.log("work done");

    // When the work is done, reset the button to original state
    btn.innerHTML = 'Create Account';

    // After the animation finishes, show SweetAlert
    swal({
        title: "Account Created",
        text: "Please Check the table below to confirm your account",
        icon: "success",
        button: "OK",
    });
    storeData();
    document.getElementById("signup_form").reset();
}, 3000);

// Clear the timeout if the button is clicked before the animation finishes
btn.addEventListener('click', function () {
    clearTimeout(timeout);
    btn.innerHTML = 'Create Account'; // Reset button text
});
```

The implementation of HTML session storage within the website represents a significant enhancement to data management and user experience. By leveraging session storage, the website can store and retrieve data dynamically, enabling personalized interactions and streamlined data access.

JavaScript Event Handling in Website Development

An essential component of web development is JavaScript event handling, which lets programmers design dynamic and interactive user interfaces. I have made considerable use of a variety of JavaScript events throughout the creation of my websites in order to improve functionality, verify user input, and offer real-time feedback.

1. onClick() Event

- Applied the onClick event to submit buttons to display loading animation using the class loader in the inner HTML, with a timeout function before data visualization. It is initialized in the **FormValidation.js page**
- This event is triggered when a user clicks on an element. It is commonly used to initiate actions such as form submission, display modal dialogs, or implement interactive features like dropdown menus.

```
// alert("All fields are valid");
// this.innerHTML = "<div class='loader'></div>";
var btn = document.getElementById("user_register");
btn.innerHTML = "<div class='loader'></div>";

// Start a timer for the animation to finish
var timeout = setTimeout(function () {
    console.log("work done");

    // When the work is done, reset the button to original state
    btn.innerHTML = 'Create Account';

    // After the animation finishes, show SweetAlert
    swal({
        title: "Account Created",
        text: "Please Check the table below to confirm your account",
        icon: "success",
        button: "OK",
    });
    storeData();
    document.getElementById("signup_form").reset();
}, 3000);

// Clear the timeout if the button is clicked before the animation finishes
btn.addEventListener('click', function () {
    clearTimeout(timeout);
    btn.innerHTML = 'Create Account'; // Reset button text
});
```

2. onFocus() Event

- The onFocus() event is triggered when an element gains focus. It is often used to provide visual feedback to users by highlighting active or selected elements and enhancing accessibility by focusing on form fields or interactive elements for keyboard navigation. It is initialized in the **FormValidation.js page**

```
document.getElementById('addr').style.borderColor = "red"
address.focus();
return false;
```

3. onSubmit() Event

- Utilized the onSubmit event on form tags to validate form data before submission, directing to formValidations.js for validation.
- This event is triggered when a form is submitted. It is used to validate form data before submission, process form submissions by sending data to server-side scripts, and display confirmation messages or redirect users to specific pages based on form submission outcomes. It is initialized in all the Html for **pages**

```
<div class="form-box">
    <form name="signup_form" id="signup_form" class="form-box" onsubmit="return validate_form(event);">
        <table>
            <tr>
```

4. onKeyup() Event

- The onKeyup() event is triggered when a key is released after being pressed. It is commonly used to implement real-time validation of input fields such as passwords or search queries, providing immediate feedback to users regarding input validity or formatting requirements. Implemented on the usersignup.html page using the onKeyup event to display a message box with password validation criteria. It is initialized in the **usersignup.html page**

```
// When the user starts to type something inside the password field
myInput.onkeyup = function () {
    // Validate lowercase letters
    var lowerCaseLetters = /[a-z]/g;
    if (myInput.value.match(lowerCaseLetters)) {
        letter.classList.remove("invalid");
        letter.classList.add("valid");
    } else {
        letter.classList.remove("valid");
        letter.classList.add("invalid");
    }

    // Validate capital letters
    var upperCaseLetters = /[A-Z]/g;
    if (myInput.value.match(upperCaseLetters)) {
        capital.classList.remove("invalid");
        capital.classList.add("valid");
    } else {
```

5. onBlur() Event

- Assigned the onBlur event to individual input fields in login pages for user and admin, facilitating real-time validation of email fields as users finish typing. It is initialized in the **usersignin.html and adminsignin.html pages**
- This event is triggered when an element loses focus. It is used to conduct validation checks on form fields or input elements as users finish typing, displaying error messages or validation prompts in real-time to guide users towards correct input.

```
<div class="inputbox" id="eid">
    <ion-icon name="mail-outline"></ion-icon>
    <input type="text" id="emailid" name="emailid" onblur="return check_email();">
    <label for="">Email</label>
</div>
```