Project Report

Project Title:

Tour and Travel Agency Website.

Name: Abdul Raheman.

Inter-ID: MST04-0048.

Submission Date: (02/1/2025)

Table of Contents:

1. Introduction:

This project involves the creation of a fully functional web application for a travel agency. The primary objective is to provide users with an intuitive platform to browse travel packages, manage bookings, and interact with dynamic content. The website aims to simplify the process of travel planning while offering a visually appealing and user-friendly interface. This project highlights the power of web technologies in delivering a seamless user experience.

---

2. Project Overview:

The Tour and Travel Agency Website provides the following key features:

- A visually engaging homepage with dynamic video and navigation elements.
- A packages section showcasing various travel destinations with pricing, images, and detailed descriptions.
- A booking system that dynamically calculates total costs based on user input.
- A registration and login system for personalized user access.
- Responsive design to ensure usability across different devices and screen sizes.

---

3. Technologies Used:

- Languages: HTML, CSS, JavaScript
- Libraries/Frameworks: Swiper.js for creating sliders
- Tools:

  Browser DOM manipulation for dynamic content updates

  Built-in JavaScript methods for form validation and data handling -

  CSS for responsive and visually appealing layouts

---

4. System Architecture:

The system architecture is designed for simplicity and efficiency:

- **Frontend:** Utilizes HTML, CSS, and JavaScript to manage user interactions and display content dynamically.
- **Dynamic Content Management**: JavaScript functions update the webpage based on user actions, such as selecting a travel package or changing guest counts.
- **Backend Simulation**: User data and session details are temporarily stored in-memory, simulating abackend for demonstration purposes.

System Workflow:

1. Users browse packages on the homepage.
2. Users proceed to the booking page, where dynamic updates occur based on selections.
3. Registered users can log in for a personalized experience.

5.Implementation Details:

Overview

The JavaScript code in this file handles the functionality of a dual-form system consisting of a **login form** and a **registration form**. It includes dynamic form switching, validation for user inputs, and a mock registration and login process using basic logic. The script manipulates the DOM (Document Object Model) to provide a seamless and interactive user experience.

Key Features and Functionality

1. **Form Switching Logic**
    - Two functions (**switchToRegistration** and **switchToLogin**) are implemented to toggle the visibility of the login and registration forms.
    - The heading text dynamically updates to reflect the current form ("**Login Page"** or **"Registration Page").**
2. **Input Validation**
    - Both the registration and login forms validate user inputs to ensure correctness.
    - Registration validation checks:
        - Username: Minimum 4 characters.
        - Email: Must match a valid email pattern.
        - Password: Minimum 6 characters and must contain at least one number.
    - Login validation ensures the email and password match the stored user data.
3. **Error Messaging**
    - Error messages are displayed to users in case of invalid inputs during registration or incorrect credentials during login.
    - Errors are shown dynamically within designated elements.
4. **Clear Input Fields**
    - After successful registration or login, all input fields in the respective form are cleared for better user experience.
5. **Mock User Storage**
    - The user data (**username, email, and password**) is stored in a variable (**registeredUser**) to simulate a real registration process.
    - No backend or persistent storage is used, making this a client-side-only implementation.
6. **Redirection**
    - On successful login, the user is redirected to a home page (**index.html**) using **window. location. href**.

Detailed Explanation of JavaScript Code

1. Switching Between Forms

Function: switchToRegistration:

```
function switchToRegistration() {
  document.getElementById("loginForm").classList.remove("active");
  document.getElementById("registrationForm").classList.add("active");
  document.getElementById("formHeading").textContent = "Registration Page";
}
```

- Hides the login form by removing the **active** class.
- Displays the registration form by adding the **active** class.
- Updates the heading to "**Registration Page**."

Function: switchToLogin:

```
function switchToLogin() {
  document.getElementById("registrationForm").classList.remove("active");
  document.getElementById("loginForm").classList.add("active");
  document.getElementById("formHeading").textContent = "Login Page";
}
```

- Performs the reverse of **switchToRegistration.**
- Updates the heading to "**Login Page**."

2. Input Validation and Registration:

Function: handleRegistration:

```
function handleRegistration() {
  const username = document.getElementById("regUsername").value.trim();
  const email = document.getElementById("regEmail").value.trim();
  const password = document.getElementById("regPassword").value.trim();
  const errorElement = document.getElementById("registrationError");

  if (username.length < 4) {
    errorElement.textContent = "Username must be at least 4 characters.";
    return;
  }

  const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
  if (!emailRegex.test(email)) {
    errorElement.textContent = "Please enter a valid email address.";
    return;
  }

  if (password.length < 6 || !/\d/.test(password)) {
    errorElement.textContent = "Password must be at least 6 characters and contain a number.";
    return;
  }

  errorElement.textContent = "";
  registeredUser = { username, email, password };
  alert("Registration successful!");
  clearFormInputs("registrationForm");
  switchToLogin(); // Switch to login form
}
```

- **Username Validation**: Ensures a minimum length of 4 characters.
- **Email Validation**: Uses a regex pattern to verify the email format.
- **Password Validation**: Ensures a minimum length of 6 characters and the inclusion of at least one number.
- On successful validation:
  - Stores the user details in the **registeredUser** variable.
  - Displays a success alert.
  - Clears the registration form inputs.
  - Switches to the login form.

3. Login Validation

Function: `handleLogin`:

```
function handleLogin() {
  const email = document.getElementById("loginEmail").value.trim();
  const password = document.getElementById("loginPassword").value.trim();
  const errorElement = document.getElementById("loginError");

  if (!registeredUser) {
    errorElement.textContent = "No registered user found. Please register first.";
    return;
  }

  if (registeredUser.email !== email || registeredUser.password !== password) {
    errorElement.textContent = "Invalid email or password.";
    return;
  }

  errorElement.textContent = "";
  alert("Login successful!");
  clearFormInputs("loginForm");
  window.location.href = "index.html"; // Redirect to home page
}
```

- Validates:
  - **Email and Password**: Matches the input with the data stored in **registeredUser.**
  - If no user is registered, displays an error message.
  - If credentials do not match, displays an "Invalid email or password" message.
- On successful login:
  - Displays a success alert.
  - Clears the login form inputs.
  - Redirects the user to the homepage.

4. Clear Form Inputs

Function: clearFormInputs:

```
function clearFormInputs(formId) {
  const form = document.getElementById(formId);
  form.querySelectorAll("input").forEach(input => {
    input.value = "";
  });
}
```

- Resets all input fields in a specified form by setting their values to an empty string.
- Called after successful registration and login.

Key Concepts Used in the Code:

1. **DOM Manipulation**
   - Adding/removing classes and updating text content dynamically.
2. **Event Handling**
   - Button clicks are handled using **onclick** attributes to trigger the corresponding JavaScript functions.
3. **Regex for Validation**
   - Ensures input fields meet specific formatting rules (e.g., **email validation**).
4. **Client-Side Data Storage**
   - Stores registered user details in a JavaScript object (**registeredUser**).
5. **Redirection**
   - Simulates a real-world flow by redirecting to a homepage upon successful login.

5.2 Package selection and booking file:

checkout.js file

Purpose of the Code:

The code aims to create an interactive and dynamic checkout page that allows users to:

- Select a package based on details passed via URL parameters.
- Dynamically update pricing and tax calculations based on the number of guests.
- Validate user input during form submission.
- Provide a responsive and seamless user experience on the checkout page.

Detailed Explanation

1. Retrieving URL Parameters:

```
const urlParams = new URLSearchParams(window.location.search);
const packageName = urlParams.get('package');
const packagePrice = parseFloat(urlParams.get('price'));
```

- This section extracts query parameters (***package, price***) from the URL using ***URLSearchParams.***
- The package parameter is assigned to ***packageName***.
- The price parameter is converted to a number using ***parseFloat*** and stored in ***packagePrice***.

The code extracts query parameters like `package`, `price` from the URL

to dynamically update the checkout page

2. Updating Package Name and Price:

```
if (document.getElementById('package-name')) {
    document.querySelector('.package').textContent = packageName;
    document.getElementById('price').textContent = `$${packagePrice}`;
}
```

- Checks if an element with ID ***package-name*** exists on the page.
- Updates the content of the. package and #***price*** elements with the package name and formatted price, respectively.

The `package` and `price` are displayed on the checkout page dynamically.

3. Handling Guest Count and Pricing:

```
let numGuests = 1;
const pricePerGuest = packagePrice; // Base price for the package
```

- Initializes the number of guests (***numGuests***) to 1 and sets the ***pricePerGues***t variable to the base price of the package.

Updating Price Dynamically:

```
function updatePrice() {
    const taxRate = 0.1; // 10% tax
    const totalPrice = pricePerGuest * numGuests;
    const taxAmount = totalPrice * taxRate;

    document.querySelector('.total-price').textContent = `Total:
    document.querySelector('.tax-info').textContent = `Tax: $${t
    document.getElementById('num-guests').textContent = numGuest
}
```

- Calculates the total price and tax amount based on the number of guests.
- Updates the following elements dynamically:
  - ***. total-price:*** Displays the total price including tax.
  - ***. tax-info:*** Displays the calculated tax amount.
  - ***#num-guests:*** Displays the current number of guests.

- **Increasing and Decreasing Guest Count;**

```
// Increase the number of guests
document.getElementById('increase-btn')?.addEventListener('click', function() {
    numGuests++;
    updatePrice();
});

// Decrease the number of guests
document.getElementById('decrease-btn')?.addEventListener('click', function() {
    if (numGuests > 1) {
        numGuests--;
        updatePrice();
    }
});
```

- Adds event listeners to ***increase-btn*** and ***decrease-btn*** buttons to modify numGuests and recalculate the total price using ***updatePrice ().***

Users can increase or decrease the number of guests, and the total price and tax are calculated accordingly.

4. Form Submission Handling:

```
document.getElementById('checkout-form')?.addEventListener('submit', function(event) {
    event.preventDefault();

    const name = document.getElementById('name')?.value;
    const email = document.getElementById('email')?.value;
    const phone = document.getElementById('phone')?.value;
    const cardNumber = document.getElementById('card-number')?.value;
    const expiryDate = document.getElementById('expiry-date')?.value;
    const cvv = document.getElementById('cvv')?.value;

    if (name && email && phone && cardNumber && expiryDate && cvv) {
        alert('Booking Confirmed! Redirecting to home page.');
        window.location.href = 'index.html'; // Redirect to package page after booking
    } else {
        alert('Please fill all the fields.');
    }
});
```

- Prevents default form submission behavior.
- Retrieves user input from form fields (***name, email***, etc.).
- Validates that all fields are filled:
    - If valid: Displays a success message and redirects to the home page.
    - If invalid: Displays an alert asking users to fill in all required fields

Form validation ensures that all required fields are filled before submission.

5. Window onload Initialization:

```
window.onload = function() {
    // Retrieve URL parameters
    const urlParams = new URLSearchParams(window.location.search);

    // Get the package details from URL
    const packageName = urlParams.get('package');
    const packagePrice = urlParams.get('price');
    const packageImage = urlParams.get('image');

    // Check if the image URL exists and update the content dynamically
    if (packageImage) {
        // Update the image source dynamically
        document.getElementById('checkout-package-img').src = packageImage;
    }

    // Update the text content dynamically
    if (packageName) {
        document.getElementById('checkout-package-name').textContent = packageName;
    }

    if (packagePrice) {
        document.getElementById('checkout-package-price').textContent = '$' + packagePrice;
    }

    // Optional: You can also add a description based on the package
    document.getElementById('checkout-package-description').textContent = "Relax on the bea
```

- Executes when the page loads.
- Retrieves URL parameters and dynamically updates the following elements:
    - #***checkout-package-img***: Updates the package image source.
    - #***checkout-package-name***: Updates the package name.
    - #***checkout-package-price***: Updates the package price.
    - #***checkout-package-description***: Sets a default package description.

On page load, the code dynamically updates elements like images and descriptions based on URL parameters.

5.3 Dynamic UI Updates

scripts.js

1. Variable Declarations

Search Button and Search Bar:

```
let searchBtn = document.querySelector('#search-btn');
let searchBar = document.querySelector('.search-bar-container');
```

- **Purpose**: These variables store references to the search button (#***search-btn***) and search bar container (***. search-bar-container***) in the DOM.
- **Functionality**: These elements enable toggling the visibility of the search bar when the search button is clicked.

Login Form Elements:

```
let formBtn = document.querySelector('#login-btn');
let loginForm = document.querySelector('.login-form-container');
let formClose = document.querySelector('#form-close');
```

- **Purpose**: These variables manage the login form.
  - **formBtn**: Opens the login form.
  - **loginForm**: Represents the login form container.
  - **formClose**: Closes the login form.

Navbar and Menu Button:

```
let menu = document.querySelector('#menu-bar');
let navbar = document.querySelector('.navbar');
```

- **Purpose**: These variables handle the navigation menu.
  - **menu**: Triggers the visibility of the navbar.
  - **navbar**: Represents the navigation bar container.

Video Buttons:

```
let videoBtn = document.querySelectorAll('.vid-btn');
```

- **Purpose**: Collects all video buttons **(. vid-btn**) into a Node List.
- **Functionality**: Allows users to select and play videos dynamically.

---

2. Scroll Event Handler:

```
window.onscroll = () => {
 searchBtn.classList.remove('fa-times');
 searchBar.classList.remove('active');
 menu.classList.remove('fa-times');
 navbar.classList.remove('active');
 loginForm.classList.remove('active');
};
```

- **Purpose**: Resets all active/toggled states (search bar, menu, navbar, login form) when the user scrolls.
- **How It Works**:
  - **onscroll** triggers whenever the user scrolls the page.
  - Classes like **fa-times** (for toggling icons) and **active** (for visibility) are removed to ensure a clean UI.

---

3. Menu Button Click Event

```
menu.addEventListener('click', () => {
 menu.classList.toggle('fa-times');
 navbar.classList.toggle('active');
});
```

- **Purpose**: Toggles the visibility of the navigation menu.
- **How It Works**:
  - **addEventListener** attaches a click event to the **menu** button.
  - **classList.toggle** adds or removes the **fa-times** and **active** classes to change the menu icon and display the navbar.

---

4. Search Button Click Event:

```
searchBtn.addEventListener('click', () => {
 searchBtn.classList.toggle('fa-times');
 searchBar.classList.toggle('active');
});
```

- **Purpose**: Toggles the visibility of the search bar.
- **How It Works**:
  - Similar to the menu button, toggling the **fa-times** and active classes allows showing or hiding the search bar.

5. Login Form Open and Close:

```
formBtn.addEventListener('click', () => {
 loginForm.classList.add('active');
});

formClose.addEventListener('click', () => {
 loginForm.classList.remove('active');
});
```

- **Purpose**: Opens and closes the login form.
- **How It Works**:
  - Clicking **formBtn** adds the **active** class to display the login form.
  - Clicking **formClose** removes the **active** class to hide the form.

---

6. Video Button Functionality:

```
videoBtn.forEach(btn =>{
 btn.addEventListener('click', ()=>{
  document.querySelector('.controls .active').classList.remove('active');
  btn.classList.add('active');
  let src = btn.getAttribute('data-src');
  document.querySelector('#video-slider').src= src;
 });
});
```

- **Purpose**: Allows users to switch and play videos interactively.
- **How It Works**:
  - Loops through all **videoBtn** elements.
  - Removes the **active** class from the currently active button and assigns it to the clicked button.
  - Updates the **src** attribute of the video slider **(#video-slider**) to play the new video.

---

7. Swiper Slider Initialization:

```
var swiper = new Swiper(".review-slider", {
 spaceBetween: 20,
 loop: true,
 autoplay: {
  delay: 2500,
  disableOnInteraction: false,
 },
 breakpoints: {
  640: { slidesPerView: 1 },
  768: { slidesPerView: 2 },
  1024: { slidesPerView: 3 },
 },
});
```

- **Purpose**: Creates a Swiper slider for customer reviews.
- **How It Works**:
  - **spaceBetween**: Sets 20px spacing between slides.
  - **loop**: Enables infinite scrolling.
  - **autoplay**: Slides change automatically every 2500ms.
  - **breakpoints**: Adjusts the number of slides based on screen size.

Brand Slider:

```
var swiper = new Swiper(".brand-slider", {
 spaceBetween: 20,
 loop: true,
 autoplay: {
  delay: 2500,
  disableOnInteraction: false,
 },
 breakpoints: {
  450: { slidesPerView: 2 },
  768: { slidesPerView: 3 },
  991: { slidesPerView: 4 },
  1200: { slidesPerView: 5 },
 },
});
```

- **Purpose**: Similar to the review slider but tailored for brand logos.
- **How It Works**:
  - Adjusts the number of visible slides based on screen width, ranging from 2 to 5 slides.

Key Features of the Code

1. **Interactive UI**: Enables toggling menus, search bars, and forms for a dynamic user experience.
2. **Responsive Design**: Swiper sliders adjust based on screen size, improving usability across devices.
3. **Dynamic Video Switching**: Users can easily select and play different videos.
4. **User-Friendly Navigation**: Collapsible menus enhance navigation on mobile devices.

Suggestions for Improvement

1. **Error Handling**: Add checks to ensure elements exist before accessing them to prevent runtime errors.
2. **Accessibility**: Include ARIA attributes for better screen reader support.
3. **Performance Optimization**: Use debouncing for the scroll event handler to improve performance.
4. **Code Organization**: Use modular functions or ES6 classes for better maintainability.

6. Challenges Faced:

- **Dynamic Content Management**: Implementing real-time updates for booking calculations requiredcareful handling of JavaScript events and DOM manipulation.

- **Responsive Design**: Ensuring compatibility across multiple devices demanded extensive testing and CSS adjustments.

- **User Input Validation**: Designing robust validation for registration and login forms to handle edgecases effectively.

- **Resolution Strategies**:
  - Used modular JavaScript functions to streamline dynamic updates.
  - Adopted media queries and testing tools to refine responsiveness.
  - Applied regex patterns for comprehensive input validation.

---

7. Future Enhancements:

- **Backend Integration**: Connect the platform to a database for persistent user data storage.

- **Advanced Filtering**: Allow users to search packages by date, destination, and price range.

- **Payment Gateway**: Integrate a secure payment system to process bookings.

- **User Dashboard**: Provide users with a dashboard to view and manage their bookings.

- These enhancements aim to transform the platform into a comprehensive travel management solution.

---

8. Conclusion:

The Tour and Travel Agency Website demonstrates the potential of web technologies in building interactive and functional platforms. By combining responsive design, dynamic content updates, and intuitive user interfaces, the project delivers a seamless experience for travel enthusiasts. Future expansions and integrations will further enhance its utility and reach.

---

9. References:

- Swiper.js documentation
- MDN Web Docs (HTML, CSS, JavaScript)
- W3Schools (JavaScript form validation)

Thank you.