```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href="style.css">
    <title>Rest and Spread Operator in JavaScript</title>
</head>

<body>

    <h1>Rest & Spread Operators - Website Example</h1>

    <h1>Rest Operator is used to combine multiple values into an array</h1>

    <h2>1. Fetched Users (Password Removed) Using REST Operator</h2>

    <div id="user-list"></div>

    <h1>Spread Operator is used to split an array into multiple values</h1>

    <h2>2. Website Settings Using Spread Operator</h2>

    <div id="settings"></div>

    <h2>3. Dynamic Form Inputs Using Rest Operator</h2>

    <form id="userForm">

        <input type="text" placeholder="First Name" required> <br><br>

        <input type="text" placeholder="Last Name" required> <br><br>

        <div id="extra-fields"></div>

        <button type="button" id="addField">Add Extra Field</button>
```

```
38
39          <br> <br>
40
41          <button type="submit">Submit</button>
42
43      </form>
44
45      <script src="script.js"></script>
46
47  </body>
48
49  </html>
```

```
/* ----------------------------------------------------------------

1. Example: Fetch API Data & Use REST Operator to Remove Password

---------------------------------------------------------------- */

// Fake array of users (like an API response)

const fakeUsers = [

    { id: 1, name: "Heet", email: "heet@example.com", password: "secret123" },

    { id: 2, name: "Aisha", email: "aisha@example.com", password: "pass456" }
];

/*

We want to display user data, BUT for security reasons, we should never display the password.

👉 Trick: Use object destructuring with the REST operator (...) to remove the `password` field while keeping the other properties.

*/

const safeUsers = fakeUsers.map((user) => {

    // Extract `password`, then collect the remaining properties into safeUser

    const { password, ...safeUser } = user;

    return safeUser; // return the "safe" user object without password
});

// Select the <div id="user-list"> from HTML to display users

const userList = document.getElementById("user-list");

// Loop through safeUsers and create HTML elements for each
```

```
38
39 safeUsers.forEach((user) => {
40
41     const div = document.createElement("div"); // create a new <div>
42
43     div.className = "user-card"; // add a class for styling
44
45     // Fill the div with user info (name + email only, password removed)
46
47     div.innerHTML = `<strong>Name:</strong> ${user.name}<br> <strong>Email:</strong> ${user.email}`;
48
49     // Finally, add this div inside the #user-list container
50
51     userList.appendChild(div);
52 });
53
54
55 /* ----------------------------------------------------------------
56
57 2. Example: Merge Default Settings with User Settings (Spread Operator)
58
59 ---------------------------------------------------------------- */
60
61 // Default website settings
62
63 const defaultSettings = { theme: "light", notifications: true, language: "en" };
64
65 // Suppose the user changed some settings
66
67 const userSettings = { theme: "dark", language: "fr" };
68
69 /*
70
71 👉 Spread operator (...) allows us to merge objects easily. If properties overlap, later values overwrite earlier ones. Here,
   userSettings will overwrite defaultSettings.
72
73 */
74
```

```
const mergedSettings = { ...defaultSettings, ...userSettings };

// Show the merged settings inside the <div id="settings"> in formatted JSON

document.getElementById("settings").textContent = JSON.stringify(mergedSettings, null, 2);

/*

⚡ JSON.stringify(obj, null, 2)

   - Converts object to JSON string

   - `null` means no special filtering

   - `2` means add indentation (pretty printing with 2 spaces)
*/


/* ----------------------------------------------------------------

3. Example: Create Dynamic Form Inputs with Rest Operator

---------------------------------------------------------------- */

// Get the form and the extra-fields container from HTML

const form = document.getElementById("userForm");

const extraFieldsContainer = document.getElementById("extra-fields");

// Add functionality: when "Add Extra Field" button is clicked

document.getElementById("addField").addEventListener("click", () => {

    // Create a new <input>

    const input = document.createElement("input");

```

```
    input.type = "text"; // input type will be "text"

    input.placeholder = "Extra Field"; // placeholder text

    input.required = true; // make it mandatory

    // Append the new input inside the "extra-fields" section of the form

    extraFieldsContainer.appendChild(input);

    // Optional: add line break after each new input for clarity

    extraFieldsContainer.appendChild(document.createElement("br"));
});

// When the form is submitted

form.addEventListener("submit", (event) => {

    event.preventDefault(); // stop page from refreshing

    // Get ALL inputs inside the form (including extra ones)

    const inputs = Array.from(form.querySelectorAll("input"));

    /*

    Use destructuring with rest operator:

    - First two inputs → firstName, lastName

    - The remaining inputs → extraFields (as an array)

    */

    const [firstName, lastName, ...extraFields] = inputs.map((input) => input.value);

    // Show the collected data in an alert box
```

```
151
152     alert(`First Name: ${firstName}\nLast Name: ${lastName}\nExtra Fields:\n${extraFields.join("\n")}`);
153
154     // Reset the form (clears inputs)
155
156     form.reset();
157 });
```