Lab 2: Declarative Bindings and Data Binding Basics

Lab 2.1: Basic Declarative Binding with `data-bind`

The following example demonstrates a simple Knockout application that uses multiple binding types to create a responsive user interface. Inline comments explain each part of the implementation.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Knockout Declarative Binding Example</title>
  <!-- Load Knockout.js from CDN -->
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/knockout/3.5.1/knockout-
min.js"></script>
</head>
<body>

  <!-- View: HTML with declarative data-bind attributes -->
  <div id="user-profile">
    <h2>User Profile</h2>

    <!-- One-way binding: Display user name -->
    <p><strong>Name:</strong> <span data-bind="text: userName"></span></p>

    <!-- Two-way binding: Editable input field -->
    <p>
      <label>Update Name:
        <input type="text" data-bind="value: userName" />
      </label>
    </p>

    <!-- Conditional visibility -->
    <p data-bind="visible: showEmail">
      <strong>Email:</strong> <span data-bind="text: userEmail"></span>
    </p>

    <!-- Toggle visibility with a button -->
    <button data-bind="click: toggleEmail">Toggle Email</button>

    <!-- Dynamic CSS class based on status -->
    <div data-bind="css: { online: isOnline(), away: !isOnline() }">
      Status: <span data-bind="text: isOnline() ? 'Online' : 'Away'"></span>
    </div>
```

```
    <!-- Click binding with method call -->
    <button data-bind="click: greetUser">Say Hello</button>
  </div>

  <script>
    // ViewModel: JavaScript representation of the UI state
    function UserProfileViewModel() {
      // Observable properties – automatically update the UI when changed
      this.userName = ko.observable("Alice Johnson");
      this.userEmail = ko.observable("alice.johnson@example.com");
      this.showEmail = ko.observable(true);
      this.isOnline = ko.observable(true);

      // Method bound to click event
      this.greetUser = function() {
        alert("Hello, " + this.userName() + "!");
      };

      // Toggle email visibility
      this.toggleEmail = function() {
        this.showEmail(!this.showEmail());
      }.bind(this); // Ensure 'this' refers to the view model
    }

    // Apply bindings after DOM is loaded
    document.addEventListener("DOMContentLoaded", function() {
      ko.applyBindings(new UserProfileViewModel(),
document.getElementById("user-profile"));
    });
  </script>
</body>
</html>
```

Explanation of Key Concepts

Binding	Purpose	Type
`data-bind="text: userName"`	Displays the value of `userName`	One-way
`data-bind="value: userName"`	Links input field to `userName` (two-way sync)	Two-way
`data-bind="visible: showEmail"`	Shows/hides element based on boolean	One-way
`data-bind="click: toggleEmail"`	Calls `toggleEmail()` when clicked	Event
`data-bind="css: { online: isOnline() }"`	Applies CSS class dynamically	One-way
`ko.observable()`	Makes property reactive —	Core KO feature

Binding	Purpose	Type
	changes trigger UI updates	
`ko.applyBindings()`	Activates Knockout on a DOM element	Initialization

Lab 2.2: Build a Dynamic Task Manager

Objective

Create a simple **Task Manager** interface using Knockout's declarative binding features. You will build a view model and bind it to an HTML page to allow users to: - View and edit a task title - Mark a task as complete using a checkbox - Show/hide task details based on user action - Change visual style (e.g., strikethrough) when complete - Trigger alerts via button clicks

Files to Create

You will create the following files: - `task-manager.html` – The HTML view with `data-bind` attributes - `task-manager.js` – The Knockout view model

---

Step-by-Step Instructions

Step 1: Create `task-manager.html`

Create a new file named `task-manager.html`. This file will contain the user interface and references to Knockout.

```
<!-- File: task-manager.html -->
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Knockout Task Manager</title>
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/knockout/3.5.1/knockout-
min.js"></script>
  <style>
    .completed { text-decoration: line-through; color: gray; }
    .details { margin-top: 10px; padding: 10px; background: #f0f8ff; border:
1px solid #ccc; }
    button { margin-top: 5px; }
  </style>
</head>
<body>

  <div id="task-app">
```

```
    <h2>Task Manager</h2>

    <!-- 1. Editable task title -->
    <p>
      <label>Task Title:
        <input type="text" data-bind="value: title" size="40" />
      </label>
    </p>

    <!-- 2. Checkbox to mark task as complete -->
    <p>
      <label>
        <input type="checkbox" data-bind="checked: isCompleted" />
        Mark as Complete
      </label>
    </p>

    <!-- 3. Button to toggle task details -->
    <button data-bind="click: toggleDetails">Toggle Details</button>

    <!-- 4. Conditionally visible details section -->
    <div data-bind="visible: showDetails" class="details">
      <h3>Task Details</h3>
      <p>Status:
        <span data-bind="text: isCompleted() ? 'Completed' : 'Pending',
                          css: { completed: isCompleted }"></span>
      </p>
      <p>Created: <span data-bind="text: createdAt"></span></p>
    </div>

    <!-- 5. Dynamic class on title based on completion -->
    <p data-bind="css: { completed: isCompleted }">
      Current Task: <strong data-bind="text: title"></strong>
    </p>

    <!-- 6. Button to clear task -->
    <button data-bind="click: clearTask">Clear Task</button>
  </div>

  <!-- Include view model script -->
  <script src="task-manager.js"></script>
</body>
</html>
```

Explanation: - `value: title` enables two-way editing of the task title. - `checked: isCompleted` syncs the checkbox state with the view model. - `visible: showDetails` shows/hides the details block. - `css: { completed: isCompleted`

} applies strikethrough when complete. - `click: toggleDetails` and `click: clearTask` bind to view model methods.

Step 2: Create `task-manager.js`

Create a file named `task-manager.js`. This will define the view model.

```
// File: task-manager.js

function TaskViewModel() {
  // Observable properties
  this.title = ko.observable("Write documentation");
  this.isCompleted = ko.observable(false);
  this.showDetails = ko.observable(false);

  // Static property (not observable)
  this.createdAt = new Date().toLocaleString();

  // Toggle visibility of details
  this.toggleDetails = function() {
    this.showDetails(!this.showDetails());
  }.bind(this);

  // Clear task and reset
  this.clearTask = function() {
    if (confirm("Are you sure you want to clear the task?")) {
      this.title("");
      this.isCompleted(false);
    }
  }.bind(this);
}

// Apply bindings when DOM is ready
document.addEventListener("DOMContentLoaded", function() {
  ko.applyBindings(new TaskViewModel(), document.getElementById("task-app"));
});
```

> **Explanation**: - All observables (`title`, `isCompleted`, `showDetails`) are reactive. - `bind(this)` ensures correct context inside event handlers. - `confirm()` adds user interaction before clearing data. - `DOMContentLoaded` ensures the DOM is ready before applying bindings.

Expected Output

When you open `task-manager.html` in a browser:

1. The input field displays: `Write documentation`
2. The checkbox is unchecked.
3. Clicking **“Toggle Details”** shows/hides the details block with:
    - Status: “Pending” (turns to “Completed” when checked)
    - Created timestamp
4. Checking the checkbox:
    - Applies strikethrough to the task title
    - Changes status text to “Completed”
5. Clicking **“Clear Task”** prompts a confirmation, then clears title and unchecks the box.

---

4. Complete Implementation

Below are the final, fully working files as a complete implementation of the lab.

`task-manager.html` (Final)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Knockout Task Manager</title>
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/knockout/3.5.1/knockout-
min.js"></script>
  <style>
    .completed { text-decoration: line-through; color: gray; }
    .details { margin-top: 10px; padding: 10px; background: #f0f8ff; border:
1px solid #ccc; }
    button { margin-top: 5px; }
  </style>
</head>
<body>

  <div id="task-app">
    <h2>Task Manager</h2>

    <p>
      <label>Task Title:
        <input type="text" data-bind="value: title" size="40" />
      </label>
    </p>

    <p>
```

```
      <label>
        <input type="checkbox" data-bind="checked: isCompleted" />
        Mark as Complete
      </label>
    </p>

    <button data-bind="click: toggleDetails">Toggle Details</button>

    <div data-bind="visible: showDetails" class="details">
      <h3>Task Details</h3>
      <p>Status:
        <span data-bind="text: isCompleted() ? 'Completed' : 'Pending',
                         css: { completed: isCompleted }"></span>
      </p>
      <p>Created: <span data-bind="text: createdAt"></span></p>
    </div>

    <p data-bind="css: { completed: isCompleted }">
      Current Task: <strong data-bind="text: title"></strong>
    </p>

    <button data-bind="click: clearTask">Clear Task</button>
  </div>

  <script src="task-manager.js"></script>
</body>
</html>
```

---

task-manager.js (Final)

```
function TaskViewModel() {
  this.title = ko.observable("Write documentation");
  this.isCompleted = ko.observable(false);
  this.showDetails = ko.observable(false);
  this.createdAt = new Date().toLocaleString();

  this.toggleDetails = function() {
    this.showDetails(!this.showDetails());
  }.bind(this);

  this.clearTask = function() {
    if (confirm("Are you sure you want to clear the task?")) {
      this.title("");
      this.isCompleted(false);
    }
  }.bind(this);
}
```

```
document.addEventListener("DOMContentLoaded", function() {
  ko.applyBindings(new TaskViewModel(), document.getElementById("task-app"));
});
```
