Lab 4: Templating with Knockout

Lab 4.1: Rendering a List of Products Using `foreach` and `template` Binding

This example demonstrates how to use Knockout's `template` binding with `foreach` to dynamically render a list of products. It also shows how to pass data context and use conditional rendering within the template.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Knockout Templating Example</title>
  <!-- Include Knockout.js -->
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/knockout/3.5.0/knockout-
min.js"></script>
</head>
<body>

  <!-- UI: Container for product list -->
  <div id="productApp">
    <h2>Product Catalog</h2>

    <!-- Template Binding: Render each product using a named template -->
    <ul data-bind="template: { name: 'productTemplate', foreach: products
}"></ul>
  </div>

  <!-- Named Template: Defined using a script tag -->
  <!-- Type "text/html" prevents browser from executing it as JS -->
  <script type="text/html" id="productTemplate">
    <li>
      <strong data-bind="text: name"></strong> -
      <span data-bind="text: '$' + price.toFixed(2)"></span>

      <!-- Conditional rendering: Show discount badge if onSale -->
      <!-- 'if' binding only renders content when condition is true -->
      <span data-bind="if: onSale" class="sale-badge">[On Sale!]</span>

      <!-- Display category only if available -->
      <em data-bind="if: category">
        (Category: <span data-bind="text: category"></span>)
      </em>
    </li>
```

```
  </script>

  <script>
    // ViewModel: Represents the data and logic
    function ProductViewModel() {
      var self = this;

      // Observable array of product objects
      self.products = ko.observableArray([
        { name: "Laptop", price: 999.99, onSale: true, category:
"Electronics" },
        { name: "Coffee Mug", price: 12.50, onSale: false, category:
"Kitchen" },
        { name: "Wireless Headphones", price: 199.99, onSale: true },
        { name: "Notebook", price: 5.99, onSale: false, category: "Office" }
      ]);
    }

    // Apply bindings after DOM is loaded
    document.addEventListener("DOMContentLoaded", function () {
      ko.applyBindings(new ProductViewModel(),
document.getElementById("productApp"));
    });
  </script>

  <style>
    .sale-badge {
      color: red;
      font-weight: bold;
      margin-left: 8px;
    }
    ul {
      list-style-type: none;
      padding: 0;
    }
    li {
      margin: 8px 0;
      padding: 6px;
      background-color: #f9f9f9;
      border: 1px solid #ddd;
      border-radius: 4px;
    }
  </style>
</body>
</html>
```

Explanation

- **`template` binding**: The `data-bind="template: { name: 'productTemplate', foreach: products }"` instructs Knockout to:

    - Use the template defined in the `<script type="text/html" id="productTemplate">`.
    - Repeat the template for each item in the `products` observable array.
- **Script Tag Template**: Using `<script type="text/html">` allows us to define reusable HTML fragments that are not rendered until used by Knockout.
- **`foreach`**: Iterates over the `products` array and applies the template to each item.
- **`if` binding**: Conditionally renders elements (e.g., `[On Sale!]` only appears when `onSale` is `true`).
- **Data Context**: Inside the template, each product becomes the binding context, so `data-bind="text: name"` refers to the current product's `name`.
- **Performance Note**: This approach separates structure (template) from logic (ViewModel), making UI updates efficient and maintainable.

Lab 4.2: Building a Dynamic Task Manager with Nested Templates

Lab Overview

In this hands-on exercise, you will build a **Task Management Dashboard** using Knockout templating features. You will:

- Render a list of projects.
- Each project contains a list of tasks.
- Use nested templates to display tasks within each project.
- Apply conditional formatting (e.g., show "Overdue" badge).
- Use external templates via script tags.
- Debug rendering issues using browser developer tools.

You will practice key templating concepts including `foreach`, `if`, nested contexts, and template reuse.

Learning Outcomes

By the end of this lab, you will be able to:

- Define and use external templates in Knockout.
- Nest templates to represent hierarchical data.
- Pass and manage data context across template levels.
- Apply conditional logic within templates.

- Structure clean, modular UI code using templates.

Prerequisites

- Modern web browser (Chrome, Firefox)
- Text editor (VS Code, Sublime, etc.)
- Internet connection (to load Knockout.js from CDN)

File Structure

Create the following files in your project directory:

```
task-manager/
├── index.html
```

Step-by-Step Instructions

Step 1: Create `index.html`

Create a new file named `index.html` and open it in your editor.

> We will build this file incrementally.

Step 2: Add HTML Boilerplate and Knockout.js

Add the basic HTML structure and include Knockout.js from CDN.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Knockout Task Manager</title>
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/knockout/3.5.0/knockout-
min.js"></script>
</head>
<body>
  <div id="taskApp">
    <h1>Task Manager Dashboard</h1>
    <!-- Task list will be rendered here -->
  </div>
</body>
</html>
```

Step 3: Define External Templates

Below the `<div id="taskApp">`, add two `<script type="text/html">` blocks to define templates for **projects** and **tasks**.

```
<!-- Template: Project -->
<script type="text/html" id="projectTemplate">
  <div class="project">
    <h3 data-bind="text: name"></h3>
    <p><strong>Deadline:</strong> <span data-bind="text:
deadline"></span></p>

    <!-- Nested Template: Render tasks for this project -->
    <ul data-bind="template: { name: 'taskTemplate', foreach: tasks }"></ul>
  </div>
</script>

<!-- Template: Task -->
<script type="text/html" id="taskTemplate">
  <li>
    <span data-bind="text: title"></span>
    <small data-bind="text: 'Priority: ' + priority"></small>

    <!-- Conditional: Show 'Overdue' if due date is past and not completed --
>
    <span data-bind="if: isOverdue() && !isCompleted()" class="overdue">
      [OVERDUE]
    </span>

    <!-- Conditional: Show 'Completed' badge -->
    <span data-bind="if: isCompleted" class="completed">[Completed]</span>
  </li>
</script>
```

Note: The `isOverdue()` method will be defined in the ViewModel.

Step 4: Add Template Binding to Render Projects

Inside the `taskApp` div, below the `<h1>`, bind the `projectTemplate` to the `projects` array.

```
<!-- Render all projects using template -->
<div data-bind="template: { name: 'projectTemplate', foreach: projects
}"></div>
```

Step 5: Implement the ViewModel

Add a `<script>` block before the closing `</body>` tag to define the ViewModel.

```
<script>
  function Task(title, priority, dueDate, isCompleted) {
    this.title = title;
    this.priority = priority;
    this.dueDate = new Date(dueDate);
    this.isCompleted = ko.observable(isCompleted || false);

    // Computed: Check if task is overdue
    this.isOverdue = ko.computed(function () {
      return this.dueDate < new Date() && !this.isCompleted();
    }, this);
  }

  function Project(name, deadline, tasksData) {
    this.name = name;
    this.deadline = deadline;
    this.tasks = tasksData.map(function (taskData) {
      return new Task(
        taskData.title,
        taskData.priority,
        taskData.dueDate,
        taskData.isCompleted
      );
    });
  }

  function TaskManagerViewModel() {
    var self = this;

    self.projects = ko.observableArray([
      new Project("Website Redesign", "2025-04-15", [
        { title: "Design Mockups", priority: "High", dueDate: "2025-03-10",
isCompleted: true },
        { title: "Frontend Development", priority: "High", dueDate: "2025-04-
05", isCompleted: false },
        { title: "Content Migration", priority: "Medium", dueDate: "2025-04-
12", isCompleted: false }
      ]),
      new Project("Mobile App Launch", "2025-05-20", [
        { title: "UI Prototyping", priority: "High", dueDate: "2025-03-20",
isCompleted: true },
        { title: "Backend API", priority: "Critical", dueDate: "2025-04-10",
isCompleted: false },
        { title: "Beta Testing", priority: "Medium", dueDate: "2025-05-05",
isCompleted: false }
      ])
    ]);
  }
```

```
  // Apply bindings when DOM is ready
  document.addEventListener("DOMContentLoaded", function () {
    ko.applyBindings(new TaskManagerViewModel(),
document.getElementById("taskApp"));
  });
</script>
```

Step 6: Add Basic Styling (Optional)

Add a `<style>` block in the `<head>` to improve readability.

```
<style>
  body { font-family: Arial, sans-serif; margin: 20px; }
  .project { border: 1px solid #ccc; margin: 10px 0; padding: 10px; border-
radius: 6px; background: #f9f9f9; }
  ul { list-style: none; padding-left: 10px; }
  li { margin: 5px 0; padding: 5px; background: #fff; border: 1px dashed
#ddd; }
  .overdue { color: red; font-weight: bold; }
  .completed { color: green; font-style: italic; }
</style>
```

Expected Output

When you open `index.html` in a browser, you should see:

```
Task Manager Dashboard

Website Redesign
  Deadline: 2025-04-15
  • Design Mockups – Priority: High [Completed]
  • Frontend Development – Priority: High [OVERDUE]   ← if current date >
2025-04-05
  • Content Migration – Priority: Medium

Mobile App Launch
  Deadline: 2025-05-20
  • UI Prototyping – Priority: High [Completed]
  • Backend API – Priority: Critical [OVERDUE]        ← if current date >
2025-04-10
  • Beta Testing – Priority: Medium
```

If today's date is before the due dates, the `[OVERDUE]` labels will not appear. Adjust due dates to test.

Troubleshooting Tips

Issue	Solution
Nothing appears	Ensure `ko.applyBindings()` runs and `id="taskApp"` matches.
Template not found	Check `id` of `<script type="text/html">` matches `name` in binding.
`isOverdue` not working	Confirm `dueDate` is a `Date` object, not a string.
Binding errors in console	Open DevTools → Console → check for Knockout error messages.

---

4. Complete Implementation

Below is the **full, ready-to-run** implementation of the lab exercise.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Knockout Task Manager</title>
  <script
src="https://cdnjs.cloudflare.com/ajax/libs/knockout/3.5.0/knockout-
min.js"></script>
  <style>
    body { font-family: Arial, sans-serif; margin: 20px; }
    .project { border: 1px solid #ccc; margin: 10px 0; padding: 10px; border-
radius: 6px; background: #f9f9f9; }
    ul { list-style: none; padding-left: 10px; }
    li { margin: 5px 0; padding: 5px; background: #fff; border: 1px dashed
#ddd; }
    .overdue { color: red; font-weight: bold; }
    .completed { color: green; font-style: italic; }
  </style>
</head>
<body>
  <div id="taskApp">
    <h1>Task Manager Dashboard</h1>

    <!-- Bind projects using template -->
    <div data-bind="template: { name: 'projectTemplate', foreach: projects
}"></div>
  </div>

  <!-- Template: Project -->
  <script type="text/html" id="projectTemplate">
    <div class="project">
```

```
      <h3 data-bind="text: name"></h3>
      <p><strong>Deadline:</strong> <span data-bind="text:
deadline"></span></p>
      <ul data-bind="template: { name: 'taskTemplate', foreach: tasks
}"></ul>
    </div>
  </script>

  <!-- Template: Task -->
  <script type="text/html" id="taskTemplate">
    <li>
      <span data-bind="text: title"></span>
      <small data-bind="text: 'Priority: ' + priority"></small>
      <span data-bind="if: isOverdue() && !isCompleted()"
class="overdue">[OVERDUE]</span>
      <span data-bind="if: isCompleted" class="completed">[Completed]</span>
    </li>
  </script>

  <script>
    function Task(title, priority, dueDate, isCompleted) {
      this.title = title;
      this.priority = priority;
      this.dueDate = new Date(dueDate);
      this.isCompleted = ko.observable(isCompleted || false);

      this.isOverdue = ko.computed(function () {
        return this.dueDate < new Date() && !this.isCompleted();
      }, this);
    }

    function Project(name, deadline, tasksData) {
      this.name = name;
      this.deadline = deadline;
      this.tasks = tasksData.map(function (taskData) {
        return new Task(
          taskData.title,
          taskData.priority,
          taskData.dueDate,
          taskData.isCompleted
        );
      });
    }

    function TaskManagerViewModel() {
      var self = this;

      self.projects = ko.observableArray([
        new Project("Website Redesign", "2025-04-15", [
```

```
          { title: "Design Mockups", priority: "High", dueDate: "2025-03-10",
isCompleted: true },
          { title: "Frontend Development", priority: "High", dueDate: "2025-
04-05", isCompleted: false },
          { title: "Content Migration", priority: "Medium", dueDate: "2025-
04-12", isCompleted: false }
        ]),
        new Project("Mobile App Launch", "2025-05-20", [
          { title: "UI Prototyping", priority: "High", dueDate: "2025-03-20",
isCompleted: true },
          { title: "Backend API", priority: "Critical", dueDate: "2025-04-
10", isCompleted: false },
          { title: "Beta Testing", priority: "Medium", dueDate: "2025-05-05",
isCompleted: false }
        ])
      ]);
    }

    document.addEventListener("DOMContentLoaded", function () {
      ko.applyBindings(new TaskManagerViewModel(),
document.getElementById("taskApp"));
    });
  </script>
</body>
</html>
```

Summary

This lab demonstrated how to:

- Use external templates with `script` tags.
- Nest templates to render hierarchical data (projects → tasks).
- Apply conditional rendering (`if`) based on computed observables.
- Manage data context across template levels.
- Structure a maintainable and scalable UI using Knockout templating.

You now have a solid foundation for building dynamic, data-driven interfaces using Knockout's templating system.