Below are the questions are just for practice:

Question 1

A small grocery shop wants a simple page where users can enter item names and prices, and it will calculate the total bill.

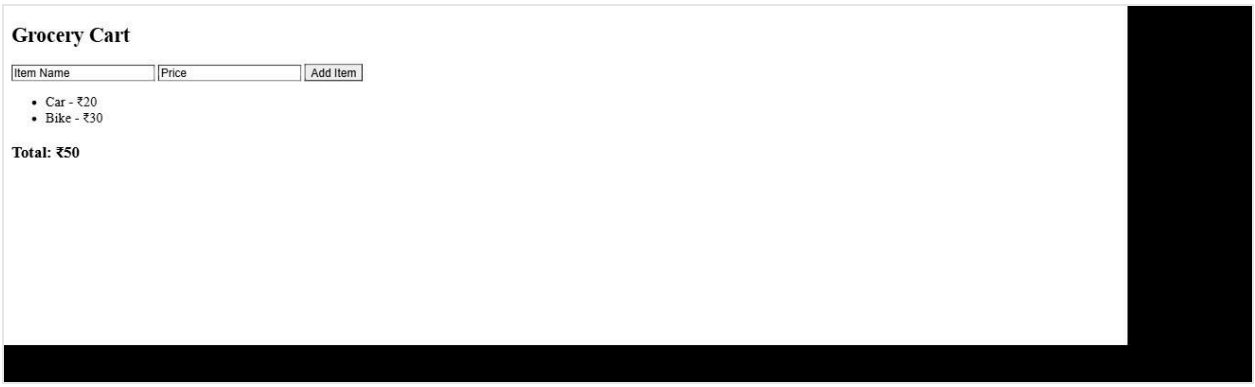
Requirements:

- An input box for **item name**
- An input box for **item price**
- Button: **Add Item**
- Display list of items
- Calculate and show **Total Amount** dynamically

Expected Skills Tested:

- DOM manipulation
- Event handling
- Arrays

Output Snapshot:



Question 2:

You are building part of a movie-ticket booking UI where a user selects seats.

Requirements:

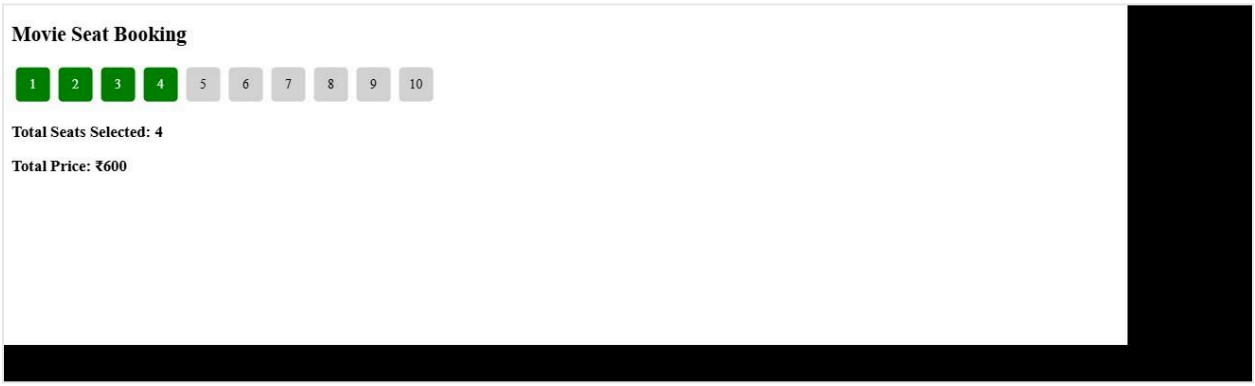
- Show 10 seats (buttons)
- When user clicks a seat:
    - If available → turn seat to green and add to selected list
    - If already selected → turn back to white and remove
- Show **Total Seats Selected**
- Show **Total Price = seats × 150**

Expected Skills Tested:

- DOM events
- Object/Array updates
- Conditional rendering

- UI state handling

Output Snapshot:



Question 3:

Build a mini project-management tool where a user can add tasks with priority and mark them as completed.

Requirements:

- Input: **Task Name**, **Priority (High/Low)**
- Button: **Add Task**
- Show tasks in a list:
    - Each task shows:
        - Task name
        - Priority badge
        - Status: Pending / Completed
        - Buttons: **Edit**, **Delete**, **Mark Completed**
- Editing should update the task *without page reload*

- Marking completed should change style (green background, strikethrough text)

Expected Skills Tested:

- CRUD operations
- Dynamic DOM rendering
- Event bubbling
- UI state management

Output Snapshot:

Question 4:

Animated Shopping Cart

Scenario:

You are building the product page for an e-commerce site.

Task:

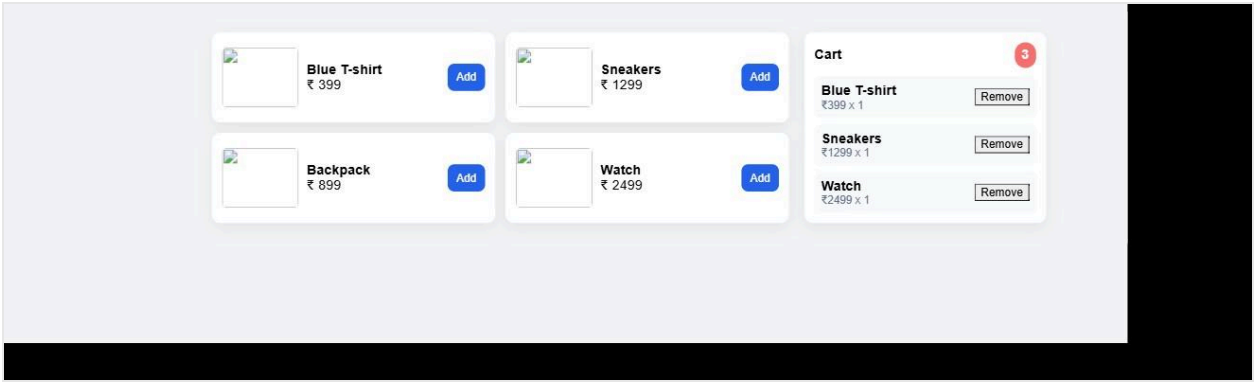
Create a simple product UI:

1. Show 4 products (image + name + price).
2. Each product has an **“Add to Cart”** button.
3. When clicking "Add to Cart":
    - Product smoothly **flies to the cart icon** (small animation).
    - Cart item count increases with a zoom animation.
4. Clicking the cart icon:
    - Open a small cart list sliding from the right.
5. Cart must support:
    - Item display
    - Increase quantity
    - Remove item

Image Url:

https://picsum.photos/id/1011/200/140
https://picsum.photos/id/1012/200/140
https://picsum.photos/id/1013/200/140
https://picsum.photos/id/1014/200/140
https://picsum.photos/id/1015/200/140

Output Snapshot:



Question 5:

Online Status Indicator With Pulse Animation

Scenario:

You need to simulate a user's online/offline indicator.

Task:

Create a status box showing:

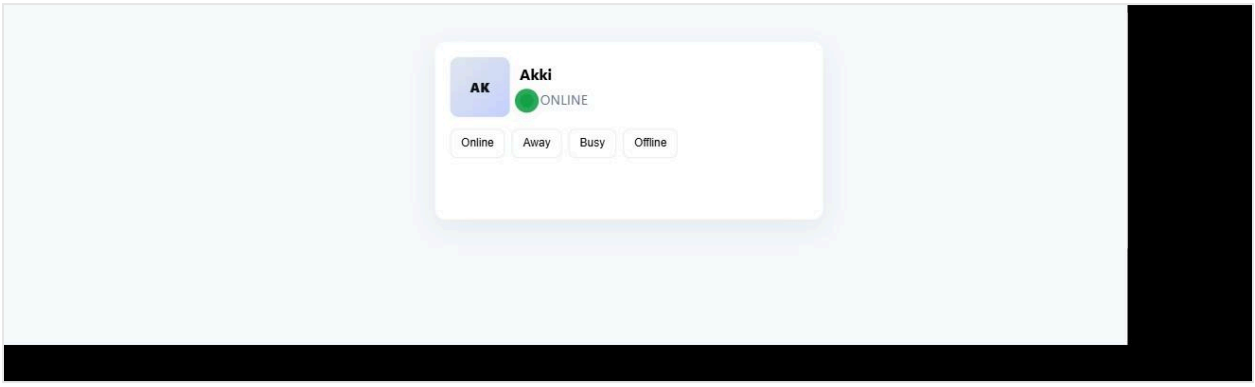
Username
Status Indicator

Features:

1. Status can be:
    - Online (green)
    - Away (yellow)
    - Busy (red)
    - Offline (gray)

2. Buttons to change status.
3. On status change:
    - Indicator circle does:
        - **Scale pulse animation**
        - color transition
4. Show a message box with **slide-in animation** saying:
   “User is now ONLINE/AWAY/BUSY/OFFLINE"

Output Snapshot:



Question 6:

A tech company is developing a **user registration module** for its upcoming learning portal. To improve security, they want a **password strength indicator** that guides users to create a better password.

The UX team has provided requirements:

- When a user types a password, the system must **instantly evaluate** its strength.
- The strength should be categorized into **Easy**, **Medium**, or **Strong**.
- A colored bar (password strength bar) should update based on the strength.

Your job is to build the **Registration Form** using **HTML, CSS & JavaScript DOM** and implement the password strength checker **as per rules below**.

Question Statement

Design a **Registration Form** that includes a password field with a **live strength indicator** using JavaScript DOM.

The password strength rating must follow these rules:

RULES FOR PASSWORD STRENGTH CHECKER

EASY Password (Weak)

A password is considered **EASY** if:

- Length is **less than 6 characters**
- Regardless of content (letters, numbers, symbols)

UI requirements:

- Show text: **“Easy”** (color: red)
- Show strength bar width: **30%**
- Bar color: **red (#ff0000)**

MEDIUM Password

A password is considered **MEDIUM** if:

- Length is **at least 6 characters**
  AND
- It includes:

    - Letters (A-Z or a-z)
    - Numbers (0–9)
      BUT
- No special characters (!@#$ etc.)

UI requirements:

- Show text: **“Medium”** (color: orange)
- Show strength bar width: **70%**
- Bar color: **orange (#ffa500)**

STRONG Password (Best)

A password is considered **STRONG** if:

- Length is **8 or more characters**
  AND
- Includes ALL:
    - Letters
    - Numbers
    - Special characters (!@#$%^&*)

UI requirements:

- Show text: **“Strong”** (color: green)
- Bar width: **100%**
- Bar color: **green (#00cc44)**

Output Snapshot:

Question 7:

Product Explorer

Objective:

Work with an API to fetch data, implement search, and display products using card-based UI.

Instructions:

1. Fetch & Display Products

Using:

https://dummyjson.com/products

Display each product as a card showing:

- Product Title

- Thumbnail Image

- Price

- Rating

2. Add a Search Feature-

Create a search box.

- On typing a keyword (e.g., phone, laptop), call the API:
https://dummyjson.com/products/search?q={keyword}

- Show search results dynamically

API: https://dummyjson.com/products/search?q=${query}

Output Snapshot:

Product Explorer

Search products...

Product Image
Essence Mascara Lash Princess
₹ 9.99
★ 2.56

Product Image
Eyeshadow Palette with Mirror
₹ 19.99
★ 2.86

Product Image
Powder Canister
₹ 14.99
★ 4.64

Product Image
Red Lipstick
₹ 12.99
★ 4.36

Product Image
Red Nail Polish
₹ 8.99
★ 4.32

Product Image
Calvin Klein CK One
₹ 49.99
★ 4.37

Product Image
Chanel Coco Noir Eau De
₹ 129.99
★ 4.26

Product Image
Dior J'adore
₹ 89.99
★ 3.8

Product Image
Dolce Shine Eau de
₹ 69.99
★ 3.96

Product Image
Gucci Bloom Eau de
₹ 79.99
★ 2.74

Product Image
Annibale Colombo Bed
₹ 1899.99
★ 4.77

Product Image
Annibale Colombo Sofa
₹ 2499.99
★ 3.92

Product Image
Bedside Table African Cherry
₹ 299.99
★ 2.87

Product Image
Knoll Saarinen Executive Conference Chair
₹ 499.99
★ 4.88

Product Image
Wooden Bathroom Sink With Mirror
₹ 799.99
★ 3.59

Product Image
Apple
₹ 1.99
★ 4.19

Product Image
Beef Steak
₹ 12.99
★ 4.47

Product Image
Cat Food
₹ 8.99
★ 3.13

Product Image
Chicken Meat
₹ 9.99
★ 3.19

Product Image
Cooking Oil
₹ 4.99
★ 4.8

Product Image
Cucumber
₹ 1.49
★ 4.07

Product Image
Dog Food
₹ 10.99
★ 4.55

Product Image
Eggs
₹ 2.99
★ 2.53

Product Image
Fish Steak
₹ 14.99
★ 3.78

Product Image
Green Bell Pepper
₹ 1.29
★ 3.25

Product Image
Green Chili Pepper
₹ 0.99
★ 3.66

Product Image
Honey Jar
₹ 6.99
★ 3.97

Product Image
Ice Cream
₹ 5.49
★ 3.39

Product Image

Product Image

Question 8:

A small business wants a simple browser-based system to **create invoices**, **calculate totals**, and **manage invoice records**.
Design a UI and build JavaScript logic to manage invoices dynamically.

Requirements

1. Add Invoice

Fields:

- Invoice Number
- Customer Name
- Item Name
- Quantity
- Price per Item
- Auto-calculated Total

2. Validate Input

- No empty fields
- Quantity & price must be positive
- Invoice number must be unique

3. Store Invoices in Array

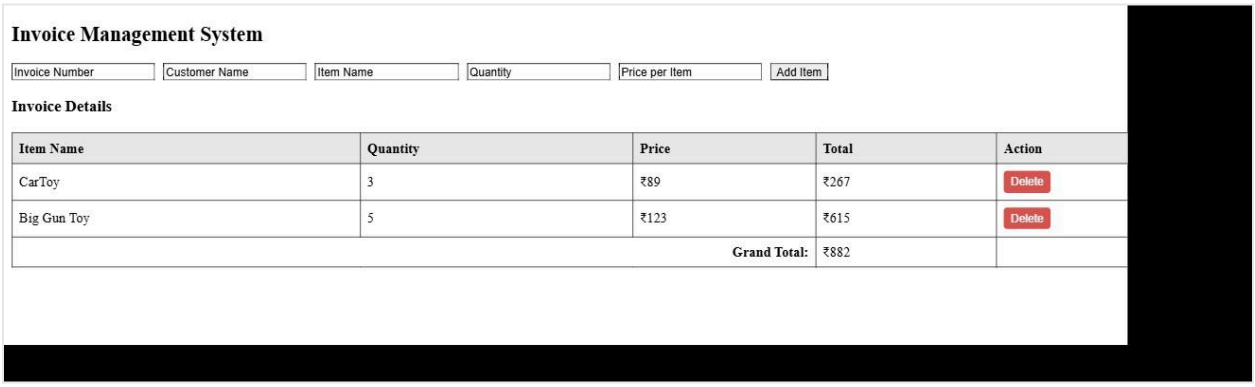
4. Display All Invoices

Show:

- Invoice No
- Customer
- Item
- Qty
- Price
- Total
- A **Delete** button (text-based)

5. Delete Any Invoice

Output Snapshot:



Question 9:

Objective:

Practice dynamic element creation, DOM-based table manipulation, and event-driven sorting/filtering.

Instructions:

- Create a form with fields: Name, Age, Country.

- On clicking “Add Row”, append a new row to the dynamic table.

- Implement the following features:

1. Column Sorting:

   Clicking on Name, Age, or Country header toggles ascending/descending sorting.

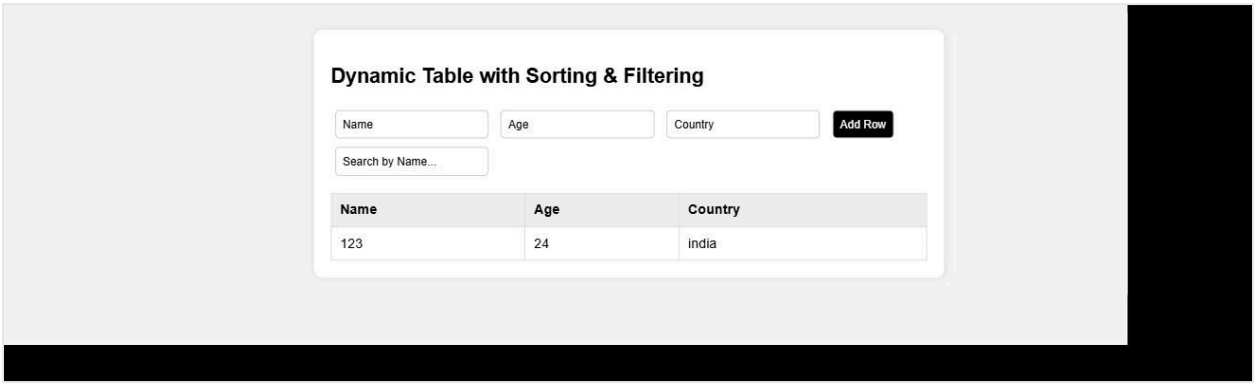
2. Live Filtering:

   Add a search box that instantly filters table rows by Name.

3. Row Highlighter:

   Hovering over any row should highlight it.

Output Snapshot:



Question 10:

A hotel wants a simple front-desk web application to **add room details**, **validate them**, and **show them in a list**.
Your job is to create the **UI + JavaScript logic** to manage the hotel’s room data.

Build a small dashboard where a staff member can:

1. Add a new room

Room details include:

- Room Number
- Room Type (Single / Double / Deluxe)
- Price per Night
- Availability (Available / Booked)

2. Validate inputs

- Room number cannot be empty
- Price must be a positive number
- Room number should not duplicate an already added room

3. Store the rooms in an Array

- Each room stored as an object

```
{ roomNo: 101, type: "Single", price: 2000, status: "Available" }
```

4. Display rooms in a card/list/table format

- Whenever a new room is added, update the DOM and show all rooms
- Highlight “Booked” rooms in red and “Available” rooms in green

5. Delete any room

Each room card/table row must have a **Delete** button to remove it from the list.

Expected UI

A form + room display section:

- **Form**
    - Room No → `<input>`
    - Type → `<select>`
    - Price → `<input>`
    - Status → `<select>`
    - Add Room → `<button>`
- **Room list**
    - Cards or table rows dynamically added with JavaScript

Expected Logic

- Form validation
- Prevent duplicate room numbers
- Array manipulation (push, filter)
- DOM manipulation (append, remove)
- Dynamic UI updates

Output Snapshot:

Question 11:

A local cricket tournament committee wants a lightweight **web-based score-management system** that can run entirely in a browser without any backend. They need a live dashboard where a scorer can update every ball of the match and the system automatically calculates the score, overs, batsmen stats, bowler stats, and logs each ball.

You have been hired to build this **Cricket Match Scorecard Dashboard** using **pure HTML, CSS, and JavaScript** (No frameworks, no libraries).

Task Requirements

Design and implement a working dashboard that supports the following:

1. Match Setup

- Default teams: *Team A* vs *Team B*
- Overs: *20 overs*
- Autoload Playing XI: 11 players for each team.

2. Live Score Management

Your dashboard must allow scorekeepers to record the following ball events:

- Runs: **0, 1, 2, 3, 4, 6**
- **Wide ball**
- **No-ball**
- **Wicket**
- **Undo** last ball action
- Strike rotation based on odd runs and over completion.

3. Auto Calculations

The application must automatically compute:

- Total score
- Total wickets
- Overs: (balls bowled → e.g., 4.3)
- Run rate
- Batsman stats: Runs, Balls, 4s, 6s
- Bowler stats: Overs, Runs conceded, Wickets
- Over-by-over ball log

4. UI Components

Your dashboard must include:

- **Score Display Panel**
- **Batting Scorecard Table**
- **Bowling Summary Table**
- **Ball Event Control Panel**

- **Over Log Panel**

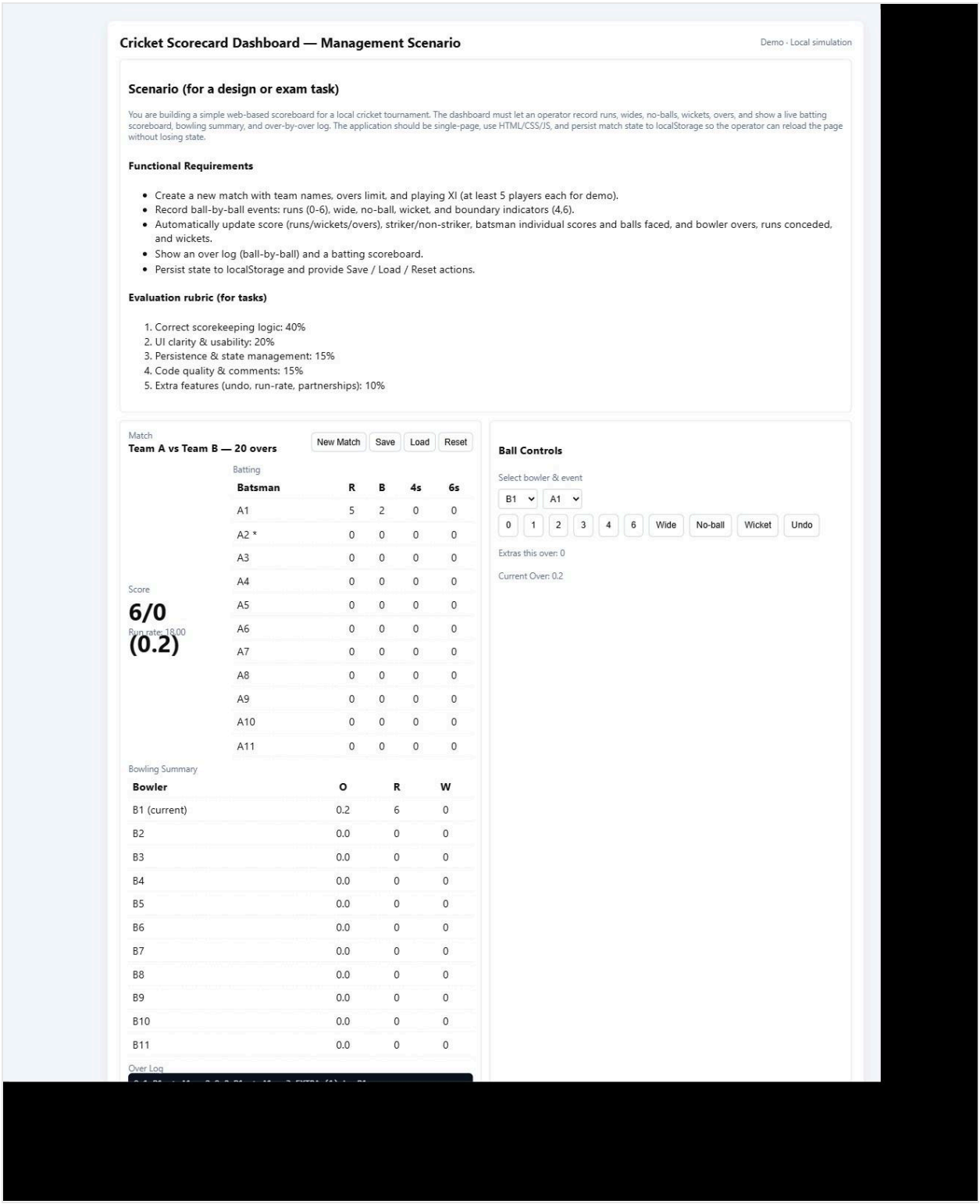
5. Data Persistence

- Use **localStorage** to Save, Load, and Reset the match.

6. Bonus (Optional)

- Partnership tracking
- Run-rate graph
- Two innings toggle
- Dark mode

Output Snapshot:



Cricket Scorecard Dashboard — Management Scenario

Demo · Local simulation

Scenario (for a design or exam task)

You are building a simple web-based scoreboard for a local cricket tournament. The dashboard must let an operator record runs, wides, no-balls, wickets, overs, and show a live batting scoreboard, bowling summary, and over-by-over log. The application should be single-page, use HTML/CSS/JS, and persist match state to localStorage so the operator can reload the page without losing state.

Functional Requirements

- Create a new match with team names, overs limit, and playing XI (at least 5 players each for demo).
- Record ball-by-ball events: runs (0-6), wide, no-ball, wicket, and boundary indicators (4,6).
- Automatically update score (runs/wickets/overs), striker/non-striker, batsman individual scores and balls faced, and bowler overs, runs conceded, and wickets.
- Show an over log (ball-by-ball) and a batting scoreboard.
- Persist state to localStorage and provide Save / Load / Reset actions.

Evaluation rubric (for tasks)

1. Correct scorekeeping logic: 40%
2. UI clarity & usability: 20%
3. Persistence & state management: 15%
4. Code quality & comments: 15%
5. Extra features (undo, run-rate, partnerships): 10%

Match

Team A vs Team B — 20 overs

New Match | Save | Load | Reset

Batting

Batsman	R	B	4s	6s
A1	5	2	0	0
A2 *	0	0	0	0
A3	0	0	0	0
A4	0	0	0	0
A5	0	0	0	0
A6	0	0	0	0
A7	0	0	0	0
A8	0	0	0	0
A9	0	0	0	0
A10	0	0	0	0
A11	0	0	0	0

Score

6/0

Run rate: 18.00

(0.2)

Bowling Summary

Bowler	O	R	W
B1 (current)	0.2	6	0
B2	0.0	0	0
B3	0.0	0	0
B4	0.0	0	0
B5	0.0	0	0
B6	0.0	0	0
B7	0.0	0	0
B8	0.0	0	0
B9	0.0	0	0
B10	0.0	0	0
B11	0.0	0	0

Over Log

Ball Controls

Select bowler & event

B1 | A1

0 | 1 | 2 | 3 | 4 | 6 | Wide | No-ball | Wicket | Undo

Extras this over: 0

Current Over: 0.2