

Project 1: Recipe Roulette 🎲 🔍

Project 1: Recipe Roulette 🎲🔍

You're tired of eating the same meals every week. You have a bunch of recipes saved but can never decide what to cook. You wish there was a way to randomly pick a recipe, track what you've cooked recently, and mark your favorites. Time to build a Recipe Roulette app!

What to Build:

A single-page application where users can:

Add New Recipes (Form with validation):

Recipe name (required, min 3 characters)

Ingredients (textarea, required)

Cooking time in minutes (number input, required, min 5)

Difficulty level (radio buttons: Easy, Medium, Hard)

Category checkboxes (Breakfast, Lunch, Dinner, Snack)

Show real-time validation errors below each field

Submit button should be disabled until all fields are valid

Recipe Display Section:

Show all recipes as cards (created dynamically using createElement)

Each card shows: name, time, difficulty, categories

Each card has: "Cook This!" button, "Favorite ★" button, "Delete 🗑️" button

Use event delegation on the container div for all button clicks

Random Recipe Picker:

"Spin the Wheel!" button that randomly highlights and selects one recipe

Use Math.random() to pick

Animate the selection (add/remove CSS classes)

Move selected recipe to "Recently Cooked" section

Sections to Create:

All Recipes section

Favorites section (filter favorites)

Recently Cooked section (last 5 cooked recipes)

Use DOM traversal to move recipes between sections

Storage:

Save all recipes to localStorage (as JSON)

Save favorites array and recently cooked array separately

Load everything on page refresh

DOM Concepts Covered:

Forms & validation

Creating & modifying DOM elements

Event delegation

localStorage

querySelector/querySelectorAll

Event basics

Array operations for UI

Element states (enable/disable buttons)

Project 2: Study Group Scheduler



Project 2: Study Group Scheduler

You and your friends want to form study groups for different subjects, but coordinating times is a mess. Everyone has different free slots. You need an app where people can propose study sessions, others can join, and you can see who's available when. Let's build a Study Group Scheduler!

What to Build:

User Registration (Constructor Function Pattern):

Create a User constructor function with properties: name, email, subjects (array)

When someone "signs up", create a new User instance

Store all users in an array

Display current users in a sidebar (with appendChild)

Study Session Creation (ES6 Class):

Create a StudySession class with:

Constructor: subject, date, time, maxParticipants, createdBy

Method: addParticipant(userName) - checks if session is full

Method: removeParticipant(userName)

Method: getSessionInfo() - returns formatted string

Private field: #participants = []

Session Management:

Form to create new study session (date input, time input, number input for max participants)

Display all sessions as a table (use createElement for rows)

Each row shows: subject, date, time, participants count/max, "Join" button

Use this keyword in session methods to access session data

Interactive Features:

Click on a session row to expand and show participant names

Use DOM traversal: parentElement, nextElementSibling to toggle details

"Join" button adds current user to session (use closest() to find session element)

Disable "Join" button if session is full

Show/hide sections based on state

Filtering & Sorting:

Filter sessions by subject (select dropdown with change event)

Sort sessions by date (use array methods, then re-render)
Search functionality using input event for real-time filtering

Storage:

Save all users and sessions to localStorage
Use JSON.stringify for complex objects
Load and reconstruct User instances and StudySession instances on page load

DOM Concepts Covered:

Constructor Functions & Instances
ES6 Classes with private fields
Objects & State Management
DOM Traversal
Forms with multiple input types
Creating elements dynamically
localStorage with JSON
The 'this' keyword (Classes 7, 8, 10)

Project 3: Campus Lost & Found Board



Project 3: Campus Lost & Found Board

You lost your water bottle in the library. Someone found a phone in the cafeteria. There's no central place to post lost/found items on campus. Let's build a digital lost & found board where people can post items they've lost or found, and match them up!

What to Build:

Post Item Form:

Type: Radio buttons (Lost / Found)
Item name (text input with real-time character count)
Description (textarea)
Location (text input with autocomplete suggestions)
Date (date input)
Contact info (email input with validation)
Photo URL (optional, url input type)
Color (text input)
Use input and blur events for validation
preventDefault on form submit

Item Board Display:

Two sections: "Lost Items" and "Found Items"
Items displayed as cards created with createElement
Each card shows all item details
Use insertBefore to add new items at the top (most recent first)
Use template literals for building card HTML with innerHTML

Smart Matching System (State Management):

Create a state object that tracks all items
When a "Found" item is posted, check if any "Lost" items have similar keywords
Show potential matches with visual highlighting (CSS class manipulation)
"This Might Be Mine!" button on matches
Use.textContent vs innerHTML appropriately for security

Interactive Features with Event Delegation:

Click on any item card to expand/collapse full details
"Mark as Resolved" button (removes item and updates counter)
"Contact Owner" button (shows contact info using closest() to find parent)
Edit button (populate form with existing data using getAttribute)
Use event delegation on the board container for all clicks

DOM Traversal & Navigation:

"Similar Items" section under each item

Use `querySelectorAll` to find all items

Use `parentElement` and `children` to navigate card structure

Filter and show related items using `nextElementSibling`

Statistics Dashboard:

Total lost items counter

Total found items counter

Total resolved items counter

Update counts dynamically (don't re-render entire board)

Use `querySelector` to target specific counter elements

Storage & Persistence:

`localStorage` for all items

`sessionStorage` for current user's draft post

Auto-save form data every 2 seconds

Restore draft if user refreshes page

DOM Concepts Covered:

Event Delegation on dynamic content

All form input types & validation

DOM Traversal patterns

Creating/removing elements

State Management with objects

`localStorage` & `sessionStorage`

Event basics with multiple event types

`querySelector` combinations

Project 4: Fitness Challenge Tracker



Project 4: Fitness Challenge Tracker 💪🏃

You want to get fit but lose motivation quickly. What if you could challenge your friends to fitness goals, track progress together, and see who's winning? Build a Fitness Challenge Tracker where you can create challenges, log workouts, and compete!

What to Build:

Challenge Class (ES6):

Create Challenge class with:

Constructor: name, type (Running/Push-ups/Steps), goal, duration (days), participants

Private field: #progress = {} (stores each participant's daily progress)

Method: addProgress(participant, date, value)

Method: getLeaderboard() - returns sorted array

Method: calculatePercentage(participant) - returns completion %

Getter: isComplete() - checks if anyone reached goal

Participant Management (Constructor Function):

Participant constructor with: name, age, fitnessLevel

Instance method: logWorkout(challenge, value)

Store participants array

Display participant cards with stats

Challenge Creation Form:

Challenge name (text, required, min 5 chars)

Type (select dropdown: Running km, Push-ups, Steps, Cycling)

Goal number (number input, min 1)

Duration in days (number input, min 1, max 30)

Select participants (multiple checkboxes from participant list)

Validation: show error messages, disable submit until valid

Use form.elements to access all inputs

Active Challenges Display:

Create challenge cards dynamically

Each card shows: name, type, goal, days remaining

Progress bars for each participant (create with createElement)

Use CSS classes to show progress percentage

Color coding: < 30% red, 30-70% yellow, > 70% green

Log Workout Interface:

Select challenge (dropdown populated from active challenges)

Select participant (radio buttons)

Enter today's value (number input)

Date input (defaults to today)

On submit: update progress in Challenge instance, re-render leaderboard

Use `setAttribute` to mark updated elements

Leaderboard Section:

Table created with `createElement` (thead, tbody, rows)

Columns: Rank, Name, Progress, Percentage

Sort by progress value

Highlight winner with special CSS class

Click on participant row to see their daily logs (DOM traversal to show/hide details)

Dynamic Interaction with Call/Apply/Bind:

Create utility functions that use `.call()` to borrow methods

Use `.bind()` for event handlers that need specific context

Use `.apply()` for calculations with array of values

Example: `Challenge.prototype.addProgress.call(anotherObject, ...)`

Filtering & State:

Filter challenges by type (use array filter, re-render)

Show completed vs active challenges (toggle with buttons)

State object pattern to manage current view

Update-render cycle on every state change

Storage:

Save all challenges to `localStorage`

Save all participants

Reconstruct Challenge and Participant instances on load

Handle data migration if structure changes

DOM Concepts Covered:

ES6 Classes with private fields & getters
Constructor Functions with instance methods
Call, Apply, Bind
State Management patterns
Form validation with all input types
Creating complex element structures
DOM Traversal for showing details
localStorage with complex objects
Array operations (finding min/max, sorting)

Project 5: Movie Night Planner 🎬 🍿

Project 5: Movie Night Planner 🎬🍿

Every weekend you and roommates argue about which movie to watch. Everyone has different tastes. You need a democratic system where everyone can suggest movies, vote, and the app picks based on preferences. Plus, you want to track what you've watched so you don't repeat!

What to Build:

Movie Class with Prototype Methods:

Constructor: title, genre, duration, suggestedBy, votes = []

Add methods to prototype (NOT inside constructor):

Movie.prototype.upvote(userName) - adds vote if not already voted

Movie.prototype.downvote(userName) - removes vote

Movie.prototype.getVoteCount() - returns total votes

Movie.prototype.hasUserVoted(userName) - returns boolean

User System (ES6 Class):

User class with: username, favoriteGenres (array), watchHistory

Method: suggestMovie(title, genre, duration) - returns new Movie instance

Method: addToWatchHistory(movieTitle)

Private field: #password (demonstrate encapsulation)

Movie Suggestion Form:

Username (text input, stored in sessionStorage for session)

Movie title (text with input event for validation)

Genre (select: Action, Comedy, Drama, Horror, Sci-Fi, Romance)

Duration in minutes (number input)

Streaming platform checkboxes (Netflix, Prime, Disney+, YouTube)

Release year (number input, validation: 1900-2024)

Form validation with real-time feedback

Clear form after successful submit

Movie List with Event Delegation:

Display all suggested movies in a scrollable container

Each movie is a card (use createElement for structure)

Cards show: title, genre, duration, suggested by, vote count

Buttons on each card: 👍 Upvote, 👎 Downvote, 🗑️ Remove, ▶️ Watch Now

Use event delegation on parent container
Use `event.target.matches()` to identify which button was clicked
Use `closest('.movie-card')` to find the movie element

Voting System with State Management:

State object tracks: `currentUser`, `allMovies`, `watchedMovies`
User can only vote once per movie
Visual feedback: highlight voted movies for current user
Disable vote buttons if not logged in (element state management)
Real-time vote count update (find element with `querySelector`, update `textContent`)

Smart Recommendations:

"Pick Tonight's Movie" button
Algorithm: highest votes + matches user's favorite genres + not in watch history
Animate the selection process (add/remove classes with `setTimeout`)
Show winner in a modal/popup (create with `createElement`, append to body)
Use `prepend()` to add winner to "Tonight's Choice" section

Filter & Sort Controls:

Filter by genre (select dropdown with change event)
Filter by platform (checkboxes with event listeners)
Sort by: votes, duration, recent (buttons with event delegation)
Search by title (input event for real-time search)
Use `querySelectorAll` to find all movie cards, show/hide with CSS

Watch History:

Separate section for watched movies
When "Watch Now" is clicked, move movie from suggestions to history
Use `element.remove()` to remove from suggestions
Use `appendChild()` to add to history
Show statistics: total watched, favorite genre (most watched)

DOM Traversal for Nested Features:

Click movie card to expand and show full details (streaming platforms, year)
Use `parentElement` to find card container
Use `querySelector` within card to find details section
Toggle details with `style.display` or CSS class
Use `firstElementChild`, `lastElementChild` for navigation

Multi-User Features:

Login/Logout system (simple, username only)
Show different users in a sidebar
Click on user to see their suggestions and votes
Use `getAttribute` and `setAttribute` to mark user-specific elements

Persistence & Storage:

`localStorage`: all movies, all users, watch history
`sessionStorage`: current logged-in user
Auto-save on every action
Load and reconstruct `Movie` instances and `User` instances
Handle edge cases: empty storage, corrupted data

Event Bubbling & Propagation:

Demonstrate `stopPropagation()` when needed
Understand `event.target` vs `event.currentTarget`
Handle clicks on nested elements properly

DOM Concepts Covered:

Prototype methods
ES6 Classes with private fields
Event Delegation with `matches()` and `closest()`
`Call/Apply/Bind` context
Event bubbling, `target` vs `currentTarget`
Complete form handling
Creating/removing/moving elements
All DOM traversal methods
State Management patterns
`localStorage` & `sessionStorage`
Multiple event types
Array operations