Mobile Programming - Lab 05

ReactJS Lab 2: Student Dashboard

2025-10-27

Lab Project: Student Dashboard

Project Goal

Build a **Dashboard** that combines composition and conditional rendering & lists. You will implement component composition, lifting state, controlled inputs, conditional UI states, filtering, sorting, and light validation. CSS is provided — **class names are enforced** in the lab so focus on React logic, not styling.

Note

If you want to use AI tools to help you, make sure to exclude any code that would violate academic integrity. Only ask for help on general React concepts, or syntax in general terms, instead of asking for help on the whole lab project.
At least try to understand what is the response you get from AI tools before using it directly!

Files you will create

```
src/
  App.jsx
  index.css  <-------- delete
  main.jsx   <-------- remove the import of index.css
  styles/
    lab-styles.css <-- provided (copy/paste)
  components/
    StudentList.jsx
    StudentItem.jsx
    StudentForm.jsx
    StudentControls.jsx
```

Important: **Do not** invent new class names for structure/visuals. Use the class names specified in this lab. The provided CSS targets them.

Setup

1. Create a new React project (Vite recommended):

```
npm create vite@latest student-dashboard -- --template react
cd student-dashboard
npm install
npm run dev
```

2. Replace `src/App.jsx` with the lab skeleton described below and add the `styles/lab-styles.css` file (contents provided at the end). Import the CSS in `src/App.jsx` with `import './styles/lab-styles.css';`.

Guidelines & Constraints (read carefully)

- **Minimal code is provided** — students must implement state updates, handlers, and composition. Small helper snippets are shown only where necessary.
- **Class names are required**. If a student uses different names they will break the provided CSS and lose points.
- Focus on **React behaviour**: composition, props, state, conditional rendering, list keys, and controlled components. Do not add layout CSS; use the provided stylesheet.
- Use functional components and React hooks (`useState`). No class components or advanced hooks.

Milestone 1 — Skeleton & Composition

Goal: Build component files and render a static list using composition.

Tasks

1. In `App.jsx` define `initialStudents` (minimal ready code):

```
const initialStudents = [
  { id: 1, name: 'Ali', grade: 85 },
  { id: 2, name: 'Siti', grade: 72 },
  { id: 3, name: 'Rahim', grade: 55 },
];
```

2. Create components with the exact filenames shown earlier. Implement minimal exports so imports work. **Do not** implement business logic yet — only structure and markup with required class names.

- `StudentList.jsx` should accept prop `students` and render a `<ul className="student-list">` that maps `students` to `<StudentItem>`.
- `StudentItem.jsx` should accept prop `student` and return an `<li className="student-item">` containing two elements with class names: `.student-name` and `.student-grade`.
- `StudentForm.jsx` should render a small form with class name `student-form` (inputs should have class `input` and the submit button class `btn`). For now, the form can `return null` or be inert — you'll wire it in the next milestone.
- `StudentControls.jsx` will later host filter/sort controls. For now create a skeleton component that returns a `<div className="controls">`.

3. In `App.jsx` import and render the components in this order inside a root `<div className="app">`:

```
<Header className="header">Student Dashboard</Header>
<StudentForm />
<StudentControls />
<StudentList students={initialStudents} />
```

4. Verify markup shows the static list. No dynamic behaviour yet.

Milestone 2 — State, Adding & Validation

Goal: Add state in `App.jsx`, implement the add-student flow as a controlled component, and basic validation.

Tasks

1. Convert `App.jsx` to hold state:

```
const [students, setStudents] = useState(initialStudents);
```

2. Implement controlled inputs **inside `StudentForm.jsx`** (do not keep form state in `App.jsx`):

- `StudentForm` should manage `name` and `grade` with `useState`.
- On submit, validate: name must not be empty, grade must be a number between 0 and 100.

- If validation passes, call `onAdd({ id: Date.now(), name, grade: Number(grade) })` — provided via props from `App.jsx`.
- After successful submit, clear inputs.

3. In `App.jsx` implement `addStudent(newStudent)` and pass it as `onAdd` prop into `StudentForm`.
4. **Edge cases:** If a student with the same `name` already exists (case-insensitive), the form must reject it and display a small inline error message element with class `form-error` (inside `StudentForm`).

Milestone 3 — Conditional Rendering + Lists

Goal: Implement pass/fail display, empty state, and delete operation.

Tasks

1. **Pass/Fail UI (in `StudentItem.jsx`)**

- Compute `passed = student.grade >= 60`.
- The root `<li>` must have an extra class `student-pass` or `student-fail` depending on the result (in addition to `student-item`). This is important for the provided CSS rules.
- Show `Pass` or `Fail` next to the grade inside an element with class `student-status`.

2. **Empty-state (in `StudentList.jsx`)**

- If `students.length === 0`, render a `<p className="no-data">No students yet - use the form above.</p>` instead of the `<ul>`.

3. **Delete a student**

- Add a small delete `<button className="delete-btn">Delete</button>` inside each `StudentItem`.
- `StudentItem` should accept an `onDelete(id)` prop and call it when the button is clicked.
- Implement `deleteStudent(id)` in `App.jsx` using `.filter()` and pass it down.

Milestone 4 — Filters, Sorting & Derived Data

Goal: Add controls to filter by pass/fail, search by name, and sort by grade.

Tasks

1. **StudentControls.jsx** must render the following UI with exact class names:

- `<div className="controls">`
  - `<div className="filters">` containing three `<button className="filter-btn">` elements with values `all`, `pass`, `fail`. Active filter button must have an additional class `active`.

- `<input className="input search" placeholder="Search by name" />` for name search (controlled by `App.jsx` or `StudentControls` — choose one and document your choice in code comments).
- `<button className="btn sort-btn">Sort: High → Low</button>` toggles between high→low and low→high.

2. **App-level derived list logic** (in `App.jsx`):

- Keep `filter`, `search`, and `sort` state in `App.jsx`.
- Compute `visibleStudents` by chaining `.filter()` and `.sort()` operations based on these states.
- Pass `visibleStudents` into `StudentList`.

3. **Conditional messages**

- If search yields zero matches, show `<p className="no-data">No students match "<em>SEARCH_TERM</em>"</p>`.

Milestone 5 — Final Checks & Best Practices

Checklist students must pass before submitting:

- Every list item uses `key={student.id}`.
- No uncontrolled inputs — all form inputs must be controlled.
- Deleting and adding students updates the `students` state immutably.
- Filter and search work together and respect sorting.
- CSS class names from the spec are used exactly.

Provided CSS (copy to `src/styles/lab-styles.css`)

```css
/* lab-styles.css - students should not change these class names */

.app {
  max-width: 760px;
  margin: 0 auto;
  font-family: system-ui, -apple-system, 'Segoe UI', Roboto, 'Helvetica Neue',
  ↪  Arial;
  text-align: center;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  padding: 24px;
}

.header {
  text-align: center;
```

```
  margin-bottom: 12px;
}
.student-form {
  display: flex;
  gap: 8px;
  margin-bottom: 12px;
  align-items: center;
  justify-content: center;
  width: inherit;
}
.input {
  padding: 8px 10px;
  border: 1px solid #d0d0d0;
  border-radius: 6px;
}
.input-grade {
  width: 100px;
}
.btn {
  padding: 8px 12px;
  border-radius: 6px;
  border: none;
  background: #0b5cff;
  color: white;
  cursor: pointer;
}
.controls {
  display: flex;
  gap: 12px;
  align-items: center;
  margin-bottom: 12px;
  justify-content: center;
  flex-wrap: wrap;
}
.filters { display: flex; gap: 6px; }
.filter-btn { padding: 6px 8px; border-radius: 6px; border: 1px solid #ccc;
 ↪  background: white; cursor: pointer; }
.filter-btn.active { background: #0b5cff; color: white; border-color: #084ecb; }
.student-list { list-style: none; padding: 0; margin: 0; width: 100%; }
.student-item { display: flex; justify-content: space-between; padding: 10px;
 ↪  border-bottom: 1px solid #eee; align-items: center; width: 100%; }
.student-name { font-weight: 600; }
.student-grade { margin-left: 12px; }
.student-status { font-size: 0.9em; margin-left: 8px; }
.student-pass { background: #f6ffef; }
.student-fail { background: #fff6f6; }
.delete-btn { background: transparent; border: 1px solid #ff6b6b; color: #ff6b6b;
 ↪  padding: 6px 8px; border-radius: 6px; cursor: pointer; }
.no-data { padding: 12px; text-align: center; color: #666; }
.form-error { color: #b00020; font-size: 0.9em; margin-left: 8px; }
.search { width: 220px; }
.sort-btn { padding: 6px 10px; }
```

```css
.stats { margin-bottom: 8px; font-size: 0.95em; color: #333; }
```

Starter hints

- `App.jsx` should import the stylesheet at top:

  ```
  import './styles/lab-styles.css';
  ```

- How to add `onAdd` prop to `StudentForm` (example signature only — do not copy full implementation):

  ```
  <StudentForm onAdd={(student) => setStudents(prev => [...prev, student])} />
  ```

- `StudentItem` should call `onDelete(student.id)` when delete button pressed.

Submission

You to push this project to a GitHub repo and submit the repo URL. Graders should clone and run `npm install && npm run dev`. So the whole project must be included, not just snippets. You must add the `node_modules` folder to `.gitignore`.
