Assignment: Contact Card Dashboard

Overview

Build a two-panel Contact Card Dashboard to practice Angular `@Input`, `@Output`, event/property binding, and lifecycle hooks—without any typed inputs.

Layout

- Left Panel (Listing)
  - A row of filter buttons: All, Favourites, Family, Friends, Classmates.
  - A contact list showing cards for the currently active filter.
- Right Panel (Details)
  - Shows the selected contact's full details.

Data & Mock Content

- Create 50 mock contacts.
- Each contact must include:
  - `id`, `name`, `phone`, `email`, `gender`, `address`
  - `groups: string[]` where a contact can be in multiple groups (any of: `Favourites`, `Family`, `Friends`, `Classmates`).

Behaviour

- Clicking a filter button updates the left list to show only contacts in that group (use All to show everything).
- Clicking a contact in the left list selects it and displays its details in the right panel.
- The right panel must include group toggle buttons for:
  - Mark as Favourite / Unfavourite
  - Add/Remove from Family
  - Add/Remove from Friends
  - Add/Remove from Classmates
- State changes are immediate:
  - If a contact is marked as Favourite, it must instantly appear when the Favourites filter is active.

    - The same applies when adding/removing the contact from Family, Friends, or Classmates while those filters are active.

Technical Requirements

- No text inputs or forms—only buttons/toggles and click events.
- State up, data down:
    - The parent component owns the source of truth: contacts list, selected contact, and current filter.
    - Children receive data via @Input and communicate changes via @Output.
- Use event binding for clicks and property binding for visual states (e.g., active filter button, favourite badge).
- Demonstrate lifecycle hooks

Acceptance Criteria

- Left panel shows filter buttons and a list that updates correctly for All/Favourites/Family/Friends/Classmates.
- 50 contacts are present with a spread across groups; several contacts belong to multiple groups.
- Clicking a contact displays full details (name, phone, email, gender, address, group badges) in the right panel.
- Toggling Favourite/Family/Friends/Classmates on the details view immediately updates:
    - the contact's state,
    - the current filtered list.
- No text boxes—only buttons and click actions.

Extension: Contact Card Dashboard (Routing, Add Contact, Search).

Goals

Extend your existing Contact Card Dashboard to use Angular routing, Reactive Forms, and a type-restricted search with a highlight pipe and a custom input directive.

New Features (on top of your current dashboard)

- Routing

  - Add a route to open the dashboard: `/dashboard`.
  - Add a second route to open an "Add Contact" page: `/contacts/new`.
- Add Contact Page
  - Build a separate page (component) at `/contacts/new`.
  - Use Reactive Forms (no template-driven forms) with suitable fields to capture the full contact model:
    - `id`, `name`, `phone`, `email`, `gender`, `address`, `groups: string[]` (from: Favourites, Family, Friends, Classmates).
  - On clicking Add, persist the new contact into the app's state and route back to `/dashboard`.
  - The newly added contact should immediately appear in the All list and in any relevant group filters.
- Search on Dashboard
  - Add a search input above the left-panel listing on `/dashboard`.
  - Behavior: exact match on `name` only (case-insensitive).
    - As the user types, display only contacts whose `name` equals the typed text exactly (ignoring case).
    - No partial/contains search for filtering (strict equality).
  - Create and use a pipe to highlight the matched portion of the name (the entire name will be highlighted when it exactly matches).
- Input Restriction Directive
  - Create a reusable attribute directive that only allows letters A–Z / a–z and space to be entered.
  - Apply this directive to the dashboard search field.
  - The directive must gracefully allow control keys (Backspace, Delete, Arrow keys, Tab, etc.).

Architecture & Constraints

- Keep "state up, data down":
  - The parent (dashboard container) remains the single source of truth for: contacts list, selected contact, and current filter.
  - Children get data via `@Input()` and emit changes via `@Output()`.
- Immediate state updates:

    - Adding a contact updates the dashboard lists instantly.
    - Existing group toggles (Favourite/Family/Friends/Classmates) must continue to update the active filtered list immediately.
- Lifecycle hooks: Continue to demonstrate where appropriate (e.g., init, changes).
- No changes to existing group filter buttons/behavior from the base assignment, other than coexisting with the new search bar.
- Forms are now allowed (Reactive Forms only) for the Add Contact page and the search field on the dashboard.

Acceptance Criteria

- `/dashboard` shows:
    - Filter buttons (All/Favourites/Family/Friends/Classmates) working as before.
    - A search field above the list that:
        - Accepts only letters and spaces (via your directive).
        - Filters by exact, case-insensitive match on `name`.
        - Uses the highlight pipe to emphasize the matched name.
- `/contacts/new` shows a Reactive Form for all contact fields (including selecting groups).
    - Clicking Add saves the contact and navigates back to `/dashboard`.
    - The new contact appears immediately in All and in any chosen group filter.
- Existing behavior remains:
    - Clicking a contact shows details on the right panel.
    - Toggling group membership in the details view updates the contact and the current filtered list immediately.
- No template-driven forms.

What to Submit

- Source code (Angular app) with:
    - Routes configured for `/dashboard` and `/contacts/new`.
    - Add Contact component using Reactive Forms.
    - Search input with the custom alpha-space directive applied.
    - Highlight pipe used in the list UI.
    - Clear `@Input()`/`@Output()` usage and lifecycle hook demos where applicable.

- A brief README:
    - How to run the app.
    - Where the routes/components live.
    - Notes on state management (e.g., service or parent component).