Ali Hamad/Hussein baydoun
Dr. Ghadir Khalil
web application
24/11/2025

Dr-online

Dr. Online is a frontend-only React web application that simulates a small online medical community. Users register and log in as either Doctor or Patient, then interact inside a Discussion Board by creating posts, liking/unliking, commenting, editing, deleting, and filtering topics by role. There is no backend/database. All users and posts are stored in React state, so refreshing the page resets the demo data.

Technologies & Tools Used

- React JS (component-based UI)
- React Router DOM (multi-page navigation without reload)
- React Hooks (useState) (state management)
- HTML/CSS (custom page styling)
- react-icons (icons for UI)
- Demo data modules (static JS arrays)

High-Level Architecture & Data Flow

Architecture Type

A React SPA (Single Page Application).
Routing changes the visible page without refreshing the browser.

Data Flow

Global data lives in APP.js:

- current user
- all users
- all posts

- App.js sends these to pages using props.
- Pages run handlers (post, like, comment, login, register…)
- Handlers update state using setState.
- React re-renders the UI immediately.

So it follows one-way data flow:
App state → props to pages → user action → handler → update App state → UI refresh

Code Documentation

App.js (Main Controller)

Main responsibilities:

Defines global state

- user: currently logged-in user (null if guest)
- users: registered users list, seeded by demo users
- posts: discussion posts list, seeded by demo posts

Handles logout

- handleLogout() sets user back to null and alerts.

Defines routes

- Home, Doctors, Dashboard, Contact, Login, Register.
- Data and setters are passed to the right pages.

Layout wrapper

- Navbar appears above routes.
- Footer appears below routes.

Why this is correct:
Keeping global state in App.js allows multiple pages to work with the same shared data.

Navbar.js

1. Shows navigation links: Home / Doctors / Discussion / Contact.
2. Handles mobile menu open/close
   - local state menuOpen
   - toggle icon changes between hamburger and X.
3. Role-based right section
   - If user is logged in → shows:
     - Login link
     - Register link
4. Logout button triggers:
   - onLogout() from App.j
   - then closes menu

Pattern used: conditional rendering based on user.

Footer.js (Global Footer)

What it does:

- Shows quick navigation links.
- Shows social/contact icons.

- Shows copyright.
- Shows a medical disclaimer to clarify this is general info.

Home.js (Landing Page)

1. Creates isLoggedIn = !!user
2. Displays a hero intro about platform mission.
3. CTA button changes depending on login:
   - Guest → “Join the Community” → Register
   - Logged in → “Go to Discussion Board” → Dashboard
4. Shows an extra login link only if user is guest.
5. Also includes:
   - Features section
   - How-it-works explanation
   - Services cards
   - Role-based CTAs later in the page

DoctorsList.js (Static Doctor Data)

Each doctor object includes:

- id
- name
- specialty
- Bio

This is used in Doctors.js for mapping.

DoctorCard.js (Reusable Presentational Component)

What it does:

- Receives one doctor object as prop.
- Displays name, specialty, and bio.
- No state and no logic → purely for UI

Doctors.js (Doctors Directory Page)

Logic:

- Imports doctorsData
- Maps through it and renders one DoctorCard per doctor.

usersData.js (Static Users Seed)

Structure:

- id
- name
- email
- password
- role (doctor/patient)

postsData.js (Static Posts Seed)

Each post object has:\

- id
- author
- role
- text
- likes: []
- comments: []
- date

Register.js (Account Creation)

Controlled form pattern:

- Form values stored in one object state:
    - name, email, password, role
- handleChange() updates the correct field.
- Clears any previous errors

Submit logic (handleSubmit):

1. Prevents page refresh.
2. Validates required fields (name/email/password).
3. Checks if email already exists in users:
    - if yes → show error and stop.
4. If valid:
    - Creates new user object with unique id.
    - Adds user to users list.
    - Auto logs in by setting user.

        - Shows success alert.

Login.js (Authentication)

Controlled inputs:

- email & password stored in form state.
- handleChange updates state + clears old errors.

Submit logic (handleSubmit):

1. Prevents page refresh.
2. Searches users list for matching email & password.
3. If not found:
    - Sets error message.
4. If found:
    - Sets global user.
    - Marks logged state true.
    - Shows success alert.

Dashboard.js

Role: Main interactive forum.

Access guard

- If user is null, the component returns a message asking user to login.
- This blocks posting by guests.

Local states

1. postText → new post textarea value
2. filter → “all / doctor / patient”
3. commentInput → stores comment text per post id

4. editingPostId → which post is in edit mode
5. editText → textarea for editing

Main handlers

1. Create Post (handlePost)
    - Runs on submit of the form
    - Validates non-empty text
    - Creates new post object:
        - id, author, role, text, likes[], comments[], date
    - Adds it to top of posts list
    - Clears textarea
2. Delete Post (handleDelete)
    - Removes the post by filtering out the id

3. Edit Post
    - startEdit(post) enables edit mode and loads text
    - saveEdit() updates post text and exits edit mode
    - cancelEdit() exits edit mode without saving
4. Like / Unlike (toggleLike)
    - If user name already exists in likes → remove it
    - Else → add it
    - This is done immutably using map + ternary

5. Add Comment (addComment)
    - Reads commentInput by postId
    - Validates non-empty
    - Appends {author, text} into post comments
    - Clears that post's input
6. Filtering (filteredPosts)
    - if filter = all → show posts
    - else → show only posts matching role

UI result

Dashboard renders:

- Header with name + role
- Post creation form with role-based placeholder
- Filter buttons
- List of filtered posts
- Each post card supports:
    - like button + count
    - comments section
    - edit/delete (only if author is current user)

Contact.js (Feedback Form)

State:

- One object state stores:
    - fname, email, subject, message
- submitted state tracks whether to show success message.

Handlers:

1. handleChange() updates the correct field

2. handleSubmit():
    - prevents refresh
    - validates required fields
    - alerts demo JSON
    - shows submitted message
    - resets form

Core User Flows

Flow A: Guest → Register → Discussion

1. Guest visits Home
2. Clicks Join Community
3. Registers role (doctor/patient)
4. Auto login
5. Opens Dashboard and posts

Flow B: Existing User Login

1. Guest goes Login
2. Enters credentials
3. Navbar updates
4. Can access Discussion Board fully

Flow C: Discussion Interaction

1. User posts topic
2. Other users like/comment
3. Author can edit/delete own post
4. Feed can be filtered by role

React Patterns Demonstrated

- Controlled components (all forms)
- Prop drilling for shared state
- Conditional rendering (role-based UI, access blocks)
- Immutable state updates
    - map/filter/spread to avoid mutation
- Component reusability (DoctorCard)

Edge Cases Handled

- Empty register fields
- Duplicate email on register
- Wrong login credentials
- Guest access blocked from discussion
- Empty post blocked
- Empty comment blocked
- Editing empty text blocked

```
├── App.js
├ assets/
│ ├──doctor.png
│ ├──services.png
├── pages/
│ ├── Home.js
│ ├── Doctors.js
│ ├── Dashboard.js
│ ├── Contact.js
│
├── components/
│ ├── Navbar.js
│ ├── Footer.js
│ ├── DoctorCard.js
├── auth/
│ ├── Login.js
│ ├── Register.js
├── data/
│ ├── usersData.js
│ ├── postsData.js
│ └── DoctorsList.js
│
│
│
└── demo/
└── demo.mp4
```