# JOB PORTAL

## 1. Architecture Style

- **Static Monolithic Page**:
  Entire functionality exists in a single HTML file. No client-server interaction, databases, or dynamic processing.

---

## 2. Core Components

### a) User Interface (UI)

- **Job Search Form**:
  - Inputs: Job title, location, category (dropdown).
  - Purpose: *Simulates* search criteria (no actual filtering).
- **Featured Jobs Section**:
  - Hardcoded job listings (3 examples) with:
    - Position, company, location, salary, description, and "Apply Now" link.
- **Job Posting Form**:
  - Inputs: Company name, job position, location, description (textarea), salary.
  - Purpose: *Simulates* job submission (no data storage).

### b) Data Representation

- **Static Mock Data**:
  Job listings are hardcoded in HTML. No database or API integration.
- **No State Management**:
  All data resets on page refresh (no persistence).

---

## 3. Data Flow

1. **User Interaction**:
    - User fills forms or clicks links.
2. **No Processing**:
    - Form submissions trigger browser defaults (e.g., page reload) but **do not send data anywhere**.
    - "Apply Now" links are non-functional (# placeholders).
3. **Output**:
    - Content is rendered as static HTML. Zero dynamic updates.

---

## 4. Limitations

- **No Functionality**:
  Forms cannot search, filter, or submit data.
- **No Responsiveness**:
  Layout breaks on mobile (no CSS/media queries).
- **No Security/Validation**:
  Zero input sanitization or user authentication.
- **Scalability**:
  Adding jobs requires manual HTML edits.

---

## 5. System Boundaries

- **Input**: User keystrokes/clicks (ignored by the system).
- **Output**: Predefined HTML content.
- **External Systems**: None (no APIs, databases, or networks involved).

---

## 6. Deployment

- **Hosting**: Can be served as a static file via any web server (e.g., Apache, Nginx) or opened locally in a browser.
- **Dependencies**: None (pure HTML5).

---

## Summary

This is a **non-functional prototype** demonstrating a job portal's *structure* using only HTML semantics. It lacks all dynamic features of a real job portal (search, applications, user accounts, etc.) and serves purely as a wireframe. To become operational, it would require:

- Backend (e.g., Node.js/Python) for data processing.
- Database (e.g., PostgreSQL) to store jobs/users.
- Frontend enhancements (CSS/JS) for interactivity.