Code Test – Senior Front End Engineer

Test Scenario

You're building an internal tool for content editors to manage a list of articles. The panel should allow editors to:

- View a paginated list of articles
- Search and filter articles by title or status (e.g., Published, Draft)
- Add a new article via a form (with validation)
- Edit and delete existing articles

Requirements

UI & Interactions

- Use your preferred Front-End framework of choice (**Svelte** with TypeScript is preferred)
- UI must include:
    - **List view** of articles
    - **Search input**
    - **Status filter (dropdown or tabs)**
    - **Modal or separate screen** for Add/Edit
- Basic form validation (e.g., required fields)

Data Handling

- Use **mock API endpoints** (mock JSON or services like MirageJS)
- Each article object should contain:

  {

  "id": 1, "title": "Example Article", "status": "Published", "author": "Jane Doe",

  "createdAt": "2024-05-01T12:00:00Z"

  }
- Simulate CRUD operations (Create, Read, Update, Delete)
- Show appropriate **loading** and **error** states

Accessibility & Performance

- Use semantic HTML
- Ensure keyboard accessibility and screen reader friendliness
- Lazy-load article list if large dataset is used

Testing

- Include:
    - One **unit test** (form logic or utility)
    - One **integration or UI test** (e.g., submitting a form, filtering list)

Engineering Practices

- Clear folder/component structure
- Demonstrate all the best practices you follow
- Reusable components (e.g., button, modal, input)
- Optional: use of **SvelteKit** or **micro front-end readiness**
- Git repo with meaningful commits and a README

Bonus Extensions (Optional)

- Role-based access (view-only vs. editor)
- Theme toggle (light/dark)
- Pagination or infinite scroll