General Good Frontend Coding Practices

- Code should be visually consistent
  - Use a consistent color palette throughout the UI
  - Fonts, spacing, and sizing should follow a defined design system
  - Consistent styling makes the interface feel professional and intentional
- Semantic HTML
  - Use the correct HTML element for its intended purpose
  - Improves accessibility and screen reader compatibility
  - Helps browsers and search engines understand page structure
- Responsive Design
  - UI should adapt cleanly to different screen sizes
  - Avoid hardcoded pixel values where relative units work better
  - Test on mobile, tablet, and desktop viewports
- Accessibility (A11y)
  - All images should have descriptive alt text
  - UI must be fully keyboard-navigable
  - Ensure sufficient color contrast for readability
- Separation of Concerns
  - Keep structure (HTML), styling (CSS), and behavior (JS) logically separate
  - Avoid inline styles and inline event handlers
  - Business logic should not be mixed into rendering code
- Performance
  - Minimize and bundle assets to reduce load times
  - Lazy load resources that are not immediately needed
  - Avoid unnecessary re-renders and DOM manipulations
- Input Validation & Security
  - Sanitize user input to prevent XSS attacks
  - Never expose API keys or sensitive data in client-side code
  - Never rely solely on client-side validation
- State Management
  - Keep application state predictable and centralized where appropriate
  - Avoid passing state through too many layers unnecessarily
  - Makes debugging and feature changes significantly easier
- Cross-Browser Compatibility
  - Test across major browsers (Chrome, Firefox, Safari, Edge)
  - Avoid using experimental or non-standard browser features without fallbacks
  - Inconsistent rendering across browsers is a common source of bugs
- Progressive Enhancement
  - Build a functional baseline experience first, then layer on enhancements

    - Core content and functionality should work even with JavaScript disabled
    - Ensures the widest possible audience can use the product
- Component Reusability
    - Build UI elements as self-contained, reusable components
    - Reduces duplication and keeps the codebase easier to maintain
    - Changes to a shared component are reflected everywhere it is used
- Error Feedback
    - Always inform the user when something goes wrong in a clear, friendly way
    - Avoid showing raw technical error messages to end users
    - Provide actionable guidance where possible (e.g. "Try again" or "Check your input")
- Loading & Empty States
    - Every data-driven UI element should account for loading and empty conditions
    - Spinners, skeletons, or placeholder text prevent the UI from feeling broken
    - Empty states should be informative, not just blank space
- Version Control Discipline
    - Commit small, logical changes with clear and descriptive messages
    - Use branches for new features or bug fixes
    - Makes it easier to track changes, review code, and roll back mistakes
- Avoid Deprecated Features
    - Regularly review and replace outdated HTML tags, CSS properties, and JS APIs
    - Deprecated features may be dropped by browsers without warning
    - Staying current reduces long-term technical debt