Modern Web Applications with React

Haolin Wu

UWA NLP-TLP Group

The University of Western Australia

December 4, 2024

Table of Contents

1 Brief history of the web

2 What makes up a webpage?

3 History

4 React

5 State Management

6 Testing

7 Final Considerations

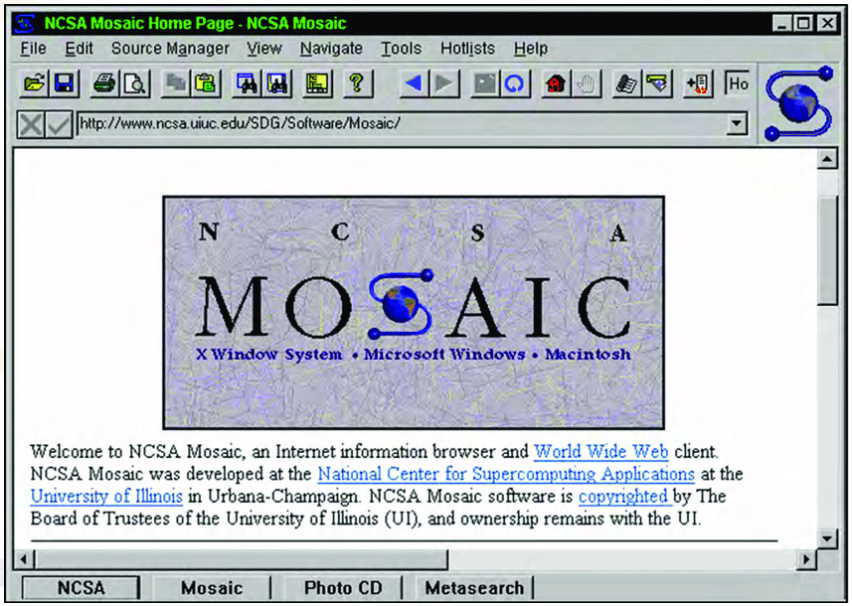
Brief history of the web

1991 - The First Website

- The first website created by Tim Berners-Lee at CERN
- Introduced HTML and HTTP

1993 - Mosaic Browser

- First graphical web browser
- Made the web visual and accessible

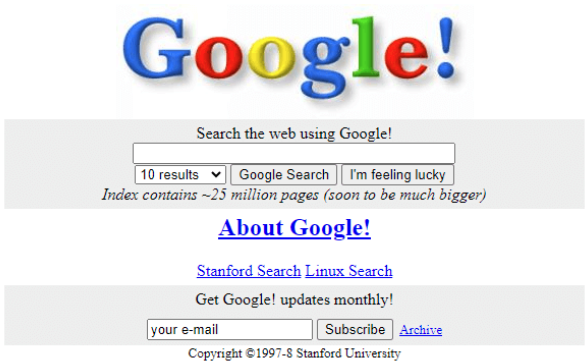


1994 - Amazon Founded

- Started as an online bookstore
- Pioneered eCommerce

1998 - Google Founded

- Revolutionized web search
- PageRank algorithm introduced



2004 - Facebook Launch

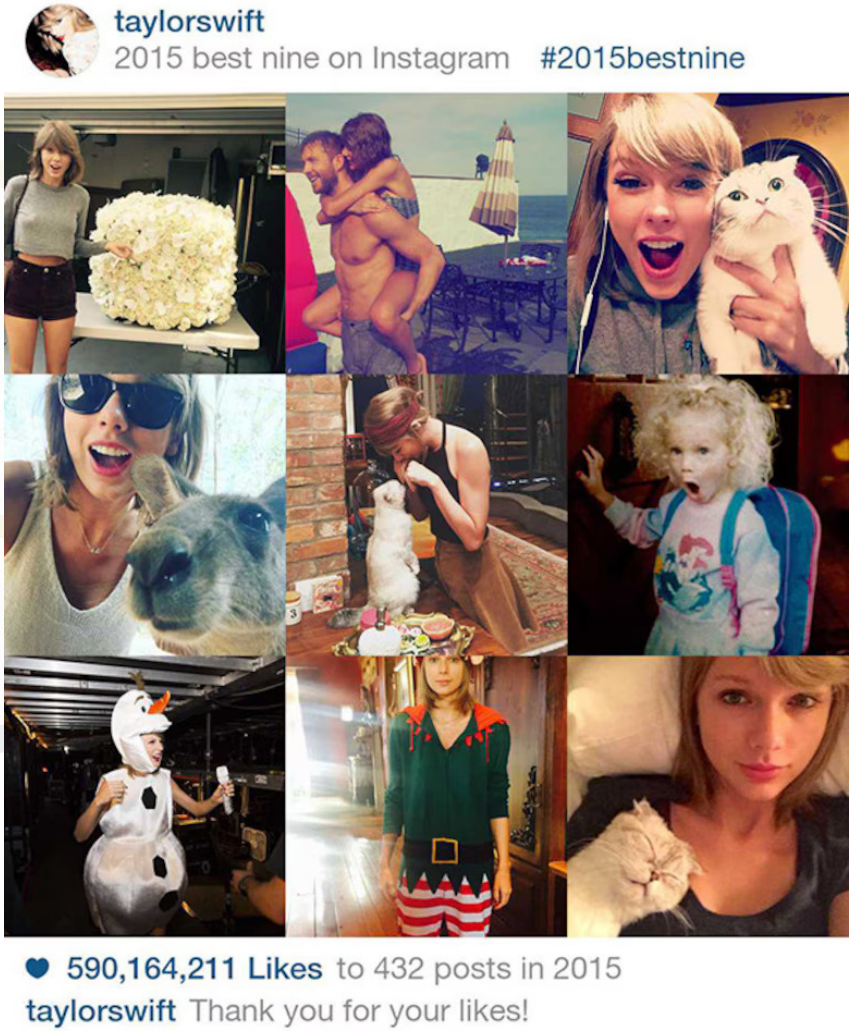
- Started at Harvard
- Social media revolution begins

2007 - iPhone Released

- Mobile web becomes accessible
- Touch interfaces transform browsing

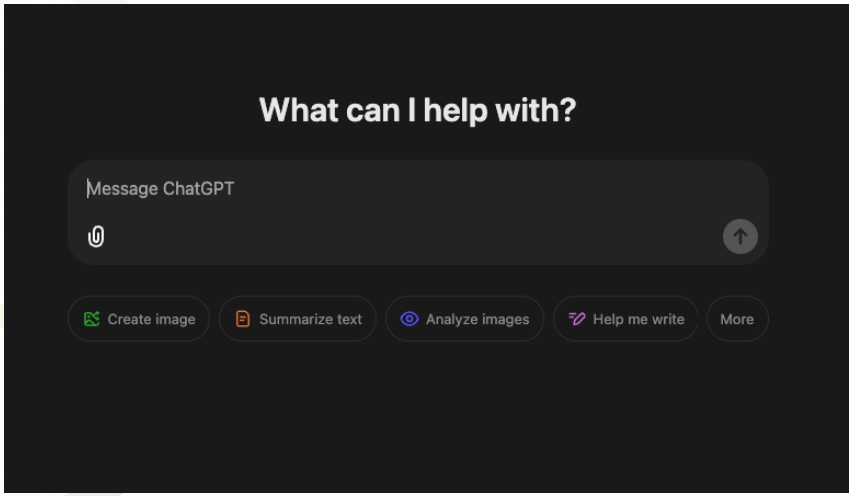
2015 - Mobile Exceeds Desktop

- Mobile traffic surpasses desktop
- Responsive design becomes crucial



2024 - Global Impact

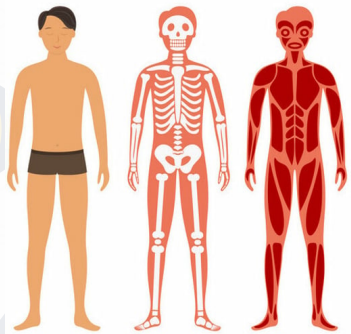
- 5 billion users globally
- Web3 and AI integration emerging



What makes up a webpage?

Core components of a webpage

- Three fundamental components:
  - HTML - Structure
  - CSS - Style
  - JavaScript - Behavior
- Think of it like a human body:
  - HTML is the skeleton
  - CSS is the appearance
  - JavaScript is the muscles & nervous system



HTML - The Structure

- Nested, tag-based language
- Each element needs opening and closing tags

```
<p>Hello</p>
```

- Tags can have attributes - each tag is an object

```
<p class="hello">Text</p>
```

CSS - The Style

- Defines the visual appearance of elements
- Uses its own language

```
.blackBackground {
  background-color: #000;
}
```

- Targets classes, specific element types or an elements attribute

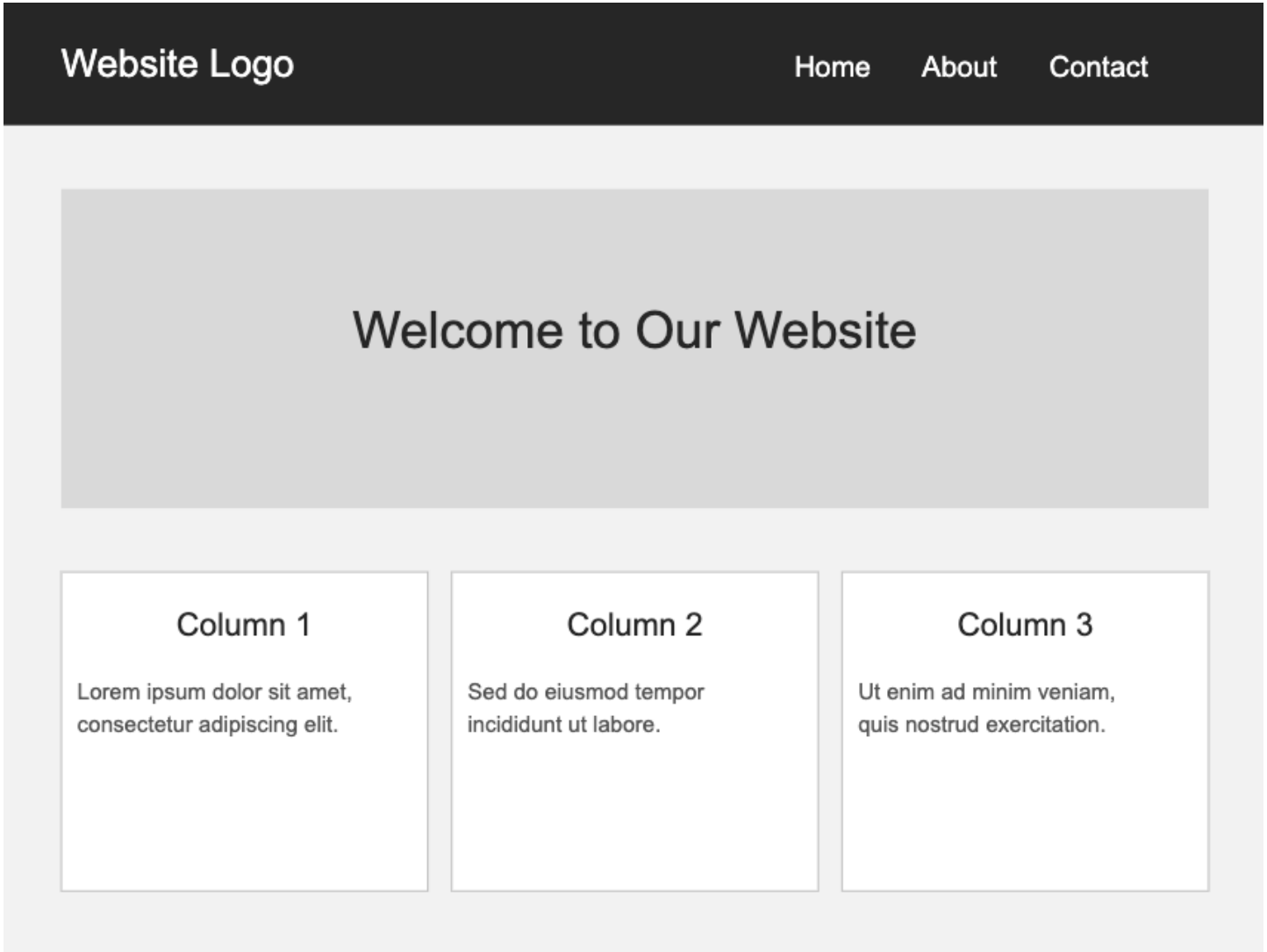
```
<p class="blackBackground">Text</p>
```

- Can be applied via style attributes directly

JavaScript - The Behavior

- The "magic" that makes webpages interactive
- Enables live changes to the page through code
- Uses the Document Object Model (DOM)
  - Entire page accessible as objects in code
  - Allows dynamic manipulation of content

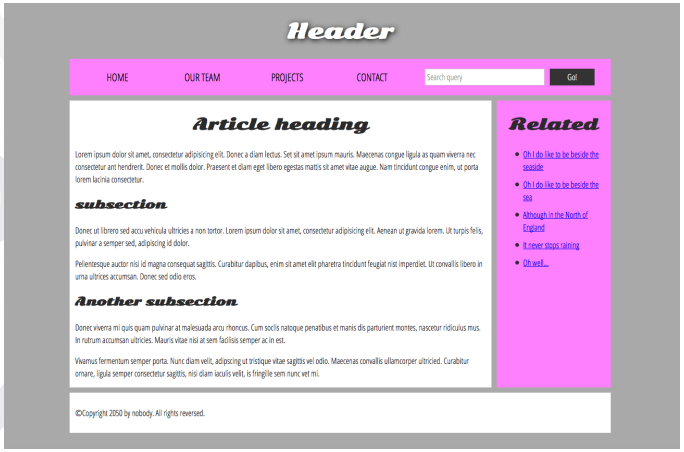
A Simple Website



History

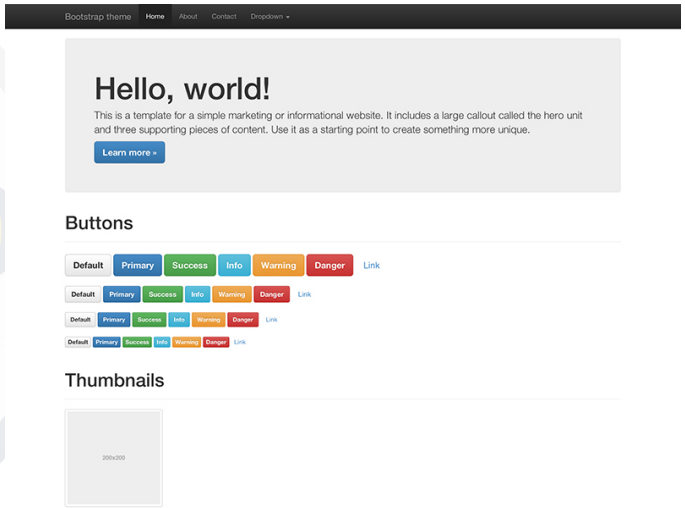
Early Web - Server-Side Era

- Basic browser capabilities
  - Limited HTML tags
  - Primitive JavaScript functionality
  - Even basic string manipulation was challenging!
- Most processing done on servers
- Static page delivery model



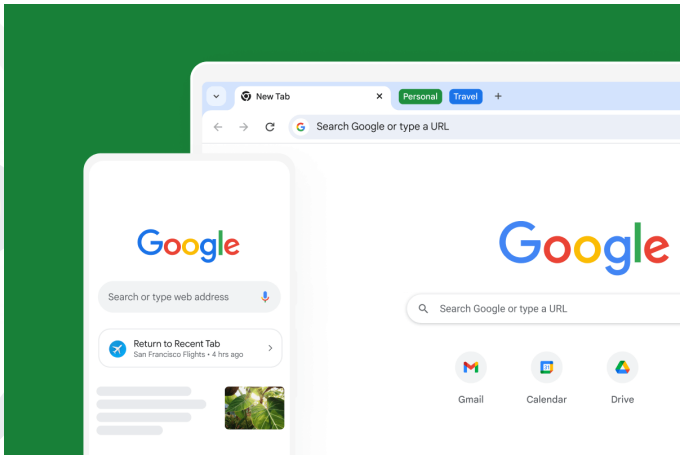
Rise of Helper Tools

- Bootstrap (2011)
  - Standardized CSS components
  - Responsive design made easy
- jQuery (2006)
  - Cross-browser compatibility
  - Simplified DOM manipulation



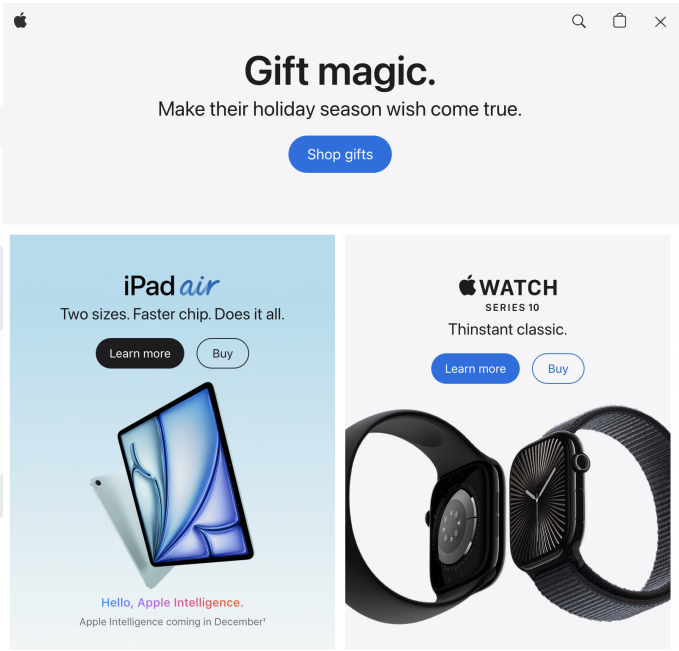
Dynamic Web Revolution

- AJAX (2005)
  - Asynchronous data loading
  - Dynamic page updates
- Hardware Evolution
  - More powerful client devices
  - Better browsers



Modern Web - Client-Side Era

- Shift to Client-Side Processing
- Modern JavaScript Frameworks
  - React (2013)
  - Angular (2010)
  - Vue (2014)
- Component-Based Architecture



Server-Side Rendering

- Server produces complete HTML for browser
- Page changes require full refresh
- Limited responsiveness
  - Navigation needs page reload
  - Actions trigger server requests
- Led to the question: How to avoid constant refreshing?

Client-Side Rendering

- JavaScript DOM manipulation
  - Create elements dynamically
  - Modify content instantly
  - Delete elements without refresh
- Responsive user experience
- Performance limited only by:
  - Code quality
  - Client hardware

Server vs Client Rendering: Trade-offs

Server-Side Benefits

- Faster initial page load
- No large JS packages
- Better SEO out of the box
- Lower client requirements

Client-Side Benefits

- Rich interactivity
- Smooth transitions
- Reduced server load
- Better user experience

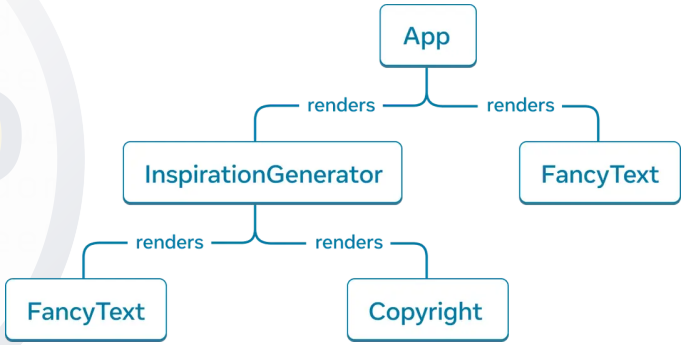
React

React: Making Client-Side Rendering Easy

- Reusable elements in familiar syntax
- JSX - HTML and JavaScript hybrid
  - Write HTML-like code in JavaScript
  - Combine markup with logic
- Smart rendering system
  - Efficient DOM updates
  - Automatic element cleanup
- Components as functions
- Changes without page refresh

React Components

- Components are:
  - Building blocks of UI
  - Reusable pieces of interface
  - JavaScript functions
- State-driven rendering
  - Show different UI based on variables
  - e.g., `isLoggedIn ? <UserDashboard /> : <LoginForm />`



Component Lifecycle

- Components have three main phases:
  - Creation (Mounting)
  - Updates (Re-rendering)
  - Deletion (Unmounting)
- State Management
  - Variables control component behavior
  - Centralized state management needed
  - Changes trigger re-renders automatically
- Next Topic: How do we manage state effectively?

Component Example

```
function UserGreeting({ username }) {
  const isLoggedIn = username != null;
  return (
    <div>
      {isLoggedIn
        ? <h1>Welcome back, {username}!</h1>
        : <h1>Please log in</h1>
      }
    </div>
  );
}
```

State Management

Understanding State

- State determines what's rendered
- Conceptually a large object

```
{
   signedIn: user,
   darkMode: true,
   currentPage: 'home'
}
```

- React re-renders based on state changes
- States can be component-specific
- Components only re-render when their state changes

Introduction to Hooks

- Functions to access component lifecycle
- Three main lifecycle phases:
  - Creation (Mounting)
  - Update (Re-rendering)
  - Deletion (Unmounting)
- Core hooks we'll explore:
  - useState
  - useEffect
  - useContext

useState Hook

- Simplest state management hook
- Provides getter and setter

```
function Counter() {
  const [count, setCount] = useState(0);
  return (
    <div>
      <p>Count: {count}</p>
      <button onClick=() => setCount(count + 1)>
        Increment
      </button>
    </div>
  );
}
```

useEffect Hook

- Runs on variable changes
- Syntax: `useEffect(function, [dependencies])`

```
useEffect(() => {
  console.log('Count changed to:', count);
}, [count]); // Runs when count changes
useEffect(() => {
  console.log('Component mounted');
}, []); // Runs once on mount
```

useContext Hook

- Global state management
- Access state from any component

```
// Create context
const ThemeContext = createContext();
// Provider component
function App() {
  const [theme, setTheme] = useState('light');
  return (
    <ThemeContext.Provider value={theme}>
      <ChildComponent />
    </ThemeContext.Provider>
  );
}
```

Testing

Testing React Apps

- No built-in testing framework
- Popular testing tools:
  - Jest - Test runner
  - React Testing Library - Component testing
  - Cypress - End-to-end testing

Jest Testing Framework

- Most popular testing solution
- Key concepts:
  - describe() - Test suite grouping
  - it()/test() - Individual test cases
  - expect() - Assertions

```
describe('Counter Component', () => {
  it('should increment count when clicked', () => {
    const { getByText } = render(<Counter />);
    const button = getByText('Increment');
    fireEvent.click(button);
    expect(getByText('Count:
1')).toBeInTheDocument();
  });
});
```

Types of Tests

Unit Tests

- Test individual components
- Isolated functionality
- Fast execution

End-to-End Tests

- Test complete user flows
- Browser simulation
- Real-world scenarios

Integration Tests

- Test component interactions
- API integration
- State management

Testing Best Practices

- Test user behavior, not implementation
- Common testing patterns:
  - Arrange - Set up test conditions
  - Act - Perform the action
  - Assert - Check the results
- Write tests that:
  - Are easy to maintain
  - Focus on critical paths
  - Mirror user interactions

Final Considerations

React vs Vanilla JavaScript

- React simplifies what's possible in vanilla JavaScript
  - Component reusability
  - State management
  - DOM manipulation
- "Don't reinvent the wheel"
  - Mature, tested solutions
  - Large community support
  - Well-documented patterns

The Trade-offs

Server-Side Rendering

- Instant initial content
- Better SEO
- No "white flash"

React (CSR)

- Initial load delay
- JS bundle download
- Richer interactions

Hybrid Solutions

- Next.js - Server-side rendered React
- Best of both worlds

Questions & Live Demo

Questions?

Static Demo: `https://static.haolin.dev`

React Demo: `https://react.haolin.dev`

Hybrid: `https://haolin.dev`