# **PART 3: Frontend Engineering**

## **Frontend Architecture**

### **Technologies**

* Framework: React.js/Next.js/Vue.js for component-based UI development
* State Management: Redux, Zustand, or Context API based on application complexity
* API Integration: Axios or fetch for API calls with proper error handling
* UI Components: Material UI, Tailwind CSS, or custom component library
* Authentication: JWT with secure token management

### **Application Structure**

frontend/

├── public/

│ ├── assets/

│ └── index.html

├── src/

│ ├── api/

│ │ ├── axios.ts

│ │ ├── conversations.ts

│ │ └── llm.ts

│ ├── components/

│ │ ├── common/

│ │ │ ├── Button.tsx

│ │ │ ├── Input.tsx

│ │ │ └── Modal.tsx

│ │ ├── conversation/

│ │ │ ├── ConversationList.tsx

│ │ │ ├── MessageBubble.tsx

│ │ │ └── MessageInput.tsx

│ │ └── layout/

│ │ ├── Header.tsx

│ │ ├── Sidebar.tsx

│ │ └── MainLayout.tsx

│ ├── contexts/

│ │ ├── AuthContext.tsx

│ │ └── ThemeContext.tsx

│ ├── hooks/

│ │ ├── useConversation.ts

│ │ ├── useLLM.ts

│ │ └── useDebounce.ts

│ ├── pages/

│ │ ├── Auth/

│ │ ├── Chat/

│ │ ├── Settings/

│ │ └── App.tsx

│ ├── state/

│ │ ├── slices/

│ │ └── store.ts

│ ├── types/

│ │ ├── conversation.ts

│ │ ├── message.ts

│ │ └── user.ts

│ ├── utils/

│ │ ├── formatting.ts

│ │ └── validation.ts

│ └── main.tsx

├── .eslintrc.js

├── tailwind.config.js

├── tsconfig.json

└── package.json

### **Best Practices**

1. Component Structure:

// Example component using atomic design principles

// atoms/Button.tsx

interface ButtonProps {

variant: 'primary' | 'secondary';

size: 'sm' | 'md' | 'lg';

children: React.ReactNode;

onClick?: () => void;

disabled?: boolean;

}

export const Button: React.FC<ButtonProps> = ({

variant = 'primary',

size = 'md',

children,

onClick,

disabled = false

}) => {

const baseClasses = "rounded font-medium transition-colors";

const variantClasses = {

primary: "bg-blue-600 text-white hover:bg-blue-700",

secondary: "bg-gray-200 text-gray-800 hover:bg-gray-300"

};

const sizeClasses = {

sm: "text-xs px-2 py-1",

md: "text-sm px-3 py-2",

lg: "text-base px-4 py-2"

};

return (

<button

className={`${baseClasses} ${variantClasses[variant]} ${sizeClasses[size]}`}

onClick={onClick}

disabled={disabled}

>

{children}

</button>

);

};

* + Use atomic design principles (atoms, molecules, organisms, templates, pages)
  + Implement lazy loading for performance optimization
  + Keep components small and focused on single responsibilities
  + Use TypeScript for type safety

1. State Management:

// Using React Context for global state

// contexts/ConversationContext.tsx

interface ConversationContextType {

conversations: Conversation[];

activeConversation: Conversation | null;

isLoading: boolean;

error: string | null;

setActiveConversation: (id: string) => void;

createConversation: () => Promise<void>;

sendMessage: (content: string) => Promise<void>;

}

export const ConversationContext = createContext<ConversationContextType | undefined>(undefined);

export const ConversationProvider: React.FC<{children: React.ReactNode}> = ({ children }) => {

const [conversations, setConversations] = useState<Conversation[]>([]);

const [activeConversation, setActiveConversation] = useState<Conversation | null>(null);

const [isLoading, setIsLoading] = useState(false);

const [error, setError] = useState<string | null>(null);

// Implement context methods...

return (

<ConversationContext.Provider value={{

conversations,

activeConversation,

isLoading,

error,

setActiveConversation: (id) => {/\* implementation \*/},

createConversation: async () => {/\* implementation \*/},

sendMessage: async (content) => {/\* implementation \*/}

}}>

{children}

</ConversationContext.Provider>

);

};

* + Centralize application state for easier management
  + Implement proper error and loading states
  + Use local state for component-specific data
  + Consider performance optimizations (context splitting, memoization)

1. API Integration:

// api/axios.ts

import axios from 'axios';

const baseURL = process.env.REACT\_APP\_API\_URL || 'http://localhost:8000/api/v1';

const api = axios.create({

baseURL,

headers: {

'Content-Type': 'application/json',

},

});

// Request interceptor for auth token

api.interceptors.request.use(

(config) => {

const token = localStorage.getItem('token');

if (token) {

config.headers.Authorization = `Bearer ${token}`;

}

return config;

},

(error) => Promise.reject(error)

);

// Response interceptor for error handling

api.interceptors.response.use(

(response) => response,

(error) => {

// Handle token expiration

if (error.response?.status === 401) {

localStorage.removeItem('token');

window.location.href = '/login';

}

return Promise.reject(error);

}

);

export default api;

* + Create a centralized API client
  + Implement request/response interceptors
  + Handle authentication and error cases
  + Use environment variables for configuration

1. Performance:
   * Implement code splitting with lazy loading

// Lazy loading routes

const Settings = React.lazy(() => import('./pages/Settings'));

function App() {

return (

<Suspense fallback={<div>Loading...</div>}>

<Routes>

<Route path="/settings" element={<Settings />} />

</Routes>

</Suspense>

);

}

* + Optimize renders with React.memo and useMemo
  + Implement virtualization for long lists
  + Use web workers for computationally intensive tasks

1. User Experience:
   * Design responsive layouts for all device sizes
   * Implement skeleton screens for loading states
   * Provide immediate feedback for user actions
   * Ensure accessibility compliance (WCAG guidelines)

## **Frontend Infrastructure**

### **Static Hosting**

1. AWS Options
   * S3 for static website hosting
   * CloudFront for CDN and edge caching
   * Route 53 for DNS management
   * Certificate Manager for SSL certificates
2. GCP Options
   * Cloud Storage for static website hosting
   * Cloud CDN for content delivery
   * Cloud DNS for domain management
   * Certificate Manager for SSL certificates

### **Build and Deployment**

1. Build Process

# Example GitHub Actions workflow for frontend deployment

name: Deploy Frontend

on:

push:

branches: [main]

paths:

- 'frontend/\*\*'

jobs:

build-and-deploy:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v2

- name: Set up Node.js

uses: actions/setup-node@v2

with:

node-version: '16'

- name: Install dependencies

run: |

cd frontend

npm ci

- name: Build

run: |

cd frontend

npm run build

env:

REACT\_APP\_API\_URL: ${{ secrets.API\_URL }}

- name: Deploy to S3

uses: jakejarvis/s3-sync-action@master

with:

args: --acl public-read --follow-symlinks --delete

env:

AWS\_S3\_BUCKET: ${{ secrets.AWS\_S3\_BUCKET }}

AWS\_ACCESS\_KEY\_ID: ${{ secrets.AWS\_ACCESS\_KEY\_ID }}

AWS\_SECRET\_ACCESS\_KEY: ${{ secrets.AWS\_SECRET\_ACCESS\_KEY }}

AWS\_REGION: 'us-east-1'

SOURCE\_DIR: 'frontend/build'

- name: Invalidate CloudFront

uses: chetan/invalidate-cloudfront-action@v2

env:

DISTRIBUTION: ${{ secrets.CLOUDFRONT\_DISTRIBUTION\_ID }}

PATHS: '/\*'

AWS\_REGION: 'us-east-1'

AWS\_ACCESS\_KEY\_ID: ${{ secrets.AWS\_ACCESS\_KEY\_ID }}

AWS\_SECRET\_ACCESS\_KEY: ${{ secrets.AWS\_SECRET\_ACCESS\_KEY }}