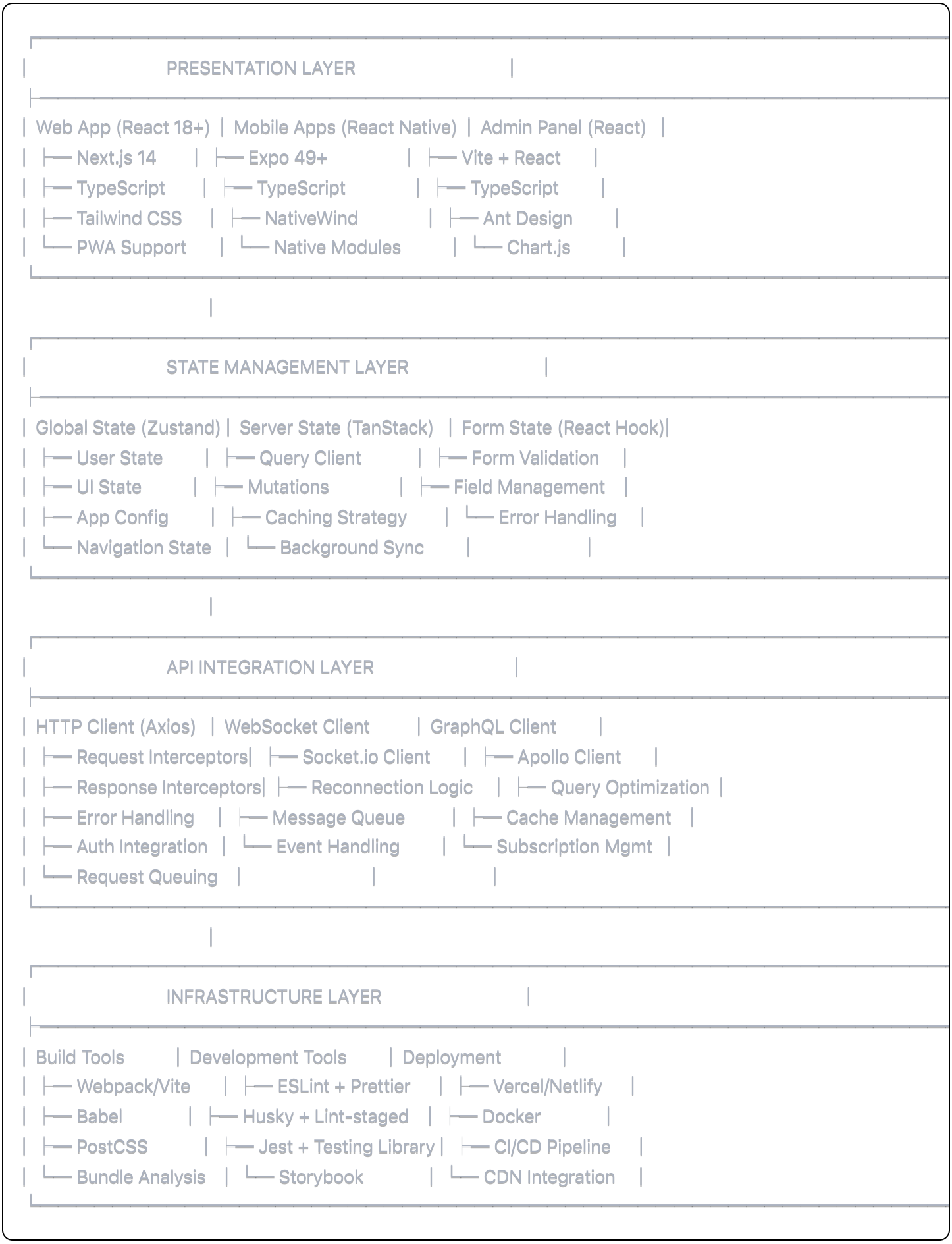
# Frontend Technical Architecture Deep Dive

## 1. Frontend Layer Architecture Overview




## 2. Component Architecture Pattern

## 2.1 Atomic Design System

typescript

```
// Atomic Structure
src/
├── components/
│   ├── atoms/          // Basic building blocks
│   │   ├── Button/
│   │   ├── Input/
│   │   ├── Icon/
│   │   └── Typography/
│   ├── molecules/      // Simple component combinations
│   │   ├── SearchBox/
│   │   ├── UserCard/
│   │   └── EventCard/
│   ├── organisms/      // Complex UI sections
│   │   ├── Header/
│   │   ├── EventList/
│   │   └── UserProfile/
│   ├── templates/      // Page layouts
│   │   ├── DashboardLayout/
│   │   └── EventLayout/
│   └── pages/          // Route components
│       ├── HomePage/
│       ├── EventPage/
│       └── ProfilePage/
```

## 2.2 Component Structure Pattern

typescript

```
// Component Template Structure
interface ComponentProps {
  // Props interface
}

interface ComponentState {
  // Local state interface
}

const Component: React.FC<ComponentProps> = ({
  // Destructured props
}) => {
  // Hooks (state, effects, custom hooks)
  // Event handlers
  // Computed values
  // Render logic

  return (
    // JSX with proper TypeScript typing
  );
};

export default Component;
```

## 3. State Management Architecture

### 3.1 Global State (Zustand)

typescript

```typescript
// stores/userStore.ts
interface UserState {
  user: User | null;
  isAuthenticated: boolean;
  preferences: UserPreferences;
  setUser: (user: User) => void;
  logout: () => void;
  updatePreferences: (prefs: Partial<UserPreferences>) => void;
}

export const useUserStore = create<UserState>((set, get) => ({
  user: null,
  isAuthenticated: false,
  preferences: defaultPreferences,

  setUser: (user) => set({ user, isAuthenticated: true }),
  logout: () => set({ user: null, isAuthenticated: false }),
  updatePreferences: (prefs) => set(state => ({
    preferences: { ...state.preferences, ...prefs }
  }))
}));

// stores/uiStore.ts
interface UIState {
  theme: 'light' | 'dark';
  sidebarOpen: boolean;
  modals: Record<string, boolean>;
  notifications: Notification[];
  toggleTheme: () => void;
  toggleSidebar: () => void;
  openModal: (modalId: string) => void;
  closeModal: (modalId: string) => void;
  addNotification: (notification: Notification) => void;
  removeNotification: (id: string) => void;
}
```

### 3.2 Server State (TanStack Query)

typescript

```
// hooks/api/useEvents.ts
export const useEvents = (filters?: EventFilters) => {
  return useQuery({
    queryKey: ['events', filters],
    queryFn: () => eventService.getEvents(filters),
    staleTime: 5 * 60 * 1000, // 5 minutes
    cacheTime: 10 * 60 * 1000, // 10 minutes
    refetchOnWindowFocus: false,
    retry: 3,
  });
};

export const useCreateEvent = () => {
  const queryClient = useQueryClient();

  return useMutation({
    mutationFn: eventService.createEvent,
    onSuccess: () => {
      queryClient.invalidateQueries(['events']);
      queryClient.invalidateQueries(['user-events']);
    },
    onError: (error) => {
      // Handle error with toast notification
      toast.error(error.message);
    }
  });
};

// React Query Configuration
const queryClient = new QueryClient({
  defaultOptions: {
    queries: {
      retry: (failureCount, error) => {
        if (error.status === 404) return false;
        return failureCount < 3;
      },
      staleTime: 5 * 60 * 1000,
      cacheTime: 10 * 60 * 1000,
    },
    mutations: {
      retry: 1,
    }
```

```
  }
});
```

## 4. API Integration Layer

### 4.1 HTTP Client Configuration

typescript

```typescript
// services/api/client.ts
import axios, { AxiosInstance, AxiosRequestConfig } from 'axios';
import { useUserStore } from '@/stores/userStore';

class ApiClient {
  private client: AxiosInstance;
  private requestQueue: Array<() => Promise<any>> = [];
  private isRefreshing = false;

  constructor() {
    this.client = axios.create({
      baseURL: process.env.NEXT_PUBLIC_API_BASE_URL,
      timeout: 30000,
      headers: {
        'Content-Type': 'application/json',
      }
    });

    this.setupInterceptors();
  }

  private setupInterceptors() {
    // Request Interceptor
    this.client.interceptors.request.use(
      (config) => {
        const token = useUserStore.getState().user?.token;
        if (token) {
          config.headers.Authorization = `Bearer ${token}`;
        }

        // Add request ID for tracing
        config.headers['X-Request-ID'] = generateRequestId();

        return config;
      },
      (error) => Promise.reject(error)
    );

    // Response Interceptor
    this.client.interceptors.response.use(
      (response) => response,
      async (error) => {
        const originalRequest = error.config;
```

```
    if (error.response?.status === 401 && !originalRequest._retry) {
      if (this.isRefreshing) {
        return new Promise((resolve) => {
          this.requestQueue.push(() => resolve(this.client(originalRequest)));
        });
      }

      originalRequest._retry = true;
      this.isRefreshing = true;

      try {
        await this.refreshToken();
        this.processQueue();
        return this.client(originalRequest);
      } catch (refreshError) {
        this.clearQueue();
        useUserStore.getState().logout();
        window.location.href = '/login';
        return Promise.reject(refreshError);
      } finally {
        this.isRefreshing = false;
      }
    }

    return Promise.reject(this.handleError(error));
  }
 );
}

private handleError(error: any) {
  const errorResponse = {
    message: error.response?.data?.message || 'An error occurred',
    status: error.response?.status,
    code: error.response?.data?.code,
    details: error.response?.data?.details
  };

  // Log error for monitoring
  console.error('API Error:', errorResponse);

  return errorResponse;
}
```

```typescript
  async get<T>(url: string, config?: AxiosRequestConfig): Promise<T> {
    const response = await this.client.get(url, config);
    return response.data;
  }

  async post<T>(url: string, data?: any, config?: AxiosRequestConfig): Promise<T> {
    const response = await this.client.post(url, data, config);
    return response.data;
  }

  async put<T>(url: string, data?: any, config?: AxiosRequestConfig): Promise<T> {
    const response = await this.client.put(url, data, config);
    return response.data;
  }

  async delete<T>(url: string, config?: AxiosRequestConfig): Promise<T> {
    const response = await this.client.delete(url, config);
    return response.data;
  }
}

export const apiClient = new ApiClient();
```

### 4.2 Service Layer Pattern

typescript

```typescript
// services/eventService.ts
class EventService {
  private readonly basePath = '/events';

  async getEvents(filters?: EventFilters): Promise<PaginatedResponse<Event>> {
    const params = new URLSearchParams();

    if (filters) {
      Object.entries(filters).forEach(([key, value]) => {
        if (value !== undefined && value !== null) {
          params.append(key, String(value));
        }
      });
    }

    return apiClient.get(`${this.basePath}?${params.toString()}`);
  }

  async getEvent(id: string): Promise<Event> {
    return apiClient.get(`${this.basePath}/${id}`);
  }

  async createEvent(event: CreateEventRequest): Promise<Event> {
    return apiClient.post(this.basePath, event);
  }

  async updateEvent(id: string, event: UpdateEventRequest): Promise<Event> {
    return apiClient.put(`${this.basePath}/${id}`, event);
  }

  async deleteEvent(id: string): Promise<void> {
    return apiClient.delete(`${this.basePath}/${id}`);
  }

  async joinEvent(eventId: string): Promise<void> {
    return apiClient.post(`${this.basePath}/${eventId}/join`);
  }

  async leaveEvent(eventId: string): Promise<void> {
    return apiClient.delete(`${this.basePath}/${eventId}/join`);
  }
}
```

```
export const eventService = new EventService();
```

## 4.3 WebSocket Integration

typescript

```typescript
// services/websocket/socketClient.ts
import { io, Socket } from 'socket.io-client';
import { useUserStore } from '@/stores/userStore';

class SocketClient {
  private socket: Socket | null = null;
  private reconnectAttempts = 0;
  private maxReconnectAttempts = 5;
  private reconnectDelay = 1000;

  connect() {
    const token = useUserStore.getState().user?.token;

    this.socket = io(process.env.NEXT_PUBLIC_WEBSOCKET_URL!, {
      auth: { token },
      transports: ['websocket'],
      upgrade: true,
      rememberUpgrade: true,
    });

    this.setupEventHandlers();
  }

  private setupEventHandlers() {
    if (!this.socket) return;

    this.socket.on('connect', () => {
      console.log('WebSocket connected');
      this.reconnectAttempts = 0;
    });

    this.socket.on('disconnect', (reason) => {
      console.log('WebSocket disconnected:', reason);
      if (reason === 'io server disconnect') {
        this.socket?.connect();
      }
    });

    this.socket.on('connect_error', (error) => {
      console.error('WebSocket connection error:', error);
      this.handleReconnection();
    });
```

```
  // Event-specific handlers
  this.socket.on('event:update', this.handleEventUpdate);
  this.socket.on('notification:new', this.handleNewNotification);
  this.socket.on('user:status', this.handleUserStatusUpdate);
}

private handleReconnection() {
  if (this.reconnectAttempts < this.maxReconnectAttempts) {
    this.reconnectAttempts++;
    setTimeout(() => {
      this.socket?.connect();
    }, this.reconnectDelay * this.reconnectAttempts);
  }
}

private handleEventUpdate = (data: EventUpdatePayload) => {
  // Update React Query cache
  queryClient.setQueryData(['events', data.eventId], data.event);

  // Show notification if user is involved
  const userStore = useUserStore.getState();
  if (data.affectedUsers?.includes(userStore.user?.id)) {
    toast.info(`Event "${data.event.title}" has been updated`);
  }
};

private handleNewNotification = (notification: Notification) => {
  const uiStore = useUIStore.getState();
  uiStore.addNotification(notification);

  // Show browser notification if permission granted
  if (Notification.permission === 'granted') {
    new Notification(notification.title, {
      body: notification.message,
      icon: '/icon-192x192.png',
      tag: notification.id
    });
  }
};

emit(event: string, data: any) {
  this.socket?.emit(event, data);
}
```

```
  disconnect() {
    this.socket?.disconnect();
    this.socket = null;
  }
}

export const socketClient = new SocketClient();

// Custom hook for WebSocket
export const useSocket = () => {
  const [isConnected, setIsConnected] = useState(false);

  useEffect(() => {
    socketClient.connect();

    const handleConnect = () => setIsConnected(true);
    const handleDisconnect = () => setIsConnected(false);

    socketClient.socket?.on('connect', handleConnect);
    socketClient.socket?.on('disconnect', handleDisconnect);

    return () => {
      socketClient.socket?.off('connect', handleConnect);
      socketClient.socket?.off('disconnect', handleDisconnect);
      socketClient.disconnect();
    };
  }, []);

  return {
    isConnected,
    emit: socketClient.emit.bind(socketClient)
  };
};
```

## 5. Mobile App Specific Architecture

### 5.1 React Native Structure

typescript

```
// React Native Specific Features
src/
├── native/
│   ├── modules/        // Native modules
│   │   ├── LocationModule/
│   │   ├── CameraModule/
│   │   └── PushNotifications/
│   ├── navigation/     // React Navigation
│   │   ├── AppNavigator/
│   │   ├── AuthNavigator/
│   │   └── TabNavigator/
│   ├── permissions/    // Permission handling
│   └── storage/        // Async Storage, MMKV
```

## 5.2 Navigation Setup

typescript

```
// navigation/AppNavigator.tsx
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';

const Stack = createNativeStackNavigator();
const Tab = createBottomTabNavigator();

export const AppNavigator = () => {
  const { isAuthenticated } = useUserStore();

  return (
    <NavigationContainer>
      {isAuthenticated ? <AuthenticatedNavigator /> : <AuthNavigator />}
    </NavigationContainer>
  );
};

const AuthenticatedNavigator = () => (
  <Tab.Navigator>
    <Tab.Screen name="Home" component={HomeScreen} />
    <Tab.Screen name="Events" component={EventsScreen} />
    <Tab.Screen name="Profile" component={ProfileScreen} />
  </Tab.Navigator>
);
```

## 6. Performance Optimization

### 6.1 Code Splitting Strategy

typescript

```
// Dynamic imports for route-based code splitting
const HomePage = lazy(() => import('@/pages/HomePage'));
const EventPage = lazy(() => import('@/pages/EventPage'));
const ProfilePage = lazy(() => import('@/pages/ProfilePage'));

// Component-based code splitting
const EventMap = lazy(() => import('@/components/EventMap'));

// Preload critical routes
const preloadRoutes = () => {
  import('@/pages/EventPage');
  import('@/pages/ProfilePage');
};
```

## 6.2 Caching Strategy

typescript

```
// Service Worker for caching
// sw.js
const CACHE_NAME = 'app-v1';
const STATIC_ASSETS = [
  '/',
  '/static/js/bundle.js',
  '/static/css/main.css',
  '/manifest.json'
];

self.addEventListener('install', (event) => {
  event.waitUntil(
    caches.open(CACHE_NAME)
      .then(cache => cache.addAll(STATIC_ASSETS))
  );
});

// API response caching
self.addEventListener('fetch', (event) => {
  if (event.request.url.includes('/api/')) {
    event.respondWith(
      caches.open('api-cache').then(cache => {
        return cache.match(event.request).then(response => {
          if (response) {
            // Serve from cache, but also fetch update
            fetch(event.request).then(fetchResponse => {
              cache.put(event.request, fetchResponse.clone());
            });
            return response;
          }
          return fetch(event.request).then(fetchResponse => {
            cache.put(event.request, fetchResponse.clone());
            return fetchResponse;
          });
        });
      })
    );
  }
});
```

## 6.3 Image Optimization

typescript

```tsx
// components/OptimizedImage.tsx
interface OptimizedImageProps {
  src: string;
  alt: string;
  width?: number;
  height?: number;
  priority?: boolean;
  sizes?: string;
}

export const OptimizedImage: React.FC<OptimizedImageProps> = ({
  src,
  alt,
  width,
  height,
  priority = false,
  sizes = '100vw'
}) => {
  const [isLoaded, setIsLoaded] = useState(false);
  const [error, setError] = useState(false);

  return (
    <div className="relative overflow-hidden">
      {!isLoaded && !error && (
        <div className="absolute inset-0 bg-gray-200 animate-pulse" />
      )}

      <Image
        src={src}
        alt={alt}
        width={width}
        height={height}
        priority={priority}
        sizes={sizes}
        className={`transition-opacity duration-300 ${
          isLoaded ? 'opacity-100' : 'opacity-0'
        }`}
        onLoad={() => setIsLoaded(true)}
        onError={() => setError(true)}
      />

      {error && (
        <div className="absolute inset-0 bg-gray-100 flex items-center justify-center">
```

```
        <span className="text-gray-400">Failed to load image</span>
      </div>
    )}
  </div>
 );
};
```

## 7. Error Handling & Monitoring

### 7.1 Error Boundary

typescript

```tsx
// components/ErrorBoundary.tsx
interface ErrorBoundaryState {
  hasError: boolean;
  error?: Error;
}

export class ErrorBoundary extends Component<
  PropsWithChildren<{}>,
  ErrorBoundaryState
> {
  constructor(props: PropsWithChildren<{}>) {
    super(props);
    this.state = { hasError: false };
  }

  static getDerivedStateFromError(error: Error): ErrorBoundaryState {
    return { hasError: true, error };
  }

  componentDidCatch(error: Error, errorInfo: ErrorInfo) {
    // Log to monitoring service
    console.error('Error Boundary caught an error:', error, errorInfo);

    // Send to error tracking service
    if (process.env.NODE_ENV === 'production') {
      // Sentry, LogRocket, etc.
      this.logError(error, errorInfo);
    }
  }

  private logError(error: Error, errorInfo: ErrorInfo) {
    // Implementation depends on monitoring service
  }

  render() {
    if (this.state.hasError) {
      return (
        <div className="min-h-screen flex items-center justify-center">
          <div className="text-center">
            <h2 className="text-2xl font-bold mb-4">Something went wrong</h2>
            <p className="text-gray-600 mb-4">
              We apologize for the inconvenience. Please try refreshing the page.
            </p>
```

```
          <button
            onClick={() => window.location.reload()}
            className="bg-blue-500 text-white px-4 py-2 rounded hover:bg-blue-600"
          >
            Refresh Page
          </button>
        </div>
      </div>
    );
  }

  return this.props.children;
 }
}
```

## 7.2 Global Error Handler

typescript

```typescript
// utils/errorHandler.ts
interface ErrorContext {
  component?: string;
  action?: string;
  userId?: string;
  metadata?: Record<string, any>;
}

class ErrorHandler {
  private static instance: ErrorHandler;

  static getInstance(): ErrorHandler {
    if (!ErrorHandler.instance) {
      ErrorHandler.instance = new ErrorHandler();
    }
    return ErrorHandler.instance;
  }

  handleError(error: Error, context?: ErrorContext) {
    const errorData = {
      message: error.message,
      stack: error.stack,
      timestamp: new Date().toISOString(),
      url: window.location.href,
      userAgent: navigator.userAgent,
      context
    };

    // Log locally
    console.error('Application Error:', errorData);

    // Send to monitoring service in production
    if (process.env.NODE_ENV === 'production') {
      this.sendToMonitoring(errorData);
    }

    // Show user-friendly message
    this.showUserNotification(error);
  }

  private sendToMonitoring(errorData: any) {
    // Implementation for error monitoring service
    fetch('/api/errors', {
```

```
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(errorData)
    }).catch(() => {
      // Fail silently for error reporting
    });
  }

  private showUserNotification(error: Error) {
    const userFriendlyMessage = this.getUserFriendlyMessage(error);
    toast.error(userFriendlyMessage);
  }

  private getUserFriendlyMessage(error: Error): string {
    if (error.message.includes('Network')) {
      return 'Network connection issue. Please check your internet connection.';
    }
    if (error.message.includes('401')) {
      return 'Session expired. Please log in again.';
    }
    if (error.message.includes('403')) {
      return 'You do not have permission to perform this action.';
    }
    if (error.message.includes('404')) {
      return 'The requested resource was not found.';
    }
    if (error.message.includes('500')) {
      return 'Server error. Please try again later.';
    }
    return 'An unexpected error occurred. Please try again.';
  }
}

export const errorHandler = ErrorHandler.getInstance();

// Global error event listeners
window.addEventListener('error', (event) => {
  errorHandler.handleError(event.error, {
    component: 'Global',
    action: 'Runtime Error'
  });
});

window.addEventListener('unhandledrejection', (event) => {
```

```
  errorHandler.handleError(new Error(event.reason), {
    component: 'Global',
    action: 'Unhandled Promise Rejection'
  });
});
```

## 8. Testing Strategy

### 8.1 Testing Structure

typescript

```
// tests/
├── __mocks__/        // Mock implementations
├── fixtures/         // Test data
├── utils/            // Test utilities
├── unit/             // Unit tests
├── integration/      // Integration tests
└── e2e/              // End-to-end tests
```

### 8.2 Component Testing Example

typescript

```tsx
// components/EventCard/EventCard.test.tsx
import { render, screen, fireEvent, waitFor } from '@testing-library/react';
import { QueryClient, QueryClientProvider } from '@tanstack/react-query';
import { EventCard } from './EventCard';

const createTestQueryClient = () => new QueryClient({
  defaultOptions: {
    queries: { retry: false },
    mutations: { retry: false }
  }
});

const renderWithProviders = (component: React.ReactElement) => {
  const queryClient = createTestQueryClient();
  return render(
    <QueryClientProvider client={queryClient}>
      {component}
    </QueryClientProvider>
  );
};

describe('EventCard', () => {
  const mockEvent = {
    id: '1',
    title: 'Test Event',
    description: 'Test Description',
    date: '2024-01-01T10:00:00Z',
    location: 'Test Location'
  };

  it('renders event information correctly', () => {
    renderWithProviders(<EventCard event={mockEvent} />);

    expect(screen.getByText('Test Event')).toBeInTheDocument();
    expect(screen.getByText('Test Description')).toBeInTheDocument();
    expect(screen.getByText('Test Location')).toBeInTheDocument();
  });

  it('handles join event action', async () => {
    const mockJoinEvent = jest.fn().mockResolvedValue({});

    renderWithProviders(
      <EventCard event={mockEvent} onJoinEvent={mockJoinEvent} />
```

```
    );

    const joinButton = screen.getByText('Join Event');
    fireEvent.click(joinButton);

    await waitFor(() => {
      expect(mockJoinEvent).toHaveBeenCalledWith(mockEvent.id);
    });
  });
});
```

This architecture provides a robust, scalable frontend that effectively connects to your microservices backend through multiple communication channels (HTTP, WebSocket, potentially GraphQL), with comprehensive error handling, performance optimization, and testing strategies.