Modern Navigation Helper - Usage Guide

🚀 Why Functional Approach?

Advantages over Class-based Helper:

1. **Better Tree Shaking**: Only import functions you need
2. **Hooks Integration**: Natural React hooks workflow
3. **Composability**: Easy to combine with other hooks
4. **Testing**: Easier to mock and test individual functions
5. **Performance**: No class instantiation overhead
6. **TypeScript**: Better type inference and safety

📖 Basic Usage

1. Simple Navigation in Components

typescript

```typescript
// screens/HomeScreen.tsx
import React from 'react';
import { View, Button } from 'react-native';
import { useNavigationHelper } from '../hooks/useNavigationHelper';

const HomeScreen = () => {
  const { navigateToProperties, navigateToProfile, goBack } = useNavigationHelper();

  return (
    <View>
      <Button
        title="View Properties"
        onPress={() => navigateToProperties()}
      />
      <Button
        title="My Profile"
        onPress={navigateToProfile}
      />
      <Button
        title="Go Back"
        onPress={goBack}
      />
    </View>
  );
};
```

2. Navigation with Parameters

typescript

```typescript
// screens/PropertyListScreen.tsx
import React from 'react';
import { FlatList, TouchableOpacity } from 'react-native';
import { useNavigationHelper } from '../hooks/useNavigationHelper';

const PropertyListScreen = () => {
 const { navigateToPropertyDetails, navigateToEditProperty } = useNavigationHelper();

 const handlePropertyPress = (property: Property) => {
  navigateToPropertyDetails(property.id, property.title);
 };

 const handleEditPress = (propertyId: string) => {
  navigateToEditProperty(propertyId);
 };

 return (
  <FlatList
   data={properties}
   renderItem={({ item }) => (
    <TouchableOpacity onPress={() => handlePropertyPress(item)}>
     {/* Property card content */}
    </TouchableOpacity>
   )}
  />
 );
};
```

3. Specialized Hooks Usage

typescript

```typescript
// screens/AuthScreen.tsx
import React from 'react';
import { useAuthNavigation } from '../hooks/useNavigationHelper';

const AuthScreen = () => {
 // Only import auth-related navigation functions
 const {
  navigateToLogin,
  navigateToRegister,
  navigateToForgotPassword
 } = useAuthNavigation();

 return (
  <View>
   <Button title="Login" onPress={navigateToLogin} />
   <Button title="Register" onPress={navigateToRegister} />
   <Button title="Forgot Password" onPress={navigateToForgotPassword} />
  </View>
 );
};
```

🎯 Advanced Usage Patterns

1. Conditional Navigation

typescript

```typescript
// components/PropertyCard.tsx
import React from 'react';
import { useNavigationHelper } from '../hooks/useNavigationHelper';
import { useAuth } from '../hooks/useAuth';

const PropertyCard = ({ property, isOwner }) => {
 const {
  navigateToPropertyDetails,
  navigateToMyPropertyDetails,
  navigateToLogin
 } = useNavigationHelper();
 const { isAuthenticated } = useAuth();

 const handleCardPress = () => {
  if (!isAuthenticated) {
   navigateToLogin();
   return;
  }

  if (isOwner) {
   navigateToMyPropertyDetails(property.id);
  } else {
   navigateToPropertyDetails(property.id, property.title);
  }
 };

 return (
  <TouchableOpacity onPress={handleCardPress}>
   {/* Card content */}
  </TouchableOpacity>
 );
};
```

2. Navigation with State Management

typescript

```typescript
// hooks/usePropertyActions.ts
import { useNavigationHelper } from './useNavigationHelper';
import { useAppDispatch } from './redux';
import { setSelectedProperty } from '../store/propertySlice';

export const usePropertyActions = () => {
  const { navigateToPropertyDetails } = useNavigationHelper();
  const dispatch = useAppDispatch();

  const viewPropertyDetails = (property: Property) => {
    // Update global state
    dispatch(setSelectedProperty(property));

    // Navigate to details
    navigateToPropertyDetails(property.id, property.title);
  };

  return {
    viewPropertyDetails,
  };
};

// Usage in component
const PropertyList = () => {
  const { viewPropertyDetails } = usePropertyActions();

  return (
    <FlatList
      data={properties}
      renderItem={({ item }) => (
        <TouchableOpacity onPress={() => viewPropertyDetails(item)}>
          {/* Property item */}
        </TouchableOpacity>
      )}
    />
  );
};
```

3. Navigation Guards with Hooks

typescript

```typescript
// hooks/useNavigationGuard.ts
import { useEffect } from 'react';
import { useNavigationHelper } from './useNavigationHelper';
import { useAuth } from './useAuth';
import { useUser } from './useUser';

export const useNavigationGuard = (
  requiredPermissions?: string[],
  redirectTo?: string
) => {
  const { isAuthenticated } = useAuth();
  const { user } = useUser();
  const { navigateToLogin, resetToHome } = useNavigationHelper();

  useEffect(() => {
    if (!isAuthenticated) {
      navigateToLogin();
      return;
    }

    if (requiredPermissions && user) {
      const hasPermissions = requiredPermissions.every(permission =>
        user.permissions.includes(permission)
      );

      if (!hasPermissions) {
        redirectTo ? navigation.navigate(redirectTo) : resetToHome();
      }
    }
  }, [isAuthenticated, user, requiredPermissions]);
};

// Usage in protected screens
const AdminScreen = () => {
  useNavigationGuard(['admin', 'user_management']);

  return <AdminDashboard />;
};
```

4. Navigation with Analytics

typescript

```typescript
// hooks/useAnalyticsNavigation.ts
import { useNavigationHelper } from './useNavigationHelper';
import { analytics } from '../utils/analytics';

export const useAnalyticsNavigation = () => {
  const navigationHelper = useNavigationHelper();

  const trackAndNavigate = (screenName: string, params?: any) => {
    // Track navigation event
    analytics.track('screen_view', {
      screen_name: screenName,
      ...params,
    });

    // Navigate using appropriate function
    const navigationFunction = navigationHelper[`navigateTo${screenName}`];
    if (navigationFunction && typeof navigationFunction === 'function') {
      navigationFunction(params);
    }
  };

  return {
    ...navigationHelper,
    trackAndNavigate,
  };
};
```

5. Debounced Navigation

typescript

```typescript
// hooks/useDebouncedNavigation.ts
import { useCallback } from 'react';
import { useNavigationHelper } from './useNavigationHelper';
import { debounce } from 'lodash';

export const useDebouncedNavigation = (delay = 300) => {
  const navigationHelper = useNavigationHelper();

  const debouncedNavigate = useCallback(
    debounce((navigationFn: Function, ...args: any[]) => {
      navigationFn(...args);
    }, delay),
    [delay]
  );

  const navigateToPropertyDetails = useCallback(
    (propertyId: string, title?: string) => {
      debouncedNavigate(navigationHelper.navigateToPropertyDetails, propertyId, title);
    },
    [debouncedNavigate, navigationHelper.navigateToPropertyDetails]
  );

  return {
    ...navigationHelper,
    navigateToPropertyDetails,
  };
};
```

🧪 Testing

1. Mocking the Navigation Helper

```typescript
// __tests__/mocks/navigationHelper.ts
export const mockNavigationHelper = {
  navigateToPropertyDetails: jest.fn(),
  navigateToLogin: jest.fn(),
  goBack: jest.fn(),
  resetToAuth: jest.fn(),
  // Add other methods as needed
};

// Mock the hook
jest.mock('../hooks/useNavigationHelper', () => ({
  useNavigationHelper: () => mockNavigationHelper,
}));
```

2. Testing Components with Navigation

```typescript
// __tests__/PropertyCard.test.tsx
import React from 'react';
import { render, fireEvent } from '@testing-library/react-native';
import PropertyCard from '../PropertyCard';
import { mockNavigationHelper } from './mocks/navigationHelper';

describe('PropertyCard', () => {
  beforeEach(() => {
    jest.clearAllMocks();
  });

  it('navigates to property details when pressed', () => {
    const property = { id: '123', title: 'Test Property' };

    const { getByTestId } = render(
      <PropertyCard property={property} />
    );

    fireEvent.press(getByTestId('property-card'));

    expect(mockNavigationHelper.navigateToPropertyDetails)
      .toHaveBeenCalledWith('123', 'Test Property');
  });
});
```

⚡ Performance Optimizations

1. Memoized Navigation Callbacks

typescript

```typescript
// components/PropertyList.tsx
import React, { useMemo, useCallback } from 'react';
import { useNavigationHelper } from '../hooks/useNavigationHelper';

const PropertyList = ({ properties }) => {
  const { navigateToPropertyDetails } = useNavigationHelper();

  const navigationCallbacks = useMemo(() =>
    properties.reduce((acc, property) => {
      acc[property.id] = () => navigateToPropertyDetails(property.id, property.title);
      return acc;
    }, {}),
    [properties, navigateToPropertyDetails]
  );

  return (
    <FlatList
      data={properties}
      renderItem={({ item }) => (
        <TouchableOpacity onPress={navigationCallbacks[item.id]}>
          {/* Property content */}
        </TouchableOpacity>
      )}
    />
  );
};
```

2. Lazy Navigation Functions

typescript

```typescript
// hooks/useLazyNavigation.ts
import { useMemo } from 'react';
import { useNavigationHelper } from './useNavigationHelper';

export const useLazyNavigation = (screenNames: string[]) => {
  const navigationHelper = useNavigationHelper();

  return useMemo(() => {
    const lazyFunctions = {};

    screenNames.forEach(screenName => {
      const functionName = `navigateTo${screenName}`;
      if (navigationHelper[functionName]) {
        lazyFunctions[functionName] = navigationHelper[functionName];
      }
    });

    return lazyFunctions;
  }, [screenNames, navigationHelper]);
};
```

🔧 Custom Navigation Patterns

1. Tab Switch with Data Preloading

typescript

```typescript
// hooks/usePreloadNavigation.ts
export const usePreloadNavigation = () => {
  const { navigateToProperties } = useNavigationHelper();
  const { prefetchProperties } = usePropertiesApi();

  const navigateToPropertiesWithPreload = useCallback(async () => {
    // Start preloading data
    const propertiesPromise = prefetchProperties();

    // Navigate immediately
    navigateToProperties();

    // Data will be ready when screen loads
    await propertiesPromise;
  }, [navigateToProperties, prefetchProperties]);

  return {
    navigateToPropertiesWithPreload,
  };
};
```

2. Navigation with Loading States

typescript

```typescript
// hooks/useLoadingNavigation.ts
import { useState } from 'react';
import { useNavigationHelper } from './useNavigationHelper';

export const useLoadingNavigation = () => {
  const [isNavigating, setIsNavigating] = useState(false);
  const navigationHelper = useNavigationHelper();

  const navigateWithLoading = useCallback(async (
    navigationFn: Function,
    ...args: any[]
  ) => {
    setIsNavigating(true);

    try {
      // Add artificial delay if needed for UX
      await new Promise(resolve => setTimeout(resolve, 100));
      navigationFn(...args);
    } finally {
      setIsNavigating(false);
    }
  }, []);

  return {
    ...navigationHelper,
    isNavigating,
    navigateWithLoading,
  };
};
```

📱 Production Tips

1. Error Handling

```typescript
// Add error boundaries around navigation-heavy screens
const NavigationErrorBoundary = ({ children }) => {
  const { resetToHome } = useNavigationHelper();

  return (
    <ErrorBoundary
      onError={(error) => {
        console.error('Navigation Error:', error);
        resetToHome();
      }}
    >
      {children}
    </ErrorBoundary>
  );
};
```

2. Deep Linking Integration

```typescript
// hooks/useDeepLink.ts
export const useDeepLink = () => {
  const { navigateToPropertyDetails } = useNavigationHelper();

  useEffect(() => {
    const handleDeepLink = (url: string) => {
      const propertyMatch = url.match(/\/property\/(.+)/);
      if (propertyMatch) {
        navigateToPropertyDetails(propertyMatch[1]);
      }
    };

    // Listen for deep link events
    Linking.addEventListener('url', handleDeepLink);

    return () => {
      Linking.removeEventListener('url', handleDeepLink);
    };
  }, [navigateToPropertyDetails]);
};
```

3. Navigation State Debugging

typescript

```typescript
// utils/navigationLogger.ts
export const useNavigationLogger = () => {
  const navigationHelper = useNavigationHelper();

  if (__DEV__) {
    const loggedHelper = {};

    Object.keys(navigationHelper).forEach(key => {
      if (typeof navigationHelper[key] === 'function' && key.startsWith('navigateTo')) {
        loggedHelper[key] = (...args) => {
          console.log(`🧭 Navigation: ${key}`, args);
          return navigationHelper[key](...args);
        };
      } else {
        loggedHelper[key] = navigationHelper[key];
      }
    });

    return loggedHelper;
  }

  return navigationHelper;
};
```

🎨 UI Integration Patterns

1. Navigation Button Components

typescript

```typescript
// components/NavigationButton.tsx
import React from 'react';
import { TouchableOpacity, Text, StyleSheet } from 'react-native';
import { useNavigationHelper } from '../hooks/useNavigationHelper';

interface NavigationButtonProps {
  to: keyof ReturnType<typeof useNavigationHelper>;
  params?: any;
  children: React.ReactNode;
  style?: any;
  disabled?: boolean;
}

const NavigationButton: React.FC<NavigationButtonProps> = ({
  to,
  params,
  children,
  style,
  disabled = false,
}) => {
  const navigationHelper = useNavigationHelper();

  const handlePress = () => {
    if (!disabled && navigationHelper[to]) {
      if (params) {
        navigationHelper[to](params);
      } else {
        navigationHelper[to]();
      }
    }
  };

  return (
    <TouchableOpacity
      style={[styles.button, style, disabled && styles.disabled]}
      onPress={handlePress}
      disabled={disabled}
    >
      <Text style={styles.buttonText}>{children}</Text>
    </TouchableOpacity>
  );
};

// Usage
<NavigationButton to="navigateToProperties">
  View Properties
</NavigationButton>
```

```
<NavigationButton
 to="navigateToPropertyDetails"
 params={{ propertyId: '123', title: 'Modern Apartment' }}
>
 View Details
</NavigationButton>
```

2. Navigation Menu Component

typescript

```typescript
// components/NavigationMenu.tsx
import React from 'react';
import { View, Text, TouchableOpacity } from 'react-native';
import { useNavigationHelper } from '../hooks/useNavigationHelper';
import { useAuth } from '../hooks/useAuth';

const NavigationMenu = () => {
  const { user } = useAuth();
  const {
    navigateToHome,
    navigateToProperties,
    navigateToMyProperties,
    navigateToProfile,
    navigateToSettings,
    navigateToAdminDashboard,
  } = useNavigationHelper();

  const menuItems = [
    { label: 'Home', action: navigateToHome, icon: 'home' },
    { label: 'Properties', action: navigateToProperties, icon: 'search' },
    {
      label: 'My Properties',
      action: navigateToMyProperties,
      icon: 'building',
      requiresAuth: true
    },
    {
      label: 'Profile',
      action: navigateToProfile,
      icon: 'user',
      requiresAuth: true
    },
    { label: 'Settings', action: navigateToSettings, icon: 'cog' },
    {
      label: 'Admin',
      action: navigateToAdminDashboard,
      icon: 'shield',
      requiresRole: 'admin'
    },
  ];

  const filteredItems = menuItems.filter(item => {
    if (item.requiresAuth && !user) return false;
    if (item.requiresRole && user?.role !== item.requiresRole) return false;
    return true;
  });
  return (
    <View style={styles.menu}>
      {filteredItems.map((item, index) => (
        <TouchableOpacity
          key={index}
          style={styles.menuItem}
          onPress={item.action}
        >
          <Icon name={item.icon} size={20} />
          <Text style={styles.menuText}>{item.label}</Text>
        </TouchableOpacity>
      ))}
    </View>
  );
};
```

3. Breadcrumb Navigation

typescript

```typescript
// components/Breadcrumb.tsx
import React from 'react';
import { View, Text, TouchableOpacity } from 'react-native';
import { useNavigationState } from '@react-navigation/native';
import { useNavigationHelper } from '../hooks/useNavigationHelper';

const Breadcrumb = () => {
  const navigationState = useNavigationState(state => state);
  const { goBack, resetToHome } = useNavigationHelper();

  const generateBreadcrumbs = () => {
    const routes = navigationState?.routes || [];
    return routes.map((route, index) => ({
      name: route.name,
      params: route.params,
      isLast: index === routes.length - 1,
    }));
  };

  const breadcrumbs = generateBreadcrumbs();

  if (breadcrumbs.length <= 1) return null;

  return (
    <View style={styles.breadcrumb}>
      <TouchableOpacity onPress={resetToHome}>
        <Text style={styles.breadcrumbItem}>Home</Text>
      </TouchableOpacity>

      {breadcrumbs.map((crumb, index) => (
        <React.Fragment key={index}>
          <Text style={styles.separator}> / </Text>
          <TouchableOpacity
            onPress={() => {
              if (!crumb.isLast) {
                // Navigate back to this level
                const backSteps = breadcrumbs.length - index - 1;
                for (let i = 0; i < backSteps; i++) {
                  goBack();
                }
              }
            }}
          >
            <Text style={[
              styles.breadcrumbItem,
              crumb.isLast && styles.currentItem
            ]}>
              {formatScreenName(crumb.name)}
            </Text>
          </TouchableOpacity>
        </React.Fragment>
      ))}
    </View>
  );
};
```

🔄 State Persistence Patterns

1. Navigation State Restoration

typescript

```typescript
// hooks/useNavigationPersistence.ts
import AsyncStorage from '@react-native-async-storage/async-storage';
import { useNavigationHelper } from './useNavigationHelper';

const NAVIGATION_STATE_KEY = 'APP_NAVIGATION_STATE';

export const useNavigationPersistence = () => {
  const navigationHelper = useNavigationHelper();

  const saveNavigationState = async (state: any) => {
    try {
      await AsyncStorage.setItem(NAVIGATION_STATE_KEY, JSON.stringify(state));
    } catch (error) {
      console.warn('Failed to save navigation state:', error);
    }
  };

  const restoreNavigationState = async () => {
    try {
      const savedState = await AsyncStorage.getItem(NAVIGATION_STATE_KEY);
      return savedState ? JSON.parse(savedState) : null;
    } catch (error) {
      console.warn('Failed to restore navigation state:', error);
      return null;
    }
  };

  const clearNavigationState = async () => {
    try {
      await AsyncStorage.removeItem(NAVIGATION_STATE_KEY);
    } catch (error) {
      console.warn('Failed to clear navigation state:', error);
    }
  };

  return {
    saveNavigationState,
    restoreNavigationState,
    clearNavigationState,
  };
};
```

2. Context-Aware Navigation

typescript

```typescript
// contexts/NavigationContext.tsx
import React, { createContext, useContext, useReducer } from 'react';
import { useNavigationHelper } from '../hooks/useNavigationHelper';

interface NavigationState {
  currentTab: string;
  previousScreens: string[];
  navigationHistory: Array<{ screen: string; timestamp: number }>;
}

const NavigationContext = createContext<{
  state: NavigationState;
  helpers: ReturnType<typeof useNavigationHelper>;
  addToHistory: (screen: string) => void;
  clearHistory: () => void;
} | null>(null);

export const NavigationProvider = ({ children }) => {
  const helpers = useNavigationHelper();

  const [state, dispatch] = useReducer(navigationReducer, {
    currentTab: 'Home',
    previousScreens: [],
    navigationHistory: [],
  });

  const addToHistory = (screen: string) => {
    dispatch({
      type: 'ADD_TO_HISTORY',
      payload: { screen, timestamp: Date.now() }
    });
  };

  const clearHistory = () => {
    dispatch({ type: 'CLEAR_HISTORY' });
  };

  return (
    <NavigationContext.Provider value={{
      state,
      helpers,
      addToHistory,
      clearHistory,
    }}>
      {children}
    </NavigationContext.Provider>
  );
};

export const useNavigationContext = () => {
  const context = useContext(NavigationContext);
  if (!context) {
    throw new Error('useNavigationContext must be used within NavigationProvider');
  }
  return context;
};
```

🧩 Integration with External Libraries

1. Redux Integration

typescript

```typescript
// hooks/useReduxNavigation.ts
import { useDispatch, useSelector } from 'react-redux';
import { useNavigationHelper } from './useNavigationHelper';
import { setCurrentScreen, addToNavigationHistory } from '../store/navigationSlice';

export const useReduxNavigation = () => {
  const dispatch = useDispatch();
  const navigationState = useSelector(state => state.navigation);
  const navigationHelper = useNavigationHelper();

  const enhancedNavigation = Object.keys(navigationHelper).reduce((acc, key) => {
    if (typeof navigationHelper[key] === 'function' && key.startsWith('navigateTo')) {
      acc[key] = (...args) => {
        // Update Redux state
        const screenName = key.replace('navigateTo', '');
        dispatch(setCurrentScreen(screenName));
        dispatch(addToNavigationHistory({
          screen: screenName,
          params: args,
          timestamp: Date.now(),
        }));

        // Call original navigation function
        return navigationHelper[key](...args);
      };
    } else {
      acc[key] = navigationHelper[key];
    }
    return acc;
  }, {});

  return {
    ...enhancedNavigation,
    navigationState,
  };
};
```

2. Analytics Integration

typescript

```typescript
// hooks/useAnalyticsNavigation.ts
import { useNavigationHelper } from './useNavigationHelper';
import Analytics from '@react-native-firebase/analytics';

export const useAnalyticsNavigation = () => {
  const navigationHelper = useNavigationHelper();

  const trackScreenView = async (screenName: string, params?: any) => {
    try {
      await Analytics().logScreenView({
        screen_name: screenName,
        screen_class: screenName,
      });

      if (params) {
        await Analytics().logEvent('navigation_with_params', {
          screen_name: screenName,
          params: JSON.stringify(params),
        });
      }
    } catch (error) {
      console.warn('Analytics tracking failed:', error);
    }
  };

  // Wrap navigation functions with analytics
  const enhancedNavigation = Object.keys(navigationHelper).reduce((acc, key) => {
    if (typeof navigationHelper[key] === 'function' && key.startsWith('navigateTo')) {
      acc[key] = async (...args) => {
        const screenName = key.replace('navigateTo', '');
        await trackScreenView(screenName, args[0]);
        return navigationHelper[key](...args);
      };
    } else {
      acc[key] = navigationHelper[key];
    }
    return acc;
  }, {});

  return enhancedNavigation;
};
```

🚀 Best Practices Summary

✅ Do's

1. **Use Specialized Hooks**: Use `useAuthNavigation`, `usePropertyNavigation` for focused functionality
2. **Type Everything**: Ensure all navigation functions are properly typed
3. **Handle Edge Cases**: Always check authentication, permissions, and network state
4. **Optimize Performance**: Use `useCallback` and `useMemo` for navigation handlers
5. **Test Navigation**: Write tests for critical navigation flows
6. **Log Navigation**: Add analytics and debugging logs for navigation events
7. **Handle Errors**: Implement error boundaries and fallback navigation
8. **Use Context**: Leverage React Context for navigation state management

❌ Don'ts

1. **Don't Use Class Components**: Stick to functional components with hooks
2. **Don't Navigate in Render**: Always navigate in event handlers or effects
3. **Don't Ignore Back Navigation**: Always handle the back button properly
4. **Don't Hardcode Routes**: Use the navigation helper functions consistently
5. **Don't Forget Loading States**: Show loading indicators during navigation
6. **Don't Skip Error Handling**: Always handle navigation failures gracefully
7. **Don't Overcomplicate**: Keep navigation logic simple and predictable

---

📊 Migration from Class-Based Helper

If you're migrating from the class-based helper:

typescript

```typescript
// Before (Class-based)
const helper = new NavigationHelper(navigation);
helper.navigateToProperties();

// After (Functional)
const { navigateToProperties } = useNavigationHelper();
navigateToProperties();

// Before (Class instantiation in component)
const MyComponent = ({ navigation }) => {
  const helper = useMemo(() => new NavigationHelper(navigation), [navigation]);

  return (
    <Button onPress={() => helper.navigateToHome()} title="Home" />
  );
};

// After (Direct hook usage)
const MyComponent = () => {
  const { navigateToHome } = useNavigationHelper();

  return (
    <Button onPress={navigateToHome} title="Home" />
  );
};
```

The functional approach provides better performance, easier testing, and more React-like patterns while maintaining all the functionality of the class-based approach.