Ανάπτυξη Κινητών Εφαρμογών

Εργαστήριο

Δημήτρης Ρίγγας

riggas@ionio.gr

Τμήμα Πληροφορικής

Ιόνιο Πανεπιστήμιο



Εργαστήριο 5 – React Native Development

Navigation & Shared State

React Navigation, Context API, useReducer



Στόχος:

- ✓ **Navigation** σε mobile εφαρμογές - τύποι navigators
- ✓ **React Navigation 7**: εγκατάσταση & `NavigationContainer`
- ✓ **Native Stack** - μεταφορά παραμέτρων, header options
- ✓ **Bottom Tabs** - icons, badges
- ✓ **Nested navigators**
- ✓ **Prop Drilling Problem** και λύσεις
- ✓ **Context API** & **useContext**
- ✓ **useReducer Hook** - Complex State Logic
- ✓ **Context + useReducer** για global state
- ✓ **Best Practices** και οργάνωση κώδικα

Navigation σε React Native

Οι mobile εφαρμογές **δεν** έχουν URLs και browser history — η πλοήγηση είναι **στοίβα οθονών** και **tabs**, με native animations & gestures.

Τύποι Navigators:

1. **Stack**: Push/Pop screens, back button & swipe-back
2. **Bottom Tabs**: Εναλλαγή μεταξύ τμημάτων της εφαρμογής
3. **Drawer**: Πλαϊνό μενού
4. **Material Top Tabs**: Swipeable tabs

Βιβλιοθήκες:

React Navigation	Η πιο διαδεδομένη. Navigators ως components (JS config) — **αυτή χρησιμοποιούμε**
Expo Router	File-based routing (`app/` folder), default στα νέα Expo templates
React Native Navigation (Wix)	Πλήρως native, πιο σύνθετο setup

> Οι έννοιες (stack, tabs, params, nesting) είναι **ίδιες** σε όλες.

React Navigation - Εγκατάσταση & Setup

```
npx expo install @react-navigation/native \
  @react-navigation/native-stack \
  @react-navigation/bottom-tabs \
  react-native-screens \
  react-native-safe-area-context \
  @expo/vector-icons
```

- **`npx expo install`** → εκδόσεις συμβατές με το Expo SDK
- **`NavigationContainer`**: **ένα**, στη ρίζα — κρατάει το navigation state
- **`createNativeStackNavigator`**: χρησιμοποιεί τα native stack primitives (iOS `UINavigationController`, Android Fragments)

```jsx
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';

const Stack = createNativeStackNavigator();

export default function App() {
  return (
    <NavigationContainer>
      <Stack.Navigator>
        <Stack.Screen
          name="Users"
          component={UsersScreen}
          options={{ title: 'Χρήστες' }}
        />
        <Stack.Screen
          name="UserDetails"
          component={UserDetailsScreen}
        />
      </Stack.Navigator>
    </NavigationContainer>
  );
}
```

Stack Navigator - Header Options

- `screenOptions` (στο navigator): ισχύει για **όλα** τα screens
- `options` (στο screen): μόνο για αυτό — **υπερισχύει**
- Το `options` μπορεί να είναι **function** του `route` → δυναμικός τίτλος
- Από μέσα στο screen: `navigation.setOptions({ ... })`

Συχνά options: `title`, `headerShown`, `headerStyle`, `headerTintColor`, `headerRight`, `presentation: 'modal'`

```
const Stack = createNativeStackNavigator();

const UsersStack = () => (
  <Stack.Navigator
    initialRouteName="Users"
    screenOptions={{
      headerStyle: { backgroundColor: '#3498db' },
      headerTintColor: '#fff',
      headerTitleStyle: { fontWeight: 'bold' }
    }}
  >
    <Stack.Screen
      name="Users"
      component={UsersScreen}
      options={{ title: 'Χρήστες' }}
    />
    <Stack.Screen
      name="UserDetails"
      component={UserDetailsScreen}
      options={({ route }) => ({
        title: route.params.user.name
      })}
    />
  </Stack.Navigator>
);
```

Navigation - Μεταφορά Παραμέτρων

```
// UsersScreen.js – Στέλνει παραμέτρους
const UsersScreen = ({ navigation }) => (
  <FlatList
    data={users}
    renderItem={({ item }) => (
      <UserCard
        name={item.name}
        onPress={() =>
          navigation.navigate('UserDetails', { user: item })
        }
      />
    )}
  />
);

// UserDetailsScreen.js – Λαμβάνει παραμέτρους
const UserDetailsScreen = ({ route, navigation }) => {
  const { user } = route.params;
  return (
    <View>
      <Text>{user.name}</Text>
      <Text>{user.email}</Text>
      <Button title="Πίσω" onPress={() => navigation.goBack()} />
    </View>
  );
};
```

Props κάθε screen:

- `navigation`: `navigate`, `goBack`, `push`, `popToTop`, `setOptions`
- `route`: `name`, `params`

Εκτός screens: `useNavigation()`, `useRoute()`

Καλές πρακτικές για params:

- Μόνο **serializable** τιμές (JSON) — **όχι** functions
- Περνάμε το **ελάχιστο** (π.χ. `{ userId: 3 }`) και η οθόνη βρίσκει τα δεδομένα από κοινή πηγή (API, Context)
- Τα params είναι σαν τα **query parameters** ενός URL

Bottom Tab Navigator

```
import { Ionicons } from '@expo/vector-icons';
import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';

const Tab = createBottomTabNavigator();

const TAB_ICONS = {
  UsersTab: 'people',
  Favorites: 'heart',
  About: 'information-circle',
};

const Tabs = () => (
  <Tab.Navigator
    screenOptions={({ route }) => ({
      tabBarIcon: ({ focused, color, size }) => {
        const name = TAB_ICONS[route.name];
        return (
          <Ionicons
            name={focused ? name : `${name}-outline`}
            size={size}
            color={color}
          />
        );
      },
      tabBarActiveTintColor: '#3498db',
      tabBarInactiveTintColor: 'gray',
    })}
  >
```

```
    <Tab.Screen
      name="UsersTab"
      component={UsersStack}
      options={{ title: 'Χρήστες', headerShown: false }}
    />
    <Tab.Screen
      name="Favorites"
      component={FavoritesScreen}
      options={{ title: 'Αγαπημένα', tabBarBadge: 3 }}
    />
    <Tab.Screen
      name="About"
      component={AboutScreen}
      options={{ title: 'Σχετικά' }}
    />
  </Tab.Navigator>
);
```

- Icons: https://icons.expo.fyi
- `tabBarBadge`: αριθμός/κείμενο στο tab (`undefined` = χωρίς badge)
- Κάθε tab κρατάει το **δικό του** state (π.χ. ανοιχτό stack)

Nested Navigators

```
NavigationContainer
└── Tab.Navigator
    ├── "UsersTab" → UsersStack
    │     ├── "Users"
    │     └── "UserDetails"
    ├── "Favorites"
    └── "About"
```

- Ένα navigator μπορεί να είναι `component` ενός screen άλλου navigator
- **Ένα** `NavigationContainer` μόνο
- Αν και ο γονέας και το παιδί έχουν header → **δύο headers**: `headerShown: false` στον έναν

```jsx
// navigation/UsersStack.js
const Stack = createNativeStackNavigator();

export default function UsersStack() {
  return (
    <Stack.Navigator>
      <Stack.Screen name="Users" component={UsersScreen} />
      <Stack.Screen name="UserDetails" component={UserDetailsScreen} />
    </Stack.Navigator>
  );
}
```

```jsx
// Από το tab "Favorites" σε screen του nested stack
navigation.navigate('UsersTab', {
  screen: 'UserDetails',
  params: { user: item },
});
```

Το Πρόβλημα του Prop Drilling

Prop Drilling: Μεταφορά props μέσα από πολλά επίπεδα components

```
// ❌ Πρόβλημα: Τα MiddleComponent & IntermediateComponent
//    δεν χρειάζονται το user, απλά το περνάνε παρακάτω

const App = () => {
  const [user, setUser] = useState({ name: 'John', role: 'admin' });
  return <MiddleComponent user={user} />;
};

const MiddleComponent = ({ user }) => {
  return <IntermediateComponent user={user} />;
};

const IntermediateComponent = ({ user }) => {
  return <DeepComponent user={user} />;
};

const DeepComponent = ({ user }) => {
  return <Text>{user.name}</Text>; // Τελικά χρησιμοποιείται εδώ
};
```

Προβλήματα: Δύσκολη συντήρηση, refactoring, testing — και με navigators τα screens **δεν** δέχονται εύκολα custom props

Το Πρόβλημα του Prop Drilling: Οπτικοποίηση του Προβλήματος

```
App { favorites }                    ← κοινός πρόγονος
└── Tab.Navigator                    [δεν το χρησιμοποιεί]
    ├── UsersStack                   [δεν το χρησιμοποιεί]
    │   ├── UsersScreen              [✓ ❤️ στις κάρτες]
    │   │   └── UserCard             [✓]
    │   └── UserDetailsScreen        [✓ toggle]
    └── FavoritesScreen              [✓ λίστα]
```

Τα `favorites` χρειάζονται σε **4** components, σε **διαφορετικά** κλαδιά του δέντρου — και πρέπει να περάσουν μέσα από **navigators**!

Context API - Λύση στο Prop Drilling: Global State χωρίς Props Chains

Το **Context API** επιτρέπει sharing data σε όλο το component tree χωρίς props:

```
import { createContext, useState } from 'react';

// 1. Δημιουργία Context
const UserContext = createContext(null);

// 2. Provider – Παρέχει τα data
const App = () => {
  const [user, setUser] = useState({ name: 'John', role: 'admin' });

  return (
    <UserContext.Provider value={{ user, setUser }}>
      <MiddleComponent />
    </UserContext.Provider>
  );
};

// 3. Τα intermediate components ΔΕΝ χρειάζονται props!
const MiddleComponent = () => <IntermediateComponent />;
const IntermediateComponent = () => <DeepComponent />;
```

useContext Hook: Κατανάλωση Context Data

Το `useContext` Hook καταναλώνει Context data, οπότε δεν χρειάζονται **καθόλου props** στα ενδιάμεσα components:

```
import { useContext } from 'react';

const DeepComponent = () => {
  // Απευθείας πρόσβαση στο user!
  const { user, setUser } = useContext(UserContext);

  const updateName = (newName) => {
    setUser(prev => ({ ...prev, name: newName }));
  };

  return (
    <View>
      <Text>Όνομα: {user.name}</Text>
      <Text>Ρόλος: {user.role}</Text>
      <Button
        title="Αλλαγή Ονόματος"
        onPress={() => updateName('Maria')}
      />
    </View>
  );
};
```

“ Όταν αλλάζει το `value` του Provider, κάνουν **re-render όλα** τα components που καλούν `useContext` για αυτό το Context. ”

Context API - Παράδειγμα με Provider & Χρήση του Custom Hook

```jsx
// context/UserContext.js
import { createContext, useState, useContext, useMemo } from 'react';

const UserContext = createContext(null);

// Custom Provider component
export const UserProvider = ({ children }) => {
  const [user, setUser] = useState(null);

  const login = (userData) => setUser(userData);
  const logout = () => setUser(null);

  // ✅ Memoize το value ώστε να μη δημιουργείται
  //    νέο object σε κάθε render
  const value = useMemo(() => ({
    user,
    isAuthenticated: user !== null,
    login,
    logout,
  }), [user]);

  return (
    <UserContext.Provider value={value}>
      {children}
    </UserContext.Provider>
  );
};

// Custom Hook για εύκολη χρήση
export const useUser = () => {
  const context = useContext(UserContext);
  if (!context) {
    throw new Error('useUser must be used within UserProvider');
  }
  return context;
};
```

```jsx
// App.js
const App = () => (
  <UserProvider>
    <NavigationContainer>
      {/* ... */}
    </NavigationContainer>
  </UserProvider>
);

// LoginScreen.js
const LoginScreen = () => {
  const { login } = useUser();

  const handleLogin = () => {
    login({ id: 1, name: 'Maria', email: 'maria@ionio.gr' });
  };

  return <Button title="Login" onPress={handleLogin} />;
};

// ProfileScreen.js
const ProfileScreen = () => {
  const { user, logout, isAuthenticated } = useUser();

  if (!isAuthenticated) return <Text>Παρακαλώ συνδεθείτε</Text>;

  return (
    <View>
      <Text>Όνομα: {user.name}</Text>
      <Text>Email: {user.email}</Text>
      <Button title="Logout" onPress={logout} />
    </View>
  );
};
```

Context API - Καλές Πρακτικές

1. Χωρίστε τα Contexts ανά domain

Μια αλλαγή στο theme δεν πρέπει να κάνει re-render τα components του cart.

2. Memoize το value για performance

Το `useMemo` απομνημονεύει (memoize) μια τιμή, ώστε να μην ξαναϋπολογίζεται σε κάθε render αν δεν έχουν αλλάξει οι εξαρτήσεις της.

Χωρίς αυτό, σε κάθε render ο Provider δημιουργεί **νέο** object → το Context θεωρεί ότι η τιμή άλλαξε → re-render σε **όλους** τους consumers.

3. Custom hook (`useUser` , `useFavorites`) με έλεγχο για Provider

```
// ✅ Ξεχωριστά contexts
<UserProvider>
  <ThemeProvider>
    <FavoritesProvider>
      <NavigationContainer>...</NavigationContainer>
    </FavoritesProvider>
  </ThemeProvider>
</UserProvider>

// ❌ Ένα μεγάλο context για όλα
<AppProvider value={{ user, theme, language, cart, ... }}>
```

```
const value = useMemo(() => ({
  user, login, logout
}), [user]); // Αποφεύγει unnecessary re-renders
```

Πότε ΟΧΙ Context: state που χρειάζεται μόνο ένα component ή γονέας-παιδί → `useState` + props.

useReducer Hook - Complex State Logic

Το `useReducer` είναι ιδανικό για **πολύπλοκη state logic**: όλες οι αλλαγές σε **μία pure function** `(state, action) → newState`

```
import { useReducer } from 'react';

// Reducer function
const cartReducer = (state, action) => {
  switch (action.type) {
    case 'ADD_ITEM':
      return {
        ...state,
        items: [...state.items, action.payload]
      };
    case 'REMOVE_ITEM':
      return {
        ...state,
        items: state.items.filter(item => item.id !== action.payload)
      };
    default:
      throw new Error(`Unknown action: ${action.type}`);
  }
};
```

“ **Immutability**: ο reducer επιστρέφει **νέο** object/array (`...spread` , `map` , `filter`) — **ποτέ** `push` , `splice` ή ανάθεση στο `state` . ”

useReducer Hook - Χρήση

```
const initialState = { items: [] };

const ShoppingCart = () => {
  const [state, dispatch] = useReducer(cartReducer, initialState);

  const addItem = (product) => {
    dispatch({
      type: 'ADD_ITEM',
      payload: { ...product, quantity: 1 }
    });
  };

  const removeItem = (id) => {
    dispatch({ type: 'REMOVE_ITEM', payload: id });
  };

  return (
    <View>
      {state.items.map(item => (
        <View key={item.id}>
          <Text>{item.name} x {item.quantity}</Text>
          <Button title="Remove" onPress={() => removeItem(item.id)} />
        </View>
      ))}
    </View>
  );
};
```

useState vs useReducer

Πότε χρησιμοποιούμε ποιο;

useState	useReducer
Simple state (primitives)	Complex state (objects, arrays)
Λίγες state updates	Πολλές state updates
Independent state changes	Related state transitions
Μικρά components	Μεγάλα components / Context

useState Example:

```
const [count, setCount] = useState(0);
const [isOpen, setIsOpen] = useState(false);
```

useReducer Example:

```
const [state, dispatch] = useReducer(formReducer, {
  username: '', password: '', email: '', errors: {}
});
```

Context + useReducer - Συνδυασμός για Global State

```
// context/FavoritesContext.js
const FavoritesContext = createContext(null);

function favoritesReducer(state, action) {
  switch (action.type) {
    case 'TOGGLE': {
      const exists = state.some(u => u.id === action.payload.id);
      return exists
        ? state.filter(u => u.id !== action.payload.id)
        : [...state, action.payload];
    }
    case 'CLEAR':
      return [];
    default:
      throw new Error(`Άγνωστο action: ${action.type}`);
  }
}

export function FavoritesProvider({ children }) {
  const [favorites, dispatch] = useReducer(favoritesReducer, []);

  const value = useMemo(() => ({
    favorites,
    isFavorite: id => favorites.some(u => u.id === id),
    toggleFavorite: user => dispatch({ type: 'TOGGLE', payload: user }),
    clearFavorites: () => dispatch({ type: 'CLEAR' }),
  }), [favorites]);

  return (
    <FavoritesContext.Provider value={value}>
      {children}
    </FavoritesContext.Provider>
  );
}
```

```
export function useFavorites() {
  const context = useContext(FavoritesContext);
  if (!context) {
    throw new Error('useFavorites: εκτός FavoritesProvider');
  }
  return context;
}
```

```
// UserDetailsScreen.js
const { isFavorite, toggleFavorite } = useFavorites();

<Button
  title={isFavorite(user.id) ? '💔 Αφαίρεση' : '❤️ Προσθήκη'}
  onPress={() => toggleFavorite(user)}
/>

// Tabs (μέσα στον Provider!)
const { favorites } = useFavorites();
<Tab.Screen
  name="Favorites"
  options={{ tabBarBadge: favorites.length || undefined }}
  /* ... */
/>
```

Ίδιο pattern με **Redux** — σε μικρότερη κλίμακα.

Best Practices - Performance

Βελτιστοποίηση Performance

1. `memo` για components που κάνουν re-render με ίδια props

```
import { memo } from 'react';

const UserCard = memo(({ name, email }) => {
  return <View>...</View>;
});
```

2. `useCallback` για functions που περνάνε σε memo components

```
const handlePress = useCallback(() => {
  doSomething(value);
}, [value]); // Νέα function μόνο όταν αλλάζει το value
```

3. `useMemo` για expensive calculations

```
const sortedUsers = useMemo(() => {
  // [...users]: το sort() αλλάζει το αρχικό array!
  return [...users].sort((a, b) =>
    a.name.localeCompare(b.name)
  );
}, [users]);
```

> **Μετρήστε πρώτα** (React DevTools Profiler) — μη βάζετε `memo` / `useCallback` / `useMemo` παντού "προληπτικά".

Best Practices - Code Organization

Οργάνωση Κώδικα

```
usersapp/
├── App.js                # Providers + NavigationContainer
├── components/           # Reusable UI components
│   └── UserCard.js
├── screens/              # Ένα component ανά οθόνη
│   ├── UsersScreen.js
│   ├── UserDetailsScreen.js
│   ├── FavoritesScreen.js
│   └── AboutScreen.js
├── navigation/           # Navigators
│   └── UsersStack.js
├── context/              # Context providers + custom hooks
│   └── FavoritesContext.js
├── hooks/                # Custom hooks (useFetch, ...)
└── constants/            # Colors, API URLs, etc.
```

Και τώρα.. *hands on*!

Συνεχίζουμε στο:

https://github.com/riggas-ionio/mobile-apps/tree/class-2026/lab05



Παραπομπές & Πηγές

- Official Documentation
  - React Navigation - Getting started
  - React Navigation - Nesting navigators
  - React - Passing Data Deeply with Context
  - React - Extracting State Logic into a Reducer
  - React - Scaling Up with Reducer and Context
  - Expo - Icons

