Ανάπτυξη Κινητών Εφαρμογών

Νικόλαος Κορφιάτης & Δημήτρης Ρίγγας

nkorf@ionio.gr | riggas@ionio.gr

Τμήμα Πληροφορικής

Ιόνιο Πανεπιστήμιο

Περιεχόμενα (Table of Contents)

Μαθησιακοί Στόχοι

- Ανάλυση της δομή και ροής μιας mobile εφαρμογής, περιλαμβάνοντας το lifecycle, τη διαχείριση της κατάστασης και τη σύνδεση με APIs.

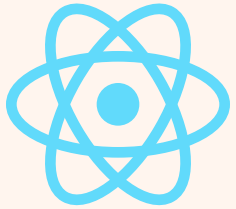
Μέρη ενότητας

Τι είναι το React Native;

React Native είναι ένα open-source framework που αναπτύχθηκε από τη Meta (Facebook) για τη δημιουργία mobile εφαρμογών χρησιμοποιώντας JavaScript και React.

Βασικά Χαρακτηριστικά:

- **Cross-platform development**: Ένας κώδικας για iOS και Android
- **Native performance**: Rendering σε πραγματικά native components
- **Hot Reloading**: Άμεση προεπισκόπηση αλλαγών
- **Large ecosystem**: Χιλιάδες libraries και community support
- **Learn once, write anywhere**: Χρήση React skills



React Native vs Άλλες προσεγγίσεις (όπως έχουμε δει)

Native Development

- **iOS**: Swift/Objective-C, Xcode
- **Android**: Kotlin/Java, Android Studio
- ✅ Άριστη performance
- ❌ Διπλός κώδικας και development time

Hybrid (Cordova, Ionic)

- Web technologies σε WebView
- ❌ Χειρότερη performance και UX

React Native

- **JavaScript** → Native components
- ✅ Near-native performance
- ✅ Shared codebase (~70-90%)
- ✅ Faster development cycle

Αρχιτεκτονική React Native

Το **Bridge** είναι το κλειδί: επικοινωνία JS ↔️ Native μέσω JSON messages.

JavaScript για Mobile Apps

Modern JavaScript (ES6+) Features

```
// Arrow Functions
const greet = (name) => `Hello, ${name}!`;

// Destructuring
const { username, email } = user;
const [first, second, ...rest] = items;

// Template Literals
const message = `User ${username} logged in at ${timestamp}`;

// Spread Operator
const newArray = [...oldArray, newItem];
const updatedUser = { ...user, status: 'active' };

// Async/Await
const fetchData = async () => {
  try {
    const response = await fetch(API_URL);
    const data = await response.json();
    return data;
  } catch (error) {
    console.error(error);
  }
};
```

JavaScript: Promises & Asynchronous Code

Στις mobile εφαρμογές, οι **asynchronous operations** είναι κρίσιμες (network requests, database queries, file operations).

```
// Promise Chain
fetch('https://api.example.com/users')
  .then(response => response.json())
  .then(data => {
    console.log('Users:', data);
    return data;
  })
  .catch(error => console.error('Error:', error))
  .finally(() => console.log('Request completed'));

// Async/Await (Preferred)
async function loadUserData() {
  try {
    const response = await fetch('https://api.example.com/user/1');
    const userData = await response.json();
    return userData;
  } catch (error) {
    throw new Error(`Failed to load user: ${error.message}`);
  }
}
```

TypeScript vs JavaScript

JavaScript (Dynamic Typing)

```
function calculateArea(width, height) {
  return width * height;
}

calculateArea(5, 10);      // ✅ 50
calculateArea('5', '10');  // ⚠️ '510' (string concatenation!)
```

TypeScript (Static Typing)

```
function calculateArea(width: number, height: number): number {
  return width * height;
}

calculateArea(5, 10);      // ✅ 50
calculateArea('5', '10');  // ❌ Compile error!
```

Πλεονεκτήματα TypeScript: Type safety, Better IDE support, Fewer runtime errors, Self-documenting code

TypeScript: Interfaces & Types

```typescript
// Interface για User object
interface User {
  id: number;
  username: string;
  email: string;
  avatar?: string;  // Optional property
  role: 'admin' | 'user' | 'guest';  // Union type
}

// Type για Component Props
type ButtonProps = {
  title: string;
  onPress: () => void;
  disabled?: boolean;
  variant?: 'primary' | 'secondary';
};

// Χρήση
const user: User = {
  id: 1,
  username: 'john_doe',
  email: 'john@example.com',
  role: 'user'
};

const MyButton: React.FC<ButtonProps> = ({ title, onPress, disabled }) => {
  // Component implementation
};
```

React Fundamentals: Components

- Function Components (βασισμένα σε JSX) ✅ Προτεινόμενα

```
import React from 'react';
import { View, Text, StyleSheet } from 'react-native';

const WelcomeCard = ({ username, isLoggedIn }) => {
  return (
    <View style={styles.container}>
      <Text style={styles.title}>Welcome, {username}!</Text>
      {isLoggedIn && <Text>You are logged in</Text>}
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    padding: 20,
    backgroundColor: '#f0f0f0',
  },
  title: {
    fontSize: 24,
    fontWeight: 'bold',
  }
});

export default WelcomeCard;
```

React Fundamentals: Components

- Class Components ❌ Legacy

```
import React, { Component } from 'react';
import { View, Text } from 'react-native';

class WelcomeCard extends Component {
  constructor(props) {
    super(props);
    this.state = {
      counter: 0
    };
  }
  componentDidMount() {
    console.log('Component mounted');
  }
  componentWillUnmount() {
    console.log('Component will unmount');
  }
  render() {
    return (
      <View>
        <Text>Welcome, {this.props.username}!</Text>
        <Text>Counter: {this.state.counter}</Text>
      </View>
    );
  }
}
```

Σημείωση: Function Components + Hooks είναι το σύγχρονο standard.

React Hooks: useState

Το **useState** hook επιτρέπει state management σε function components.

```
import React, { useState } from 'react';
import { View, Text, Button } from 'react-native';

const Counter = () => {
  // Δήλωση state variable
  const [count, setCount] = useState(0);
  const [name, setName] = useState('Guest');

  const increment = () => {
    setCount(count + 1);
    // ή με function για safety
    setCount(prevCount => prevCount + 1);
  };

  return (
    <View>
      <Text>Count: {count}</Text>
      <Text>User: {name}</Text>
      <Button title="Increment" onPress={increment} />
    </View>
  );
};
```

Κανόνες Hooks: Μόνο στο top level, μόνο σε function components.

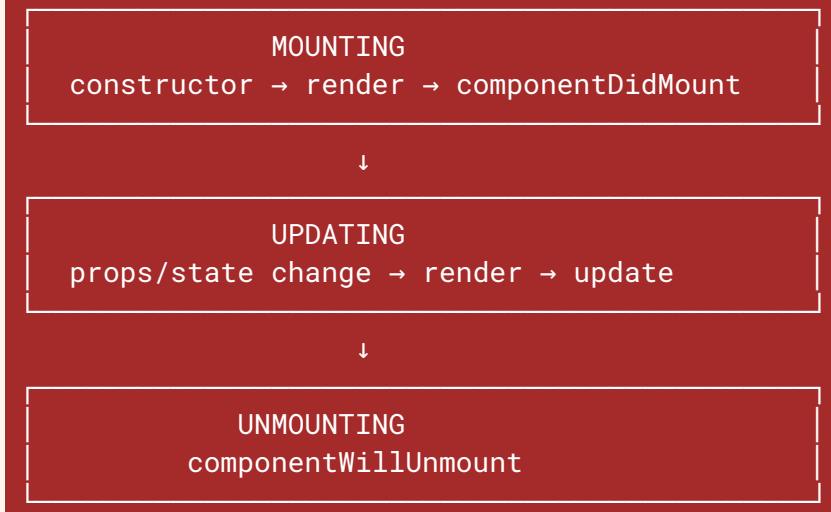
React Hooks: useEffect

Το **useEffect** hook διαχειρίζεται side effects (data fetching, subscriptions, timers).

```
import React, { useState, useEffect } from 'react';
import { View, Text, ActivityIndicator } from 'react-native';

const UserProfile = ({ userId }) => {
  const [user, setUser] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    // Εκτελείται μετά το render
    const fetchUser = async () => {
      try {
        const response = await fetch(`/api/users/${userId}`);
        const data = await response.json();
        setUser(data);
      } catch (error) {
        console.error(error);
      } finally {
        setLoading(false);
      }
    };

    fetchUser();

    // Cleanup function
    return () => {
      console.log('Cleanup on unmount or userId change');
    };
  }, [userId]); // Dependency array - re-run όταν αλλάζει το userId

  if (loading) return <ActivityIndicator />;
  return <Text>{user?.name}</Text>;
};
```

Component Lifecycle στο React Native

Lifecycle Phases



Με Function Components & Hooks:

```
useEffect(() => {
  // ComponentDidMount logic
  console.log('Component mounted');

  return () => {
    // ComponentWillUnmount logic
    console.log('Component will unmount');
  };
}, []); // Empty array = run once
```

Component-based Architecture

Principles

1. **Single Responsibility**: Κάθε component κάνει ένα πράγμα καλά
2. **Reusability**: Components μπορούν να επαναχρησιμοποιηθούν
3. **Composition**: Σύνθετα UIs από απλά components
4. **Props Flow Down**: Data flow από parent → child

```
App
├── Header
│   ├── Logo
│   └── Navigation
├── MainContent
│   ├── Sidebar
│   └── ContentArea
│       ├── ArticleList
│       │   └── ArticleCard (×N)
│       └── Pagination
└── Footer
```

Props & Component Composition

```jsx
// Reusable Button Component
const CustomButton = ({ title, onPress, variant = 'primary', icon }) => {
  const buttonStyle = variant === 'primary'
    ? styles.primaryButton
    : styles.secondaryButton;

  return (
    <TouchableOpacity style={buttonStyle} onPress={onPress}>
      {icon && <Image source={icon} style={styles.icon} />}
      <Text style={styles.buttonText}>{title}</Text>
    </TouchableOpacity>
  );
};

// Usage - Composition
const LoginScreen = () => {
  return (
    <View>
      <CustomButton
        title="Login"
        onPress={handleLogin}
        variant="primary"
        icon={require('./assets/login-icon.png')}
      />
      <CustomButton
        title="Cancel"
        onPress={handleCancel}
        variant="secondary"
      />
    </View>
  );
};
```

State Management

State Management

Πρόβλημα: Prop Drilling

```
// App → Screen → List → Item → Button
// Πρέπει να περάσουμε props μέσα από πολλά levels

App (user data)
  → Screen (pass user)
    → List (pass user)
      → Item (pass user)
        → Button (finally use user!)
```

Λύση: **Context API** (built-in React)

- Ένας τρόπος για να μοιράζεσαι δεδομένα (state) μεταξύ πολλών components χωρίς να χρειάζεται να τα περάσεις χειροκίνητα μέσω props από το ένα component στο άλλο.
- Χρησιμοποιείται όταν έχουμε δεδομένα που πρέπει να είναι προσβάσιμα από πολλά επίπεδα της εφαρμογής, όπως:
  - Το theme (π.χ. dark/light mode)
  - Ο χρήστης που είναι συνδεδεμένος
  - Οι ρυθμίσεις της εφαρμογής

Context API: Βασική Χρήση

```jsx
import React, { createContext, useContext, useState } from 'react';

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

// 2. Provider Component
export const UserProvider = ({ children }) => {
  const [user, setUser] = useState(null);

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

  return (
    <UserContext.Provider value={{ user, login, logout }}>
      {children}
    </UserContext.Provider>
  );
};

// 3. 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;
};
```

Δημιουργίας ενός context:

```jsx
const MyContext = React.createContext();
```

Επιστρέφει ένα αντικείμενο με δύο βασικά στοιχεία:

- `MyContext.Provider`
  Ο Provider είναι το component που παρέχει τα δεδομένα στα υπόλοιπα components.
- `const contextValue = useContext(MyContext);`
  Hook που επιτρέπει σε ένα component να καταναλώσει τα δεδομένα του context. Αντικαθιστά την παλαιότερη μέθοδο `MyContext.Consumer`.

Context API: Παράδειγμα Χρήσης

```
// App.js - Wrap με Provider
import { UserProvider } from './contexts/UserContext';

const App = () => {
  return (
    <UserProvider>
      <NavigationContainer>
        <AppNavigator />
      </NavigationContainer>
    </UserProvider>
  );
};

// Οποιοδήποτε nested component
import { useUser } from './contexts/UserContext';

const ProfileScreen = () => {
  const { user, logout } = useUser();

  if (!user) {
    return <Text>Please login</Text>;
  }

  return (
    <View>
      <Text>Welcome, {user.name}!</Text>
      <Button title="Logout" onPress={logout} />
    </View>
  );
};
```

Context API: Παράδειγμα Χρήσης 2 (theme selection)

```jsx
// ThemeContext.js
import React, { createContext, useContext, useState } from 'react';

const ThemeContext = createContext();

export const ThemeProvider = ({ children }) => {
  const [theme, setTheme] = useState('light');

  const toggleTheme = () =>
    setTheme((prev) => (prev === 'light' ? 'dark' : 'light'));

  return (
    <ThemeContext.Provider value={{ theme, toggleTheme }}>
      {children}
    </ThemeContext.Provider>
  );
};

export const useTheme = () => useContext(ThemeContext);
```

```jsx
import React from 'react';
import { View, Text, Button } from 'react-native';
import { useTheme } from './ThemeContext';

export default function SettingsScreen() {
  const { theme, toggleTheme } = useTheme();

  return (
    <View>
      <Text>Current theme: {theme}</Text>
      <Button title="Toggle Theme" onPress={toggleTheme} />
    </View>
  );
}
```

Context API: Παράδειγμα Χρήσης 3 (χρήση useReducer)

```jsx
// CartContext.js
import React, { createContext, useContext, useReducer } from 'react';

const CartContext = createContext();

const initialState = [];

function cartReducer(state, action) {
  switch (action.type) {
    case 'ADD_ITEM':
      return [...state, action.payload];
    case 'REMOVE_ITEM':
      return state.filter((item) => item.id !== action.payload.id);
    case 'CLEAR_CART':
      return [];
    default:
      return state;
  }
}

export const CartProvider = ({ children }) => {
  const [cart, dispatch] = useReducer(cartReducer, initialState);

  return (
    <CartContext.Provider value={{ cart, dispatch }}>
      {children}
    </CartContext.Provider>
  );
};

export const useCart = () => useContext(CartContext);
```

```jsx
import React from 'react';
import { View, Button, Text } from 'react-native';
import { useCart } from './CartContext';

export default function ProductScreen() {
  const { cart, dispatch } = useCart();

  const addItem = () => {
    dispatch({ type: 'ADD_ITEM', payload: { id: 1, name: 'Product A' } });
  };

  return (
    <View>
      <Button title="Add to Cart" onPress={addItem} />
      <Text>Items in cart: {cart.length}</Text>
    </View>
  );
}
```

Η useReducer είναι ένα React Hook που χρησιμοποιείται για πιο σύνθετη διαχείριση κατάστασης (state), ειδικά όταν το state έχει πολλές μεταβολές ή εξαρτάται από διαφορετικές ενέργειες (actions).

- reducer: Μια συνάρτηση που καθορίζει πώς αλλάζει το state με βάση την ενέργεια (action).
- initialState: Η αρχική τιμή του state.
- dispatch: Συνάρτηση που καλείται για να στείλεις μια ενέργεια (action) και να ενημερώσεις το state.

React Navigation

React Navigation: Εισαγωγή

Το **React Navigation** είναι το standard library για navigation σε React Native apps.

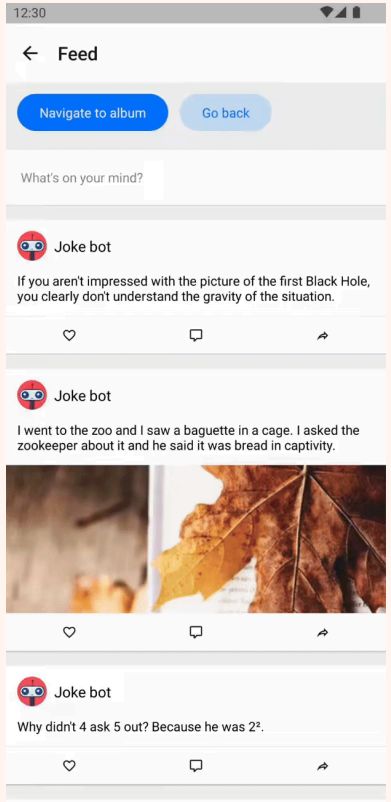
Τύποι Navigation:

1. **Stack Navigator**: Push/pop screens (native style)
2. **Tab Navigator**: Bottom tabs
3. **Drawer Navigator**: Sidebar menu

Stack Navigator: Παράδειγμα

Stack Navigator provides a way for your app to transition between screens where each new screen is placed on top of a stack.

By default the stack navigator is configured to have the familiar iOS and Android look & feel: new screens slide in from the right on iOS, use OS default animation on Android. But the animations can be customized to match your needs.

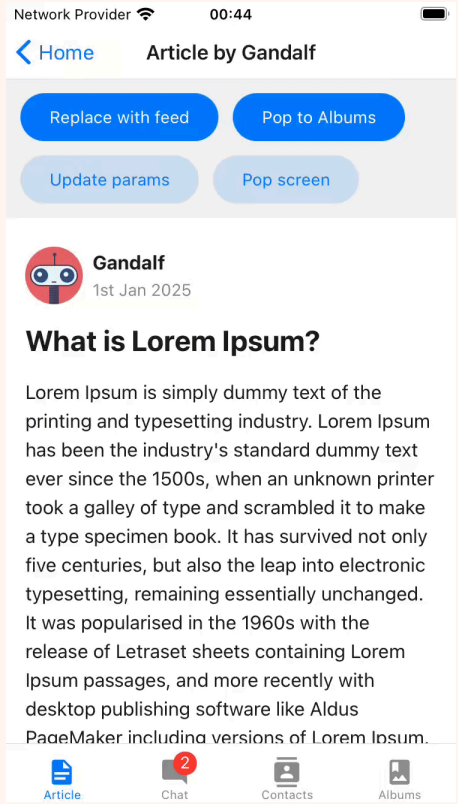


“ Δείτε: ”

- https://reactnavigation.org/assets/7.x/stack-android.mp4
- https://reactnavigation.org/assets/7.x/stack-ios.mp4

Tab Navigator

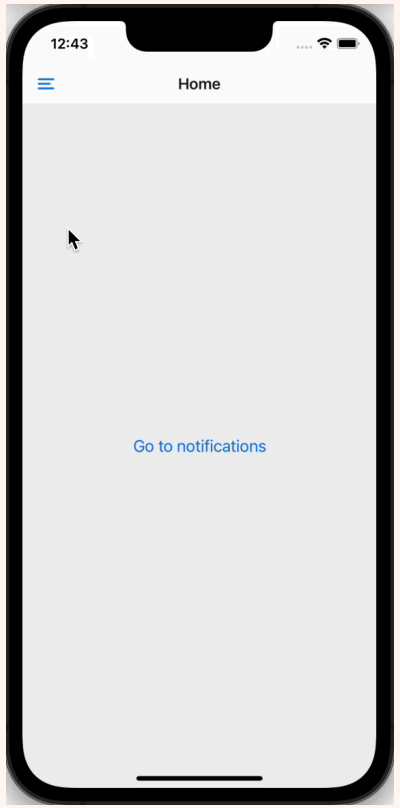
A simple tab bar on the bottom of the screen that lets you switch between different routes. Routes are lazily initialized -- their screen components are not mounted until they are first focused.



“ Δείτε: https://reactnavigation.org/assets/7.x/bottom-tabs.mp4 ”

Drawer Navigator

Common pattern in navigation is to use drawer from left (sometimes right) side for navigating between screens.



“ Δείτε: https://reactnavigation.org/assets/navigators/drawer/drawer.mp4 ”

Παραπομπές & Πηγές

- Official Documentation
  - https://reactnative.dev/docs/getting-started
  - https://reactnavigation.org/docs/getting-started