Ανάπτυξη Κινητών Εφαρμογών

Εργαστήριο

Δημήτρης Ρίγγας

riggas@ionio.gr

Τμήμα Πληροφορικής

Ιόνιο Πανεπιστήμιο



Εργαστήριο 3 – React Native Development

State & Interaction

Expo Go, useState, Touch & Input, Λίστες



Στόχος:

- ✓ **Expo Go** - Η εφαρμογή σε πραγματική συσκευή
- ✓ **React Hooks** - Εισαγωγή και κανόνες
- ✓ **useState Hook** - Διαχείριση local state (τιμές, objects, functional updates)
- ✓ **Touch events** - `TouchableOpacity`, `Pressable`, `onPress`
- ✓ **TextInput** - Controlled components
- ✓ **Conditional rendering & styling**
- ✓ **Λίστες από state** - `map`, `key`, immutable updates

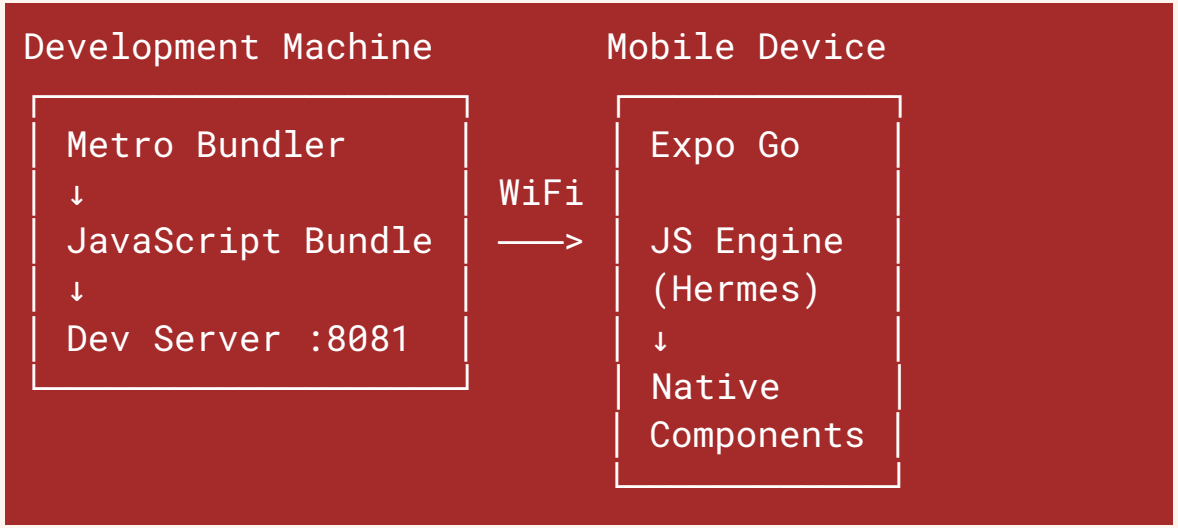
Συνεχίζουμε στο project `firstapp` του Lab 02

Expo Go - Η εφαρμογή στο κινητό

1. Εγκατάσταση **Expo Go** από App Store / Google Play
2. Κινητό & υπολογιστής στο **ίδιο WiFi**
3. `npx expo start` στον υπολογιστή
4. Scan το QR code
   - iOS: εφαρμογή Camera
   - Android: μέσα από το Expo Go

⚠️ **Δίκτυο πανεπιστημίου**: αν δεν συνδέεται
`npx expo start --tunnel`

⚠️ Το Expo Go υποστηρίζει **μόνο** την τελευταία έκδοση του Expo SDK → ενημερώστε το Expo Go ή το project (`npx expo install expo@latest --fix`)



- **Metro**: μετατρέπει JSX → JavaScript bundle
- **Expo Go**: έτοιμη app με pre-built native modules
- Αλλαγή κώδικα → νέο bundle → άμεση ενημέρωση, **χωρίς** native build

React Hooks - Εισαγωγή: Τι είναι τα Hooks;

Τα **Hooks** είναι functions που μας επιτρέπουν να "hook into" React features από **functional components**:

Πριν τα Hooks (Class Components):

- State: `this.state`, `this.setState()`
- Lifecycle: `componentDidMount()`, `componentDidUpdate()`
- Πολύπλοκος κώδικας, δύσκολη επαναχρησιμοποίηση

Με Hooks (Functional Components):

- State: `useState()`
- Side Effects: `useEffect()`
- Πιο απλός, καθαρός κώδικας
- Καλύτερη επαναχρησιμοποίηση logic

React Hooks - Κανόνες Χρήσης

1. Καλέστε Hooks μόνο στο top level

2. Καλέστε Hooks μόνο από React Functions

- Functional Components ✅
- Custom Hooks ✅
  Functions that themselves use hooks and start with `use`
- Regular JavaScript functions ❌

```
// ✅ Σωστό
const MyComponent = () => {
  const [count, setCount] = useState(0);

  return <View>...</View>;
};

// ❌ Λάθος - μέσα σε if
const MyComponent = () => {
  if (condition) {
    const [count, setCount] = useState(0); // NO!
  }
};

// ❌ Λάθος - μέσα σε regular js function
function doSomething() {
  const [count, setCount] = useState(0);
}
```

useState Hook - Διαχείριση Local State

Το `useState` είναι το πιο βασικό Hook για state management.

Το useState είναι ένα React Hook που επιτρέπει σε ένα function component να θυμάται δεδομένα μεταξύ renderings.
Χωρίς αυτό, κάθε φορά που το component θα "ξαναζωγραφιζόταν", όλες οι τιμές θα χάνονταν.

Μέρος	Ρόλος
`count`	Η **μεταβλητή κατάστασης** (state variable). Κρατά την τρέχουσα τιμή.
`setCount`	Η **συνάρτηση ενημέρωσης** (setter). Αλλάζει την τιμή του state.
`useState(0)`	Η αρχικοποίηση της τιμής (`0` στην αρχή).

Κάθε φορά που καλείτε το setCount, η React:

- Αλλάζει την αποθηκευμένη τιμή του count.
- Ξανακάνει render το component.
- Εμφανίζει τη νέα τιμή στην οθόνη.

Το state δεν είναι απλό variable — είναι δεμένο με τον κύκλο ζωής του component.

```jsx
import { useState } from 'react';

const Counter = () => {
  // Δήλωση state variable
  const [count, setCount] = useState(0);
  //     ↑state  ↑setter    ↑initial value

  const increment = () => {
    setCount(count + 1); // Ενημέρωση state
  };

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

useState - Πολλαπλές State Μεταβλητές

Μπορούμε να έχουμε πολλές state μεταβλητές στο ίδιο component:

```
const UserProfile = () => {
  const [name, setName] = useState('');
  const [age, setAge] = useState(0);
  const [email, setEmail] = useState('');
  const [isActive, setIsActive] = useState(false);

  return (
    <View>
      <TextInput
        value={name}
        onChangeText={setName}
        placeholder="Όνομα"
      />
      <TextInput
        value={email}
        onChangeText={setEmail}
        placeholder="Email"
      />
      <Switch value={isActive} onValueChange={setIsActive} />
    </View>
  );
};
```

useState - State ως Object

Για σύνθετα δεδομένα, χρησιμοποιούμε objects.

Όταν χρησιμοποιούμε useState με ένα object το React κρατάει ολόκληρο το αντικείμενο user σαν μία “μονάδα” state.

Όταν καλούμε setUser() με νέο αντικείμενο, η React **αντικαθιστά ολόκληρο** το προηγούμενο state με αυτό που δίνουμε.

Αν προσπαθήσουμε να ενημερώσουμε μόνο ένα πεδίο χωρίς το spread:
`setUser({ name: newName }); // ❌`
τότε το νέο state θα είναι πχ:
`{ name: 'Maria' }`
και όλα τα άλλα πεδία (email, age, preferences) θα χαθούν, επειδή δεν μεταφέρθηκαν στο νέο object.

Η σωστή λύση: Spread operator (`...`).

Αυτό λέγεται immutable update — δηλαδή, δημιουργούμε ένα νέο object αντί να αλλάζουμε το παλιό.

⚠️ Η React αναγνωρίζει ότι το state άλλαξε μόνο αν πάρει ένα νέο αντικείμενο. Αν αλλάξουμε απευθείας την ιδιότητα (user.name = 'Maria') χωρίς να καλέσουμε setUser, το React δεν θα κάνει re-render, γιατί δεν “βλέπει” τη μεταβολή.

```jsx
const UserProfile = () => {
  const [user, setUser] = useState({
    name: '',
    email: '',
    age: 0,
    preferences: {
      theme: 'light',
      notifications: true
    }
  });

  // !!! Πρέπει να κρατάμε τα υπόλοιπα properties
  const updateName = (newName) => {
    setUser({
      ...user,            // Spread existing properties
      name: newName       // Override specific property
    });
  };

  return <View>...</View>;
};
```

useState - Functional Updates

Όταν το νέο state εξαρτάται από το προηγούμενο, χρησιμοποιούμε **functional update**.

Το πρόβλημα των async updates είναι ότι η React δεν ενημερώνει το state αμέσως.
Αντίθετα, προγραμματίζει την ενημέρωση για το επόμενο render.
Αυτό σημαίνει ότι αν κληθεί πολλές φορές `setCount(count + 1)` στη σειρά όλες αυτές οι κλήσεις βλέπουν το ίδιο "παλιό" count.

Η λύση είναι η χρήση συνάρτησης μέσα στο setCount. Η παράμετρος prev να έχει την πιο πρόσφατη τιμή του count. Το prev δεν είναι κάποια "μαγική" λέξη· μπορεί να χρησιμοποιηθεί οποιοδήποτε όνομα παραμέτρου το οποίο απλώς αντιπροσωπεύει την τιμή *currentStateValue*.

```
const Counter = () => {
  const [count, setCount] = useState(0);

  // ❌ Πρόβλημα με async updates
  const incrementThreeTimes = () => {
    setCount(count + 1);
    setCount(count + 1);
    setCount(count + 1);
    // Result: count + 1 (not + 3!)
  };

  // ✅ Σωστό με functional update
  const incrementThreeTimesSafe = () => {
    setCount(prev => prev + 1);
    setCount(prev => prev + 1);
    setCount(prev => prev + 1);
    // Result: count + 3 ✓
  };
};
```

Touch Events: `onPress`

Component	Πότε
`Button`	Γρήγορα, **ελάχιστο** styling (native εμφάνιση)
`TouchableOpacity`	Δικό σας styling, "σβήνει" (opacity) όταν πατιέται
`Pressable`	Νεότερο, πιο ευέλικτο API (`pressed` state, `onLongPress`)

- Το `onPress` δέχεται **συνάρτηση**, όχι κλήση:
  `onPress={increment}` ✅
  `onPress={increment()}` ❌ (εκτελείται σε κάθε render!)
  `onPress={() => setCount(0)}` ✅
- `disabled={true}` → δεν ανταποκρίνεται (το στυλ το αλλάζετε εσείς)

```jsx
import { Text, TouchableOpacity, Pressable } from 'react-native';

<TouchableOpacity
  style={styles.button}
  onPress={() => setCount(prev => prev + 1)}
  disabled={count >= 10}
>
  <Text style={styles.buttonText}>+</Text>
</TouchableOpacity>

<Pressable
  onPress={reset}
  style={({ pressed }) => [
    styles.button,
    pressed && { opacity: 0.6 },
  ]}
>
  <Text style={styles.buttonText}>Reset</Text>
</Pressable>
```

TextInput - Controlled Component

Η τιμή του input **ζει στο state**:

- `value={name}` → το input δείχνει πάντα το state
- `onChangeText={setName}` → κάθε πλήκτρο ενημερώνει το state
- Μία "πηγή αλήθειας": μπορούμε να το αδειάσουμε με `setName('')`

Χρήσιμα props για κινητό:

- `placeholder`, `keyboardType="email-address"`
- `autoCapitalize="words"` / `"none"`
- `returnKeyType="done"` + `onSubmitEditing={...}`
- `secureTextEntry` (κωδικοί)

```
const [name, setName] = useState('');
const [greeting, setGreeting] = useState('');

const handleGreeting = () => {
  const trimmed = name.trim();
  setGreeting(trimmed
    ? `Γεια σου, ${trimmed}!`
    : 'Παρακαλώ εισάγετε το όνομά σας!');
  setName('');           // άδειασμα του input
};

<TextInput
  style={styles.input}
  placeholder="Το όνομά σας..."
  value={name}
  onChangeText={setName}
  returnKeyType="done"
  onSubmitEditing={handleGreeting}
/>
```

Conditional Rendering & Styling

Rendering — εμφάνιση ή όχι:

```
{atMax && <Text>🎉 Φτάσατε το όριο!</Text>}

{greeting ? (
  <Text style={styles.greeting}>{greeting}</Text>
) : null}
```

⚠️ `{count && <Text>...</Text>}` με `count === 0` εμφανίζει `0` → error
"Text strings must be rendered within a `<Text>` *component"*
✅ `{count > 0 && ...}`

Styling — style arrays (τα επόμενα υπερισχύουν):

```
const counterColor =
  count === 0 ? '#95a5a6'
  : count % 2 === 0 ? '#27ae60'
  : '#e74c3c';

<Text style={[styles.counterText,
              { color: counterColor }]}>
  Counter: {count}
</Text>

<TouchableOpacity
  style={[styles.button,
          atMax && styles.buttonDisabled]}
  disabled={atMax}
>
```

`false` / `null` μέσα στο array αγνοούνται

Λίστες από State: `map` + `key`

```
const [history, setHistory] = useState([]);

// Προσθήκη – ΝΕΟ array (νεότερο πρώτο)
setHistory(prev => [
  { id: Date.now().toString(), name: trimmed },
  ...prev,
]);

// Αφαίρεση / καθαρισμός
setHistory(prev => prev.filter(h => h.id !== id));
setHistory([]);

// Rendering
{history.map(item => (
  <Text key={item.id}>• {item.name}</Text>
))}
```

- ❌ `history.push(...)` → αλλάζει το **ίδιο** array, το React δεν "βλέπει" αλλαγή → **δεν** γίνεται re-render
- ✅ `[...prev, x]`, `.filter()`, `.map()` → νέο array (βλ. Lab 01, Άσκηση 6)
- `key`: **σταθερό & μοναδικό** ανά στοιχείο — βοηθά το React να καταλάβει τι άλλαξε
  - *Warning: Each child in a list should have a unique "key" prop*
- Μεγάλες λίστες που δεν χωράνε στην οθόνη → `ScrollView` / `FlatList` στο **Lab 04**

Bonus: Χωρισμός σε Components

```
App  (state: name, greeting, count)
├── GreetingInput   (value, onChangeText, label)
├── GreetingButton  (onPress, title)
├── GreetingDisplay (greeting)
└── Counter         (count)
```

- Το **state μένει στο** **App**
- Τα παιδιά παίρνουν **τιμές** και **callbacks** μέσω props
- Αναλυτικά για components & props → **Lab 04**

```jsx
function GreetingButton({ onPress, title }) {
  return (
    <TouchableOpacity style={styles.button}
                      onPress={onPress}>
      <Text style={styles.buttonText}>{title}</Text>
    </TouchableOpacity>
  );
}

// στο App
<GreetingButton title="Χαιρετισμός"
                onPress={handleGreeting} />
```

Και τώρα.. *hands on*!

Συνεχίζουμε στο:

https://github.com/riggas-ionio/mobile-apps/tree/class-2026/lab03



Παραπομπές & Πηγές

- **Official Documentation**
  - React – useState
  - React – Updating Objects / Arrays in State
  - React Native – Handling Touches · Pressable · TextInput
  - Expo Go
  - JavaScript Info

