THE COMPLETE MOBILE APP DEVELOPER ROADMAP

Go from zero to a mobile app developer in 12 months

Mosh Hamedani

Hi! I am Mosh Hamedani, a software engineer with over 20 years of experience.

Over the past 10 years, I've had the privilege of teaching millions of people how to code and become professional software engineers through my YouTube channel and online courses.

It's my mission to make software engineering accessible to everyone. Join me on this journey and unlock your potential in the world of coding!

https://codewithmosh.com

Table of Content

Introduction **4**
- Target Audience 4
- Resources 4

Essential Skills and Learning Timeline **5**
- React Native Path 5
- Flutter Path 5

Top Tips Every Beginner Should Know **6**

Native vs Cross-Platform Development **8**
- Native Development 8
- Cross-Platform Development 8
- React Native or Flutter? 9

Programming Languages **10**
- HTML/CSS 11
- JavaScript 13
- TypeScript 15
- Dart 16

Git **18**

Data Structures & Algorithms **19**

Design Patterns **20**

Mobile Frameworks **21**
- React 22
- React Native 23
- Flutter 24

Project Ideas **26**

Introduction

This guide is designed to help you navigate the essential skills needed to become a successful mobile app developer. Whether you're just starting out or looking to enhance your existing skills, this roadmap will provide a clear and structured path.

Target Audience

This guide is for:

- **Beginners** who want to know what they need to learn to land a mobile app developer job.
- **Experienced individuals** looking to level up their skills and fill in the gaps in their knowledge.

Resources

For detailed tutorials and full courses, check out the following resources:

- **YouTube Channel**: https://www.youtube.com/c/programmingwithmosh
- **Full Courses**: https://codewithmosh.com

Essential Skills and Learning Timeline

There are several paths to becoming a mobile app developer, with React Native and Flutter being the top two choices. React Native offers more job opportunities but comes with a steeper learning curve. Flutter provides fewer opportunities but is an easier route. We'll dive into these paths throughout this document.

React Native Path

Skill	Time Required	Learning Phase
HTML/CSS	1 month	Beginner
JavaScript	2 months	Beginner
TypeScript	1 month	Beginner
Git	2 weeks	Beginner
Data Structures & Algorithms	2 months	Intermediate
Design Patterns	2 months	Intermediate
React	2 months	Advanced
React Native	2 months	Advanced
Total	**1 Year**	

Flutter Path

Skill	Time Required	Learning Phase
Dart	2 month	Beginner
Git	2 weeks	Beginner
Data Structures & Algorithms	2 months	Intermediate
Design Patterns	2 months	Intermediate
Flutter	2 months	Advanced
Total	**8 Months**	

Top Tips Every Beginner Should Know

1. **Start Small and Build Up:** Begin with the basics and gradually move to more complex topics. Don't rush; build a strong foundation.

2. **Practice Consistently:** Set aside time each day or week to practice coding. Consistency is key to retaining knowledge and improving skills.

3. **Work on Projects:** Apply what you learn by working on real projects. This helps reinforce your knowledge and gives you practical experience.

4. **Ask for Help:** Ask ChatGPT for help or post your questions on StackOverflow to get help from the community. Additionally, participate in answering other people's questions. This is a great way to learn and reinforce your knowledge.

5. **Join a Community:** Engage with other learners and professionals. Join online forums, attend meetups, and participate in coding challenges. This can provide support, motivation, and valuable insights.

6. **Learn to Debug:** Debugging is a crucial skill. Practice finding and fixing errors in your code. It improves problem-solving skills and helps you understand how code works.

7. **Try Rubber Duck Debugging:** Explain your code and the problem you're facing to an inanimate object like a rubber duck. This method, known as rubber duck debugging, can help you think more clearly and often leads to discovering the solution on your own.

8. **Read Documentation:** Familiarize yourself with official documentation. It's an essential skill for understanding and using new tools and technologies effectively.

9. **Stay Updated:** Mobile app development is always evolving. Follow industry blogs, news, and social media to stay informed about the latest trends and updates.

10. **Be Patient and Persistent:** Learning to code is a journey. Be patient with yourself and stay persistent, even when things get tough. Progress takes time and effort.

11. **Embrace Change:** The mobile app industry is always evolving with new versions of languages and tools being released frequently. Dealing with breaking changes is a common challenge, and many courses and books can become outdated quickly. Be patient and embrace these changes. Practicing how to handle them will prepare you for real-world scenarios, as this is a regular part of a mobile app developer's job.

12. **Take Regular Breaks:** Stepping away from your desk and taking regular breaks can refresh your mind and help you find solutions to problems. Sometimes, a short walk or a change of scenery is all you need to spark new ideas and improve your productivity.

Good luck, and happy coding!

Mosh

Native vs Cross-Platform Development

There are basically two ways to build mobile apps: native development and cross-platform development.

Native Development

With native development, we build an app that runs specifically on a particular platform like iOS or Android. This approach gives us full control over the capabilities of the device, resulting in excellent performance and a superior user experience. However, creating the same app for a different platform requires a separate project in a different language and ecosystem, which means two separate projects and two sets of bugs to fix.

Cross-Platform Development

Cross-platform (or multi-platform) comes to the rescue here. By reusing the same code for different platforms, we can reduce development time and potentially reduce bugs. Fixing a bug once means it's fixed across multiple platforms.

These days, most companies prefer to use cross-platform development to reduce costs. If you're starting out, I recommend focusing on cross-platform development for better job opportunities.

There are several popular cross-platform solutions available:

- React Native
- Flutter
- .NET MAUI
- Kotlin Multiplatform

React Native and Flutter are the two most popular options.

React Native or Flutter?

The choice between React Native and Flutter depends on two main factors:

1. **Job Opportunities:** There are often more job opportunities available for React Native developers compared to Flutter developers. Do your own research to see the number of job opportunities for React Native vs Flutter where you live.

2. **Learning Curve:** While there may be more jobs available for React Native developers, there are also many more things to learn to become proficient in React Native. Flutter, on the other hand, is an easier and shorter path.

If you want more job opportunities, go for React Native. If you want an easier route, go for Flutter.

Programming Languages

Once you've chosen your cross-platform solution, the next step is to pick the right programming language(s).

React Native is built on top of web technologies, so you'll need a basic understanding of HTML and CSS, along with a solid grasp of JavaScript and ideally TypeScript.

For Flutter, you'll need to learn Dart.

React Native Path	Flutter Path
HTML/CSS	Dart
JavaScript	
TypeScript (optional but recommended)	

As you can see, the React Native path has a deeper learning curve. Plus, the JavaScript ecosystem can be chaotic and often overwhelming for beginners. In contrast, Dart is a cleaner, more beginner-friendly language without the quirks and complexities of JavaScript.

Remember to compare job opportunities for React Native and Flutter in your area. This will help you decide which path is the best fit for you.

Next, we'll explore each of these languages and the specific concepts you need to know.

HTML/CSS

HTML (Hypertext Markup Language) and CSS (Cascading Style Sheets) are the foundational languages of web development. HTML defines the structure of web pages, while CSS styles them.

While you don't need to master HTML/CSS for React Native, I recommend spending a few weeks learning the basics.

Time required: 1-2 weeks

Learning resources: YouTube Tutorial | Full Course

Essential HTML Concepts

- **Basic Tags:** <html>, <head>, <body>, <title>
- **Text Formatting:** <h1> to <h6>, <p>,
, ,
- **Lists:** , <ol>,
- **Links:** <a>, href, target
- **Images:** <img>, src, alt, width, height
- **Tables:** <table>, <tr>, <td>, <th>, colspan, rowspan
- **Forms:** <form>, <input>, <textarea>, <button>, <select>, <option>, <label>

Essential CSS Concepts

- **Selectors:** element, class, id, attribute, pseudo-class, pseudo-element
- **Box Model:** margin, border, padding, content
- **Positioning:** static, relative, absolute, fixed, sticky
- **Display:** block, inline, inline-block, none, flex, grid

- **Flexbox:** justify-content, align-items, flex-direction, flex-wrap
- **Grid:** grid-template-columns, grid-template-rows, gap, grid-area
- **Typography:** font-family, font-size, font-weight, line-height, text-align, text-decoration
- **Colors:** color, background-color, opacity, rgba, hex, hsl
- **Units:** px, em, rem, %, vh, vw

JavaScript

JavaScript is a programming language that adds interactivity and dynamic behavior to web pages. It handles tasks like user interactions, form validation, animations, and fetching data from servers, making web pages more engaging and functional.

Time required: 6-8 weeks

Learning resources: YouTube Tutorial | Full Course

Essential Concepts

- **Variables:** declarations (var, let, const), scope (block, functional, global), hoisting
- **Data Types:** primitive types (strings, number, boolean, undefined, null, Symbol), Object, typeof operator
- **Type Casting:** explicit casting, implicit casting, type conversion vs coercion
- **Operators:** assignment, comparison, arithmetic, bitwise, logical, conditional
- **Equality Comparisons:** ==, ===, Object.is
- **Control Flow:** if, else, switch
- **Loops:** for, for...in, for...of, while, do...while, break, continue
- **Functions:** function declaration, function expression, arrow functions, parameters, return values
- **Arrays:** creation, methods (push, pop, shift, unshift, map, filter, reduce)
- **Objects:** creation, properties, methods, this keyword
- **Classes**

- **Data Structures:** Map, WeakMap, Set, WeakSet, JSON
- **Error Handling:** try, catch, finally, throw, Error objects
- **Asynchronous JavaScript:** Promises, async/await, callbacks, callback hell
- **DOM Manipulation:** document.getElementById, document.querySelector, addEventListener, innerHTML, style
- **Events:** click, submit, load, change, focus, blur, event propagation (bubbling and capturing)
- **Working with APIs:** fetch
- **Browser Storage:** local storage, web storage
- **Modules:** ECMAScript Modules

TypeScript

TypeScript is a superset of JavaScript that adds static typing and other features, making code more robust and maintainable. It helps catch errors early during development and is widely used in large-scale applications.

While not essential, TypeScript is often used in newer React Native projects to enhance code quality. I recommend spending a few weeks learning TypeScript to increase your job opportunities.

Time required: 2-3 weeks

Learning resources: YouTube Tutorial | Full Course

Essential Concepts

- **Basics Types:** string, number, boolean, array, tuple, enum, any, void, null, undefined, never, unknown
- **Type Assertion:** as keyword, <> syntax
- **Interfaces:** defining, extending, optional properties, readonly properties, dynamic keys
- **Classes:** properties, methods, constructors, inheritance, access modifiers (public, private, protected)
- **Functions:** type annotations, optional and default parameters, rest parameters
- **Generics:** generic functions, generic classes
- **Modules:** import, export, namespaces
- **Utility Types:** Partial, Pick, Omit, Readonly, Record, Exclude, etc

Dart

Dart is a client-optimized language for fast apps on any platform. It's optimized for building user interfaces with features like sound null safety, async-await for asynchronous programming, and rich standard libraries.

Time Required: 2 months

Essential Concepts

- **Variables:** declarations (var, final, const), types (int, double, String, bool), type inference
- **Data Types:** numbers (int, double), strings (String), booleans (bool), lists (List), sets (Set), maps (Map), Runes, Symbols
- **Operators:** arithmetic, equality, relational, logical, bitwise, assignment
- **Control Flow:** if, else, switch, assert
- **Loops:** for, for...in, while, do...while, break, continue
- **Functions:** function declaration, arrow functions, parameters (required, optional, named), return values, anonymous functions, closures
- **Exception Handling:** try, catch, finally, throw, custom exceptions
- **Collections:** lists (fixed-length, growable), sets, maps, collection if, collection for, spread operators
- **Classes and Objects:** class declaration, constructors (default, named, factory), properties, methods, this keyword, inheritance, mixins, abstract classes, interfaces
- **Generics:** using generics in classes, methods, and collections

- **Libraries and Packages:** importing libraries, creating and using packages, pub package manager
- **Asynchronous Programming:** Futures, async/await, Stream, StreamBuilder
- **Null Safety:** sound null safety, nullable types, late keyword, null-aware operators (?., ??, ??=)
- **Meta Programming:** annotations, reflection

Git

Git is a version control system that tracks changes in code, allowing multiple developers to collaborate efficiently. It helps manage and maintain different versions of code, facilitates branching and merging, and stores the project history.

Time required: 1-2 weeks

Learning resources: YouTube Tutorial | Full Course

Essential Concepts

- **Setup and Configuration**: init, clone, config
- **Staging**: status, add, rm, mv, commit, reset
- **Inspect and Compare**: log, diff, show
- **Branching**: branch, checkout, merge
- **Remote Repositories**: remote, fetch, pull, push
- **Temporary Commits**: stash
- **GitHub**: fork, pull request, code review

Data Structures & Algorithms

Data structures and algorithms are core topics taught to computer science students but often skipped by self-taught developers. However, mastering them is crucial for boosting your programming and problem-solving skills. They're also frequently tested in tech interviews, so understanding these concepts will give you a significant advantage when job hunting.

Time required: 2 months

Learning resources: YouTube Tutorial | Full Course

Essential Concepts

- **Big O Notation**
- **Arrays and Linked Lists**
- **Stacks and Queues**
- **Hash Tables**
- **Trees and Graphs**: Binary trees, AVL trees, heaps, tries, graphs
- **Sorting Algorithms**: Bubble sort, selection sort, insertion sort, merge sort, quick sort, counting sort, bucket sort
- **Searching algorithms**: Linear search, binary search, ternary search, jump search, exponential search
- **String Manipulation Algorithms**: Reversing a string, reversing words, rotations, removing duplicates, most repeated character, anagrams, palindrome
- **Recursion**

Design Patterns

Design patterns are proven solutions to common software design problems. There are 23 classic design patterns that were documented in the book: “Design Patterns: Elements of Reusable Object-oriented Software” by the Gang of Four. Many of these patterns are used in mobile frameworks. So learning them will give you a deeper understanding of object-oriented design principles and how these mobile frameworks work under the hood.

Time required: 2 months

Learning resources: YouTube Tutorial | Full Course

Essential Concepts

- **Object-oriented Programming:** Classes, interfaces, encapsulation, abstraction, inheritance, polymorphism, coupling
- **Creational Patterns:** Prototype, singleton, factory method, abstract factory, builder
- **Structural Patterns:** Composite, adapter, decorator, facade, flyweight, bridge, proxy
- **Behavioral Patterns:** Memento, state, iterator, strategy, template method, command, observer, mediator, chain of responsibility, visitor

Mobile Frameworks

There are many frameworks for building cross-platform mobile apps, with React Native and Flutter being the two most popular ones.

React Native is a popular framework developed by Facebook for building mobile apps using JavaScript. It's built on top of React, a library for building user interfaces. This means that to effectively use React Native, you should first learn React. React Native allows you to create truly native apps while sharing the majority of your code across different platforms.

Flutter is a framework developed by Google for building natively compiled applications for mobile, web, and desktop from a single codebase. It uses the Dart programming language and provides a rich set of pre-designed widgets to create beautiful and responsive UIs. Flutter is known for its fast development cycles and expressive, flexible UI.

React

React is a popular JavaScript library for building user interfaces, particularly single-page applications. It allows developers to create reusable UI components, manage application state efficiently, and handle dynamic data changes.

Time required: 6-8 weeks

Learning resources: YouTube Tutorial | Full Course

Essential Concepts

- **Basics:** Components, props, state, JSX
- **Rendering:** Conditional rendering, rendering lists
- **Hooks:** useState, useEffect, useReducer, useRef, custom hooks
- **Styling:** Using vanilla CSS, CSS modules, CSS-in-JS
- **Forms:** react-hook-forms, zod
- **Data Fetching:** fetch API, axios
- **State Management:** Lifting state up, Context API, React Query
- **Routing:** React Router

React Native

React Native is a framework for building cross-platform mobile applications using React. It allows developers to write code once and deploy it to both iOS and Android platforms, leveraging native components for a seamless user experience.

Time required: 6-8 weeks

Learning resources: YouTube Tutorial | Full Course

Essential Concepts

- **Basics:** View, Text, Image, Touchables, Button, Alert, platform-specific code
- **Layouts:** dimensions, detecting orientation, Flexbox, absolute and relative
- **Styling:** StyleSheet, borders, shadows, padding, styling text, icons, platform-specific styles
- **Input Components:** TextInput, Switch, Picker, custom pickers
- **Navigation:** React Navigation, stack navigator, tab navigator, drawer navigator
- **Native Modules:** linking native code, using third-party libraries
- **Offline Support:** detecting network status, caching, AsyncStorage
- **Authentication:** auth providers
- **Notifications:** push notification services
- **Distribution:** optimizing assets, building, error reporting, environment management

Flutter

Flutter is an open-source framework by Google for building natively compiled applications for mobile, web, and desktop from a single codebase. It uses the Dart programming language and provides a rich set of pre-designed widgets to create beautiful and responsive UIs.

Time required: 2 months

Essential Concepts

- **Widgets:** StatelessWidget, StatefulWidget, basic widgets (Text, Image, Icon, Button, Scaffold, Container, Row, Column)
- **Layouts:** alignment, padding, margin, sizing, BoxDecoration, Flex, Expanded, ListView, GridView
- **State Management:** setState, InheritedWidget, Provider, Riverpod, Bloc
- **Navigation:** Navigator, routes, named routes, passing data between screens
- **Forms and Input:** Form, TextFormField, handling form validation, focus management
- **Networking:** HTTP requests using http package, JSON parsing, error handling
- **Asynchronous Programming:** Futures, async/await, Stream, StreamBuilder
- **Storage:** local storage, file storage, SQLite
- **Animations:** implicit animations (AnimatedContainer, AnimatedOpacity), explicit animations (AnimationController, Tween, AnimatedBuilder)
- **Themes and Styles:** defining and using themes, custom fonts, icons

- **Testing:** unit testing, widget testing, integration testing using flutter_test package

- **Deployment:** preparing for app release, publishing to Play Store and App Store

Project Ideas

Fitness Tracker

Create a fitness tracker application to log workouts and track progress.

- Log workouts with details like type, duration, and calories burned
- Display a summary of daily/weekly workouts
- Track progress with charts and statistics
- Set and track fitness goals

Expense Tracker

Develop an expense tracker application to manage personal finances.

- Add new expenses with details such as amount, category, and date
- Display a list of expenses with filtering options
- Visualize expenses with charts (e.g., pie chart for categories)
- Calculate total expenses and display summary statistics

News App

Create a news application that fetches and displays the latest news articles.

- Fetch news articles from a news API
- Display a list of news articles with headlines and images
- View detailed news articles
- Filter news by categories (e.g., sports, technology, politics)

E-commerce App

Build an e-commerce application with product listings, a shopping cart, and checkout functionality.

- Display a list of products with images, prices, and descriptions
- Add products to a shopping cart
- View the shopping cart with a list of selected products and total price
- Checkout process with order summary

Learning to code is a journey. Be patient with yourself and stay persistent, even when things get tough.

- Mosh