THE COMPLETE BACK-END DEVELOPER ROADMAP

Go from zero to a back-end 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**

Top Tips Every Beginner Should Know **6**

Languages **8**
- Python 9
- Java 10
- Project Ideas 11

Git **13**

Data Structures & Algorithms **14**

Design Patterns **15**

Databases **16**
- MySQL 17
- Project Ideas 18
- MongoDB 21

Web Frameworks **22**
- Django 23
- Spring Boot 24
- Project Ideas 25

Additional Skills **29**

Introduction

This guide is designed to help you navigate the essential skills needed to become a successful back-end 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 back-end 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

Back-end development has many tools and technologies. Trying to learn them all is impossible and not practical. This guide focuses on the most important and widely used skills and tools to help you get the best job opportunities.

I've selected these skills because they are in high demand. Mastering them will give you a strong foundation and make you a competitive job candidate.

For the first 12 months, focus only on the tools and technologies listed in this document. Instead of trying to learn too many things at once, build a strong foundation with these essential skills. You can always learn other tools and technologies on the job as you go.

Skill	Time required	Learning Phase
Programming	2 months	Beginner
Git	2 weeks	Beginner
Data Structures & Algorithms	2 months	Beginner
Design Patterns	2 months	Intermediate
Databases	2 months	Intermediate
Web Framework	2 months	Advanced
Total	**11 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:** Back-end 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. **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

Languages

You have many choices when it comes to selecting a programming language for backend development, and choosing the right one depends on various factors like project requirements, team expertise, and performance needs.

Language	Use Case
JavaScript	Full-stack development (used both on client and server)
Python and Ruby	Rapid prototyping and fast development cycles
Java and C#	Building large-scale, enterprise-grade applications
Go	For performance-critical and concurrent applications.

Python, Java, and JavaScript are among the most popular languages for backend development.

My Recommendation

- Pick **only one language** from this list.
- To decide what language is right for you, do some research and find the job opportunities available for each language where you live.
- For beginners, I'd recommend **Python** because it's the easiest.
- For more serious leaners, I'd recommend **Java** because it is a classic language and is taught to computer science students. Once you learn Java, you can easily learn other languages, particularly C-based languages (C, C#, C++, JavaScript, etc).

Python

Python is a highly popular language for backend development, known for its simplicity, readability, and extensive library support. It's widely used for building scalable and robust web applications, thanks to frameworks like Django and Flask.

Time required: 2 months

Course: Complete Python Mastery

Essential Concepts

- **Basics**: Variables, data types, type conversion
- **Control Flow**: Comparison operators, logical operators, if, elif, else, for, for..else, iterables, while
- **Functions**: Defining functions, arguments, keyword args, default args, xargs, xxargs, scope
- **Data Structures**: Arrays, lists, tuples, sets, stacks, queues, dictionaries, comprehensions, generator expressions
- **Exception Handling**: try/except, with statements, raising exceptions
- **Object-oriented Programming**: Classes, constructors, instance vs class members, magic methods, private members, properties, inheritance, method overriding, Object class, abstract base classes, polymorphism, duck typing
- **Modules**: Built-in modules, creating modules, packages, sub-packages
- **Python Standard Library**: Working with paths, files, directories, CSV, JSON, date/times, random values
- **Package Management**: Pypi, pip, virtual environments, pipenv, Pipfile

Java

Java is a robust and versatile language, perfect for backend development. Known for its performance and scalability, Java is a top choice for building large-scale, enterprise-level applications.

Time required: 2 months

Course: The Ultimate Java Series

Essential Concepts

- **Basics**: Variables, primitive and reference types, constants, casting, arrays, arithmetic expressions
- **Control Flow**: Comparison operators, logical operators, ternary operator, if/else, switch/case, for, foreach, while, do..while, break, continue
- **Object-oriented Programming**: Classes, objects, getters/setters, constructors, method overloading, static members, inheritance, access modifiers, method overriding, upcasting/downcasting, polymorphism, abstract classes, final classes, interfaces
- **Exceptions**: Exceptions hierarchy, try/catch/finally, throw, custom exceptions
- **Generics**: Generic classes/methods/interfaces, constraints, type erasure, type parameters
- **Collections Framework**: Iterable, Iterator, Collection, List, Comparable, Comparator, Queue, Set, Map
- **Functional interfaces**: Lambda expressions, Consumer, Supplier, Function, Predicate, BinaryOperator, UnaryOperator
- **Streams**: Creating streams, mapping, filtering, slicing, sorting, peeking, reducing

Project Ideas

When you're just starting out and learning a programming language, building command-line applications is a great way to practice your skills. Here are a few project ideas, ordered from simple to complex:

1. Calculator

Build a basic calculator that can perform operations like addition, subtraction, multiplication, and division. This will give you practice with functions and control flow.

2. Number Guessing Game

Develop a game where the program randomly selects a number, and the user has to guess it. Provide feedback if the guess is too high or too low. This project helps you understand loops and conditional statements.

3. Unit Converter

Build a unit converter that can convert between different units of measurement (e.g., kilometers to miles, Celsius to Fahrenheit). This project will strengthen your understanding of functions and user input.

4. Password Generator

Develop a password generator that creates random, secure passwords based on user-defined criteria (e.g., length, inclusion of special characters). This project will help you understand random number generation and string handling.

5. Word Counter

Create a program that counts the number of words, characters, and lines in a given text file. This will give you experience with file I/O operations and string manipulation.

6. To-Do List

Create a simple to-do list application where users can add, remove, and mark tasks as complete. This project will help you practice working with lists and user input.

7. Simple Quiz

Create a quiz application that asks the user multiple-choice questions and provides feedback on their answers. This will help you practice working with lists, conditionals, and user input.

8. Contact Book

Design a command-line application to store and manage contacts. Users should be able to add, view, and delete contacts. This will help you practice working with data structures like lists or dictionaries.

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

Course: The Ultimate Git Course

Essential Concepts

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

Course: The Ultimate Data Structures & Algorithms 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 web frameworks, particularly Spring, Django, and ASP.NET Core. So learning these design patterns will give you a deeper understanding of object-oriented design principles and how these web frameworks work under the hood.

Time required: 2 months

Course: The Ultimate Design Patterns 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

Databases

Understanding databases is a fundamental skill for backend developers. There are two main types of databases you should know about: *Relational* and *NoSQL*. Each has its own use cases and benefits, and knowing when to use which can make a big difference in your applications.

Relational Databases

- **Storage:** Data is stored in tables with rows and columns, much like a spreadsheet.
- **Examples**: MySQL, PostgreSQL, SQLite, SQL Server, and Oracle.
- **Use Case**: Best for applications that require complex queries and reporting, such as banking and financial systems.

NoSQL Databases

- **Storage:** Data is stored without a predefined structure, making it more flexible for different types of data.
- **Examples**: MongoDB, CouchDB, and Cassandra.
- **Use Case**: Best for applications that require flexible data models, such as real-time analytics, content management systems, and IoT applications.

You don't need to learn all these database management systems. Only one is enough to get started. You can learn about the others when needed.

My recommendation: To build a strong foundation, start with MySQL for relational databases followed by MongoDB for NoSQL.

MySQL

MySQL is a widely used open-source relational Database Management System (DBMS) that's perfect for beginners. Whether you're building a small web application or a large enterprise system, MySQL provides a solid foundation for managing your data.

Time required: 1 month

Course: The Complete SQL Mastery

Essential Concepts

- **Querying Data**: SELECT, WHERE, logical operators (AND, OR, NOT), IN, BETWEEN, LIKE, REGEXP, IS NULL, ORDER BY, LIMIT
- **Joins**: Inner joins, outer joins, self joins, natural joins, cross joins, unions
- **Complex Queries**: Aggregate functions (MAX, MIN, AVG, SUM, COUNT), GROUP BY, HAVING, ROLLUP, subqueries
- **Data Manipulation**: INSERT, UPDATE, DELETE
- **Views**
- **Stored Procedures and Functions**
- **Triggers and Events**
- **Transactions**: Transaction isolation levels, BEGIN, COMMIT, ROLLBACK
- **Database Design**: Tables, relationships, primary keys, foreign keys, normalization
- **Indexes**
- **Security**: Managing users and privileges

Project Ideas

Here are a few project ideas for practicing your MySQL skills, ordered from simple to complex:

1. Simple Address Book

Design a database for storing contact information. Create tables for storing names, phone numbers, and email addresses. Write queries to add new contacts, update existing ones, delete contacts, and retrieve all contacts.

2. Library Management System

Create a database schema for a library. Design tables for books, authors, and borrow records. Write queries to:

- Add new books and authors
- Record borrowing and returning of books
- List all available books
- Find books by a specific author

3. Student Grades Database

Design a database to store student grades, including tables for students, courses, and grades. Write queries to:

- Add students and courses
- Record student grades
- Calculate average grades for a student
- List students with the highest grades in a course

4. Inventory Management System

Create a schema for an inventory system. Design tables for products, suppliers, and stock levels. Write queries to:

- Add new products and suppliers
- Update stock levels
- List all products with low stock
- Find suppliers for a specific product

5. Event Registration System

Design a database for managing event registrations, including tables for events, attendees, and registrations. Write queries to:

- Add new events and attendees
- Register attendees for events
- List all attendees for a specific event
- Find events a particular attendee has registered for

6. Employee Management System

Develop a database schema for an employee management system with tables for employees, departments, roles, and salaries. Write queries to:

- Add employees and departments
- Update employee roles and salaries
- Generate department-wise employee lists
- Calculate average salaries by department

7. Online Store Database

Create a schema for an online store, including tables for products, customers, orders, and payments. Write queries to:

- Add new products and customers
- Record orders and payments
- List all orders for a customer
- Calculate total sales for a period

8. Movie Rental System

Design a database for a movie rental store, with tables for movies, customers, rentals, and returns. Write queries to:

- Add movies and customers
- Record rentals and returns
- List all currently rented movies
- Calculate late fees for overdue returns

MongoDB

MongoDB is a powerful, flexible, and scalable NoSQL database that is perfect for handling large volumes of unstructured or semi-structured data.

If you're just starting out, you can skip MongoDB as it's often not required by entry-level jobs. However, familiarity with NoSQL databases can be an advantage.

Time requires: 1 month

Essential Concepts

- **Basics**: SQL vs NoSQL, documents and collections, data types
- **Methods:** insert(), find(), update(), deleteOne(), bulkeWrite()
- **Comparison Operators:** $eq, $gt, $lt, $lte, $gte, $ne
- **Logical Operators:** $and, $or, $not, $nor
- **Array Operators:** $in, $nin, $all, $elemMatch, $size
- **Element Operators:** $exists, $type, $regex
- **Projection Operators:** $project, $include, $exclude, $slice
- **Indexes:** Single field, compound, text
- **Aggregation:** $match, $group, $sort, $project, $skip, $limit, $unwind, $lookup, $sum
- **Transactions**
- **Security**: Managing user roles, authentication, and authorization

Web Frameworks

Web frameworks provide us with tools and a structured approach to building backend applications. They simplify the development process by offering pre-built components and functionalities, so we can focus on creating our application rather than dealing with repetitive tasks like routing, database connections, and user authentication. By using web frameworks, we can develop faster, write cleaner code, and ensure our applications are scalable and maintainable.

Language	Web Framework
Python	Django
Java	Spring Boot
JavaScript	Express.js
C#	ASP.NET Core
Ruby	Ruby on Rails
Go	Gin

You don't need to learn all of these frameworks. Just focus on learning one that matches the programming language you're skilled at.

Django

Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. It's known for its "batteries-included" philosophy, providing a wide range of built-in features to help you build robust and scalable web applications quickly.

Time required: 2 months

Course: The Ultimate Django Series

Essential Concepts

- **Basics:** Models, views, URLs, templates
- **Models:** Creating models, fields, relationships, generic relationships
- **Migrations:** Creating, running, reverting
- **Django ORM:** Managers, QuerySets, CRUD operations, sorting, limiting, selecting and deferring fields
- **Admin Interface:** Customizing pages, custom actions, custom forms
- **RESTful APIs:** Resources, HTTP methods, API views, serializers
- **Advanced API Concepts:** Class-based views, mixins, generic views, ViewSets, routers, filtering, searching, sorting, pagination
- **Authentication:** Managing user models, profiles, groups, permissions
- **Deployment**

Spring Boot

Spring Boot is a powerful and widely-used framework for building enterprise-level applications in Java. It simplifies the development process by providing pre-configured templates and reducing the need for boilerplate code, enabling rapid development and deployment.

Time required: 2 months

Essential Concepts

- **Spring Core:** Dependency Injection (DI), Spring IOC, Spring AOP, Spring MVC, annotations, configuration
- **Spring Boot:** Starters, auto configuration, actuators, embedded server, hibernate
- **Database Integration:** Spring Data JPA, Spring Data MongoDB
- **Security:** Spring Security (authentication, authorization, OAuth2, JWT authentication)
- **Testing:** JPA Test, MockMVC
- **Deployment**

Project Ideas

Here are some project ideas, ordered from simple to complex:

1. Personal Blog API

Create an API for a personal blog where users can create, read, update, and delete posts. Implement user authentication and basic CRUD operations.

Key Features:

- User authentication (signup, login, logout)
- CRUD operations for blog posts
- Endpoints for managing user profiles

2. To-Do List API

Build an API for a to-do list application where users can manage their tasks. Include user authentication and the ability to categorize tasks.

Key Features:

- User authentication
- CRUD operations for tasks
- Categorization and filtering of tasks
- Endpoints for task management

3. Simple E-commerce API

Develop an API for a basic e-commerce site where users can browse products, add them to a cart, and checkout. Include user authentication and order management.

Key Features:

- User authentication
- CRUD operations for products
- Shopping cart management
- Order processing and management

4. Event Management API

Create an API for managing events, where users can create events, register for events, and see a list of attendees.

Key Features:

- User authentication
- CRUD operations for events
- Registration and attendee management
- Endpoints for event management

5. Social Media API

Build an API for a social media platform where users can create profiles, post updates, follow other users, and like posts.

Key Features:

- User authentication
- CRUD operations for posts
- User following and unfollowing

- Like and comment functionality

6. Online Learning Platform API

Develop an API for an online learning platform where instructors can create courses, and students can enroll in and complete courses. Include progress tracking and course reviews.

Key Features:

- User authentication
- CRUD operations for courses
- Enrollment and progress tracking
- Course review management

7. Job Board API

Create an API for a job board where companies can post job listings, and users can apply for jobs. Include user authentication and resume uploads.

Key Features:

- User authentication
- CRUD operations for job listings
- Job application submission and tracking
- Resume management

8. Real-Time Chat API

Build an API for a real-time chat application where users can join chat rooms and send messages. Implement user authentication and real-time message updates.

Key Features:

- User authentication
- CRUD operations for chat rooms
- Real-time messaging using WebSockets
- Message history and retrieval

Additional Skills

As you advance in your backend development career, learning additional skills can help you reach senior levels. Here are some key areas to focus on:

- **Database Management:** Query optimization, replication, sharding, transactions, concurrency
- **API Design:** GraphQL, gRPC
- **Security:** OAuth, JWT, TLS/SSL, common security vulnerabilities (SQL injection, XSS, CSRF)
- **Scalability and Performance**: Load balancing, caching (Redis, Memcached), asynchronous processing with message queues (RabbitMQ)
- **System Design and Architecture:** Microservices, Service-Oriented Architecture (SOA), Event-Driven Architecture, Domain-Driven Design (DDD), Command Query Responsibility Segregation (CQRS), Event Sourcing
- **DevOps and CI/CD:** Jenkins, GitHub Actions, Docker, Kubernetes
- **Cloud Computing:** AWS, Azure, Google Cloud

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

- Mosh