Backend Development Roadmap

Phase 1: Core Node.js Mastery (Week 1–2)

🧠 Key Concepts

- **Asynchronous Programming**
  - Callback functions
  - Promises
  - `async/await` patterns
- **Event Loop & Concurrency**
  - Call Stack, Task Queue, Event Queue
  - Event loop phases and timers
- **Streams**
  - Types: Readable, Writable, Duplex, Transform
  - Use cases: Large file processing, data piping
- **Error Handling**
  - Error-first callback style
  - `try/catch` with `async/await`
  - Custom error classes
- **Performance Optimization**
  - Non-blocking code practices
  - Avoiding memory leaks
  - Monitoring performance bottlenecks
- **Cluster Module**
  - Forking child processes
  - Load balancing across CPU cores

- **Core Node Modules**
  - `fs`, `path`, `os`, `crypto`, `url`, `buffer`
- **Memory Management**
  - Detecting and debugging memory leaks
  - Tools: `clinic.js`, Chrome DevTools heap profiling

Phase 2: Express.js Advanced (Week 3–4)

📌 Core Concepts

- **Express.js Fundamentals Refresher**
  - How Express handles HTTP requests/responses
  - Request-Response lifecycle

🧩 Middleware

- **Types of Middleware**
  - Application-level
  - Router-level
  - Error-handling middleware
  - Built-in vs. Custom middleware
- **Practical Use Cases**
  - Logging, body parsing, authentication, error tracking

🛡️ Security & Protection

- **CORS (Cross-Origin Resource Sharing)**
  - Configuring CORS with `cors` package
  - Pre-flight requests and headers
- **Helmet.js**
  - Setting HTTP headers for security
  - Preventing XSS, clickjacking, MIME sniffing
- **Rate Limiting & Throttling**

    - Using `express-rate-limit` and `Redis` to prevent DDoS attacks
- **Session Management**
    - Sessions vs. JWT
    - Implementing Redis-backed sessions
    - Cookie settings and security (HttpOnly, Secure, SameSite)

🛠️ Input Handling

- **Validation & Sanitization**
    - Using `express-validator`
    - Escaping user input
- **Custom Validators**
    - Writing schema-specific custom rules

🗂️ API Architecture

- **API Versioning Techniques**
    - URI versioning: `/api/v1/`
    - Header-based versioning
- **Organizing Route Handlers**
    - Modular routers
    - Error responses structure (standardizing errors)

Phase 3: MongoDB Deep Dive (Week 5–6)

🧠 NoSQL vs SQL

- Key differences: schema, joins, normalization, scalability
- When to choose MongoDB over relational databases

🧱 Schema Design with Mongoose

- **Defining Mongoose Models**
    - Data types, default values, required fields

- **Validation & Middleware**
    - Pre/post hooks
    - Async custom validations

🔍 Query Performance

- **Indexes**
    - Types: single field, compound, text, hashed
    - Performance considerations
- **Aggregation Pipeline**
    - `$match`, `$group`, `$lookup`, `$unwind`
    - Building reports and analytics

💾 Advanced MongoDB Features

- **Replication**
    - Primary-secondary setup
    - Failover scenarios
- **Sharding**
    - Horizontal partitioning across clusters
    - Use cases and limitations
- **Transactions**
    - Multi-document ACID operations
    - Sessions and rollback scenarios

🛡️ Data Integrity & Optimization

- Ensuring consistency in denormalized data
- Query profiling using MongoDB Compass and `explain()`
- Best practices in schema design and query building

Phase 4: RESTful API Architecture (Week 7–8)

🧰 API Design Patterns

- **RESTful Principles**
    - Statelessness, uniform interface, layered system
- **Resource Naming Conventions**
    - Nouns vs Verbs in URIs
    - Versioning and pagination

🛠️ CRUD Implementation

- Create, Read, Update, Delete using Express and Mongoose
- Soft deletes, timestamps, and audit trails

📚 API Documentation

- Using **Swagger / OpenAPI**
    - Auto-generate docs
    - Interactive API testing via Swagger UI

✅ Request Validation

- Middleware for payload schema checking
- Centralized validation error handling

🔄 Error Handling Strategy

- Consistent error structure (code, message, stack)
- HTTP status codes (200, 400, 401, 404, 500)

🧪 Testing the APIs

- Unit Testing with **Jest**
- API Testing with **Supertest**
- Mocking database calls for isolated tests

Phase 5: Authentication & Authorization (Week 9–10)

🔐 Authentication Basics

- **What is Authentication vs Authorization**
    - Difference and importance
    - Real-world examples and user flows

🪪 JWT (JSON Web Tokens)

- Structure of JWT: Header, Payload, Signature
- Generating & signing tokens with `jsonwebtoken`
- Verifying tokens and protecting routes
- Expiry, refresh tokens, and token rotation

🌐 OAuth2 Integration

- Using third-party auth (Google, Facebook)
- Setting up OAuth2 flows with Passport.js
- Access tokens vs ID tokens
- Handling OAuth callbacks and state

🎭 Role-Based Access Control (RBAC)

- Defining roles (admin, user, guest)
- Role permissions in route guards
- Middleware for role checks

🛡️ Advanced Security

- **2FA (Two-Factor Authentication)**
    - Generating and verifying OTPs
    - Sending OTPs via Email/SMS (using Twilio/Nodemailer)
- **Password Hashing with Bcrypt**
    - Storing hashed passwords securely
    - Salting and comparing passwords
- **Secure Password Reset Flows**

    - Token-based password reset mechanism
    - Emailing reset links securely

💼 Session & Cookie Management

- Storing sessions in Redis
- HTTP-only, secure cookies with session tokens
- Expiry handling and logout mechanisms

Phase 6: Testing & Debugging (Week 11–12)

🧪 Types of Testing

- Unit Testing: Test individual logic components (functions, utils)
- Integration Testing: Test routes and controllers
- End-to-End Testing: Simulate real user flows (Optional)

🧰 Testing Tools

- **Jest**
    - Writing test cases with `describe`, `it`, `expect`
    - Mocking functions and modules
- **Supertest**
    - Testing REST API endpoints
    - Setting up HTTP request-response scenarios

🧙 Mocking & Isolation

- Jest mocks and spies
- Mocking Mongoose and third-party libraries
- Avoiding real DB/API calls in tests

🚦 Test-Driven Development (TDD)

- Writing tests before functionality
- Red-green-refactor workflow

- Test coverage metrics

🐞 Debugging Techniques

- Node Inspector / Chrome DevTools
- VS Code Debugger setup
- Debugging async/await and promises

🔁 Continuous Testing & Automation

- Setting up **GitHub Actions** or **Travis CI**
- Auto-run tests on push/PR
- Coverage reports using tools like Coveralls or Codecov

Phase 7: Performance, Deployment & CI/CD (Week 13–14)

🚀 Performance Optimization

- Identifying bottlenecks with `clinic.js`, Node Profiler
- Caching (Redis) strategies for expensive DB queries
- Async optimizations and load testing (e.g., Artillery)

⚙️ Clustering & Load Balancing

- Using **PM2** for process management
- Scaling Node.js apps using clustering
- Sticky sessions and Redis session sharing

🐳 Docker & Containers

- Writing Dockerfiles for Node.js projects
- Using `.dockerignore`, `docker-compose.yml`
- Building, running, and networking containers

🔄 CI/CD Pipelines

- Using **GitHub Actions** or **Jenkins**

- Build-test-deploy cycles automation
- Linting and formatting automation with ESLint + Prettier

🌐 Deployment

- Platforms: **Render**, **Railway**, **Vercel**, **AWS EC2**, or **DigitalOcean**
- Managing `.env` files with environment variables
- Handling production errors/logs using tools like Sentry

🔌 WebSockets Integration

- Real-time data (chat, notifications)
- Using **Socket.IO** for WebSocket communication

🛡️ Rate Limiting Revisited

- Protecting endpoints from high-traffic abuse
- IP-based, user-based limits with Redis

Phase 8: Containerization & Microservices (Week 15–16)

🐳 Docker Essentials

- **Docker Architecture**: Images, containers, registries, volumes, networks
- **Docker CLI**: `build`, `run`, `exec`, `logs`, `prune`, `rm`, etc.
- Creating Dockerfiles for Node.js projects
- Best practices: Alpine base image, multi-stage builds, non-root users
- Building a production-ready container for a Node + Mongo app

📦 Docker Compose

- `docker-compose.yml` basics
- Running multi-container apps (Node.js + MongoDB + Redis)
- Environment variable management
- Volume mounts and service networks

🧩 Introduction to Microservices

- **Monolith vs Microservices**: Pros, cons, and tradeoffs
- Breaking down a monolith into microservices
- Designing independent services with bounded contexts
- Keeping microservices stateless

🚪 API Gateway

- Introduction to API Gateway pattern
- Tools: Nginx as reverse proxy or using **Kong**
- Centralized authentication and request routing

🔍 Service Discovery

- Why it matters in microservices
- Tools: **Consul**, **Eureka**, DNS-based discovery
- Dynamic registration and deregistration of services

📨 Event-Driven Architecture

- Benefits of asynchronous communication
- **Message brokers** overview: Kafka vs RabbitMQ
- Event producers, consumers, and message queues

🔁 Kafka Basics

- What is Kafka and when to use it
- Kafka topics, partitions, producers, and consumers
- Kafka in Node.js using `kafkajs`
- Handling retries, dead-letter queues, and message durability

🔌 Inter-Service Communication

- RESTful APIs between services
- Kafka or RabbitMQ for pub-sub messaging
- gRPC: Setup, Protobufs, and performance benefits

Phase 9: GraphQL & Kafka Deep Dive (Week 17–18)

🧬 GraphQL Basics

- What is GraphQL and how it differs from REST
- **Schemas**: Defining types, queries, mutations
- Query nesting, arguments, and aliases
- Apollo Playground for testing queries

🧠 Apollo Server with Node.js

- Setting up Apollo Server
- Modular schema design using `makeExecutableSchema`
- Context API for injecting authentication/authorization
- Connecting GraphQL to MongoDB with resolvers

🔁 GraphQL vs REST

- When to use GraphQL (complex querying, flexibility)
- REST advantages (simplicity, caching)
- Performance considerations, rate limiting, tooling

📡 GraphQL Subscriptions (Real-Time Updates)

- Using WebSocket or `graphql-ws`
- Subscribing to real-time events (e.g., chat messages, notifications)
- Publishing from backend services using PubSub

📦 Kafka in Production

- **Kafka Setup**: Zookeeper, brokers, topics
- Consuming Kafka in Node.js using `kafkajs` or `node-rdkafka`
- Efficient batch processing of messages
- Kafka Connect for data sync (MongoDB to Kafka)

🔄 Event Sourcing

- Events as the single source of truth
- Storing events instead of state
- Rebuilding state from event logs
- Benefits: audit trail, replayable workflows

📈 Kafka Streams (Optional Advanced)

- Real-time data processing with Kafka Streams API
- Filtering, joining, and transforming message streams

🔗 GraphQL Federation (Optional Advanced)

- Federating multiple GraphQL services
- Apollo Gateway setup
- Shared ownership of the schema across teams

🎯 Interview Preparation Topics (Week 19–20)

📚 1. Data Structures & Algorithms (DSA)

Mastering DSA is essential not just for coding interviews but also to improve backend logic and optimization.

✅ Core Concepts:

- **Arrays & Strings**: Sliding window, two-pointer techniques
- **Linked Lists**: Reversals, cycles, merging
- **Stacks & Queues**: Balanced parentheses, infix-postfix conversion
- **Trees & Binary Trees**: Inorder/preorder/postorder traversals, BST, AVL Trees
- **Graphs**: DFS, BFS, Dijkstra’s, Union-Find, Topological Sort
- **Heaps**: Min/Max Heap, Priority Queue use cases
- **Recursion & Backtracking**: Combinatorics, permutations, subset generation
- **Dynamic Programming**: Memoization, Tabulation, 0/1 Knapsack, LIS

- **Searching & Sorting**: Binary search, Merge sort, Quick sort, Radix sort

🛠️ Practice Platforms:

- LeetCode
- HackerRank
- Codeforces
- InterviewBit

🧠 2. Design Patterns

Understanding reusable patterns enhances your backend software architecture.

- **Singleton Pattern**: For shared database connections or caching
- **Factory Pattern**: Abstract object creation for services or strategies
- **Observer Pattern**: Event-based systems, WebSockets
- **Strategy Pattern**: For switching between algorithms at runtime
- **Decorator Pattern**: Extend functionality of classes or endpoints
- **Adapter Pattern**: Integration of third-party tools or legacy systems

Use Refactoring.Guru for crisp visual explanations.

🧱 3. SOLID Principles (Object-Oriented Design)

Helps keep backend code **clean, maintainable, and scalable**:

- **S – Single Responsibility**: One class, one job
- **O – Open/Closed**: Extend, don't modify
- **L – Liskov Substitution**: Subtypes replace parent types
- **I – Interface Segregation**: Fine-grained interfaces
- **D – Dependency Inversion**: Rely on abstractions, not concrete classes

---

👨‍💼 4. Behavioral Interview Prep

Use the **STAR Method** to structure your answers:

- **S**ituation: Set the context

- **T**ask: What was your goal?
- **A**ction: What actions did *you* take?
- **R**esult: What was the outcome?

✅ Sample Questions:

- Tell me about a time you failed at something.
- How do you handle team conflict?
- Describe a challenging backend project.
- Explain a time when you optimized a slow API.

🌐 5. Scalability & High Availability

Backend engineers must understand how to scale applications efficiently.

- **Horizontal Scaling**: Load balancing with Nginx, HAProxy
- **Vertical Scaling**: Increasing server resources
- **Replication**: MongoDB replicas for high availability
- **Caching**: Redis, CDN usage
- **Auto-scaling**: Cloud-native tools (AWS EC2 Auto Scaling)

⚡ 6. High-Traffic API Management

Design APIs that handle **millions of requests** efficiently.

- **Rate Limiting**: Prevent abuse (e.g., 100 requests/min)
- **Throttling**: Control over burst traffic
- **Load Testing Tools**: Apache JMeter, Artillery, k6
- **Circuit Breakers**: Fallbacks during service failures (e.g., Netflix Hystrix)

🧱 7. System Design (Big Picture Thinking)

Prepare to build **scalable backend systems** from scratch.

Key Topics:

- **Designing Scalable APIs**: Rate limits, retries, pagination

- **Design a URL Shortener**: Hashing, caching, database sharding
- **Design a Chat App**: WebSockets, message queues, delivery acknowledgment
- **Design an E-Commerce Backend**: Cart system, payment flows, order tracking
- **Design a Notification System**: Kafka + Webhooks or Push Services

Must-read Resources:

- System Design Primer (GitHub)
- ByteByteGo

🧬 8. Microservices Design Patterns

- **Service Registry & Discovery**: Consul, Eureka
- **Circuit Breaker Pattern**: Netflix Hystrix, Resilience4j
- **Event Sourcing**: Kafka + immutable event logs
- **SAGA Pattern**: Distributed transaction management
- **Bulkhead Pattern**: Isolate failures across services
- **Centralized Logging & Monitoring**: ELK stack, Grafana + Prometheus