RoadMap to Become a Full Stack Developer

Fundamentals - 1.5 Months

C & C++

- Number Systems
- Binary, Octal, Hexadecimal
- Binary Operations & Bits
- Logical Gates (AND, OR, NOT, XOR)

Data Structures (Only Basic Ones)

- Arrays
- LinkedLists
- Stacks
- Queues
- HashMaps

Build Following Projects (CLI Based)

- Number Guessing Game
- Telephone Directory with File Handling
- Bank Account Simulation - OOPS
- File Encryption

Web Development [Basic] (3 Months)

HTML & CSS

- Build at least 100 beautiful static landing pages.

Master Git and GitHub

- **Basic JavaScript**
  - Variables
  - Functions
  - Control Flow
  - Data Structures

- Objects & etc...
- DOM APIs
- Event handling and DOM Manipulation
- Browser Features

Build Following Projects

- Digital Clock
- Stopwatch and Timer App
- Todo App with Local Storage
- Weather Forecast based on current users location
- Monkey Typing Game
- Image Slider
- Dice Game
- Simon Game

Basic Extra Information

- What is Internet and How it works?
- What are Protocols and why it is needed?
- TCP vs UDP Protocol
- What is DNS and how DNS works?
- DNS Records - A, AAA, CNAME, TXT, NS, MX
- Client Server Architecture
- 3 Way TCP Handshakes
- UDP & WebRTC
- IP Addresses (Private vs Public)
- Subnet Masks and basics of Networking

Basic Backend Development (3 Months)

- What is a Server?
- Difference between WebServer, API Server, File Server.
- Cloud Server vs On Prem. Servers and Virtualization.
- Introduction to NodeJS
- Building REST APIs in Node
- ExpressJS, Hono or Intent JS Frameworks
- Testing of APIs with POSTMAN like tools

- Headers, Body and HTTP Methods
- DB Integration (Start with Mongodb and Mongoose)
- What is an ORM
- SQL vs No-SQL DBs
- PostgreSQl DB and Learn SQL
- Prisma, Drizzle, Knex as ORM for SQL
- Authentication and Authorization
- JWT vs Sessions

- **Build Projects**
  - Simple CRUD Applications
  - E-Commerce API
  - Task Management API
  - Weather API Wrapper

- **More Concepts**
  - Redis
  - Queue System (BullMQ or SQS)
  - Rate Limiting and Rate Limiting Strategies
  - Async and Batch Processing Jobs
  - DB Indexes and DB Optimisations

Binge watch Case Studies and Blogs of large companies

- **Soft Skills**
  - Be active on Twitter like community and tell about yourself
  - Tell what you are learning and what you are building
  - Ask for more advance topics and suggestions for what I can build next
  - Watch out for problems and try solving and creating solutions
  - Feel more **confident** on what you are doing and learning
  - Connect with Right People

React JS (2 Months)

- Why do we need React?
- React Fundamentals
  - Hooks

  - Components
  - Life Cycle
  - Styling
  - Tailwind CSS
  - State Managements
  - Redux and Zustand
- Routing in React Router DOM

- **Building Projects**
  - Todo Application with Local Storage
  - Weather Applications
  - Amazon like Working Clones (FE Only)
  - YouTube working clone using Google APIs

Full Stack Application (MERN or PERN)
  - Real Time Collaboration Google Doc
  - Multi-Vendor E-Commerce Platform
  - Social Media (X, LinkedIn) Automation Tool
  - Video Conferencing Applications (WebRTC or SFU)
  - Live Streaming Studio (RTMP)
  - Collaborative white boarding tools with ACL and more advance features
  - Zapier Like Automation Apps

Cloud
  - AWS
    - EC2
    - Load Balancers
    - SG, TG
    - Cloud Front
    - VPC and Network Gateways and Internet Gateways
    - SES, SQS, S3
    - CloudFormation Templates and Scripts
    - Private and Public VPC
    - Subnets
    - IAM Roles, Federated Identities

- API Gateways

Become Unstoppable (Advanced Learning)

1. Distributed Systems

- **Microservices Architecture**: Breaking applications into smaller, independently deployable services.
  - Service discovery and load balancing.
  - Communication strategies (REST, gRPC, Message Queues).
  - Saga patterns for distributed transactions.

- **CAP Theorem**: Trade-offs between Consistency, Availability, and Partition Tolerance in distributed systems.

- **Eventual Consistency**: Achieving data consistency over time in distributed databases (e.g., DynamoDB).

- **Sharding and Replication**: Techniques to horizontally scale databases.

- **Consensus Algorithms**: Ensuring distributed nodes agree on a single source of truth (e.g., Raft, Paxos).

2. API Design and Optimization

- **RESTful API Best Practices**: Designing versioned, scalable, and secure APIs.

- **GraphQL**: Crafting flexible and efficient query systems.

- **gRPC**: High-performance, low-latency communication for inter-service communication.

- **API Rate Limiting**: Strategies to control traffic using tools like Redis or third-party services.

- **HATEOAS**: Hypermedia as the Engine of Application State for discoverable APIs.

3. Caching Strategies

- **Cache Invalidation**: Techniques for keeping cache and data in sync (write-through, write-behind, cache-aside).

- **CDN Integration**: Leveraging edge caching for static and dynamic content.

- **Distributed Caches**: Using tools like Redis or Memcached in multi-node environments.

- **Cache Partitioning**: Avoiding bottlenecks with consistent hashing.

4. Asynchronous Processing

- **Task Queues**: Using RabbitMQ, Kafka, or BullMQ for background job processing.

- **Worker Pools**: Managing and scaling background workers efficiently.

- **Batch Processing**: Designing systems for processing large datasets in chunks.

- **Event-Driven Architectures**: Leveraging events for loose coupling and scalability (e.g., Kafka, AWS SNS/SQS).

5. Database Optimization and Management

- **Indexing**: Choosing the right type of index (B-Tree, Hash, Full-text).
- **Database Partitioning**: Vertical vs. horizontal partitioning for scaling.
- **Query Optimization**: Analyzing query plans and reducing query time.
- **Data Modeling**: Designing schemas for relational and NoSQL databases.
- **Distributed Databases**: Understanding database models like Spanner, CockroachDB, or Cassandra.
- **ACID vs. BASE**: Trade-offs between consistency and availability in database systems.

6. Authentication and Authorization

- **OAuth 2.0 and OpenID Connect**: Secure token-based authentication for APIs.
- **JWT (JSON Web Tokens)**: Stateless authentication mechanisms.
- **Session Management**: Balancing session persistence and security.
- **Role-Based Access Control (RBAC)**: Implementing granular permission management.
- **SSO (Single Sign-On)**: Integrating with external identity providers.

7. Observability and Monitoring

- **Logging**: Structured logging with tools like ELK Stack, Graylog.
- **Metrics Collection**: Using Prometheus, Grafana for performance monitoring.
- **Tracing**: Distributed tracing with tools like Jaeger or Zipkin.
- **Alerting**: Setting up alerts based on predefined thresholds.
- **Chaos Engineering**: Simulating failures to test system resilience (e.g., Netflix's Chaos Monkey).

8. Fault Tolerance and Resilience

- **Circuit Breakers**: Prevent cascading failures in microservices (e.g., Resilience4j).
- **Retries and Exponential Backoff**: Handling transient failures effectively.
- **Graceful Degradation**: Designing systems to offer reduced functionality during failures.
- **Dead Letter Queues**: Managing failed messages in message queues.

9. Security

- **Data Encryption**: Encrypting data at rest and in transit.
- **Rate Limiting and Throttling**: Preventing abuse and DoS attacks.
- **CSRF and CORS**: Protecting APIs against cross-site vulnerabilities.
- **SQL Injection and XSS Prevention**: Secure input validation and sanitization.
- **Secrets Management**: Using tools like HashiCorp Vault or AWS Secrets Manager.

10. Advanced Networking Concepts

- **Load Balancing**: Using reverse proxies like NGINX, HAProxy, or AWS ALB.
- **Reverse Proxy**: Optimizing performance and securing backend services.
- **WebSockets**: Real-time bidirectional communication.
- **TCP vs. UDP**: Choosing protocols based on use cases (e.g., WebRTC).
- **Zero Downtime Deployments**: Techniques like blue-green deployment or canary releases

11. Advanced Architectures

- **Service Mesh**: Managing microservices traffic with tools like Istio or Linkerd.
- **Serverless Architecture**: Using AWS Lambda, Google Cloud Functions for scaling on demand.
- **Edge Computing**: Moving computation closer to the user for latency-sensitive applications.
- **Hybrid Cloud**: Designing systems that work across on-premises and cloud environments

12. Data Pipelines and ETL

- **Stream Processing**: Real-time data processing with Apache Kafka or Apache Flink.
- **Batch Processing**: Large-scale ETL jobs with Apache Airflow or AWS Glue.
- **Data Warehousing**: Building scalable analytics pipelines with Snowflake or BigQuery.