Full-Stack Developer Roadmap

A step-by-step guide with mini-assignments & quiz questions

Awesome — you've picked a great path. Below is a practical, ordered roadmap you can follow from zero → full-stack, with small learning chunks, hands-on mini-assignments after each chunk, and short quiz questions to test yourself. I've included important prerequisites (networking, OS, Git, data structures, security) and a capstone project tuned to your interest (school management system). Work at your own pace and build the projects — that's how you learn.

---

1) Foundations (essential before diving full-stack)

What to learn

- Computer basics: OS, files, folders, installing software.
- Command line / terminal (bash / PowerShell basics).
- Text editors / IDEs (VS Code) and extensions.
- Git & GitHub: commits, branches, merging, PRs.
- Basic debugging & reading error messages.
- Basic networking terms: IP, DNS, HTTP/HTTPS, ports, client/server.
- Basic HTTP: requests, responses, status codes, headers.

Mini-assignments

- Install VS Code, Git; create a GitHub account; make a repo called `learning-log`.
- Create a `README.md` explaining your goals; commit + push.
- From terminal: make a folder, create a file, commit to Git, push.

Quiz questions

- What command initializes a git repo?
- What HTTP status code means “not found”?
- Name two differences between HTTP and HTTPS.

---

2) Web basics — HTML & CSS (frontend foundation)

What to learn

- HTML structure: elements, semantic tags (header, nav, article, form).
- CSS fundamentals: selectors, box model, layout.
- Responsive design: flexbox, CSS Grid, media queries.
- Accessibility basics (alt, labels, keyboard nav).
- Basic debugging with DevTools.

Mini-assignments

- Build a responsive personal profile page (header, about, contact form). Make it mobile friendly.
- Create a simple CSS grid gallery of 6 images with captions.

Quiz questions

- What does the CSS box model include (4 parts)?
- When to use `<section>` vs `<article>`?
- How do you make an element horizontally center with flexbox?

---

3) JavaScript — language fundamentals

What to learn

- Variables, types, operators, control flow.
- Functions, arrow functions, scope, closures.
- Arrays and objects.
- DOM manipulation & events.
- Asynchronous JS: callbacks, Promises, async/await.
- Dev tooling: linting (ESLint), formatting (Prettier), basic debugging.

Mini-assignments

- Build a TODO app (add, delete, mark done) that persists to `localStorage`.
- Make a small interactive widget: live search/filter a list.

Quiz questions

- What's the difference between `==` and `===`?
- How do you fetch data with `async/await`? (show a short snippet)
- What is a closure?

---

4) Frontend tooling & modern frameworks

What to learn

- Module bundlers / dev tools: Vite (or Webpack), npm/yarn.
- Component-based frameworks: React (recommended) — components, props, state, hooks.
- Routing (React Router).
- State management basics: Context API, then Redux or Zustand.

- Styling strategies: CSS Modules, Tailwind, styled-components.
- Testing basics: unit tests with Jest + React Testing Library.

Mini-assignments

- Rebuild TODO app in React (componentized).
- Create a small multi-page app (list view + detail view) using React Router.
- Add a simple global state (e.g., dark mode) via Context.

Quiz questions

- What does `useEffect` do and when does it run?
- How do you pass data from parent → child in React?
- Why choose Vite over older bundlers for development?

---

5) Backend fundamentals — APIs & Node.js

What to learn

- Node.js basics (runtime, event loop).
- Express.js: routing, middleware, error handling.
- REST API design: endpoints, verbs (GET/POST/PUT/DELETE), status codes.
- Authentication basics: sessions vs tokens (JWT).
- File uploads and serving static files.
- API testing with tools (Postman / curl) and automated tests.

Mini-assignments

- Create a REST API for notes with Express + in-memory store (CRUD).
- Add JWT authentication and protect a route.
- Write tests for at least one endpoint.

Quiz questions

- What is middleware in Express? Give an example.
- When is POST vs PUT appropriate?
- What’ s a JWT and where is it stored on the client?

---

6) Databases (SQL + NoSQL)

What to learn

- Relational DBs: PostgreSQL basics — tables, relations, joins.
- Query building and schema design (ER diagrams).
- ORMs: Prisma or TypeORM (how they map objects to DB).
- NoSQL basics: when to use MongoDB or key-value stores.
- Migrations, indexing, basic optimization.

Mini-assignments

- Model a simple school schema: students, teachers, classes, enrollments in Postgres.
- Build API endpoints to create/read students & enroll them in classes.
- Add a migration system and seed data.

Quiz questions

- What is a foreign key used for?
- When would you pick NoSQL over SQL?
- What is an index and why use it?

---

7) Authentication, Authorization & Security

What to learn

- Secure password storage (bcrypt), JWT best practices.
- Role-based access control (admin/teacher/student).
- OWASP Top 10: XSS, CSRF, injection — prevention patterns.
- HTTPS, CORS basics, secure headers.
- Rate limiting, input validation and sanitation.

Mini-assignments

- Implement role-based routes (admin only) in your API.
- Add input validation to endpoints and demonstrate protection against SQL injection (use parameterized queries/ORM).
- Configure Helmet (Express) and basic CORS settings.

Quiz questions

- Why never store passwords in plain text?
- What is CSRF and how to prevent it?
- Where should you validate user input — client or server?

---

8) Real-time & background processing

What to learn

- WebSockets (Socket.IO) for real-time notifications/chats.
- Background jobs & queues: Bull / Redis (processing reports, emails).
- Pub/Sub concepts and when to use them.

Mini-assignments

- Add a simple real-time notifications channel: when a new message is posted, clients see it instantly.
- Create a background job to generate a CSV export of student data.

Quiz questions

- When to choose WebSockets vs polling?
- Why use a job queue for sending emails?

---

9) Testing, quality & observability

What to learn

- Unit tests, integration tests, end-to-end tests (Cypress or Playwright).
- Logging (structured logs), monitoring (Prometheus, Sentry), metrics.
- Readiness/liveness probes for services.

Mini-assignments

- Add unit tests for core business logic (e.g., enrollment rules).
- Create one E2E test for student sign-up + enroll flow.

Quiz questions

- Difference between unit and integration tests?
- What is observability vs monitoring?

---

10) DevOps, deployment & cloud basics

What to learn

- Containers: Docker (Dockerfile, images, compose).
- CI/CD basics: GitHub Actions / GitLab CI — run tests and deploy.
- Cloud fundamentals: compute (VMs), serverless (Vercel, Netlify, AWS Lambda), managed DBs.
- Simple infra as code (Terraform basics optional).
- Static hosting for frontend (Vercel/Netlify) and deploying backend (Heroku, DigitalOcean App Platform, or cloud provider).
- SSL, domain setup, environment variables, secrets management.

Mini-assignments

- Dockerize your backend and database with `docker-compose` for local dev.
- Set up a GitHub Action that runs tests on PR and deploys the frontend to Vercel.
- Deploy backend to a cloud service and connect to the managed Postgres.

Quiz questions

- What is the purpose of a Dockerfile?
- What’ s the difference between serverless and a VM?

---

11) Performance, caching & scaling

What to learn

- Caching strategies: Redis, browser caching, CDN (Cloudflare).
- Database scaling basics: replication, read replicas, connection pooling.
- Horizontal vs vertical scaling.
- Load testing basics.

Mini-assignments

- Add Redis caching for a frequently requested API (e.g., class list).
- Put static assets behind a CDN and measure load improvements (use Lighthouse).

Quiz questions

- When to use Redis caching?
- What is connection pooling?

---

12) System design & architecture (high-level)

What to learn

- Monolith vs microservices tradeoffs.
- Designing for multi-tenancy (important for a system that serves many schools).
- Data partitioning, backups, and disaster recovery.
- Authentication flows and data privacy (GDPR basics if relevant).

Mini-assignments

- Draw a system diagram for a multi-tenant school system (frontend, API, DB strategy, auth, storage).
- Write a short plan for how you will backup and restore the DB.

Quiz questions

- What is multi-tenancy and one way to implement it?
- Why are backups critical and how often should you test restores?

---

13) Computer science fundamentals (important for interviews & robustness)

What to learn

- Algorithms & data structures: arrays, linked lists, trees, hash maps, sorting.
- Big O notation, complexity analysis.
- Basic system design patterns.

Mini-assignments

- Implement and test common algorithms (binary search, quicksort) in your language of choice.
- Solve 10 medium-level algorithm problems on a practice site.

Quiz questions

- What’ s the time complexity of binary search?
- When to use a hash map vs an array?

---

14) Soft skills & professional practices

What to learn

- Writing clear README, documentation, API docs (OpenAPI/Swagger).
- Code reviews, teamwork, agile basics.
- Communication with stakeholders and product thinking.

Mini-assignments

- Write API docs (basic OpenAPI spec) for one service.
- Do a mock code review: create PR with a small change and document review comments.

Capstone project — Multi-Tenant School Management System

Goal: build an MVP that demonstrates full-stack skills.

Core features

- Multi-tenant sign-up (schools separated by tenant).
- Roles: admin, teacher, student, parent.
- CRUD for students, classes, enrollments.
- File upload for student documents.
- Reports (CSV/PDF exports) and dashboard with basic charts.
- Real-time notifications (e.g., announcements).
- Auth + role-based access control.
- Deployment: frontend hosted (Vercel/Netlify), backend on a cloud service, Postgres managed DB, Redis for cache/queues.

Deliverables (what to submit)

- Repo with clearly segmented frontend/backend folders.
- Docker compose for local dev.
- README with setup instructions and architecture diagram.
- Deployed app links (frontend + API), and a short video demo (2–5 min).

Rubric / checklist (self-review)

- ✅ Authentication works and role restrictions enforced.
- ✅ CRUD operations operate and data is persisted.
- ✅ Tests: unit + at least one E2E.
- ✅ CI runs tests on PRs.
- ✅ App deployed and reachable; DB backups demonstrated.

Mini-assignments leading to capstone

- Each previous mini-assignment feeds into capstone pieces: auth + DB + APIs + frontend pages + deployment.
- Final: implement a “school admin” page to invite teachers and create classes; then add multi-tenant isolation.

Learning rhythm & recommendations

- Build first, read second. Small projects beat tutorials.
- Keep a learning log (use your `learning-log` repo).
- Pair program or get feedback — code reviews accelerate growth.
- Focus on fundamentals (HTTP, JS, SQL) rather than only frameworks.
- Learn to read docs (MDN, Postgres docs, framework docs).

Quick suggested 6-month sequence (example pacing)

- **Month 1:** Foundations + HTML/CSS + JS basics → small projects.
- **Month 2:** Deep JS + React + Vite + build 2 React apps.
- **Month 3:** Node + Express + Postgres + build REST API.
- **Month 4:** Auth, real-time, testing, Docker.
- **Month 5:** CI/CD + deploy, caching, monitoring.
- **Month 6:** Capstone + polish + interview prep.

(Adjust pace to your time and learning speed.)

If you want, I can:

- Turn this into a printable checklist you can tick off weekly.
- Generate a week-by-week study schedule tailored to how many hours per week you can commit.
- Create specific starter code templates (React + Express + Postgres) you can clone to begin.

Which of those would you like next?