BACKEND ROADMAP

120 concepts to become a backend developer

8 areas, 15 concepts each. Every concept explained in one line, the order to study each area, and two practice questions to test yourself.

How to use it: one area at a time, tick each concept only when you can explain it out loud in under a minute, then answer the two practice questions without notes.

@chalkesahil · chalkesahil.ai

AREA 01 / 08

Programming language

Study order: Pick one of Java, Node.js, Python or Go and stick to it for all eight areas.

01	**Variables and types**	How your language stores numbers, strings and objects.
02	**Functions**	Small reusable pieces of logic.
03	**Classes and objects**	Group data with the code that uses it.
04	**Collections**	Lists, maps and sets, and their costs.
05	**Error handling**	Catch failures and return useful messages.
06	**Modules and packages**	Split code across files and reuse libraries.
07	**Package manager**	npm, Maven, pip or Go modules.
08	**File and JSON handling**	Read, write and parse data.
09	**Async basics**	Do other work while waiting on the network.
10	**Concurrency basics**	Threads, event loop or goroutines.
11	**Logging**	Know what happened when something broke.
12	**Config and env variables**	Different settings for dev and prod.
13	**Unit tests**	Check one function in isolation.
14	**Debugging**	Breakpoints and reading stack traces.
15	**Big O basics**	Why some code slows down as data grows.

Practice

1. Read a JSON file of orders and print total revenue per city.
2. Your function works for 10 items but hangs for a million. Why?

AREA 02 / 08

HTTP and REST APIs

Study order: Build one API by hand before using any generator.

01	**How HTTP works**	Request, response, headers and body.
02	**HTTP methods**	GET, POST, PUT, PATCH and DELETE.
03	**Status codes**	200, 201, 400, 401, 404, 500 and what each means.
04	**REST design**	Resources as nouns, actions as methods.
05	**Routing**	Map a URL to the right handler.
06	**Request validation**	Reject bad input early.
07	**Pagination**	Return big lists page by page.
08	**Filtering and sorting**	Query parameters done right.
09	**Idempotency**	Retrying a request must not charge twice.
10	**API versioning**	Change the API without breaking apps.
11	**Middleware**	Code that runs before every route.
12	**CORS**	Let a browser app call your API.
13	**File uploads**	Accept images and documents safely.
14	**API docs**	OpenAPI so others can use your API.
15	**Postman**	Test every route by hand.

Practice

1. Design the API for a notes app with all CRUD routes.
2. A client retries a payment request. How do you avoid charging twice?

AREA 03 / 08

SQL databases

Study order: Learn PostgreSQL or MySQL properly; every backend job tests it.

01	**Tables and keys**	Primary and foreign keys.
02	**SELECT, WHERE, ORDER BY**	Read the rows you need.
03	**Joins**	Combine tables.
04	**GROUP BY**	Totals per group.
05	**Indexes**	Make lookups fast.
06	**Query plans**	EXPLAIN shows where the time goes.
07	**Normalisation**	Avoid duplicate data.
08	**Transactions**	All-or-nothing writes.
09	**ACID**	What makes a database reliable.
10	**Isolation levels**	What other transactions can see.
11	**Locks**	Two writes to the same row.
12	**ORMs**	Query from code with Prisma, Hibernate or SQLAlchemy.
13	**N+1 queries**	The classic slow-API mistake.
14	**Migrations**	Change the schema safely.
15	**Connection pooling**	Reuse database connections.

Practice

1. Design tables for a food delivery app.
2. An API page makes 101 queries. How do you fix it?

AREA 04 / 08

NoSQL and caching

Study order: Know when a database is the slow part, and what to do about it.

01	**SQL vs NoSQL**	When to pick each.
02	**Document databases**	MongoDB and flexible records.
03	**Key-value stores**	Fast lookups by one key.
04	**Redis basics**	In-memory data for speed.
05	**Cache-aside pattern**	Check the cache, then the database.
06	**TTL and expiry**	How long cached data lives.
07	**Cache invalidation**	Keep the cache from going stale.
08	**Cache stampede**	Thousands of misses at once.
09	**Session storage**	Logged-in users across servers.
10	**Rate limiting with Redis**	Count requests per user.
11	**Leaderboards**	Sorted sets for rankings.
12	**Replication**	Copies of data for reads and safety.
13	**Sharding**	Split data across machines.
14	**CAP theorem**	Consistency vs availability when networks fail.
15	**Choosing storage**	Match the store to the access pattern.

Practice

1. Your product page hits the database on every view. Add caching.
2. A price changed but users still see the old one. Why?

AREA 05 / 08

Auth and security

Study order: Build login yourself once so you understand every step.

01	**Password hashing**	bcrypt; never store plain passwords.
02	**Sessions**	Server remembers who is logged in.
03	**JWT**	Signed tokens and what is inside them.
04	**Refresh tokens**	Stay logged in without long-lived tokens.
05	**OAuth login**	Sign in with Google.
06	**Roles and permissions**	Who can do what.
07	**Cookies**	httpOnly, secure and SameSite flags.
08	**HTTPS**	Encrypt traffic between client and server.
09	**SQL injection**	Never build queries from raw input.
10	**Input validation**	Trust nothing from the client.
11	**Rate limiting**	Stop password guessing and abuse.
12	**Secrets management**	Keys out of code and git.
13	**OWASP Top 10**	The most common web risks.
14	**Audit logs**	Record who changed what.
15	**Least privilege**	Give each service only the access it needs.

Practice

1. Walk through what happens when a user logs in with a JWT.
2. How do you stop someone guessing passwords on your login API?

AREA 06 / 08

Queues and async

Study order: Move slow work out of the request so the API stays fast.

01	**Sync vs async work**	What the user waits for and what they don't.
02	**Background jobs**	Send emails and resize images later.
03	**Message queues**	RabbitMQ or SQS between services.
04	**Kafka basics**	Event streams many services can read.
05	**Producers and consumers**	Write messages, read messages.
06	**Retries**	Try again on temporary failures.
07	**Dead letter queues**	Park messages that keep failing.
08	**At-least-once delivery**	Why handlers must handle duplicates.
09	**Idempotent consumers**	Processing twice gives the same result.
10	**Cron jobs**	Scheduled tasks.
11	**Webhooks**	Other systems calling your API on events.
12	**WebSockets**	Push live updates to clients.
13	**Event-driven design**	Services react to events, not calls.
14	**Backpressure**	When consumers can't keep up.
15	**Outbox pattern**	Save to the database and send an event reliably.

Practice

1. An order API sends an email and takes 4 seconds. Make it fast.
2. A consumer processed the same payment event twice. How do you prevent it?

AREA 07 / 08

Docker and deployment

Study order: Ship every project to a live URL; recruiters click links.

01	**Linux basics**	Move around a server terminal.
02	**Git workflow**	Branches and pull requests.
03	**Docker images**	Package the app with everything it needs.
04	**Dockerfile**	Write the build steps for your image.
05	**Docker Compose**	Run the app, database and Redis together.
06	**Environment config**	Same image, different settings.
07	**CI pipelines**	Run tests on every push.
08	**Cloud basics**	Servers, storage and managed databases.
09	**Deploying an API**	Render, Railway or a cloud VM.
10	**Reverse proxy**	Nginx in front of your app.
11	**Load balancers**	Spread traffic across servers.
12	**Health checks**	Know when a server is broken.
13	**Logs and metrics**	See errors and slow routes.
14	**Zero-downtime deploys**	Release without dropping requests.
15	**Kubernetes basics**	Run and scale containers.

Practice

1. Dockerise an API with a Postgres database using Compose.
2. Your app works locally but crashes in production. Where do you look?

AREA 08 / 08

System design and interviews

Study order: Product companies test design and DSA; practise both out loud.

01	**Scaling basics**	Vertical vs horizontal.
02	**Stateless servers**	Any server can handle any request.
03	**Database scaling**	Read replicas and sharding.
04	**CDN**	Serve static files close to users.
05	**Consistent hashing**	Spread keys across servers evenly.
06	**Rate limiter design**	A classic interview question.
07	**URL shortener design**	The most common first design round.
08	**Microservices vs monolith**	When to split and when not to.
09	**API gateway**	One front door for many services.
10	**DSA: arrays and hashing**	The most asked category.
11	**DSA: trees and graphs**	Common in product company rounds.
12	**Low-level design**	Parking lot, BookMyShow, splitwise.
13	**Explaining your project**	Problem, design choices and trade-offs.
14	**Resume**	Projects with numbers, not tool lists.
15	**AI tools**	Copilot writes boilerplate; you review every line.

Practice

1. Design a URL shortener that handles 10,000 requests a second.
2. Explain your best backend project in two minutes.