Guide

TypeScript Crash Guide

1. Why TypeScript?

- Adds **types** to JavaScript for safer and more scalable code.
- Excellent for backend APIs, especially with frameworks like Hono and libraries like Drizzle, Zod.

2. Essentials

```
// Variable typing
let name: string = "SMSX";
let age: number = 22;
let isActive: boolean = true;

// Arrays & Tuples
let scores: number[] = [99, 88];
let response: [string, number] = ["OK", 200];

// Functions
function greet(user: string): string {
  return `Hello, ${user}`;
}

// Interfaces (object contracts)
interface Notification {
  id: string;
  content: string;
  sentAt: Date;
}

// Types (like interfaces but more flexible)
type Status = "sent" | "failed";
```

```
// Generics
function wrapInArray<T>(value: T): T[] {
  return [value];
}
```

3. Advanced Patterns

```
// Union & Intersection
type ApiResponse = { data: string } | { error: string };
type FullNotification = Notification & { status: Status };

// Optional & Default Parameters
function log(message: string, level: "info" | "warn" = "info") {
  console.log(`[${level}]: ${message}`);
}
```

4. Working with Zod & Drizzle

```
import { z } from "zod";

const TaskSchema = z.object({
  id: z.number(),
  title: z.string(),
});

type Task = z.infer<typeof TaskSchema>;
```

🟨 JavaScript → TypeScript Quick Transition Sheet

JS	TypeScript
`const a = 5`	`const a: number = 5`
`function f(x)`	`function f(x: number): number {}`

Dynamic object	`interface Obj { key: string }`
Array of strings	`string[]` or `Array<string>`
Return types	`(): string`
Optional properties	`interface X { a?: string }`
Default parameters	`function f(x = 1) {}` → same in TS
Object shape enforcement	Use `interface` or `type`
Working with modules	Use `import` / `export` with `.ts`

⚙️ Hono Framework Basics

Hono is a **lightweight, blazing-fast web framework** for TypeScript/JavaScript, great for microservices and edge apps.

1. Creating a Route

```
import { Hono } from "hono";

const app = new Hono();

app.get("/hello", (c) => {
  return c.text("Hello SMSX");
});

export default app;
```

2. Handling Requests & Responses

```
app.post("/submit", async (c) => {
  const body = await c.req.json();
  return c.json({ received: body });
});
```

3. Middleware Example

```
app.use("*", async (c, next) ⇒ {
  console.log("Incoming request");
  await next();
});
```

4. Modular Routes

```
// routes/tasks.ts
const taskRouter = new Hono();
taskRouter.get("/", getAllTasks);

// main app
app.route("/tasks", taskRouter);
```

5. Zod + Hono Integration

```
import { z } from "zod";
import { zValidator } from "@hono/zod-validator";

const schema = z.object({
  name: z.string(),
});

app.post(
  "/user",
  zValidator("json", schema),
  (c) ⇒ {
    const data = c.req.valid("json");
    return c.json({ message: `Hello ${data.name}` });
  }
);
```

6. Error Handling

```
app.onError((err, c) => {
  return c.json({ error: err.message }, 500);
});
```

🧭 Suggested Learning Path

1. **TS Fundamentals**: Use TypeScript in small features. Practice `type`, `interface`, `z.infer`.
2. **Hono Basics**: Build a test route, connect it with a service, add OpenAPI.
3. **Zod Integration**: Add schema validation to a route.
4. **Drizzle ORM**: Try creating a `Task` table and querying it.