TRAINING CURRICULUM

Go Training for Development Teams

Capstone Project : GoCheck — URL Health Monitor

GENERAL INFORMATION

Total Duration	5 half-days (15 hours)
Format	In-person or remote
Target Audience	Backend/fullstack developers with programming experience
Prerequisites	Command line, Git, compiled language (Java, C#, TypeScript)
Class Size	Up to 12 participants

LEARNING OBJECTIVES

Upon completion of this training, participants will be able to:

- Master the syntax and fundamental concepts of the Go language
- Apply idiomatic conventions and best practices of the Go community
- Write robust unit and integration tests
- Implement concurrency with goroutines and channels effectively
- Develop production-ready Go applications according to industry standards

ACQUIRED SKILLS

Technical Skills

- Compiling and running Go programs
- Idiomatic error handling
- Manipulating data structures (slices, maps, structs)
- Writing tests with the testing package and third-party tools
- Using concurrency (goroutines, channels, context)
- Creating structured REST APIs following clean architecture principles

Methodological Skills

- Problem analysis and solving with the Go approach
- Application debugging and profiling
- Go project organization and structuring
- Code review and adherence to community standards

TEACHING APPROACH

The training relies on a **"practice-first"**: participants face problems before receiving theoretical input. This format, designed for experienced developers, creates a memorable contrast between habits acquired in other languages and the idiomatic Go approach.

Typical Half-Day Format (3h)

Phase	Duration	Description
Briefing	15 min	Presentation of objectives, minimum building blocks to get started
Practice	45 min	Participants code, trainer circulates and unblocks
Live Review	45 min	Building the solution together, discussing alternatives
Theory & Deep Dive	1h15	Concepts, patterns, and best practices anchored in the produced code

Exception: Day 1 follows a more traditional format (short theory → immediate practice) as participants are discovering the language.

CAPSTONE PROJECT: GOCHECK

GoCheck is a **URL Health Monitor** built from scratch throughout the training. The service allows registering URLs to monitor via a REST API, periodically checking them (HTTP status, response time), and exposing the results. The business domain is intentionally simple to leave room for learning the language.

Project Progression

Day	What We Build	Key Go Concepts
Day 1	CLI that checks URLs	Structs, slices, maps, errors, net/http client
Day 2	Structured REST API (CRUD targets)	Packages, interfaces, net/http server, clean architecture
Day 3	Check endpoints + test suite	testing, httptest, table-driven tests, mocks
Day 4	Concurrent scheduling + worker pool	Goroutines, channels, context, sync, graceful shutdown
Day 5	Customized deep dive	Based on chosen module

DETAILED PROGRAM

📘 DAY 1 – INTRODUCTION TO GO

Duration : **3h00** | Format: alternating short theory / immediate practice

Operational Objectives

- Install and configure a Go development environment
- Write and run your first Go program
- Manipulate fundamental data types
- Understand error handling in Go

Block 1 — Getting Started (45 min)

Theory (20 min) : History and philosophy of Go. Application domains (APIs, microservices, CLI tools). Go SDK installation, editor setup (VSCode, GoLand). First Hello World program, basic commands : go run, go build, go mod init.

Practice (25 min) : Participants install their environment, create their gocheck module, write a program that compiles and runs.

Block 2 — Syntax and Data Structures (1h15)

Theory (30 min) : Variables and declarations (var, :=, const). Basic types (int, string, bool, float). Control structures (if, for, switch). Functions, multiple return values, pointers. Structs, slices, maps, methods on types.

Practice (45 min) : Model the Target and CheckResult structs. Create a MemoryStore with a map. Write a program that creates targets, stores them, iterates over them, and displays them.

Block 3 — The First Checker (1h)

Theory (15 min) : The net/http package on the client side. Error handling (if err != nil). The defer keyword. Time measurement with time.Now() and time.Since().

Practice (45 min) : Write the Check(url string) CheckResult function. Wire up main.go with hardcoded URLs (including one with an error, one slow via httpstat.us). Display a results table in the terminal.

Teaching Methods

- Theoretical presentation with live coding demonstrations (50%)
- Guided practical exercises (50%)

GoCheck Deliverable

Functional CLI program that checks a list of URLs and displays results (HTTP status, response time, errors) in the terminal.

✍ DAY 2 – REST API AND BEST PRACTICES

Duration : **3h00** | Format: practice-first

Operational Objectives

- Apply Go idiomatic conventions
- Organize a Go project following standards (cmd/, internal/)
- Define and implement interfaces
- Expose a REST API with net/http

Practical Phase (1h15)

Briefing (15 min) : Presentation of the target project structure and HTTP handler syntax with http.NewServeMux (Go 1.22+). Instructions :

- Reorganize Day 1 code into a cmd/ + internal/ structure (domain, checker, storage, handler)
- Extract a TargetRepository interface (Add, Get, List, Delete)
- Implement this interface in the existing MemoryStore
- Create 4 REST endpoints: POST /targets, GET /targets, GET /targets/{id}, DELETE /targets/{id}
- Test with curl

Practice (1h) : Participants work. The trainer circulates to unblock.

Review + Theory Phase (1h45)

Live review (45 min) : Building the solution together. Concepts are introduced during the review :

- Packages, visibility (uppercase/lowercase), role of internal/
- Implicit interfaces, "accept interfaces, return structs" principle
- HTTP handlers, JSON encoding, appropriate status codes
- Error handling with error wrapping (fmt.Errorf + %w)

Theoretical deep dive (30 min) : Naming conventions and Go culture (simplicity, readability). Quality tools : go fmt, go vet, golangci-lint — live demonstration on the day's code.

Refactoring demonstration (30 min) : Live application of 2–3 best practices on the reviewed code. Functional options pattern for HTTP server configuration (port, timeouts). Composition vs inheritance.

Teaching Methods

- Self-directed practice (40%)
- Interactive live review (30%)
- Tool demonstrations and refactoring (30%)

GoCheck Deliverable

Functional REST API with CRUD targets, clean package-based architecture, code validated by golangci-lint.

✅ DAY 3 – TESTING GO CODE

Duration : **3h00** | Format: practice-first

Operational Objectives

- Write effective unit tests with the testing package
- Implement HTTP integration tests
- Use mocks and stubs to isolate dependencies
- Measure and improve test coverage

Practical Phase (1h30)

Briefing (15 min) : Presentation of tools (testing, httptest, table-driven tests, subtests) with a minimal example of each. Instructions :

- Add 2 endpoints to the API : POST /targets/{id}/check (immediate check) and GET /targets/{id}/results (history)
- Extract a ResultRepository interface (Save, GetByTarget)
- Write unit tests for the checker with httptest.NewServer (simulate 200, 500, timeout, invalid URL) in table-driven test format
- Create a manual TargetRepository mock to test handlers in isolation
- Write an integration test: complete scenario create target → check → retrieve results → delete

Practice (1h15) : Participants implement the production code and tests. Longest block due to double work (functional + tests).

Review + Theory Phase (1h30)

Live review (45 min) : Building the solution test by test :

- Table-driven tests: idiomatic structure, case naming, test case slices
- Manual mock: struct with configurable fields, demonstrate it's simple and sufficient
- Integration test: httptest.NewServer with the real router, helpers t.Helper() et t.Cleanup()
- Subtests: t.Run() for readable reports

Supplementary theory (30 min) : When to mock vs when to integration test. Presentation of gomock and testify/mock for comparison with the manual approach. Build tags (//go:build integration) to separate tests. Coverage : go test -cover, go tool cover -html.

Quick exercise (15 min) : Run coverage on their project, identify an uncovered path, write the missing test.

Teaching Methods

- Guided practical exercises (50%)
- Live coding: writing tests in real-time (30%)
- Existing test review (20%)

GoCheck Deliverable

Complete test suite (unit + integration), coverage above 70%, all tests pass with go test ./...

🔄 DAY 4 – CONCURRENCY IN GO

Duration : **3h00** | Format: practice-first

Operational Objectives

- Understand Go's concurrency model (CSP)
- Implement a worker pool with goroutines and channels
- Manage the lifecycle with context.Context
- Ensure graceful application shutdown

Practical Phase (1h15)

Briefing (20 min) : Focused presentation on the necessary building blocks : goroutines, channels (buffered/unbuffered), select, sync.WaitGroup, context.Context. Presentation of the target flow :

Scheduler → chan Target → [Worker Pool] → chan CheckResult → Collector

Instructions :

- Scheduler: component that sends active targets into a channel at regular intervals, stops via context
- Worker pool: N workers (configurable) that read from the channel, execute the check with context.WithTimeout, send the result
- Collector: reads results and persists to the store
- Main: wire everything, listen for SIGINT/SIGTERM with signal.NotifyContext, graceful shutdown

Practice (55 min) : Participants implement. Most challenging part — the trainer is in high demand.

Review + Theory Phase (1h45)

Live review (1h) : Step-by-step construction, taking the time to explain pitfalls at each stage :

- Scheduler: time.NewTicker + select + ctx.Done(). Show goroutine leak if ctx.Done() is forgotten
- Worker pool: for range ch pattern, why close(ch) is necessary, WaitGroup
- Collector: cascading shutdown (scheduler → workers → collector)
- Graceful shutdown : signal.NotifyContext, shutdown order, server.Shutdown(ctx)

Supplementary theory (30 min) : Concurrency vs parallelism, the Go scheduler. Naming the patterns used (worker pool, fan-out/fan-in, pipeline). Data races: demonstration with go test -race. Presentation of errgroup as an alternative to WaitGroup.

Demonstration (15 min) : Launch GoCheck with 10–15 targets, observe worker logs in parallel, press Ctrl+C and see the clean shutdown.

Teaching Methods

- Progressive exercises (40%)
- Pattern demonstrations and debugging (30%)
- Live concurrent code debugging (30%)

GoCheck Deliverable

Complete application: REST API + concurrent scheduling + worker pool + graceful shutdown. go test -race detects no data races.

🧠 DAY 5 – CUSTOMIZED DEEP DIVE

Duration : **3h00** | Format: practice-first

This half-day is custom-built based on team needs. The GoCheck project serves as the foundation for the chosen module. Here are the available modules.

MODULE A – OPTIMIZATION & PERFORMANCE

Objectifs: Profile a Go application, identify bottlenecks, comprendre le Garbage Collector et optimize memory and CPU.

Content: pprof (CPU and memory profiling), advanced benchmarking, memory allocation analysis, hot path optimization, http.Client pooling and buffer reuse.

GoCheck Application: Benchmark the checker and store, profile the worker pool under load (100+ targets), optimize allocations.

MODULE B – ADVANCED TESTING

Objectives: Practice TDD in Go, implement end-to-end tests, automate test generation.

Content: TDD workflow with Go, BDD with frameworks (goconvey, ginkgo), property-based testing, mutation testing, non-regression strategies.

Application GoCheck: Add a feature using TDD (alerts when a check fails N times), property-based tests on the scheduler, end-to-end tests of the complete application.

MODULE C – MICROSERVICES ARCHITECTURE

Objectifs: Structure a microservices project, implement inter-service communication, manage observability.

Content: Hexagonal architecture in Go, gRPC (protobuf), observability (slog, Prometheus, OpenTelemetry), resilience patterns (retry, circuit breaker).

Application GoCheck: Add structured logging with slog, expose Prometheus metrics (check count, latency, error rate), implement a circuit breaker on failing targets.

MODULE D – SECURITY AND INTEROPERABILITY

Objectifs: Secure Go applications, manage authentication and authorization.

Content: HTTPS and TLS certificates, JWT and OAuth2, input validation and sanitization, common vulnerabilities and protections.

Application GoCheck: Add JWT authentication to the API, validate and sanitize incoming URLs (SSRF protection), switch the API to HTTPS.

Half-Day Format

Practical phase (1h–1h15): Briefing on the chosen module (20–30 min) with concrete examples on GoCheck, then practice (45 min).

Review + closing phase (1h45–2h): Module review (45 min), GoCheck project retrospective from Day 1 to Day 5 (30 min), Q&A and closing (30 min).

PRACTICAL DETAILS

Technical Prerequisites

- Laptop with administrator rights
- Operating system: Linux, macOS, or Windows
- 8 GB RAM minimum, 16 GB recommended
- Stable internet connection
- Code editor (VSCode, GoLand, Vim...)

Participant Prerequisites

- Software development experience
- Knowledge of at least one compiled language (Java, C#, C++, Rust...)
- Command line proficiency
- Basic Git knowledge
- Understanding of object-oriented programming concepts

Provided Materials

- Training slides in PDF format
- Practical exercises with detailed solutions
- Source code for examples and the complete GoCheck solution
- Additional resources and references
- Access to a private GitHub repository for 6 months

Post-Training Follow-up

- Q&A session 2 weeks after training
- Email support for 1 month

Training Assessment

- Knowledge quiz at the start of training
- Daily practical exercises via the GoCheck project
- Satisfaction questionnaire at the end of training

PRICING

In-Company Format

- Fixed rate: quote based on configuration, starting from €5,000
- Included: complete training + materials + follow-up
- Group up to 12 participants
- Location: at your premises or online

Additional Options

- Individual post-training coaching: quote on request
- Existing code audit: quote on request
- Additional customized day: quote on request

CONTACT

Want to train your team in Go?

Contact us to discuss your goals and customize this training to your technical and business context.

📧 Email: yohann.bethoule@gmail.com

🌐 Website: www.yohannbethoule.com

📞 Phone: +33 6 12 31 25 54

Document version: 2.0

Last updated: March 2026