MAQUETTE PÉDAGOGIQUE

Formation Go pour les équipes de développement

Projet fil rouge : GoCheck — Moniteur de santé d'URLs

INFORMATIONS GÉNÉRALES

Durée totale	5 demi-journées (15 heures)
Format	Présentiel ou distanciel
Public cible	Développeurs backend/fullstack avec expérience de programmation
Prérequis	Ligne de commande, Git, langage compilé (Java, C#, TypeScript)
Effectif	Jusqu'à 12 participants

OBJECTIFS PÉDAGOGIQUES

À l'issue de cette formation, les participants seront capables de :

- Maîtriser la syntaxe et les concepts fondamentaux du langage Go
- Appliquer les conventions idiomatiques et bonnes pratiques de la communauté Go
- Écrire des tests unitaires et d'intégration robustes
- Implémenter la concurrence avec goroutines et channels de manière efficace
- Développer des applications Go en production selon les standards de l'industrie

COMPÉTENCES ACQUISES

Compétences techniques

- Compilation et exécution de programmes Go
- Gestion idiomatique des erreurs
- Manipulation des structures de données (slices, maps, structs)
- Écriture de tests avec le package testing et outils tiers
- Utilisation de la concurrence (goroutines, channels, context)
- Création d'APIs REST structurées selon les principes de clean architecture

Compétences méthodologiques

- Analyse et résolution de problèmes avec l'approche Go
- Debug et profilage d'applications
- Organisation et structuration de projets Go
- Revue de code et respect des standards communautaires

APPROCHE PÉDAGOGIQUE

La formation s'appuie sur une pédagogie **« practice-first »** : les participants sont confrontés aux problèmes avant de recevoir les apports théoriques. Ce format, adapté à un public de

développeurs expérimentés, crée un contraste mémorable entre les réflexes acquis dans d'autres langages et l'approche idiomatique Go.

Format type d'une demi-journée (3h)

Phase	Durée	Description
Briefing	15 min	Présentation de l'objectif, briques minimales pour démarrer
Pratique	45 min	Les participants codent, le formateur circule et débloque
Correction live	45 min	Construction de la solution ensemble, discussion des alternatives
Théorie & approfondissement	1h15	Concepts, patterns et bonnes pratiques ancrés sur le code produit

Exception : le jour 1 suit un format plus classique (théorie courte → pratique immédiate) car les participants découvrent le langage.

PROJET FIL ROUGE : GOCHECK

GoCheck est un **moniteur de santé d'URLs** développé de A à Z au fil de la formation. Le service permet d'enregistrer des URLs à surveiller via une API REST, de les vérifier périodiquement (statut HTTP, temps de réponse) et d'exposer les résultats. Le domaine métier est volontairement simple pour laisser toute la place à l'apprentissage du langage.

Progression du projet

Jour	Ce qu'on construit	Concepts Go clés
Jour 1	CLI qui vérifie des URLs	Structs, slices, maps, errors, net/http client
Jour 2	API REST structurée (CRUD targets)	Packages, interfaces, net/http server, clean architecture
Jour 3	Endpoints de check + suite de tests	testing, httptest, table-driven tests, mocks
Jour 4	Scheduling concurrent + worker pool	Goroutines, channels, context, sync, graceful shutdown
Jour 5	Approfondissement personnalisé	Selon le module choisi

PROGRAMME DÉTAILLÉ

📘 JOUR 1 – INTRODUCTION À GO

Durée : **3h00** | Format : alternance théorie courte / pratique immédiate

Objectifs opérationnels

- Installer et configurer un environnement de développement Go
- Écrire et exécuter son premier programme Go
- Manipuler les types de données fondamentaux
- Comprendre la gestion des erreurs en Go

Bloc 1 — Prise en main (45 min)

Théorie (20 min) : Historique et philosophie de Go. Domaines d'application (APIs, microservices, outils CLI). Installation du SDK Go, configuration de l'éditeur (VSCode, GoLand). Premier programme Hello World, commandes de base : go run, go build, go mod init.

Pratique (25 min) : Les participants installent leur environnement, créent leur module gocheck, écrivent un programme qui compile et s'exécute.

Bloc 2 — Syntaxe et structures de données (1h15)

Théorie (30 min) : Variables et déclarations (var, :=, const). Types de base (int, string, bool, float). Structures de contrôle (if, for, switch). Fonctions, valeurs de retour multiples, pointeurs. Structs, slices, maps, méthodes sur les types.

Pratique (45 min) : Modéliser les structs Target et CheckResult. Créer un MemoryStore avec une map. Écrire un programme qui crée des targets, les stocke, les parcourt et les affiche.

Bloc 3 — Le premier checker (1h)

Théorie (15 min) : Le package net/http côté client. Gestion des erreurs (if err != nil). Le mot-clé defer. Mesure du temps avec time.Now() et time.Since().

Pratique (45 min) : Écrire la fonction Check(url string) CheckResult. Câbler le main.go avec des URLs en dur (dont une en erreur, une lente via httpstat.us). Afficher un tableau de résultats dans le terminal.

Méthodes pédagogiques

- Exposé théorique avec démonstrations live coding (50%)
- Exercices pratiques guidés (50%)

Livrable GoCheck

Programme CLI fonctionnel qui vérifie une liste d'URLs et affiche les résultats (statut HTTP, temps de réponse, erreurs) dans le terminal.

✍ JOUR 2 – API REST ET BONNES PRATIQUES

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

Objectifs opérationnels

- Appliquer les conventions idiomatiques Go
- Organiser un projet Go selon les standards (cmd/, internal/)
- Définir et implémenter des interfaces
- Exposer une API REST avec net/http

Phase pratique (1h15)

Briefing (15 min) : Présentation de la structure de projet cible et de la syntaxe d'un handler HTTP avec http.NewServeMux (Go 1.22+). Consignes :

- Réorganiser le code du jour 1 dans une structure cmd/ + internal/ (domain, checker, storage, handler)
- Extraire une interface TargetRepository (Add, Get, List, Delete)
- Implémenter cette interface dans le MemoryStore existant
- Créer 4 endpoints REST : POST /targets, GET /targets, GET /targets/{id}, DELETE /targets/{id}
- Tester avec curl

Pratique (1h) : Les participants travaillent. Le formateur circule pour débloquer.

Phase correction + théorie (1h45)

Correction live (45 min) : Construction de la solution ensemble. Les concepts sont introduits au fil de la correction :

- Packages, visibilité (majuscule/minuscule), rôle de internal/
- Interfaces implicites, principe « accept interfaces, return structs »
- Handlers HTTP, encodage JSON, status codes appropriés
- Gestion d'erreurs avec error wrapping (fmt.Errorf + %w)

Approfondissement théorique (30 min) : Naming conventions et culture Go (simplicité, lisibilité). Outils de qualité : go fmt, go vet, golangci-lint — démonstration live sur le code du jour.

Démonstration de refactoring (30 min) : Application en direct de 2–3 bonnes pratiques sur le code corrigé. Functional options pattern pour la configuration du serveur HTTP (port, timeouts). Composition vs héritage.

Méthodes pédagogiques

- Pratique autonome (40%)
- Correction live interactive (30%)
- Démonstrations d'outils et refactoring (30%)

Livrable GoCheck

API REST fonctionnelle avec CRUD targets, architecture propre en packages, code validé par golangci-lint.

✅ JOUR 3 – TESTER DU CODE GO

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

Objectifs opérationnels

- Écrire des tests unitaires efficaces avec le package testing
- Implémenter des tests d'intégration HTTP
- Utiliser les mocks et stubs pour isoler les dépendances
- Mesurer et améliorer la couverture de tests

Phase pratique (1h30)

Briefing (15 min) : Présentation des outils (testing, httptest, table-driven tests, subtests) avec un exemple minimal de chaque. Consignes :

- Ajouter 2 endpoints à l'API : POST /targets/{id}/check (check immédiat) et GET /targets/{id}/results (historique)
- Extraire une interface ResultRepository (Save, GetByTarget)
- Écrire des tests unitaires du checker avec httptest.NewServer (simuler 200, 500, timeout, URL invalide) en format table-driven tests
- Créer un mock manuel de TargetRepository pour tester les handlers en isolation
- Écrire un test d'intégration : scénario complet créer target → check → récupérer résultats → supprimer

Pratique (1h15) : Les participants implémentent le code de production et les tests. Bloc le plus long car double travail (fonctionnel + tests).

Phase correction + théorie (1h30)

Correction live (45 min) : Construction de la solution test par test :

- Table-driven tests : structure idiomatique, nommage des cas, slice de test cases
- Mock manuel : struct avec champs configurables, démontrer que c'est simple et suffisant
- Test d'intégration : httptest.NewServer avec le vrai router, helpers t.Helper() et t.Cleanup()
- Subtests : t.Run() pour des rapports lisibles

Théorie complémentaire (30 min) : Quand mocker vs quand tester en intégration. Présentation de gomock et testify/mock pour comparaison avec l'approche manuelle. Build tags (//go:build integration) pour séparer les tests. Couverture : go test -cover, go tool cover -html.

Exercice rapide (15 min) : Lancer la couverture sur leur projet, identifier un chemin non couvert, écrire le test manquant.

Méthodes pédagogiques

- Exercices pratiques guidés (50%)
- Live coding : écriture de tests en direct (30%)
- Revue de tests existants (20%)

Livrable GoCheck

Suite de tests complète (unitaires + intégration), couverture supérieure à 70%, tous les tests passent avec go test ./...

🔄 JOUR 4 – CONCURRENCE EN GO

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

Objectifs opérationnels

- Comprendre le modèle de concurrence de Go (CSP)
- Implémenter un worker pool avec goroutines et channels
- Gérer le cycle de vie avec context.Context
- Assurer un arrêt gracieux de l'application

Phase pratique (1h15)

Briefing (20 min) : Exposé concentré sur les briques nécessaires : goroutines, channels (buffered/unbuffered), select, sync.WaitGroup, context.Context. Présentation du flux cible :

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

Consignes :

- Scheduler : composant qui envoie les targets actives dans un channel à intervalle régulier, s'arrête via context
- Worker pool : N workers (configurable) qui lisent le channel, exécutent le check avec context.WithTimeout, envoient le résultat
- Collector : lit les résultats et persiste dans le store
- Main : câbler le tout, écouter SIGINT/SIGTERM avec signal.NotifyContext, arrêt gracieux

Pratique (55 min) : Les participants implémentent. Partie la plus challengeante — le formateur est très sollicité.

Phase correction + théorie (1h45)

Correction live (1h) : Construction pas à pas, en prenant le temps d'expliquer les pièges à chaque étape :

- Scheduler : time.NewTicker + select + ctx.Done(). Montrer la goroutine leak si on oublie ctx.Done()
- Worker pool : pattern for range ch, pourquoi close(ch) est nécessaire, WaitGroup
- Collector : fermeture en chaîne (scheduler → workers → collector)
- Arrêt gracieux : signal.NotifyContext, ordre d'arrêt, server.Shutdown(ctx)

Théorie complémentaire (30 min) : Concurrence vs parallélisme, le scheduler Go. Nommer les patterns utilisés (worker pool, fan-out/fan-in, pipeline). Data races : démonstration avec go test -race. Présentation de errgroup comme alternative au WaitGroup.

Démonstration (15 min) : Lancer GoCheck avec 10–15 targets, observer les logs des workers en parallèle, faire Ctrl+C et voir l'arrêt propre.

Méthodes pédagogiques

- Exercices progressifs (40%)
- Démonstrations de patterns et debugging (30%)
- Debugging de code concurrent en direct (30%)

Livrable GoCheck

Application complète : API REST + scheduling concurrent + worker pool + arrêt gracieux. go test -race ne détecte aucune data race.

🧠 JOUR 5 – APPROFONDISSEMENT PERSONNALISÉ

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

Cette demi-journée est construite sur mesure selon les besoins de l'équipe. Le projet GoCheck sert de base pour le module choisi. Voici les modules disponibles.

MODULE A – OPTIMISATION & PERFORMANCE

Objectifs : Profiler une application Go, identifier les goulots d'étranglement, comprendre le Garbage Collector et optimiser mémoire et CPU.

Contenu : pprof (CPU et memory profiling), benchmarking avancé, analyse des allocations mémoire, optimisation de hot paths, pool de http.Client et réutilisation de buffers.

Application GoCheck : Benchmarker le checker et le store, profiler le worker pool sous charge (100+ targets), optimiser les allocations.

MODULE B – TESTS AVANCÉS

Objectifs : Pratiquer le TDD en Go, implémenter des tests end-to-end, automatiser la génération de tests.

Contenu : TDD workflow avec Go, BDD avec frameworks (goconvey, ginkgo), property-based testing, tests de mutation, stratégies de non-régression.

Application GoCheck : Ajouter une fonctionnalité en TDD (alertes quand un check échoue N fois), property-based tests sur le scheduler, tests end-to-end de l'application complète.

MODULE C – ARCHITECTURE MICROSERVICES

Objectifs : Structurer un projet microservices, implémenter la communication inter-services, gérer l'observabilité.

Contenu : Architecture hexagonale en Go, gRPC (protobuf), observabilité (slog, Prometheus, OpenTelemetry), patterns de résilience (retry, circuit breaker).

Application GoCheck : Ajouter des logs structurés avec slog, exposer des métriques Prometheus (nombre de checks, latence, taux d'erreur), implémenter un circuit breaker sur les targets en échec.

MODULE D – SÉCURITÉ ET INTEROPÉRABILITÉ

Objectifs : Sécuriser les applications Go, gérer l'authentification et l'autorisation.

Contenu : HTTPS et certificats TLS, JWT et OAuth2, input validation et sanitization, vulnérabilités communes et protections.

Application GoCheck : Ajouter une authentification JWT sur l'API, valider et sanitiser les URLs entrantes (protection SSRF), passer l'API en HTTPS.

Format de la demi-journée

Phase pratique (1h–1h15) : Briefing du module choisi (20–30 min) avec exemples concrets sur GoCheck, puis pratique (45 min).

Phase correction + clôture (1h45–2h) : Correction du module (45 min), rétrospective du projet GoCheck du jour 1 au jour 5 (30 min), Q&A et clôture (30 min).

MODALITÉS PRATIQUES

Prérequis techniques

- Ordinateur portable avec droits administrateur
- Système d'exploitation : Linux, macOS ou Windows
- 8 Go de RAM minimum, 16 Go recommandé
- Connexion internet stable
- Éditeur de code (VSCode, GoLand, Vim...)

Prérequis participants

- Expérience en développement logiciel
- Connaissance d'au moins un langage compilé (Java, C#, C++, Rust...)
- Maîtrise de la ligne de commande
- Connaissance de base de Git
- Compréhension des concepts de programmation orientée objet

Supports fournis

- Slides de formation au format PDF
- Exercices pratiques avec corrections détaillées
- Code source des exemples et du corrigé GoCheck complet
- Ressources complémentaires et références
- Accès à un repository GitHub privé pendant 6 mois

Suivi post-formation

- Session de questions-réponses 2 semaines après la formation
- Support par email pendant 1 mois

Évaluation de la formation

- Quiz de connaissances en début de formation
- Exercices pratiques quotidiens via le projet GoCheck
- Questionnaire de satisfaction en fin de formation

TARIFICATION

Format intra-entreprise

- Tarif forfaitaire : sur devis selon configuration, à partir de 5000€
- Inclus : formation complète + supports + suivi
- Groupe : jusqu'à 12 participants
- Lieu : dans vos locaux ou en ligne

Options supplémentaires

- Coaching individuel post-formation : sur devis
- Audit de code existant : sur devis
- Jour supplémentaire personnalisé : sur devis

CONTACT

Vous souhaitez former votre équipe à Go ?

Contactez-nous pour échanger sur vos objectifs et personnaliser cette formation selon votre contexte technique et métier.

📧 Email : yohann.bethoule@gmail.com

🌐 Site web : www.yohannbethoule.com

📞 Téléphone : +33 6 12 31 25 54

Version du document : 2.0

Dernière mise à jour : Mars 2026