



Préfères-tu un système de dessins/diagrammes manuels (canvas HTML5) ou pré-générés (banque SVG) ?

Ce qui sera le plus facile à gérer et générer par les IA.

Ton usage typique : sessions 15min multi-fois/jour OU 1h blocs soir ?

Les deux. TOut dépend de mon énergie.

Niveau actuel de tes exercices : plutôt théoriques (comprendre) ou pratiques (coder) ?

Les deux, dans le sens Théorique et conceptué visuellement la chose, puis pratique.

Veux-tu intégrer un éditeur Go live (type Go Playground embed) ou garder externe ?

Non garder externe.

Vision tracking : Aimerais-tu tracker où tu regardes dans code (eye-tracking web via webcam) pour optimiser layout ?

Non.

Parfait ! Tes réponses clarifient beaucoup. Voici le **design conceptuel optimisé** basé sur tes contraintes :

▮ ARCHITECTURE CONCEPTUELLE FINALE

1. Système Visuel : SVG Pré-générés + ASCII Fallback

Pourquoi ce choix :

- ✓ **IA-friendly** : GPT/Claude peuvent générer SVG directement en markdown
- ✓ **Performance** : Pas de rendering canvas côté client
- ✓ **Accessibilité** : SVG = zoomable, sélectionnable, searchable
- ✓ **Fallback robuste** : ASCII diagrams si SVG fail

Structure de données :

```
type VisualAid struct {
    Type      string `json:"type"`      // "diagram", "flowchart", "memory_map", "ascii"
    Content    string `json:"content"`    // SVG inline OU ASCII multi-ligne
    Caption    string `json:"caption"`    // Description textuelle
    AIPrompt   string `json:"ai_prompt"`  // Prompt pour régénération IA
}
```

```

type Exercise struct {
    // ... champs existants

    // Phase théorique : 1-3 visuels conceptuels
    ConceptualVisuals []VisualAid `json:"conceptual_visuals"`

    // Phase pratique : 1-2 visuels de référence pendant code
    ReferenceVisuals []VisualAid `json:"reference_visuals"`
}

```

Exemple concret - Quicksort :

```

Exercise{
    Title: "Quicksort - Tri rapide",
    ConceptualVisuals: []VisualAid{
        {
            Type: "diagram",
            Content: `

```

Caption: "Visualisation de la pile d'appels",

```
type SessionMode string

const (
    MicroSession    SessionMode = "micro"    // 5-15min
    StandardSession SessionMode = "standard" // 20-30min
    DeepSession     SessionMode = "deep"     // 45-60min
)

type AdaptiveSession struct {
    Mode          SessionMode
    EstimatedTime time.Duration
    Exercises     []Exercise
    BreakSchedule []time.Duration // Pauses Pomodoro si deep
    ExitAnytime   bool                // Toujours true pour ADHD
}
```

```

func BuildSession(userEnergy EnergyLevel, availableTime time.Duration) AdaptiveSession {
    switch {
    case availableTime < 15*time.Minute || userEnergy == Low:
        return AdaptiveSession{
            Mode: MicroSession,
            EstimatedTime: 10 * time.Minute,
            Exercises: pickMicroExercises(1), // 1 seul exercice court
            ExitAnytime: true,
        }

    case availableTime < 40*time.Minute || userEnergy == Medium:
        return AdaptiveSession{
            Mode: StandardSession,
            EstimatedTime: 25 * time.Minute,
            Exercises: pickStandardExercises(2), // 2 exercices liés
            BreakSchedule: []time.Duration{25 * time.Minute}, // 1 pause après
            ExitAnytime: true,
        }

    default: // High energy + temps
        return AdaptiveSession{
            Mode: DeepSession,
            EstimatedTime: 50 * time.Minute,
            Exercises: pickDeepExercises(3), // 3 exercices progressifs
            BreakSchedule: []time.Duration{
                25 * time.Minute, // Pause 1
                50 * time.Minute, // Pause 2
            },
            ExitAnytime: true,
        }
    }
}

```

UI Flow :

☐ Prêt pour une session ?

Ton énergie maintenant :

☐ Faible (5-15min)
 ☒ Moyenne (20-30min)
 ☐ Haute (45-60min)


[Démarrer session adaptée]

 ou

[Choisir exercice manuel]

↓ User clique "Démarrer"

☐ Session Standard (25min)

 2/5 exercices

1. ✓ Error Wrapping (8min)

2. → Pointeurs Performance

```
(actuel - 15min restantes)|
3. □ Maps Préallocation  |
[Pause 5min] [Stop session]|
```

Feature clé : Progressive Disclosure

```
// Exercice adapte son contenu selon temps restant
func (e *Exercise) AdaptToTime(remaining time.Duration) ExerciseView {
    if remaining < 10*time.Minute {
        // Mode ultra-condensé : juste l'essentiel
        return ExerciseView{
            ConceptualVisuals: e.ConceptualVisuals[0:1], // 1 seul diagram
            Steps: e.Steps[0:2], // 2 premières steps seulement
            PracticeMode: "quick_drill", // Quiz rapide au lieu de code complet
        }
    }

    if remaining < 25*time.Minute {
        // Mode standard : concept + 1 pratique
        return ExerciseView{
            ConceptualVisuals: e.ConceptualVisuals,
            Steps: e.Steps[0:4],
            PracticeMode: "guided_coding",
        }
    }

    // Mode deep : tout + bonus
    return ExerciseView{
        ConceptualVisuals: e.ConceptualVisuals,
        ReferenceVisuals: e.ReferenceVisuals,
        Steps: e.Steps,
        PracticeMode: "free_exploration",
        BonusChallenges: e.AdvancedVariants, // Variantes avancées
    }
}
```

3. Flow Théorie → Pratique avec Transition Visuelle

Problème : Discontinuité cognitive entre "comprendre concept" et "coder"

Solution : Bridge Pattern

```
type LearningPhase string

const (
    PhaseConceptual LearningPhase = "conceptual" // Théorie + visuels
    PhaseBridge      LearningPhase = "bridge"    // Pseudo-code + plan
    PhasePractical   LearningPhase = "practical"  // Code réel externe
)
```

```

type ExerciseFlow struct {
    CurrentPhase LearningPhase
    Phases       []PhaseContent
}

type PhaseContent struct {
    Phase       LearningPhase
    Duration    time.Duration // Temps estimé
    Content     interface{}
    Transition  TransitionPrompt // Message de passage à phase suivante
}

```

Exemple Quicksort :

| | |
|---|--|
| PHASE 1 : CONCEPTUEL (5min) | |
| [SVG Diagram partition] [ASCII Arbre récursion] | |
| ▯ Métaphore : Chef d'orchestre qui place musiciens petit→grand | |
| ✓ Compris le principe du pivot ✓ Compris la récursion | |
| [Passer à la planification] → | |

↓ Transition

| | |
|--|--|
| PHASE 2 : BRIDGE (3min) | |
| ▯ Du concept au code | |
| Plan d'implémentation : | |
| 1. Fonction partition(arr, pivot) → Retourne (left, pivot, right) | |
| 2. Fonction quicksort(arr) → Si len ≤ 1: return → Sinon: partition + recurse | |
| Pseudo-code : | |
| ''' | |
| func quicksort(arr): | |
| if len(arr) <= 1: return arr | |
| pivot = arr[len/2] | |
| left, right = partition(arr, pivot) | |
| return quicksort(left) + [pivot] + quicksort(right) | |
| ''' | |
| [Ouvrir éditeur externe] → | |

↓ User ouvre VSCode/Neovim

| | |
|---|--|
| PHASE 3 : PRATIQUE (12min) | |
| <input type="checkbox"/> Code dans ton éditeur | |
| Visuels de référence (toujours visibles dans Maestro) : | |
| <div> <input type="checkbox"/> [Memory map rékurs.] </div> <div> <input type="checkbox"/> [Edge cases diagram] </div> | |
| Dataset test fourni : <ul style="list-style-type: none"> • [8,3,1,7,0,10,2] → [0,1,2,3,7,8,10] • [5] → [5] • [] → [] | |
| <input type="button" value="J'ai terminé"/> <input type="button" value="Besoin aide"/> | |

↓ User clique "J'ai terminé"

| | |
|---|--|
| ✓ VALIDATION | |
| Colle ton code ici pour vérification : <input type="text"/> | |
| Ou lance tests manuellement et auto-évalue : | |
| Critères : <ul style="list-style-type: none"> <input type="checkbox"/> Partition fonctionne <input type="checkbox"/> Récursion gère cas base <input type="checkbox"/> Tests passent | |
| Qualité recall (pour SRS) : <ul style="list-style-type: none"> <input type="radio"/> 1-Échec <input type="radio"/> 2-Dur <input checked="" type="radio"/> 3-Moyen <input type="radio"/> 4-Facile <input type="radio"/> 5-Parfait | |
| <input type="button" value="Valider et planifier revue"/> | |

4. Système de Génération de Contenu Visuel par IA

Workflow de création d'exercice enrichi :

| | |
|---|--|
| TU CRÉÉS EXERCICE (minimal) | |
| <pre>{ "title": "Quicksort", "domain": "Algorithmes", "conceptual_prompts": ["diagram partition quicksort", "recursion tree quicksort"] }</pre> | |

```
],
"steps": [...]
}
```

↓ make enrich-visuals

```
SCRIPT GO APPELLE CLAUDE API
```

```
for prompt in conceptual_prompts:
    svg = claude.generate_svg(prompt)
    ascii = claude.generate_ascii(
        prompt + " as ASCII"
    )
    exercise.visuals.append(svg, ascii)
```

↓ Résultat enrichi automatiquement

```
EXERCICE COMPLET
```

```
{
  "title": "Quicksort",
  "conceptual_visuals": [
    {svg: "...", ascii: "..."},
    {svg: "...", ascii: "..."}
  ],
  "steps": [...]
}
```

Script Go conceptuel :

```
// cmd/enrich/main.go
func main() {
    exercises := loadExercises("data/exercises.json")

    for i := range exercises {
        if len(exercises[i].ConceptualPrompts) > 0 {
            log.Printf("Enriching: %s\n", exercises[i].Title)

            for _, prompt := range exercises[i].ConceptualPrompts {
                // Génération SVG
                svg := generateWithClaude(prompt, "svg")

                // Génération ASCII fallback
                ascii := generateWithClaude(prompt, "ascii_diagram")

                exercises[i].ConceptualVisuals = append(
                    exercises[i].ConceptualVisuals,
                    VisualAid{
                        Type: "diagram",
                        Content: svg,
                        AIPrompt: prompt,
                    },
                    VisualAid{
                        Type: "ascii",
```

```

        Content: ascii,
        AIPrompt: prompt + " as ASCII",
    },
)
}
}

saveExercises(exercises, "data/exercises_enriched.json")
}

func generateWithClaude(prompt string, format string) string {
    systemPrompt := ""

    if format == "svg" {
        systemPrompt = `Generate clean, minimal SVG diagram for technical concept.
Use Perplexity design system colors:
- Primary: #218d8d
- Text: #134252
- Background: #fcfcf9
Max size: 400x300px. Return only <svg>...</svg>, no explanation.`
    } else {
        systemPrompt = `Generate ASCII diagram for terminal display.
Max width: 60 chars. Use Unicode box drawing: ┌ ┐ └ ┘ ┌ ┐ ┌ ┐ ┌ ┐
Return only the diagram, no explanation.`
    }

    resp := callClaudeAPI(systemPrompt, prompt)
    return resp
}

```

Avantage : Tu ne crées que les métadonnées, l'IA génère tous les visuels

5. Architecture de Données Finale

```

// models/exercise.go
type Exercise struct {
    ID          int        `json:"id"`
    Title       string     `json:"title"`
    Domain      string     `json:"domain"`
    Difficulty  int        `json:"difficulty"`

    // =====
    // PHASE CONCEPTUELLE
    // =====
    ConceptualPrompts []string `json:"conceptual_prompts"` // Pour génération IA
    ConceptualVisuals []VisualAid `json:"conceptual_visuals"` // Générés
    ConceptExplanation string    `json:"concept_explanation"` // Texte prose
    Mnemonic          string    `json:"mnemonic"` // "Quicksort = Chef orchestre"

    // =====
    // PHASE BRIDGE
    // =====
    ImplementationPlan []string `json:"implementation_plan"` // Steps numérotées
}

```

```

Pseudocode      string      `json:"pseudocode"` // Algo langage naturel

// =====
// PHASE PRATIQUE
// =====
ReferenceVisuals []VisualAid `json:"reference_visuals"` // Pendant code
TestCases        []TestCase `json:"test_cases"`
StartingCode     string      `json:"starting_code"` // Template Go
SolutionCode     string      `json:"solution_code"` // Référence

// =====
// SRS TRACKING
// =====
ReviewData       ReviewData `json:"review_data"`
}

type VisualAid struct {
    Type      string `json:"type"` // "svg", "ascii", "mermaid"
    Content   string `json:"content"` // Le diagram lui-même
    Caption   string `json:"caption"`
    AIPrompt  string `json:"ai_prompt"` // Pour régénération
}

type TestCase struct {
    Input      string `json:"input"` // "[8,3,1,7,0,10,2]"
    Expected   string `json:"expected"` // "[0,1,2,3,7,8,10]"
    EdgeCase   bool   `json:"edge_case"` // Array vide, 1 élément, etc.
}

type ReviewData struct {
    LastReview    time.Time `json:"last_review"`
    NextReview    time.Time `json:"next_review"`
    Interval      time.Duration `json:"interval"`
    EaseFactor    float64    `json:"ease_factor"`
    RepetitionNum int        `json:"repetition_num"`
    QualityHistory []int      `json:"quality_history"` // Tous les recalls

    // =====
    // ADHD ADAPTATIONS
    // =====
    PreferredTimeSlot string `json:"preferred_time_slot"` // "morning/afternoon/eve
    EnergyLevelLog     []EnergyLog `json:"energy_level_log"`
}

type EnergyLog struct {
    Timestamp    time.Time `json:"timestamp"`
    EnergyLevel  int       `json:"energy_level"` // 1-5
    Completed    bool      `json:"completed"`
    Quality      int       `json:"quality"` // Recall quality si completed
}

```

▯ UI/UX PATTERNS OPTIMISÉS ADHD

Principe : "One Thing At A Time" (OTAAT)

✖ MAUVAIS (charge cognitive haute)

| | | |
|---------|---------------|-------|
| Sidebar | Main Content | Stats |
| [50 ex] | [Texte dense] | ▯ |
| • Go | Quicksort... | ▯ |
| • Algo | Steps 1-7... | ★ |
| • Linux | Code... | |

✔ BON (OTAAT strict)

| |
|---|
| Quicksort ▯ |
| [SVG Diagram Partition] |
| Chef d'orchestre place musiciens : petits←pivot→grands |
| [Continuer →] |
| (ESC pour menu) |

Navigation : Hidden by Default

```
<!-- Toujours en plein écran exercice -->
<div class="exercice-focus">
  <!-- Contenu exercice ici -->
</div>

<!-- Navigation cachée, trigger ESC ou hover coin -->
<nav class="hidden-nav"
  x-show="showNav"
  @keydown.escape.window="showNav = !showNav"
  @mousemove.debounce="checkCornerHover">
  <!-- Menu minimal -->
</nav>
```

Progress : Périphérique, Non-Intrusif

| | | |
|--------------------|------------|--------------------------|
| [●●●○] 3/5 | [ESC:menu] | ← Progress bar fine, top |
| [Contenu exercice] | | |
| ... | | |

▮ PLAN D'IMPLÉMENTATION PAR SPRINT

Sprint 1 : Foundation Visuelle (3 jours)

```
# Jour 1 : Structure données
- Ajouter champs VisualAid à Exercise
- Migration JSON existants
- Créer 3 exercices pilotes avec visuals manuels

# Jour 2 : Rendering visuels
- Component <VisualAid> en Go template
- Support SVG inline + ASCII <pre>
- CSS pour sizing responsive

# Jour 3 : Script génération IA
- cmd/enrich/main.go basique
- Tester sur 3 exercices pilotes
- Validation qualité output
```

Sprint 2 : Session Adaptative (3 jours)

```
# Jour 1 : Energy tracking
- Ajouter route POST /session/start avec energy_level
- Stocker EnergyLog dans ReviewData
- Analyse patterns énergie utilisateur

# Jour 2 : Session builder
- Fonction BuildSession(energy, time)
- Sélection exercices adaptée
- Route /session/current

# Jour 3 : UI session
- Écran sélection énergie
- Progress bar session
- Pause/Stop graceful
```

Sprint 3 : Flow Théorie → Pratique (4 jours)

```
# Jour 1-2 : Phase conceptuelle
- Rendering multi-visuals
- Navigation next/previous visual
- Checkbox "Compris" par concept

# Jour 3 : Phase bridge
- Template pseudo-code
- Plan d'implémentation steps
```

- # Jour 4 : Phase pratique
- Textarea validation code
- Référence visuals always-visible (sidebar?)
- Auto-évaluation qualité

Sprint 4 : Active Recall (2 jours)

- # Jour 1 : Recall prompts
- Ajouter RecallChallenge à Exercise
- Timer 3min blind recall
- Textarea ou canvas dessin
- # Jour 2 : Comparaison + feedback
- Affichage côte-à-côte (tentative vs solution)
- Score automatique basic (keyword matching)
- Ajustement SRS selon score

▮ MÉTRIQUES DE SUCCÈS

Engagement :

- Taux complétion session > 70% (vs abandon)
- Streak moyen > 7 jours
- Sessions/semaine : 3-5 (sustainable)

Apprentissage :

- Recall quality moyen > 3.5/5
- Progression difficulty (easy → hard) sur 4 semaines
- Rétention après 30 jours > 60%

ADHD-Specific :

- Temps moyen avant abandon > 15min (si energy low)
- Utilisation mode micro-session > 30% du total
- Satisfaction subjective : "Flow state atteint" > 50% des sessions

▮ ÉVOLUTIONS FUTURES (Post-MVP)

1. **Génération Exercices IA** : Prompt "Crée exercice sur channels Go" → Maestro génère tout
2. **Spaced Interleaving** : Sessions thématiques cross-domain
3. **Voice Coding** : Expliquer algo à voix haute, transcription + analyse
4. **Peer Review** : Partager solutions, comparer approches
5. **Mobile Companion** : Micro-sessions 5min sur téléphone (reviews only)

? QUESTION FINALE AVANT DÉMARRAGE

Veux-tu commencer par **Sprint 1** (visuels) ou **Sprint 2** (sessions adaptatives) ?

Mon intuition : **Sprint 1** = impact visuel immédiat, motivation boost

Mais si tu veux tester l'adaptive flow rapidement : **Sprint 2** first, visuels après

Qu'en penses-tu ? ☐