Dies ist eine Anleitung für einen AI-Agenten, der die Implementierung des **Adaptiven Päckchen-Systems** in eine bestehende App-Architektur vornehmen soll. Die Integration muss modular, skalierbar und didaktisch fundiert sein, um die gezielte Förderung mathematischer Kompetenzen in erweiterten Zahlenräumen (ZR 100, ZR 1000+) zu gewährleisten.

I. Integrations-, Emulations- und Entwicklungsbereiche

Die Bereiche, die am effizientesten in die bestehende App integriert, emuliert oder als eigenständige Module entwickelt werden können, basieren auf der **Modularen App-Architektur** der "Päckchen-Werkstatt".

A. Effizient Integrierbare Module (Backend-Logik)

Der effizienteste Integrationspunkt ist die **Fehleranalyse-Engine**, da sie auf festen Regeln basiert und skalierbar ist.

1. **Erweiterte Fehlerklassifikation (Regel-Engine):**
   - **Ziel:** Aufbau einer skalierbaren Regel-Engine (z.B. in Python/Flask), die Fehler nicht nur im ZR 20, sondern auch in großen Zahlenräumen (ZR 100+ bis Millionen) kategorisiert.
   - **Zu integrieren/entwickeln:** Neue Fehlertypen wie **Stellenwertverwechslung**, **Zehnerübergang im Hunderterraum**, **Subtraktion mit Übertrag** und **Nullstellenfehler** müssen in das Klassifikationsmodul aufgenommen werden.
   - **Ergebnis:** Das Modul muss ein **Fehlerprofil** aus mehreren Aufgaben generieren können, um systematische Denkfehler zu clustern (z.B. „Kind ignoriert Hunderterstelle“).
2. **Modulare Aufgabenbibliothek (JSON-Basis):**
   - **Ziel:** Die Päckchen-Generierung muss parameterbasiert und einfach erweiterbar sein.
   - **Zu integrieren/entwickeln:** Definition von Päckchen-Typen als JSON-Strukturen, die verschiedene mathematische Prinzipien abdecken (z.B. `constant_sum`, `stellenwert`). Diese JSON-Struktur sollte die Parameter für die Aufgabengenerierung (z.B. `start_a`, `step_a`, `anzahl`) festlegen.
   - **Effizienz:** Die logische Zuordnung (Fehlertyp Päckchen-Typ) kann als einfaches Mapping im Backend implementiert werden.

B. Notwendige Entwicklungen/Emulationen (Frontend/Didaktik)

Diese Bereiche erfordern neue UI-Komponenten, da sie didaktische Prozesse (Visualisierung, Argumentation) digital emulieren müssen.

1. **Visualisierungsmodul:**
   - **Ziel:** Bereitstellung digitaler "Mittel zum Forschen".
   - **Zu entwickeln:** Implementierung von Komponenten zur **dynamischen Darstellung** von **Zahlzerlegung** mit **Stellenwerttafeln**, **Zahlstrahl** (wichtig für große Zahlen) und **Plättchen** für Hunderter/Tausender. Plotly oder Matplotlib können hierfür genutzt werden. Die Visualisierung unterstützt insbesondere die Einsicht in die **Konstanz der Summe** durch die Darstellung gegensinniger Veränderungen.
2. **Reflexions- und Sprachfördermodul:**
   - **Ziel:** Förderung metakognitiver Kompetenzen und mathematischer Kommunikation.
   - **Zu entwickeln:** Integration von **Satzstartern** (z.B. „Die Tausenderstelle bleibt gleich…“) und Reflexionsfragen nach jedem Päckchen. Entwicklung von **Fehlerforschungs-Päckchen** (Musterbruch finden und begründen) und **Inversen Aufgabenstellungen** (Regel Päckchen) als spezielle UI-Formate.
3. **Kreative Förderformate (Gamification):**
   - **Ziel:** Steigerung der Motivation und des Transfers.
   - **Zu emulieren/entwickeln:** Implementierung von Zusatzformaten wie **Escape-Room-Mathematik**, **Päckchen-Detektivspiel** (Fehler finden) oder **Labyrinth-Päckchen** als spielerische Rahmen, die die Basis-Päckchen (Zerlegung, Stellenwert) einbetten.

II. Projektschritte für den AI-Agenten (Implementierungsanweisung)

Die Umsetzung sollte in **fünf Phasen** erfolgen, wobei der AI-Agent spezifische Code- und Datenstrukturen liefern soll.

Phase 1: Konzeption und Datenmodell (Setup)

Ziel: Definition der didaktischen Basis und des modularen Datenmodells. **Code-Aufgaben des AI-Agenten:**

1. **Erstellung des JSON-Schemas für die Aufgabenbibliothek:**
   ◦ Definiere die **JSON-Datenstruktur** für die Päckchen-Vorlagen, die `paeckchen_typ`, `parameter` (z.B. Zahlenraum, Schrittgröße) und die `aufgaben` selbst enthält. Stelle sicher, dass das Schema für große Zahlenräume skalierbar ist.
   ◦ Erstelle erste Templates für **Stellenwert-Päckchen** und **Subtraktions-Zerlegungspäckchen**.
2. **Definition der erweiterten Fehlerregeln:**
   ◦ Erstelle eine strukturierte Mapping-Datei (z.B. Python Dictionary oder JSON) aller relevanten Fehlertypen (inkl. ZR 100+ Typen) und deren direkter Zuordnung zu den optimalen Päckchen-Typen (z.B. `Zehnerübergang nicht erkannt` `constant_sum`, `crossing_tens`).

Phase 2: KI-gestützte Fehleranalyse (Backend-Entwicklung)

Ziel: Entwicklung der Kernlogik zur Fehlerklassifikation und Päckchen-Zuweisung. **Code-Aufgaben des AI-Agenten:**

1. **Entwicklung der Regel-Engine (Python Backend):**
   ◦ Implementiere das **Fehlerklassifikationsmodul** (`klassifySingleError`) als regelbasierte Engine, die Eingaben (JSON-Array von `{ "task": ..., "user_result": ... }`) gegen die in Phase 1 definierten Regeln prüft.
   ◦ Das Modul muss **systematische Denkfehler** identifizieren und ein konsolidiertes `error_profile` (Fehlerprofil) zurückgeben.
2. **Entwicklung des Päckchen-Zuordnungsmoduls:**
   ◦ Implementiere die Funktion zur Auswahl des didaktisch passenden Päckchen-Typs (`selectPaeckchenTypeForError`) basierend auf dem Fehlerprofil und der definierten Zuordnungsmatrix.
   ◦ Implementiere die Kernfunktionen zur Generierung der Päckchen-Aufgaben (`generate*Paeckchen`), die die Parameter aus den JSON-Templates nutzen.

Phase 3: Interaktive Päckchen-Umgebung (Frontend-Entwicklung)

Ziel: Erstellung der Benutzerschnittstelle und der interaktiven Lernumgebung. **Code-Aufgaben des AI-Agenten:**

1. **UI-Entwicklung und Modul-Integration (Streamlit/Flask Frontend):**
   ◦ Erstelle eine spielerische UI für Kinder. Implementiere die **Auswahl- und Vorschlagslogik** für die vom Backend generierten Päckchen.
   ◦ **Visualisierungs-Komponenten (Plotly/SVG):** Implementiere interaktive Visualisierungen für Stellenwerttafeln und Zahlstrahlen. Die Plättchen-Darstellungen müssen dynamisch die Zerlegung und gegensinnige Veränderung (Konstanz der Summe) bei Addition und Subtraktion abbilden können.
2. **Implementierung kreativer Formate:**
   ◦ Codiere die Logik für **Labyrinth-Päckchen** (Ergebnis nächste Startzahl) und das **Päckchen-Detektivspiel** (logische Überprüfung der Musterkette) als interaktive Aufgabenformate.

Phase 4: Reflexions- und Feedbacksystem

Ziel: Implementierung der metakognitiven und sprachlichen Förderkomponenten. **Code-Aufgaben des AI-Agenten:**

1. **Sprachfördermodul:**

◦ Implementiere die Anzeige von **Satzstartern** (z.B. „Die erste Zahl wird um … größer“) und **Reflexionsfragen** nach jedem Päckchen zur Förderung der Argumentation und Begründung.

◦ Entwickle die UI-Komponenten für die **Inverse Aufgabenstellung** (Kind gibt Regel vor, erstellt Päckchen) und die **Muster-Analyse** ("Ja/Nein"-Bewertung).

2. **Fortschritts- und Feedbacksystem:**

◦ Implementiere ein **Selbstbewertungsmodul**.

◦ Codiere die **Adaptive Sequenzierung** im Backend: Wenn ein Päckchen-Typ erfolgreich war, schaltet das System automatisch ein vertiefendes oder komplexeres Päckchen frei.

Phase 5: Lehrkraft-Dashboard und Datenexport

Ziel: Bereitstellung von Verwaltungs- und Exportfunktionen für Lehrkräfte. **Code-Aufgaben des AI-Agenten:**

1. **Datenspeicherung und Dashboard:**

◦ Richte die Speicherung der Lernverläufe und Fehlerprofile (SQLite oder Firebase) ein.

◦ Entwickle das **Lehrkraft-Dashboard**, das eine Übersicht über Fehlerprofile und Förderverläufe liefert.

2. **Exportmodul:**

◦ Implementiere einen **PDF-Generator** (z.B. mit PDFKit oder ReportLab), der die individuell generierten und differenzierten Päckchen (inklusive Visualisierungshilfen) als Arbeitsblätter exportieren kann.