EIAH Training Module – Estrutura, Prisma e Evaluators (v0)

Abaixo estão: (1) estrutura de pastas sugerida, (2) `schema.prisma` inicial, e (3) esqueleto de três evaluators com contratos Zod.

---

1) Estrutura de pastas

```
packages/
  training/
    README.md
    package.json
    tsconfig.json
    src/
      index.ts
      core/
        orchestrator/
          AgentOrchestrator.ts
          TrainingDispatcher.ts
        loops/
          sft.ts
          dpo.ts
          rl.ts
        services/
          ContextSnapshotService.ts
          DatasetBuilder.ts
          EvalRunner.ts
          BillingEngine.ts
          LoggerFactory.ts
      datasets/
        edu/
          README.md
          connectors/
            moodle.ts
          converters/
            to-learning-tasks.ts
          samples/
            eduqa-v1.jsonl
      evaluators/
        index.ts
        rubric_judge.ts
        factuality.ts
        hint_usefulness.ts
        types.ts
      policies/
        default.sft.json
```

```
        default.eval.json
      schemas/
        index.ts
        StudentProfile.ts
        LearningTask.ts
        Rubric.ts
        FeedbackEvent.ts
      utils/
        ids.ts
        time.ts
    prisma/
      schema.prisma

apps/
  workers/
    training-runner/
      src/
        main.ts
        queues/
          training-runs.ts
          eval-runs.ts
      package.json
  cli/
    src/
      commands/
        training-run.ts
        eval-run.ts
        datasets-ingest.ts
        runs-replay.ts
      package.json
```

---

2) `schema.prisma` inicial

Postgres + `@db.Json` / `Json`. Ajuste nomes conforme o seu monorepo. Índices focam consulta por `tenantId`, `workspaceId` e `runId`.

```
// packages/training/prisma/schema.prisma

generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

// ==== Enums ====
```

```
enum TrainingStatus {
  PENDING
  RUNNING
  COMPLETED
  FAILED
  CANCELLED
}

enum TrainingStepType {
  SFT
  DPO
  RL
  EVAL
}

enum TaskType {
  QA_CONTEXTUAL
  EXPLANATION
  STUDY_PLAN
  ESSAY_FEEDBACK
  SOCRATIC_TUTORING
}

// ==== Core (compartilhado) ====

model Run {
  id           String   @id @default(cuid())
  tenantId     String
  workspaceId  String
  agentId      String
  createdAt    DateTime @default(now())
  updatedAt    DateTime @updatedAt
  // relacionamento
  events       RunEvent[]
  trainingRuns TrainingRun[]

  @@index([tenantId, workspaceId])
}

model RunEvent {
  id           String   @id @default(cuid())
  runId        String
  type         String
  payload      Json
  costCents    Int       @default(0)
  createdAt    DateTime  @default(now())

  run          Run       @relation(fields: [runId], references: [id],
onDelete: Cascade)
```

```
  @@index([runId, type, createdAt])
}

// ==== Treinamento ====

model TrainingDataset {
  id           String   @id @default(cuid())
  tenantId     String
  workspaceId  String
  name         String
  version      String   @default("v1")
  description  String?
  // JSONL resumido: cabeçalhos, estatísticas, paths de blobs
  manifest     Json
  taskType     TaskType
  createdAt    DateTime @default(now())

  learningTasks LearningTask[]

  @@unique([tenantId, workspaceId, name, version])
  @@index([tenantId, workspaceId, taskType])
}

model TrainingRun {
  id           String         @id @default(cuid())
  runId        String
  tenantId     String
  workspaceId  String
  datasetId    String
  policyName   String
  status       TrainingStatus @default(PENDING)
  startedAt    DateTime?
  finishedAt   DateTime?
  totalCostCts Int            @default(0)
  meta         Json

  run          Run            @relation(fields: [runId], references: [id],
onDelete: Cascade)
  dataset      TrainingDataset @relation(fields: [datasetId], references:
[id], onDelete: Restrict)
  steps        TrainingStep[]
  evalResults  EvalResult[]

  @@index([tenantId, workspaceId, status])
  @@index([datasetId])
}

model TrainingStep {
  id           String           @id @default(cuid())
  trainingRunId String
  type         TrainingStepType
```

```
  seq          Int
  status       TrainingStatus   @default(PENDING)
  input        Json
  output       Json
  costCents    Int              @default(0)
  createdAt    DateTime         @default(now())
  finishedAt   DateTime?

  trainingRun  TrainingRun      @relation(fields: [trainingRunId],
references: [id], onDelete: Cascade)

  @@index([trainingRunId, type, seq])
}

// ==== Domínio educacional ====

model StudentProfile {
  id           String   @id @default(cuid())
  tenantId     String
  workspaceId  String
  // Identificadores pseudoanonimizados
  studentKey   String   // hash determinístico
  age          Int?
  level        String?
  goals        Json?
  preferences  Json?
  locale       String?  // ex: pt-BR
  createdAt    DateTime @default(now())

  @@unique([tenantId, workspaceId, studentKey])
  @@index([tenantId, workspaceId])
}

model LearningTask {
  id             String           @id @default(cuid())
  datasetId      String
  taskType       TaskType
  context        Json
  prompt         String
  expectedAnswer Json?
  expectedRationale Json?
  rubric         Json? // snapshot da rubrica usada na geração
  difficulty     Int?
  tags           String[]

  dataset        TrainingDataset  @relation(fields: [datasetId], references:
[id], onDelete: Cascade)
  evalResults    EvalResult[]

  @@index([datasetId, taskType])
}
```

```
model Rubric {
  id           String   @id @default(cuid())
  tenantId     String
  workspaceId  String
  name         String
  version      String   @default("v1")
  criteria     Json     // { clarity: {weight:.3}, correctness:{weight:.
5}, ... }
  createdAt    DateTime @default(now())

  @@unique([tenantId, workspaceId, name, version])
}

model EvalResult {
  id             String   @id @default(cuid())
  trainingRunId  String
  learningTaskId String
  evaluator      String   // ex: rubric_judge or factuality
  score          Float
  details        Json
  createdAt      DateTime @default(now())

  trainingRun    TrainingRun @relation(fields: [trainingRunId], references:
[id], onDelete: Cascade)
  learningTask   LearningTask @relation(fields: [learningTaskId], references:
[id], onDelete: Cascade)

  @@index([trainingRunId, evaluator])
  @@index([learningTaskId])
}
```

---

3) Esqueleto de três evaluators

3.1 `evaluators/types.ts`

```ts
// packages/training/src/evaluators/types.ts
import { z } from "zod";

export const EvalInput = z.object({
  runId: z.string(),
  taskId: z.string(),
  agentOutput: z.object({
    answer: z.string(),
    rationale: z.string().optional(),
    meta: z.record(z.any()).optional(),
  }),
  task: z.object({
```

```
    taskType: z.string(),
    context: z.any(),
    prompt: z.string(),
    expectedAnswer: z.any().optional(),
    expectedRationale: z.any().optional(),
    rubric: z.any().optional(),
  }),
});

export type EvalInputT = z.infer<typeof EvalInput>;

export const EvalOutput = z.object({
  evaluator: z.string(),
  score: z.number().min(0).max(1),
  details: z.record(z.any()),
});
export type EvalOutputT = z.infer<typeof EvalOutput>;

export interface Evaluator {
  name: string;
  supports: (taskType: string) => boolean;
  evaluate: (input: EvalInputT) => Promise<EvalOutputT>;
}
```

3.2 `evaluators/rubric_judge.ts`

```
// packages/training/src/evaluators/rubric_judge.ts
import { z } from "zod";
import { EvalInput, EvalOutput, Evaluator } from "./types";

const RubricSchema = z.object({
  criteria: z.record(
    z.object({ weight: z.number().min(0), description:
z.string().optional() })
  ),
});

type RubricT = z.infer<typeof RubricSchema>;

export const rubricJudge: Evaluator = {
  name: "rubric_judge",
  supports: (_taskType) => true,
  evaluate: async (input) => {
    const parsed = EvalInput.parse(input);
    const rubric = RubricSchema.safeParse(parsed.task.rubric);

    if (!rubric.success) {
      return EvalOutput.parse({
        evaluator: "rubric_judge",
        score: 0,
```

```
        details: { error: "invalid_rubric" },
      });
    }

    const r: RubricT = rubric.data;
    // Placeholder: regra simples baseada em palavras-chave e comprimento
    const text = parsed.agentOutput.answer.trim();
    const len = text.split(/\s+/).length;

    let base = 0;
    if (len > 30) base += 0.2; // cobertura mínima
    if (/porque|portanto|logo|assim/.test(text.toLowerCase())) base +=
0.2; // presença de justificativa

    const totalWeight = Object.values(r.criteria).reduce((s, c) => s +
c.weight, 0) || 1;
    const weighted = Math.min(1, base * (1 / totalWeight));

    return EvalOutput.parse({
      evaluator: "rubric_judge",
      score: Number(weighted.toFixed(3)),
      details: { rubric: r, heuristics: { len, hasConnectives: base >=
0.4 } },
    });
  },
};
```

3.3 `evaluators/factuality.ts`

```
// packages/training/src/evaluators/factuality.ts
import { EvalInput, EvalOutput, Evaluator } from "./types";

export const factuality: Evaluator = {
  name: "factuality",
  supports: (taskType) => ["QA_CONTEXTUAL",
"EXPLANATION"].includes(taskType),
  evaluate: async (input) => {
    const { task, agentOutput } = EvalInput.parse(input);

    // Placeholder sem LLM: compara com expectedAnswer se existir
    let score = 0.5;
    if (task.expectedAnswer && typeof task.expectedAnswer === "string") {
      const gold = String(task.expectedAnswer).toLowerCase().trim();
      const pred = agentOutput.answer.toLowerCase().trim();
      if (gold.length > 0) {
        const overlap = jaccardSimilarity(tokens(gold), tokens(pred));
        score = Math.max(0, Math.min(1, overlap));
      }
    }
```

```
    return EvalOutput.parse({
      evaluator: "factuality",
      score: Number(score.toFixed(3)),
      details: { method: "token-jaccard", hasGold:
Boolean(task.expectedAnswer) },
    });
  },
};

function tokens(s: string): Set<string> {
  return new Set(s.replace(/[^\p{L}\p{N}\s]/gu, " ").toLowerCase().split(/
\s+/).filter(Boolean));
}

function jaccardSimilarity(a: Set<string>, b: Set<string>): number {
  const inter = new Set([...a].filter(x => b.has(x))).size;
  const union = new Set([...a, ...b]).size;
  if (union === 0) return 0;
  return inter / union;
}
```

3.4 `evaluators/hint_usefulness.ts`

```
// packages/training/src/evaluators/hint_usefulness.ts
import { EvalInput, EvalOutput, Evaluator } from "./types";

/**
 * Mede utilidade de dica: concisão + presença de próximos passos acionáveis.
 * Heurística simples que pode ser trocada por LLM-as-a-judge.
 */
export const hintUsefulness: Evaluator = {
  name: "hint_usefulness",
  supports: (taskType) => ["SOCRATIC_TUTORING",
"EXPLANATION"].includes(taskType),
  evaluate: async (input) => {
    const { agentOutput } = EvalInput.parse(input);
    const text = agentOutput.answer.trim();

    const lengthPenalty = Math.max(0, 1 - (text.split(/\s+/).length / 120));
    const actionables = countMatches(text.toLowerCase(), [
      "tente",
      "faça",
      "substitua",
      "calcule",
      "verifique",
      "agora",
      "passo",
    ]);

    const hasQuestion = /\?$/.test(text) || /por que|como|o que/.test(text);
```

```
    const score = clamp(0, 1, 0.3 * lengthPenalty + 0.5 *
sigmoid(actionables) + 0.2 * Number(hasQuestion));

    return EvalOutput.parse({
      evaluator: "hint_usefulness",
      score: Number(score.toFixed(3)),
      details: { actionables, hasQuestion, lengthPenalty:
Number(lengthPenalty.toFixed(3)) },
    });
  },
};

function countMatches(text: string, phrases: string[]): number {
  return phrases.reduce((acc, p) => acc + (text.includes(p) ? 1 : 0), 0);
}

function sigmoid(x: number): number {
  return 1 / (1 + Math.exp(-x));
}

function clamp(min: number, max: number, v: number): number {
  return Math.min(max, Math.max(min, v));
}
```

3.5 `evaluators/index.ts`

```
// packages/training/src/evaluators/index.ts
import { Evaluator } from "./types";
import { rubricJudge } from "./rubric_judge";
import { factuality } from "./factuality";
import { hintUsefulness } from "./hint_usefulness";

export const EVALUATORS: Record<string, Evaluator> = {
  [rubricJudge.name]: rubricJudge,
  [factuality.name]: factuality,
  [hintUsefulness.name]: hintUsefulness,
};

export type { Evaluator } from "./types";
```

---

4) Pontos de integração rápidos

- Persistência: cada evaluator deve gerar `EvalResult` e um `RunEvent` tipo `eval` com `costCents`.
- Orquestração: `EvalRunner` recebe uma lista de evaluators por `policy` ou `--suite` e publica resultados em lote.

- Segurança: se `StudentProfile` contiver PII, usar chaves pseudônimas e KMS no nível de aplicação antes do `create`.

Pronto para copiar no monorepo. Caso queira, adiciono scripts de CLI e um `EvalRunner` mínimo em TypeScript.```