# CodeTutor — Academic Learning Infrastructure Overview

**Version 1.0 | January 2026**

**Institutional Documentation for Academic Integration**

---

## Table of Contents



---

## 1. System Purpose

### 1.1 Why CodeTutor Exists

CodeTutor was designed to address fundamental challenges in computer science education that traditional Learning Management Systems (LMS) fail to solve:

**Academic Measurement Challenge** Traditional programming courses struggle to objectively measure student capability. Grading code manually is inconsistent, time-intensive, and subject to bias. CodeTutor introduces automated, deterministic evaluation where identical code produces identical scores—removing subjectivity from technical assessment.

**Transparency Requirement** Students deserve to understand exactly how they are evaluated. CodeTutor provides immediate, granular feedback on every submission: which test cases passed, which failed, and precisely why. This transparency transforms assessment from a black box into a learning opportunity.

**Scalability Imperative** Institutions cannot sustainably evaluate thousands of code submissions weekly using human graders. CodeTutor executes and grades code automatically within seconds, enabling courses to scale from 30 to 3,000 students without proportional staffing increases.

**AI-Era Learning Adaptation** As AI tools become ubiquitous, education must shift from knowledge recall to skill verification. CodeTutor evaluates whether students can actually write working code—not whether they can describe programming concepts. This positions assessment to remain meaningful as AI assistants become standard tools.

### 1.2 Differentiation from Traditional Systems

| Dimension | Traditional LMS | Git-Based Grading | CodeTutor |
|---|---|---|---|
| **Feedback Latency** | Days to weeks | Hours to days | Seconds |
| **Grading Consistency** | Variable by grader | Dependent on scripts | Deterministic |
| **Student Iteration** | Limited attempts | Manual resubmission | Unlimited practice |
| **Progress Visibility** | End-of-term grades | Commit history only | Real-time dashboards |
| **Learning Adaptation** | None | None | Algorithmic personalization |
| **Skill Verification** | Essay/quiz based | Code review | Automated test execution |

**Key Differentiators:**

1. **Immediate Verification**: Students know within seconds whether their solution works
2. **Unlimited Practice**: Learning occurs through iteration, not single-attempt stakes
3. **Objective Standards**: Test cases define correctness, not stylistic preferences
4. **Continuous Assessment**: Progress is measured continuously, not at midterm/final checkpoints
5. **Adaptive Difficulty**: The system adjusts to each student's demonstrated ability

---

## 2. Pedagogical Model

### 2.1 Adaptive Learning Principles

CodeTutor implements evidence-based adaptive learning through a multi-factor scoring algorithm that selects questions based on individual student performance:

**Core Adaptation Factors:**

| Factor | Weight | Purpose |
|---|---|---|
| Weak Topic Prioritization | +50 pts | Students with <60% pass rate receive targeted practice |
| Difficulty Matching | +30 pts | Questions align with demonstrated skill level |
| Recent Failure Retry | +40 pts | Failed questions resurface within 24 hours |
| Topic Sequencing | +25 pts | Curriculum order preserved for new concepts |
| Streak Momentum | +20 pts | Challenge escalation when performing well |
| Spaced Repetition | +20 pts | Mastered content revisited after 7+ days |

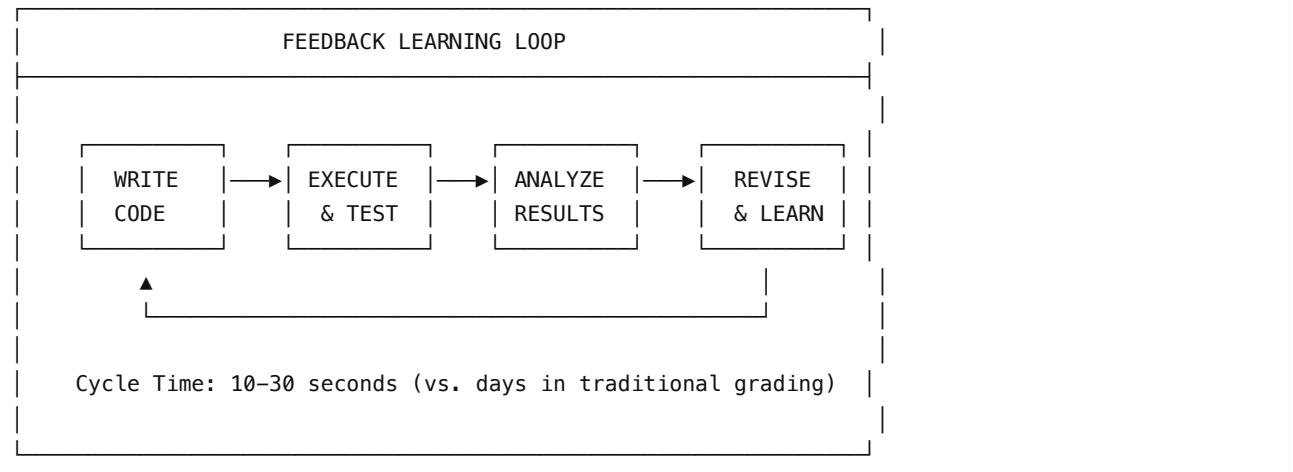
**Difficulty Adjustment Logic:**

```
IF pass_rate < 40% THEN reduce_difficulty(−30 pts)
IF pass_rate > 80% THEN increase_difficulty(+30 pts)
Target Difficulty = Skill Level × 5
```

This creates a personalized learning path where struggling students receive reinforcement while advanced students face appropriately challenging problems.

### 2.2 Feedback-Driven Learning Loops

The system implements rapid feedback cycles that accelerate learning:




**Feedback Components:**

- **Compilation Status**: Immediate syntax error identification with line numbers
- **Test Case Results**: Per-test pass/fail with expected vs. actual output
- **Execution Metrics**: Runtime duration and memory consumption
- **Error Classification**: Categorized feedback (SYNTAX, LOGIC, TIMEOUT, EDGE_CASE, etc.)
- **Progressive Hints**: On-demand assistance with point cost trade-off

### 2.3 Trial-and-Error Methodology

CodeTutor is architected around the understanding that programming skill develops through iteration:

**Design Principles:**

1. **No Penalty for Failure**: Failed attempts are learning data, not grade deductions
2. **Unlimited Submissions**: Students may retry until they succeed
3. **Preserved Progress**: Each attempt is recorded for personal reflection
4. **Incremental Improvement**: Partial progress is visible through passing test counts

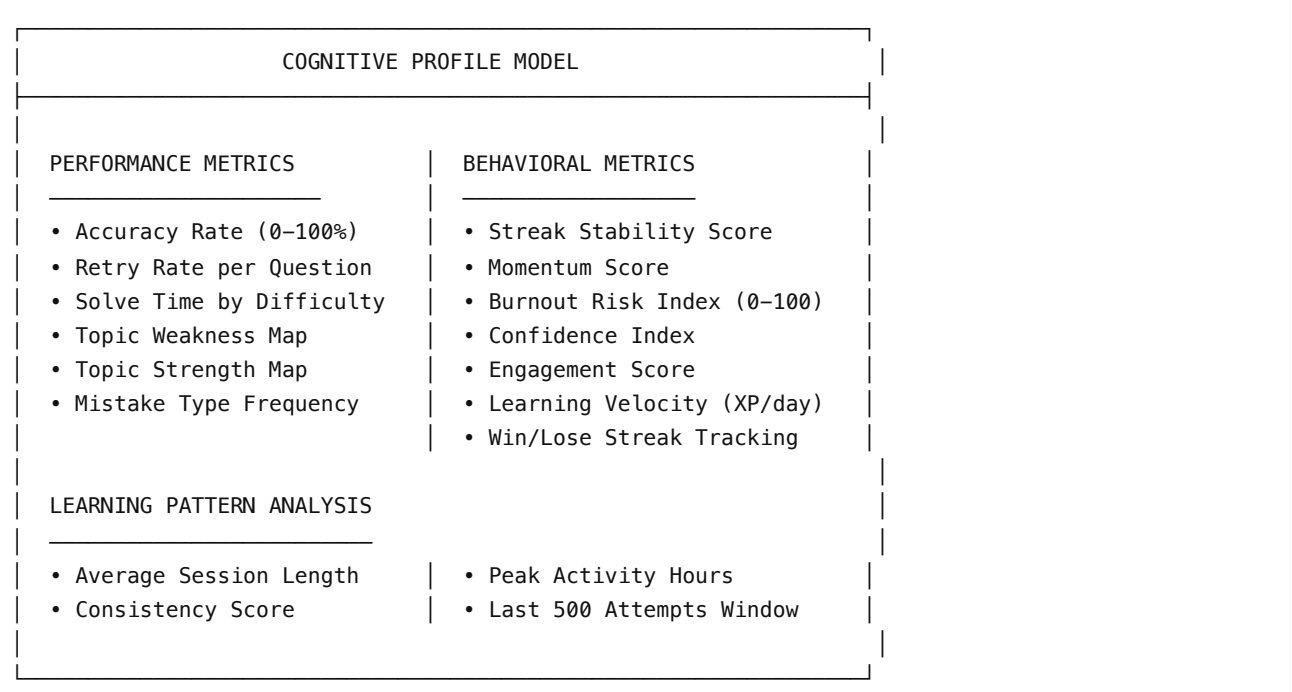
**Attempt Status Taxonomy:**

| Status | Meaning | Learning Signal |
|---|---|---|
| `PASS` | All tests successful | Mastery demonstrated |
| `FAIL` | Some tests failed | Logic refinement needed |
| `COMPILE_ERROR` | Code doesn't compile | Syntax correction required |
| `RUNTIME_ERROR` | Execution crashed | Exception handling needed |
| `TIMEOUT` | Exceeded time limit | Algorithm optimization required |
| `MEMORY_EXCEEDED` | Resource exhaustion | Space complexity issue |

### 2.4 Skill Profiling and Progression Logic

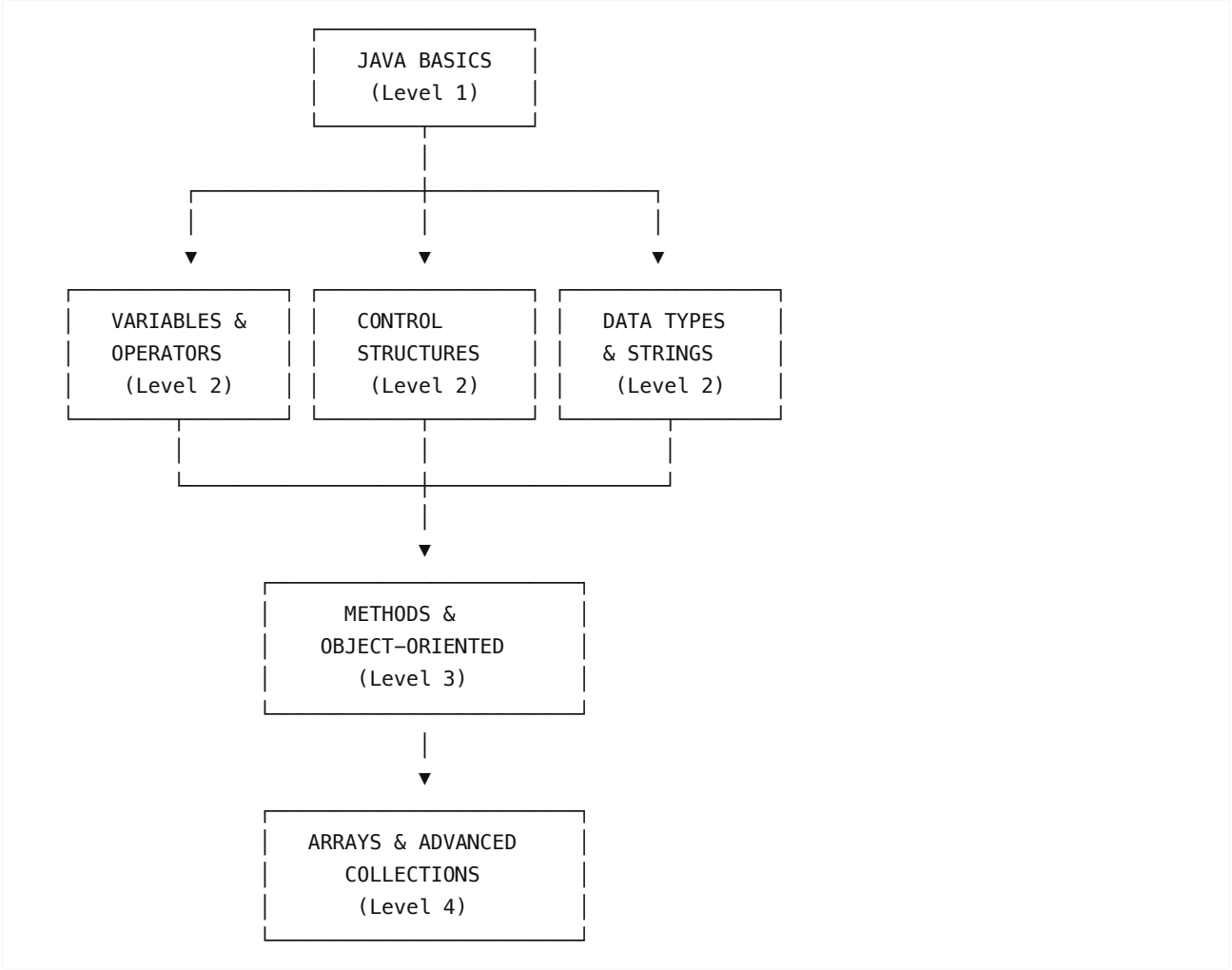
**Cognitive Profile System:**

The platform maintains comprehensive learner profiles tracking 30+ behavioral metrics:



**Skill Tree Architecture:**

Students progress through a hierarchical skill tree with prerequisite relationships:




**Progression Rewards:**

| Achievement | XP Reward |
|---|---|
| Question Solved (base) | +10 XP |
| No Hints Used | +5 XP |
| First-Try Success | +10 XP |
| Speed Bonus (<60 sec) | +5 XP |
| Daily Challenge | +25 XP |
| Daily Streak | +15 XP/day |
| Topic Mastery | +50 XP |
| Skill Node Unlock | +100 XP |

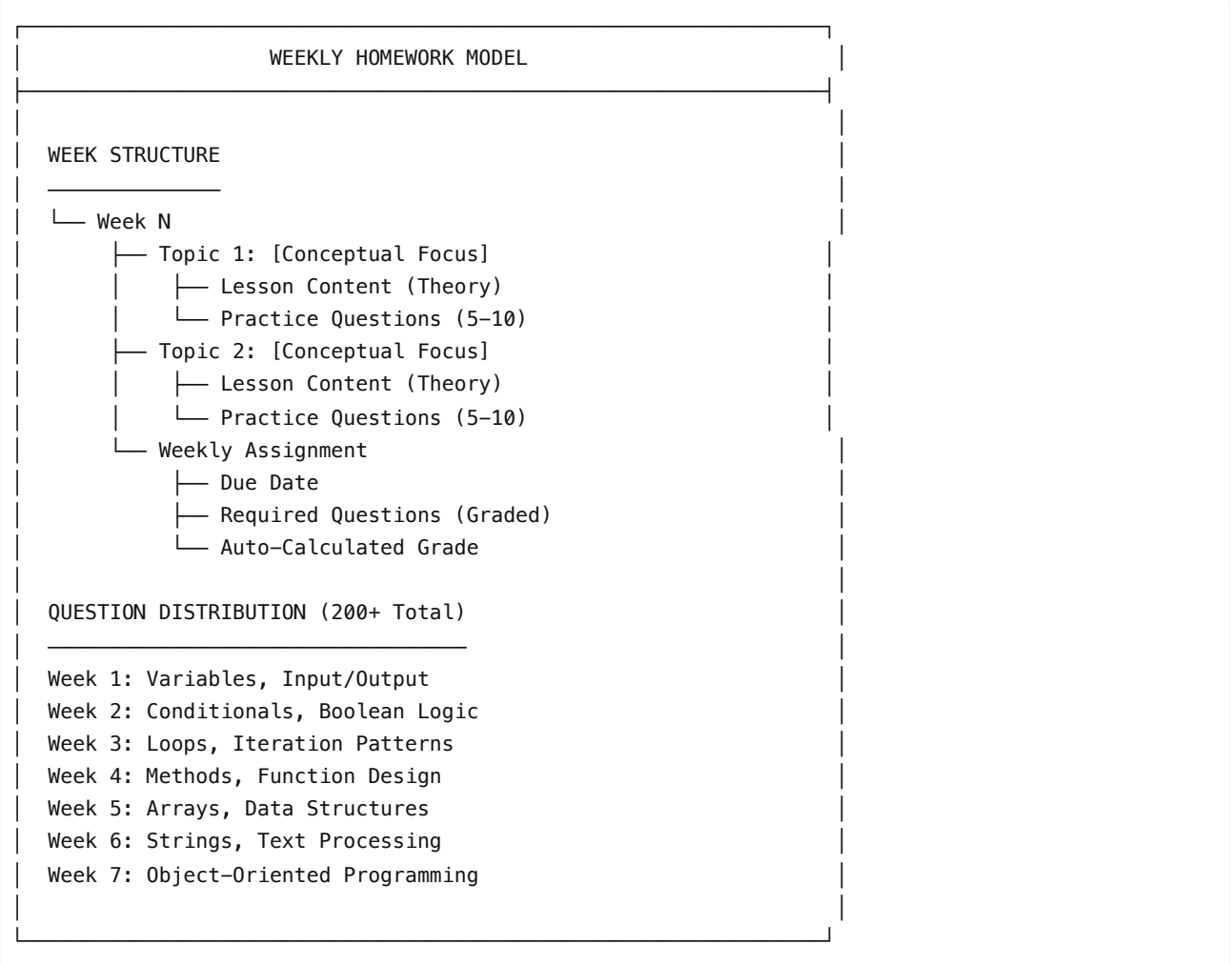
Level progression: `Level = (Total XP ÷ 250) + 1`

---

## 3. Assessment Model

### 3.1 Weekly Homework Structure

CodeTutor organizes curriculum into structured weekly assignments aligned with academic calendars:




**Assignment Configuration:**

- **Start Date**: When assignment becomes visible
- **Due Date**: Submission deadline
- **Questions**: Curated subset from topic question pool
- **Grade Weight**: Configurable per assignment
- **Late Policy**: Institution-configurable

### 3.2 Automatic Test-Case Grading

Every question includes a comprehensive test suite that serves as the definitive specification:

**Question Types:**

| Type | Description | Evaluation Method |
|---|---|---|
| `FULL_PROGRAM` | Complete Java application | Execute main(), compare output |
| `FUNCTION` | Implement specific method | Call method with test inputs |
| `FIX_BUG` | Debug provided broken code | Validate corrected behavior |
| `PREDICT_OUTPUT` | Analyze code, predict result | Compare predicted vs. actual |

**Test Case Structure:**

```
{
  "tests": [
    {
      "input": "5",
      "expectedOutput": "25",
      "description": "Square of positive integer",
      "hidden": false
    },
    {
      "input": "-3",
      "expectedOutput": "9",
      "description": "Square of negative integer",
      "hidden": false
    },
    {
      "input": "0",
      "expectedOutput": "0",
      "description": "Edge case: zero",
      "hidden": true
    }
  ]
}
```

**Grading Execution Flow:**

```
┌─────────────────────────────────────────────────────────┐
│                TEST-CASE GRADING PIPELINE               │
├─────────────────────────────────────────────────────────┤
│                                                         │
│  1. CODE SUBMISSION                                     │
│     └─► Receive student code                            │
│                                                         │
│  2. SECURITY SANITIZATION                               │
│     └─► Block dangerous patterns (reflection, I/O, threads)  │
│                                                         │
│  3. COMPILATION                                         │
│     └─► Compile Java code, capture errors               │
│                                                         │
│  4. TEST EXECUTION (per test case)                      │
│     └─► Provide input                                   │
│     └─► Execute in sandbox (25 sec timeout)             │
│     └─► Capture stdout/stderr                           │
│     └─► Compare against expectedOutput                  │
│                                                         │
│  5. RESULT AGGREGATION                                  │
│     └─► Calculate pass rate                             │
│     └─► Determine overall status                        │
│     └─► Record attempt with metrics                     │
│                                                         │
│  6. GRADE CALCULATION                                   │
│     └─► passed_tests / total_tests × question_points    │
│                                                         │
└─────────────────────────────────────────────────────────┘
```

### 3.3 Grade Calculation Logic

**Individual Question Score:**

```
Question Score = (Passed Tests / Total Tests) × Base Points

Example:
  - 8 of 10 tests pass
  - Base Points = 100
  - Score = 80/100
```

**Assignment Grade:**

```
Assignment Grade = Σ(Question Scores) / Σ(Maximum Possible)

Example:
  - Question 1: 80/100
  - Question 2: 100/100
  - Question 3: 60/100
  - Assignment Grade = 240/300 = 80%
```

**Course Grade:**

```
Course Grade = Σ(Assignment Grade × Weight) / Σ(Weights)

Example:
  - HW1 (10%): 80%  → 8.0
  - HW2 (15%): 90%  → 13.5
  - HW3 (20%): 75%  → 15.0
  - Course Grade = 36.5/45 = 81.1%
```

### 3.4 Fair Measurement Principles

**Logic Over Formatting:**

CodeTutor evaluates correctness, not style. A solution that produces correct output passes regardless of:

- Variable naming conventions
- Whitespace formatting
- Comment presence or absence
- Specific implementation approach

**What Is Graded:**

- Correctness of output for all test cases
- Handling of edge cases
- Algorithm termination (no infinite loops)
- Resource compliance (time and memory limits)

**What Is NOT Graded:**

- Code formatting or style
- Variable naming choices
- Comment quality
- Specific algorithm choice (if output is correct)

**Objective Truth Standard:**

The test suite serves as the objective specification:

- If tests pass, the solution is correct
- If tests fail, the solution needs revision

- No subjective interpretation required

---

# 4. Academic Governance

## 4.1 Admin Dashboard Purpose

The administrative interface provides institutional oversight without requiring code-level access:

```
┌─────────────────────────────────────────────────────────────────┐
│                    ADMIN DASHBOARD VIEWS                        │
├─────────────────────────────────────────────────────────────────┤
│                                                                 │
│  PLATFORM OVERVIEW                                              │
│  ─────────────────                                              │
│  • Total enrolled students                                      │
│  • Active students (last 7 days)                                │
│  • Total questions in curriculum                                │
│  • Overall platform pass rate                                   │
│  • Daily/weekly activity trends                                 │
│                                                                 │
│  STUDENT MANAGEMENT                                             │
│  ──────────────────                                             │
│  • Student roster with progress metrics                         │
│  • Individual student profiles                                  │
│  • Progress tracking by student                                 │
│  • At-risk student identification                               │
│  • Password reset capabilities                                  │
│                                                                 │
│  ASSIGNMENT MANAGEMENT                                          │
│  ─────────────────────                                          │
│  • Create/edit assignments                                      │
│  • Set due dates and weights                                    │
│  • View submission statistics                                   │
│  • Export grades (CSV/XLSX)                                     │
│  • Bulk operations                                              │
│                                                                 │
│  CONTENT MANAGEMENT                                             │
│  ──────────────────                                             │
│  • Question bank editor                                         │
│  • Topic/week organization                                      │
│  • Test case management                                         │
│  • Content versioning                                           │
│                                                                 │
└─────────────────────────────────────────────────────────────────┘
```

**Admin Role Hierarchy:**

| Role | Capabilities |
|---|---|
| `EDITOR` | Content creation and editing |
| `ADMIN` | Full content + student management |
| `SUPER_ADMIN` | All capabilities + system configuration |

## 4.2 Gradebook and Analytics Model

**Gradebook Export:**

```
┌─────────────────────────────────────────────────────────────┐
│                    GRADEBOOK DATA MODEL                     │
├─────────────────────────────────────────────────────────────┤
│                                                             │
│  Per-Student Record:                                        │
│  ──────────────────                                         │
│  • Student ID (External)                                    │
│  • Student Name                                             │
│  • Email                                                    │
│  • Per-Assignment Grades                                    │
│  • Assignment Submission Timestamps                         │
│  • Total Questions Attempted                                │
│  • Total Questions Passed                                   │
│  • Overall Pass Rate                                        │
│  • Current Level                                            │
│  • Total XP Earned                                          │
│  • Current Streak                                           │
│  • Last Active Date                                         │
│                                                             │
│  Export Formats:                                            │
│  ──────────────                                             │
│  • CSV (for spreadsheet import)                             │
│  • XLSX (formatted Excel)                                   │
│  • JSON (programmatic access)                               │
│                                                             │
└─────────────────────────────────────────────────────────────┘
```

**Analytics Dashboards:**

| Metric Category | Available Data |
|---|---|
| **Engagement** | Daily active users, session duration, login streaks |
| **Performance** | Pass rates by topic, question difficulty distribution |
| **Progress** | Completion rates, average level, XP distribution |
| **Content** | Hardest questions, most attempted, highest failure rates |
| **Time-Based** | Peak usage hours, submission patterns by day/week |

### 4.3 Risk Detection and Progress Monitoring

**At-Risk Student Detection:**

The platform automatically identifies students who may need intervention:

```
┌─────────────────────────────────────────────────────────────┐
│                    RISK DETECTION MODEL                     │
├─────────────────────────────────────────────────────────────┤
│                                                             │
│  BURNOUT RISK INDICATORS                                    │
│  ──────────────────────                                     │
│  • High retry rate (>5 attempts per question average)       │
│  • Decreasing pass rate trend                               │
│  • Session duration anomalies                               │
```

```
│  • Streak breaks after long streaks                                  │
│  • Low momentum score (<40)                                          │
│                                                                      │
│  DROPOUT RISK INDICATORS                                             │
│  ──────────────────────                                              │
│  • No activity in 7+ days                                            │
│  • Incomplete assignments near due date                              │
│  • Low engagement score                                              │
│  • Negative learning velocity                                        │
│                                                                      │
│  STRUGGLE INDICATORS                                                 │
│  ──────────────────                                                  │
│  • Topic pass rate <40%                                              │
│  • Recurring mistake patterns                                        │
│  • High hint usage                                                   │
│  • Low confidence index                                              │
│                                                                      │
│  AUTOMATED RESPONSES                                                 │
│  ──────────────────                                                  │
│  • Generate targeted practice missions                               │
│  • Surface easier "confidence boost" questions                       │
│  • Alert instructors via dashboard                                   │
│                                                                      │
└──────────────────────────────────────────────────────────────────────┘
```

**Progress Monitoring Reports:**

- Weekly progress summaries per class
- Individual student trajectory visualization
- Comparative cohort analysis
- Topic-level performance breakdowns
- Assignment completion tracking

---

# 5. Technical Architecture

## 5.1 Frontend Stack

**Core Technologies:**

| Component | Technology | Version |
|---|---|---|
| Framework | Next.js | 16.1.1 |
| UI Library | React | 19.2.3 |
| Language | TypeScript | 5.x |
| Styling | Tailwind CSS | 4.x |
| Animations | Framer Motion | 12.x |
| Code Editor | Monaco Editor | Latest |

**Component Architecture:**

```
┌──────────────────────────────────────────────────────────────────────┐
│                    FRONTEND COMPONENT TREE                           │
├──────────────────────────────────────────────────────────────────────┤
```

```
│                                                                    │
│  /src/components (100+ components)                                 │
│  │                                                                 │
│  ├── /practice                                                     │
│  │    ├── code-editor.tsx      (Monaco integration)                │
│  │    ├── question-panel.tsx   (Problem display)                   │
│  │    ├── results-panel.tsx    (Test output)                       │
│  │    ├── action-bar.tsx       (Run/Submit buttons)                │
│  │    └── execution-overlay.tsx (Live feedback)                    │
│  │                                                                 │
│  ├── /progress                                                     │
│  │    ├── progress-header.tsx  (XP/Level display)                  │
│  │    ├── skill-tree.tsx       (Visual progression)                │
│  │    ├── streak-display.tsx   (Daily streak)                      │
│  │    └── achievements.tsx     (Badge gallery)                     │
│  │                                                                 │
│  ├── /admin                                                        │
│  │    ├── dashboard.tsx        (Analytics overview)                │
│  │    ├── student-table.tsx    (Student management)                │
│  │    ├── gradebook.tsx        (Grade management)                  │
│  │    └── question-editor.tsx  (Content management)                │
│  │                                                                 │
│  └── /layout                                                       │
│       ├── dashboard-shell.tsx  (App wrapper)                       │
│       ├── sidebar.tsx          (Navigation)                        │
│       └── mobile-nav.tsx       (Responsive nav)                    │
│                                                                    │
└────────────────────────────────────────────────────────────────────┘
```

**Key Frontend Features:**

- Server-side rendering for performance
- React Server Components for reduced JavaScript
- Real-time code execution feedback
- Responsive design (mobile to desktop)
- Dark/light mode support
- Accessible components (Radix UI primitives)

### 5.2 Backend and API Architecture

**API Structure:**

```
┌────────────────────────────────────────────────────────────────────┐
│                    API ROUTE ORGANIZATION                          │
├────────────────────────────────────────────────────────────────────┤
│                                                                    │
│  /api (72 routes)                                                  │
│  │                                                                 │
│  ├── /auth                                                         │
│  │    ├── register        (User registration)                     │
│  │    ├── login           (Credentials auth)                      │
│  │    └── [...nextauth]   (NextAuth handlers)                     │
│  │                                                                 │
│  ├── /execute             (Code execution endpoint)                │
│  │    └── POST: Submit code for evaluation                         │
│  │                                                                 │
│  ├── /questions                                                    │
```

```
│   │    ├── [id]            (Get question by ID)                │
│   │    └── next-question   (Adaptive selection)                │
│   │                                                            │
│   ├── /progress                                                │
│   │    ├── stats           (User statistics)                   │
│   │    ├── streaks         (Streak data)                       │
│   │    └── achievements    (Badge progress)                    │
│   │                                                            │
│   ├── /admin                                                   │
│   │    ├── stats           (Platform statistics)               │
│   │    ├── analytics       (Detailed analytics)                │
│   │    ├── students        (Student management)                │
│   │    ├── assignments     (Homework CRUD)                     │
│   │    └── curriculum      (Content management)                │
│   │                                                            │
│   ├── /assignments                                             │
│   │    ├── [id]            (Assignment details)                │
│   │    └── submit          (Submit assignment)                 │
│   │                                                            │
│   └── /webhooks                                                │
│        └── stripe          (Payment processing)                │
│                                                                │
└────────────────────────────────────────────────────────────────┘
```

**Service Layer:**

```
┌────────────────────────────────────────────────────────────────┐
│                    BACKEND SERVICES                            │
├────────────────────────────────────────────────────────────────┤
│                                                                │
│  /src/lib                                                      │
│  │                                                             │
│  ├── /auth                                                     │
│  │    └── auth.ts          (NextAuth configuration)           │
│  │                                                             │
│  ├── /db                                                       │
│  │    └── prisma.ts        (Database client singleton)        │
│  │                                                             │
│  ├── executor.ts           (Code execution service)           │
│  │    • Health checking                                        │
│  │    • Sandbox invocation                                     │
│  │    • JDoodle fallback                                       │
│  │    • Rate limiting                                          │
│  │                                                             │
│  ├── /progression                                              │
│  │    ├── xp-calculator.ts (Point calculation)                │
│  │    ├── level-service.ts (Level management)                 │
│  │    └── streak-service.ts (Streak tracking)                 │
│  │                                                             │
│  ├── /mentor                                                   │
│  │    ├── cognitive-profiler.ts (Profile updates)             │
│  │    ├── mistake-engine.ts (Error classification)            │
│  │    └── mission-generator.ts (Adaptive missions)            │
│  │                                                             │
│  └── /security                                                 │
│       ├── rate-limiter.ts  (Request throttling)               │
```

```
│       ├── sanitizer.ts     (Code validation)                  │
│       └── audit-logger.ts  (Event logging)                    │
│                                                               │
└───────────────────────────────────────────────────────────────┘
```

### 5.3 Database Model Summary

**Entity-Relationship Overview:**

```
┌─────────────────────────────────────────────────────────────────┐
│                    DATABASE SCHEMA (PostgreSQL)                 │
├─────────────────────────────────────────────────────────────────┤
│                                                                 │
│  USER DOMAIN                                                    │
│  ──────────                                                     │
│  User ──┬── UserProgress (XP, level, streaks)                   │
│         ├── UserEnrollment ─── Course                           │
│         ├── UserTopicStats ─── Topic                            │
│         ├── CognitiveProfile (behavioral metrics)               │
│         └── ProProfile (premium features)                       │
│                                                                 │
│  CURRICULUM DOMAIN                                              │
│  ─────────────────                                              │
│  Course ─── Week ─── Topic ──┬── Lesson                        │
│                              └── Question ─── Tests (JSON)      │
│                                                                 │
│  ASSESSMENT DOMAIN                                              │
│  ─────────────────                                              │
│  Assignment ──┬── AssignmentQuestion ─── Question               │
│               └── AssignmentSubmission ─── User                 │
│                                                                 │
│  ATTEMPT DOMAIN                                                 │
│  ──────────────                                                 │
│  Attempt ──┬── User                                             │
│            ├── Question                                         │
│            ├── Status (PASS/FAIL/ERROR/TIMEOUT)                 │
│            ├── ExecutionMetrics (time, memory)                  │
│            └── TestResults (JSON)                               │
│                                                                 │
│  GAMIFICATION DOMAIN                                            │
│  ────────────────────                                           │
│  PointsLedger ─── User (XP transactions)                        │
│  Achievement ─── UserAchievement ─── User                       │
│  SkillNode ─── SkillUnlock ─── User                             │
│  League ─── LeaguePlacement ─── User                            │
│                                                                 │
│  MISTAKE TRACKING                                               │
│  ────────────────                                               │
│  MistakeLog ──┬── User                                          │
│               ├── Question                                      │
│               ├── Type (SYNTAX/LOGIC/TIMEOUT/etc.)              │
│               └── Severity (1-5)                                │
│                                                                 │
└─────────────────────────────────────────────────────────────────┘
```

**Key Tables:**

| Table | Purpose | Key Fields |
|---|---|---|
| `User` | User accounts | id, email, password (hashed), role |
| `Question` | Problem definitions | title, prompt, starterCode, tests, difficulty |
| `Attempt` | Submission records | userId, questionId, code, status, testResults |
| `Assignment` | Homework collections | title, dueDate, courseId |
| `CognitiveProfile` | Learning analytics | accuracyRate, retryRate, burnoutRisk |
| `MistakeLog` | Error patterns | mistakeType, severity, isRecurring |

### 5.4 Security Model

**Multi-Layer Security Architecture:**

```
┌─────────────────────────────────────────────────────────────────┐
│                    SECURITY ARCHITECTURE                        │
├─────────────────────────────────────────────────────────────────┤
│                                                                 │
│  LAYER 1: AUTHENTICATION                                        │
│  ──────────────────────                                         │
│  • NextAuth.js session management                               │
│  • bcrypt password hashing (cost factor 10)                     │
│  • JWT-based sessions                                           │
│  • OAuth integration (Google, GitHub)                           │
│                                                                 │
│  LAYER 2: AUTHORIZATION                                         │
│  ──────────────────────                                         │
│  • Role-based access control (USER, ADMIN)                      │
│  • Route-level middleware protection                            │
│  • Resource ownership validation                                │
│                                                                 │
│  LAYER 3: CODE SANITIZATION                                     │
│  ──────────────────────────                                     │
│  Blocked Patterns:                                              │
│  • Runtime.exec(), ProcessBuilder                               │
│  • System.exit()                                                │
│  • Reflection APIs (Class.forName, getDeclaredMethod)           │
│  • File/Network I/O (FileInputStream, Socket, URL)              │
│  • Thread.sleep() > 10 seconds                                  │
│  • GUI frameworks (javax.swing, java.awt)                       │
│                                                                 │
│  LAYER 4: EXECUTION ISOLATION                                   │
│  ────────────────────────────                                   │
│  • Docker containerization                                      │
│  • Non-root user execution                                      │
│  • Network isolation                                            │
│  • Resource limits (CPU, memory, time)                          │
│  • 25-second hard timeout                                       │
│                                                                 │
│  LAYER 5: RATE LIMITING                                         │
│  ──────────────────────                                         │
│  • Code execution: 20 requests/minute                           │
│  • Hint requests: 10 requests/minute                            │
│  • General API: 100 requests/minute                             │
```

```
│  • Redis-backed sliding window                                          │
│                                                                         │
│  LAYER 6: SECURITY HEADERS                                              │
│  ─────────────────────────                                              │
│  • X-Frame-Options: DENY                                                │
│  • X-Content-Type-Options: nosniff                                      │
│  • X-XSS-Protection: 1; mode=block                                      │
│  • Strict Referrer-Policy                                               │
│                                                                         │
│  LAYER 7: AUDIT & MONITORING                                            │
│  ──────────────────────────                                             │
│  • 30+ event types logged                                               │
│  • Sentry error tracking                                                │
│  • Security alert triggers                                              │
│  • IP and session tracking                                              │
│                                                                         │
└─────────────────────────────────────────────────────────────────────────┘
```

### 5.5 Execution Sandbox (Docker Isolation)

**Sandbox Architecture:**

```
┌─────────────────────────────────────────────────────────────────────────┐
│                    DOCKER SANDBOX DESIGN                                │
├─────────────────────────────────────────────────────────────────────────┤
│                                                                         │
│  BASE IMAGE: Alpine Linux + Eclipse Temurin JDK 17                      │
│                                                                         │
│  SECURITY CONFIGURATION:                                                │
│  ──────────────────────                                                 │
│  • Non-root user "sandbox" created                                      │
│  • Working directory: /sandbox                                          │
│  • Network utilities removed                                            │
│  • Read-only filesystem where possible                                  │
│                                                                         │
│  RESOURCE LIMITS:                                                       │
│  ────────────────                                                       │
│  • CPU: Limited to single core                                          │
│  • Memory: 256MB maximum                                                │
│  • Execution timeout: 25 seconds                                        │
│  • Disk: Temporary only, no persistence                                 │
│                                                                         │
│  EXECUTION FLOW:                                                        │
│  ──────────────                                                         │
│  1. Receive code from executor service                                  │
│  2. Write to temporary .java file                                       │
│  3. Compile with javac                                                  │
│  4. Execute with timeout wrapper                                        │
│  5. Capture stdout/stderr                                               │
│  6. Return results, destroy container                                   │
│                                                                         │
│  FALLBACK: JDoodle API                                                  │
│  ─────────────────────                                                  │
│  • Activated when Docker unavailable (Vercel deployment)                │
│  • Same evaluation logic, external execution                            │
```

```
│                                                                  │
└──────────────────────────────────────────────────────────────────┘
```

---

# 6. Data & Privacy

## 6.1 User Data Separation

**Data Isolation Principles:**

```
┌──────────────────────────────────────────────────────────────────┐
│                    DATA ISOLATION MODEL                          │
├──────────────────────────────────────────────────────────────────┤
│                                                                  │
│  STUDENT DATA BOUNDARIES                                         │
│  ──────────────────────                                          │
│                                                                  │
│  Each student can access ONLY:                                   │
│  • Their own profile and preferences                             │
│  • Their own submission history                                  │
│  • Their own progress and achievements                           │
│  • Their own cognitive profile                                   │
│  • Their own grades and feedback                                 │
│                                                                  │
│  Students CANNOT access:                                         │
│  • Other students' code submissions                              │
│  • Other students' progress or grades                            │
│  • Other students' profiles                                      │
│  • Administrative analytics                                      │
│  • Question test case implementation (hidden cases)              │
│                                                                  │
│  AGGREGATED DATA (No PII):                                       │
│  ─────────────────────────                                       │
│  • Leaderboard shows: rank, username, XP only                    │
│  • Class statistics: anonymized pass rates                       │
│  • Topic difficulty: aggregate, not individual                   │
│                                                                  │
└──────────────────────────────────────────────────────────────────┘
```

**Access Control Matrix:**

| Data Type | Student | Instructor | Admin |
|---|---|---|---|
| Own submissions | Read | Read | Read |
| Other submissions | None | Read (own class) | Read (all) |
| Own grades | Read | Read | Read |
| Class grades | None | Read | Read |
| Question content | Read | Read/Write | Full |
| System analytics | None | Limited | Full |

## 6.2 GDPR-Safe Architecture Principles

**Privacy by Design Implementation:**

```
┌─────────────────────────────────────────────────────────────┐
│                   GDPR COMPLIANCE FRAMEWORK                 │
├─────────────────────────────────────────────────────────────┤
│                                                             │
│  DATA MINIMIZATION                                          │
│  ─────────────────                                          │
│  • Collect only data necessary for educational function     │
│  • No tracking beyond learning analytics                    │
│  • No third-party data sharing for marketing                │
│                                                             │
│  PURPOSE LIMITATION                                         │
│  ──────────────────                                         │
│  • Data used solely for:                                    │
│    - Learning progress tracking                             │
│    - Academic assessment                                    │
│    - System improvement                                     │
│  • No secondary commercial use                              │
│                                                             │
│  STORAGE LIMITATION                                         │
│  ──────────────────                                         │
│  • Attempt history: Retained for course duration + 1 year   │
│  • Session data: Temporary, cleared on logout               │
│  • Analytics: Aggregated, individual data can be purged     │
│                                                             │
│  DATA SUBJECT RIGHTS                                        │
│  ──────────────────                                         │
│  • Access: Users can view all their stored data             │
│  • Rectification: Profile data editable                     │
│  • Erasure: Account deletion removes all personal data      │
│  • Portability: Export functionality available              │
│                                                             │
│  SECURITY MEASURES                                          │
│  ─────────────────                                          │
│  • Encryption at rest (database)                            │
│  • Encryption in transit (TLS 1.3)                          │
│  • Password hashing (bcrypt)                                │
│  • Access logging and audit trails                          │
│                                                             │
└─────────────────────────────────────────────────────────────┘
```

### 6.3 No Code Sharing Between Students

**Academic Integrity Enforcement:**

```
┌─────────────────────────────────────────────────────────────┐
│                   CODE ISOLATION GUARANTEE                  │
├─────────────────────────────────────────────────────────────┤
│                                                             │
│  SUBMISSION ISOLATION                                       │
│  ────────────────────                                       │
│  • Each submission tied to single user ID                   │
│  • No API endpoint exposes other users' code                │
│  • Database queries filtered by authenticated user          │
│                                                             │
│  DRAFT PROTECTION                                           │
│  ────────────────                                           │
```

```
│  • Auto-saved drafts are user-private                          │
│  • No shared editing or collaboration on assignments           │
│  • Each student has independent workspace                      │
│                                                                │
│  EXECUTION ISOLATION                                           │
│  ──────────────────                                            │
│  • Each execution in fresh container                           │
│  • No persistence between executions                           │
│  • No access to other users' execution history                 │
│                                                                │
│  INSTRUCTOR ACCESS                                             │
│  ─────────────────                                             │
│  • Instructors can view submissions for grading                │
│  • View is individual, not comparative by default              │
│  • Plagiarism detection would be separate system integration   │
│                                                                │
└────────────────────────────────────────────────────────────────┘
```

### 6.4 Academic Integrity Enforcement

**Integrity Measures:**

| Measure | Implementation |
|---|---|
| Individual Assessment | Each student's progress is independent |
| No Copy Mechanism | No clipboard sharing or export of others' code |
| Session Isolation | Authentication required for all submissions |
| Timestamp Logging | Every submission recorded with precise timing |
| IP Tracking | Submission origin logged for anomaly detection |
| Audit Trail | Complete history of all actions retained |

---

## 7. Scalability & Institution Readiness

### 7.1 Multi-Course Readiness

**Course Architecture:**

```
┌────────────────────────────────────────────────────────────────┐
│                    MULTI-COURSE ARCHITECTURE                   │
├────────────────────────────────────────────────────────────────┤
│                                                                │
│  INSTITUTION                                                   │
│  └── Course: Introduction to Java                              │
│       ├── Semester: Fall 2026                                  │
│       ├── Instructor: Prof. Smith                              │
│       ├── Weeks: 1-14                                          │
│       └── Assignments: 12                                      │
│                                                                │
│  └── Course: Data Structures                                   │
│       ├── Semester: Fall 2026                                  │
│       ├── Instructor: Prof. Johnson                            │
│       ├── Weeks: 1-14                                          │
```

```
│       └── Assignments: 10                                                │
│                                                                          │
│   └── Course: Algorithms                                                 │
│       ├── Semester: Spring 2027                                          │
│       ├── Instructor: Prof. Lee                                          │
│       ├── Weeks: 1-14                                                    │
│       └── Assignments: 14                                                │
│                                                                          │
│   CONTENT SHARING:                                                       │
│   ─────────────────                                                      │
│   • Question bank shared across courses                                  │
│   • Topics can be reused with different assignments                      │
│   • Content versioning allows course-specific variations                 │
│                                                                          │
│   ISOLATION:                                                             │
│   ──────────                                                             │
│   • Student enrollments are course-specific                              │
│   • Grades are per-course                                                │
│   • Progress tracking can be global or course-scoped                     │
│                                                                          │
└──────────────────────────────────────────────────────────────────────────┘
```

### 7.2 Multi-Class Readiness

**Class Section Support:**

```
┌──────────────────────────────────────────────────────────────────────────┐
│                    MULTI-CLASS (SECTION) SUPPORT                         │
├──────────────────────────────────────────────────────────────────────────┤
│                                                                          │
│   Course: Introduction to Java                                           │
│   │                                                                      │
│   ├── Section A (Morning)                                                │
│   │   ├── TA: Alice                                                      │
│   │   ├── Students: 30                                                   │
│   │   └── Office Hours: MWF 9am                                          │
│   │                                                                      │
│   ├── Section B (Afternoon)                                              │
│   │   ├── TA: Bob                                                        │
│   │   ├── Students: 35                                                   │
│   │   └── Office Hours: TTh 2pm                                          │
│   │                                                                      │
│   └── Section C (Evening)                                                │
│       ├── TA: Charlie                                                    │
│       ├── Students: 25                                                   │
│       └── Office Hours: MW 6pm                                           │
│                                                                          │
│   ADMINISTRATIVE VIEWS:                                                  │
│   ──────────────────────                                                 │
│   • Instructor: All sections                                             │
│   • TA: Assigned section only                                            │
│   • Analytics: Per-section and aggregated                                │
│                                                                          │
│   ASSIGNMENT MANAGEMENT:                                                 │
│   ──────────────────────                                                 │
│   • Same assignments across sections                                     │
```

```
│  • Section-specific due date adjustments possible                │
│  • Unified grading standards                                     │
│                                                                  │
└──────────────────────────────────────────────────────────────────┘
```

**Scalability Characteristics:**

| Dimension | Current Capacity | Scaling Path |
|---|---|---|
| Concurrent Users | 1,000+ | Horizontal pod scaling |
| Questions | 200+ | Database indexed |
| Submissions/Hour | 10,000+ | Queue-based processing |
| Storage | PostgreSQL | Managed database scaling |
| Execution | Docker swarm | Kubernetes orchestration |

### 7.3 Future AI Tutor Expansion Readiness

**AI Integration Architecture:**

```
┌──────────────────────────────────────────────────────────────────┐
│                    AI TUTOR EXPANSION READINESS                  │
├──────────────────────────────────────────────────────────────────┤
│                                                                  │
│  CURRENT FOUNDATION:                                             │
│  ────────────────────                                            │
│  • Cognitive Profile: 30+ metrics per student                    │
│  • Mistake Classification: 8 error categories                    │
│  • Learning Patterns: Session, peak hours, velocity              │
│  • Weakness Maps: Per-topic struggle identification              │
│                                                                  │
│  AI TUTOR INTEGRATION POINTS:                                    │
│  ──────────────────────────────                                  │
│                                                                  │
│  1. INTELLIGENT HINT SYSTEM                                      │
│     • Current: Static hint sequence                              │
│     • AI Enhancement: Context-aware hints based on               │
│       specific student mistakes                                  │
│                                                                  │
│  2. PERSONALIZED EXPLANATIONS                                    │
│     • Current: Generic lesson content                            │
│     • AI Enhancement: Explanations tailored to                   │
│       student's demonstrated understanding level                 │
│                                                                  │
│  3. ERROR ANALYSIS                                               │
│     • Current: Pattern-based classification                      │
│     • AI Enhancement: Natural language explanation               │
│       of what went wrong and how to fix it                       │
│                                                                  │
│  4. ADAPTIVE QUESTIONING                                         │
│     • Current: Algorithm-based selection                         │
│     • AI Enhancement: Dynamic question generation                │
│       targeting specific skill gaps                              │
│                                                                  │
│  5. PROGRESS COACHING                                            │
```

```
│     • Current: Automated missions                                    │
│     • AI Enhancement: Conversational progress                        │
│       reviews and motivational support                               │
│                                                                      │
│  DATA INFRASTRUCTURE READY:                                          │
│  ──────────────────────────                                          │
│  • All student interactions logged                                   │
│  • Cognitive profiles provide training data                          │
│  • Mistake logs enable error pattern learning                        │
│  • Topic performance enables targeted intervention                   │
│                                                                      │
└──────────────────────────────────────────────────────────────────────┘
```

---

## 8. Why This Is Institutional Infrastructure

### 8.1 Academic Operating System Characteristics

CodeTutor transcends the category of "educational tool" to function as institutional infrastructure:

**Operating System Analogy:**

```
┌──────────────────────────────────────────────────────────────────────┐
│        TRADITIONAL TOOLS vs. ACADEMIC INFRASTRUCTURE                 │
├──────────────────────────────────────────────────────────────────────┤
│                                                                      │
│  TRADITIONAL LMS (Tool)          │  CODETUTOR (Infrastructure)       │
│  ──────────────────────          │  ──────────────────────────       │
│  • Content delivery              │  • Complete learning process      │
│  • Assignment collection         │  • Execution + evaluation         │
│  • Grade recording               │  • Automatic measurement          │
│  • Static interaction            │  • Adaptive personalization       │
│  • Manual grading required       │  • Zero grading labor             │
│  • End-of-term feedback          │  • Continuous feedback            │
│                                                                      │
│  Like the difference between:                                        │
│  • A filing cabinet (tool)                                           │
│  • An operating system (infrastructure)                              │
│                                                                      │
└──────────────────────────────────────────────────────────────────────┘
```

**Infrastructure Characteristics:**

| Characteristic | Implementation |
|---|---|
| **Abstraction** | Hides complexity of code execution, grading, analytics |
| **Resource Management** | Manages compute resources for thousands of executions |
| **Process Scheduling** | Queues and processes submissions asynchronously |
| **Security Kernel** | Isolates untrusted code execution |
| **User Management** | Handles authentication, authorization, sessions |
| **Data Management** | Stores, retrieves, analyzes learning data |
| **System Services** | Provides APIs for integration with other systems |

### 8.2 Institutional Value Proposition

**For Students:**

- Immediate feedback accelerates learning
- Unlimited practice without grade penalty
- Clear, objective success criteria
- Personalized difficulty progression
- Gamified motivation system

**For Instructors:**

- Zero manual grading labor
- Real-time class progress visibility
- Early identification of struggling students
- Data-driven curriculum improvement
- Scalable to any class size

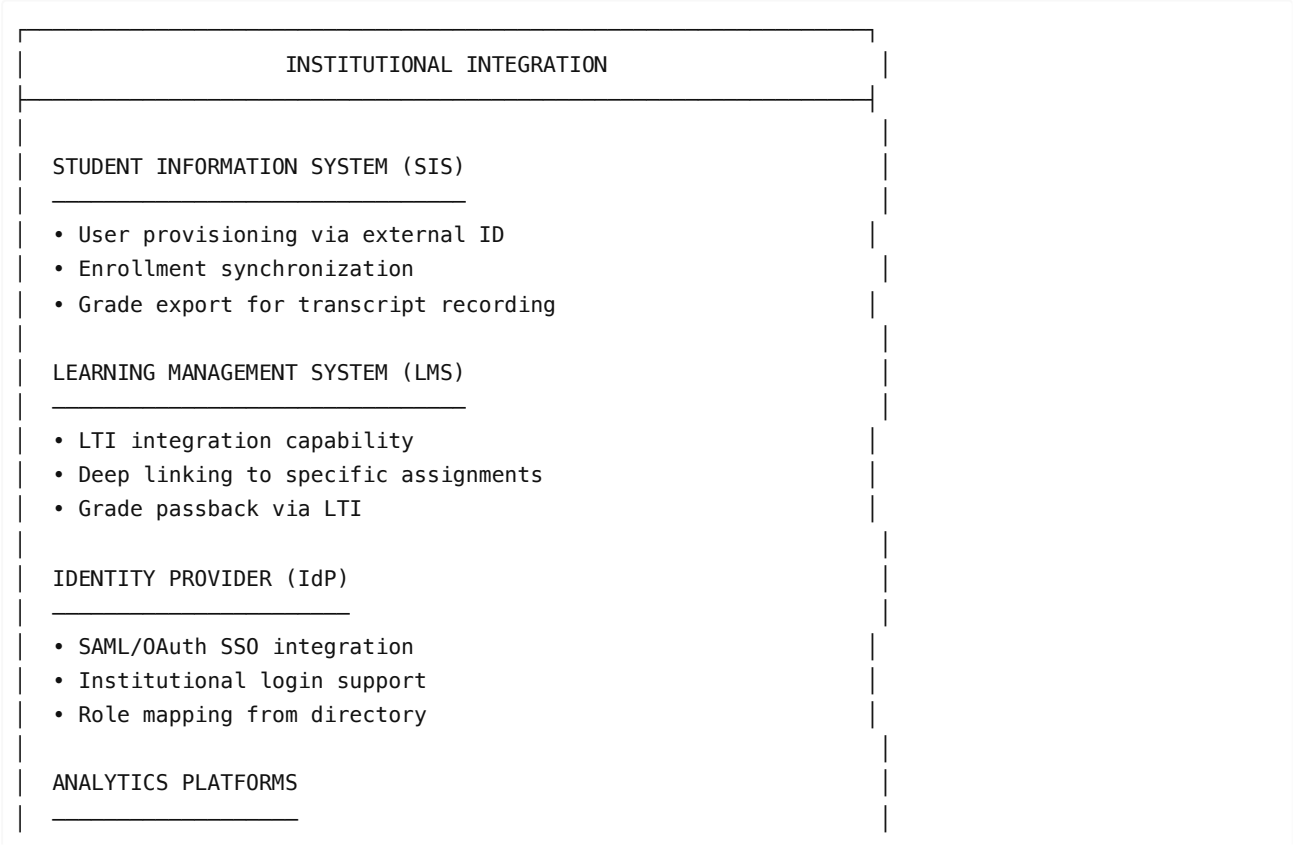
**For Institutions:**

- Standardized assessment across sections
- Objective, auditable grading
- GDPR-compliant data handling
- Reduced teaching assistant requirements
- Quality metrics for accreditation

**For the Discipline:**

- Verified skill measurement
- Resistance to AI-assisted cheating
- Authentic assessment of programming ability
- Continuous assessment vs. high-stakes exams

### 8.3 Infrastructure Integration Points

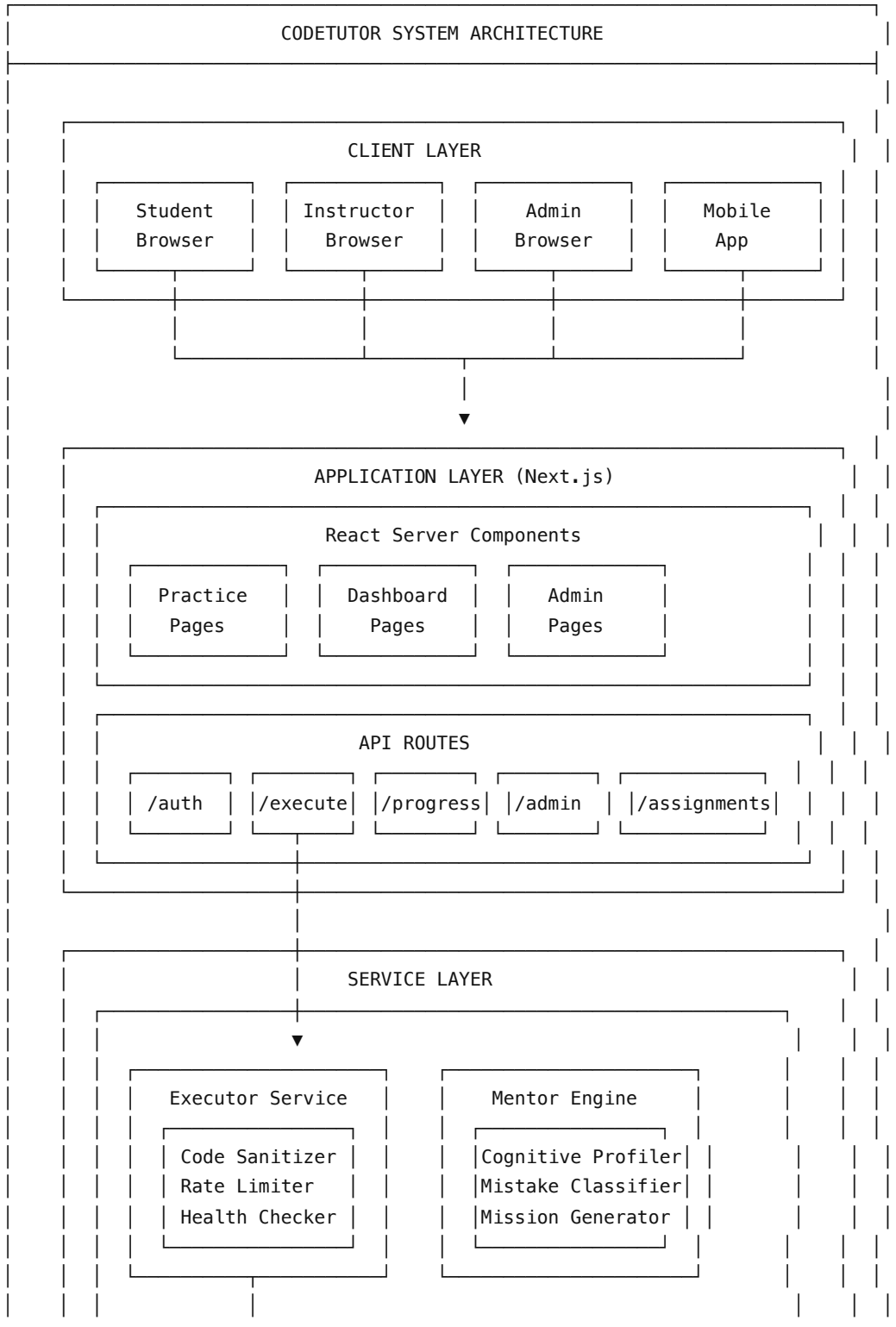


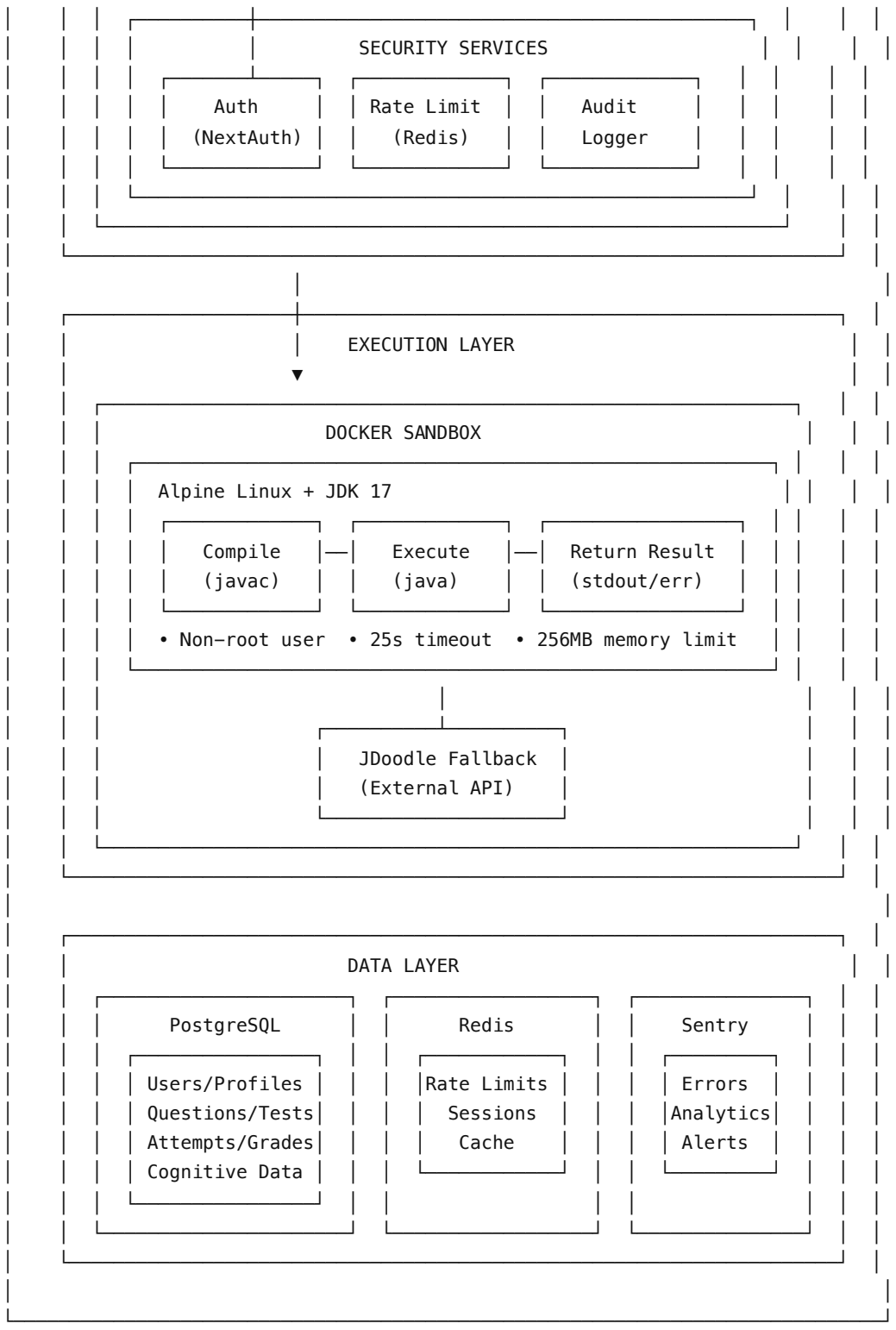
```
│  • Data export for institutional analytics                          │
│  • Learning analytics dashboards                                    │
│  • Predictive modeling data feeds                                   │
│                                                                     │
└─────────────────────────────────────────────────────────────────────┘
```

---

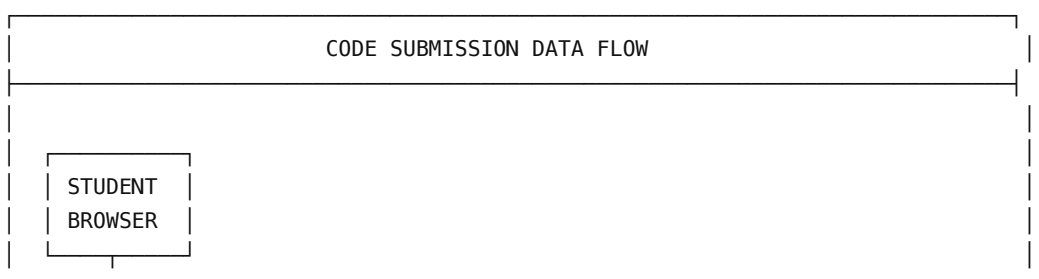
## 9. Architecture Diagrams

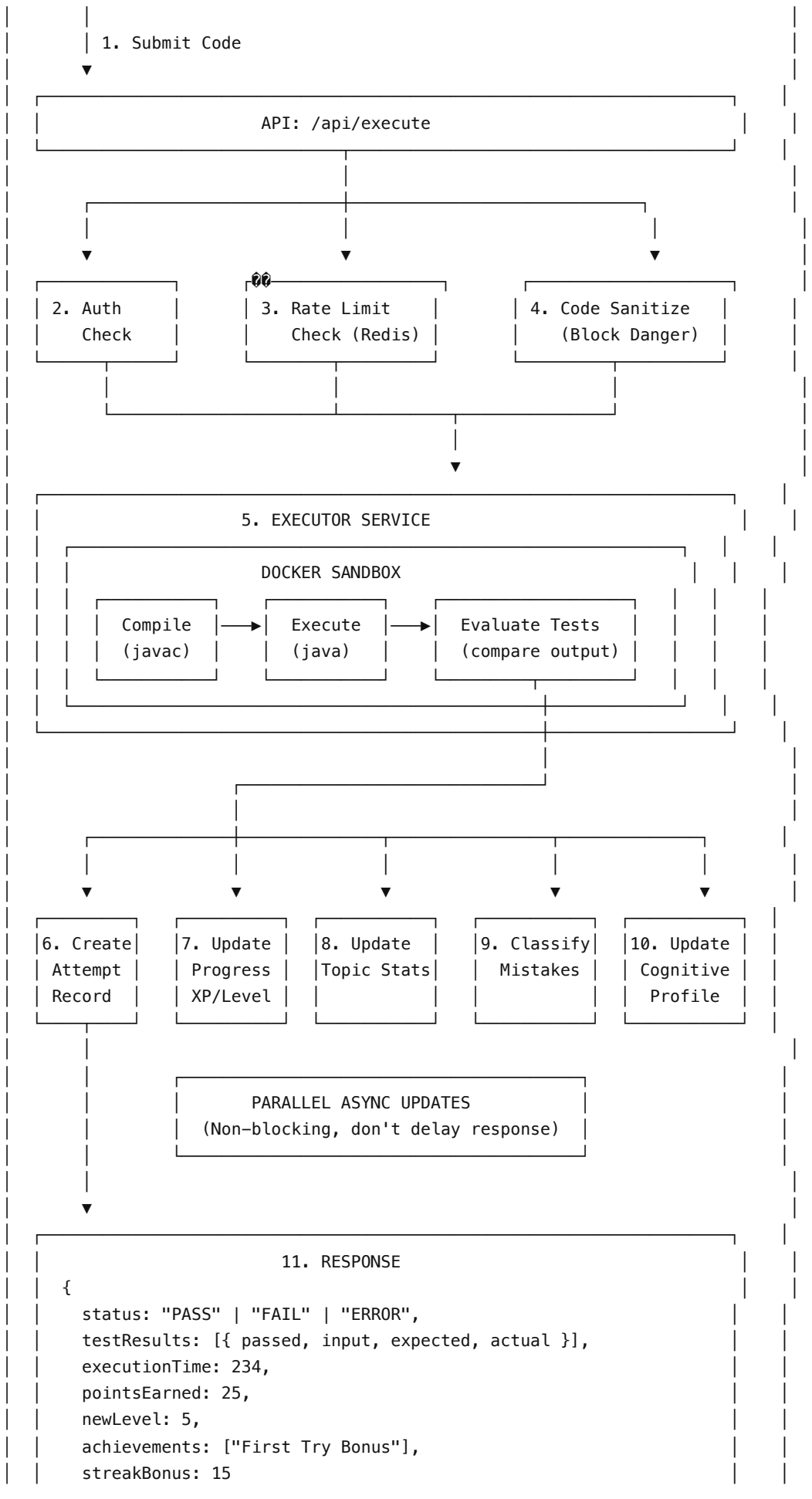
### 9.1 System Architecture Diagram

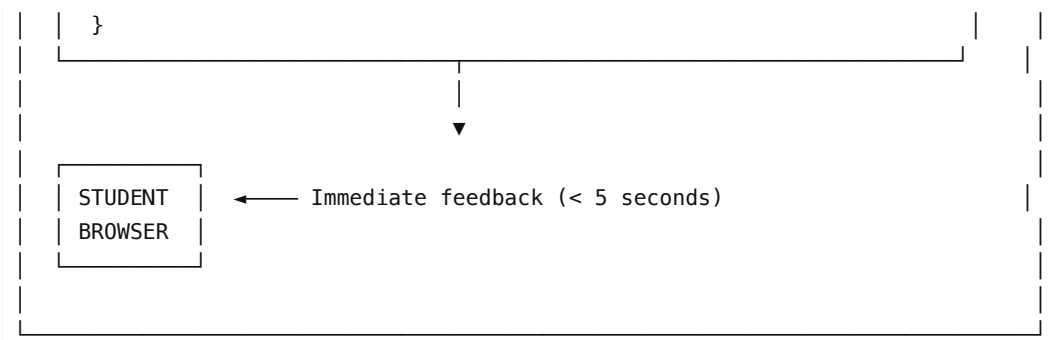




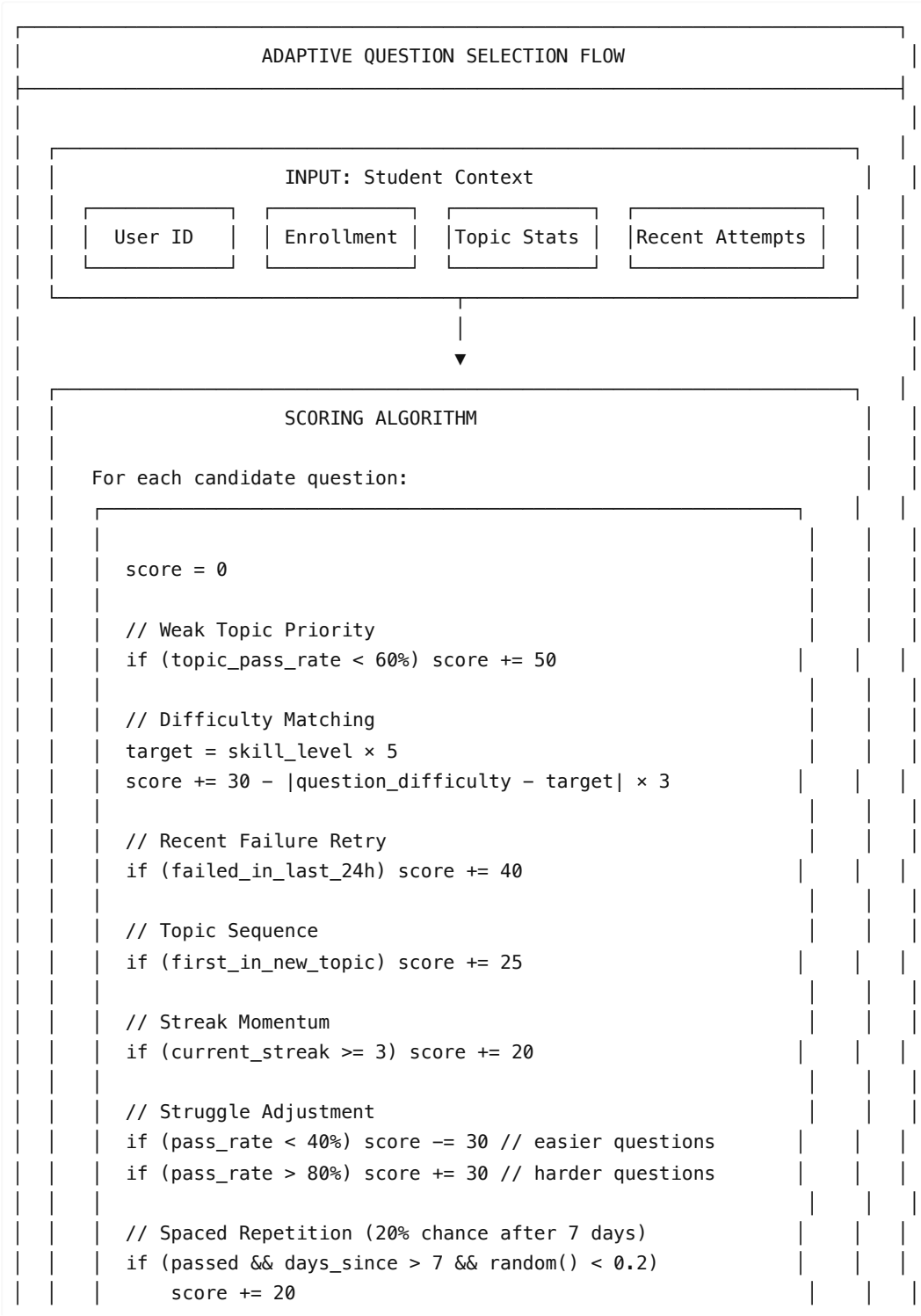

### 9.2 Data Flow Diagram

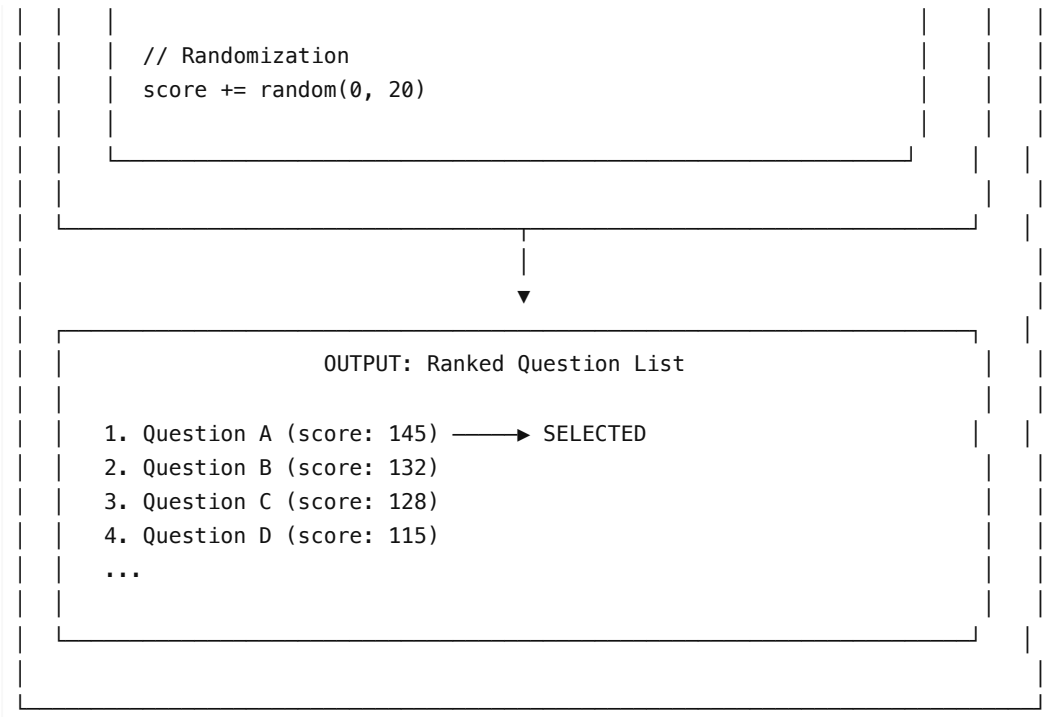



1. Submit Code
API: /api/execute
2. Auth Check
3. Rate Limit Check (Redis)
4. Code Sanitize (Block Danger)
5. EXECUTOR SERVICE
DOCKER SANDBOX
Compile (javac)
Execute (java)
Evaluate Tests (compare output)
6. Create Attempt Record
7. Update Progress XP/Level
8. Update Topic Stats
9. Classify Mistakes
10. Update Cognitive Profile
PARALLEL ASYNC UPDATES
(Non-blocking, don't delay response)
11. RESPONSE
{
status: "PASS" | "FAIL" | "ERROR",
testResults: [{ passed, input, expected, actual }],
executionTime: 234,
pointsEarned: 25,
newLevel: 5,
achievements: ["First Try Bonus"],
streakBonus: 15




### 9.3 Adaptive Learning Flow






---

## Appendix A: Glossary

| Term | Definition |
|---|---|
| **Attempt** | A single code submission for a question |
| **Cognitive Profile** | Comprehensive learning analytics for a student |
| **Hidden Test** | Test case not shown to student, prevents hardcoding |
| **Hint** | Progressive assistance available for point cost |
| **Mission** | Targeted practice generated by mentor engine |
| **Pass Rate** | Percentage of questions successfully completed |
| **Skill Node** | Unit in hierarchical skill progression tree |
| **Streak** | Consecutive days of activity |
| **Topic** | Thematic grouping of related questions |
| **XP** | Experience points earned through practice |

---

## Appendix B: Contact & Support

For institutional inquiries regarding CodeTutor deployment, integration, or customization, please contact the development team through official institutional channels.

---

**Document Version:** 1.0 **Last Updated:** January 2026 **Classification:** Academic Infrastructure Documentation
**Distribution:** Institutional Stakeholders

---

*CodeTutor — Transforming Computer Science Education Through Adaptive, Transparent, and Scalable Assessment Infrastructure*