

# CodeTutor — Academic Learning Infrastructure

## Overview

---

Version 1.0 | January 2026

Institutional Documentation for Academic Integration

---

### Table of Contents

---

1. [System Purpose](#)
2. [Pedagogical Model](#)
3. [Assessment Model](#)
4. [Academic Governance](#)
5. [Technical Architecture](#)
6. [Data & Privacy](#)
7. [Scalability & Institution Readiness](#)
8. [Why This Is Institutional Infrastructure](#)
9. [Architecture Diagrams](#)

# 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
<b>Feedback Latency</b>	Days to weeks	Hours to days	Seconds
<b>Grading Consistency</b>	Variable by grader	Dependent on scripts	Deterministic
<b>Student Iteration</b>	Limited attempts	Manual resubmission	Unlimited practice
<b>Progress Visibility</b>	End-of-term grades	Commit history only	Real-time dashboards
<b>Learning Adaptation</b>	None	None	Algorithmic personalization
<b>Skill Verification</b>	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

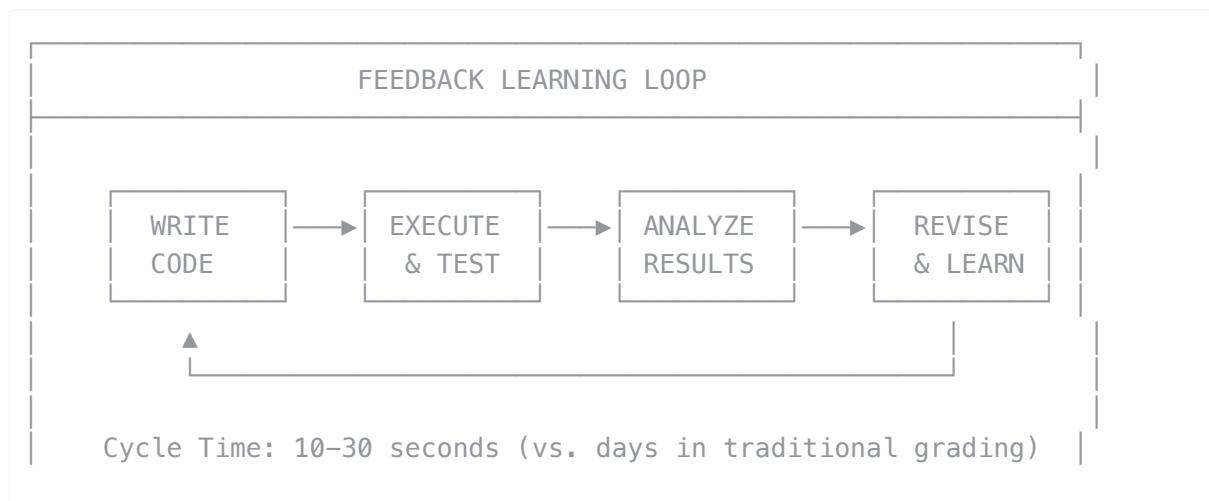
#### Difficulty Adjustment Logic:

```
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:

Category	Metrics
<b>Performance</b>	Accuracy Rate, Retry Rate, Solve Time by Difficulty
<b>Topic Analysis</b>	Weakness Map, Strength Map, Mistake Frequency
<b>Behavioral</b>	Streak Stability, Momentum, Burnout Risk (0-100)
<b>Engagement</b>	Confidence Index, Engagement Score, Learning Velocity
<b>Patterns</b>	Session Length, Peak Hours, Consistency Score

#### **Skill Tree Architecture:**

Students progress through a hierarchical skill tree with prerequisite relationships, earning XP 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 Formula:** Level = (Total XP ÷ 250) + 1

## 3. Assessment Model

---

### 3.1 Weekly Homework Structure

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

#### Week Structure:

- Week N
  - Topic 1: [Conceptual Focus]
    - Lesson Content (Theory)
    - Practice Questions (5-10)
  - Topic 2: [Conceptual Focus]
    - Lesson Content (Theory)
    - Practice Questions (5-10)
  - Weekly Assignment
    - Due Date
    - Required Questions (Graded)
    - Auto-Calculated Grade

#### Question Distribution (200+ Total):

Week	Topics
Week 1	Variables, Input/Output
Week 2	Conditionals, Boolean Logic
Week 3	Loops, Iteration Patterns
Week 4	Methods, Function Design
Week 5	Arrays, Data Structures
Week 6	Strings, Text Processing
Week 7	Object-Oriented Programming

### 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 Example:

```
{  
  "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  
    }  
  ]  
}
```

### 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 =  $\Sigma(\text{Question Scores}) / \Sigma(\text{Maximum Possible})$

##### Example:

- Question 1: 80/100
- Question 2: 100/100

- Question 3: 60/100
- Assignment Grade =  $240/300 = 80\%$

**Course Grade:**

$$\text{Course Grade} = \Sigma(\text{Assignment Grade} \times \text{Weight}) / \Sigma(\text{Weights})$$

## 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)

## 4. Academic Governance

---

### 4.1 Admin Dashboard Purpose

---

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

#### 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

#### 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 - Per-Student Record:

- Student ID (External)
- Student Name
- Email
- Per-Assignment Grades
- Assignment Submission Timestamps
- Total Questions Attempted/Passed
- Overall Pass Rate
- Current Level & XP
- Current Streak
- Last Active Date

**Export Formats:** CSV, XLSX, JSON

#### **Analytics Categories:**

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

## **4.3 Risk Detection and Progress Monitoring**

---

#### **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

## 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

#### 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 (72 routes):

Path	Purpose
/api/auth/*	User registration, login, NextAuth handlers
/api/execute	Code execution endpoint
/api/questions/*	Get question by ID, adaptive selection
/api/progress/*	User statistics, streaks, achievements
/api/admin/*	Platform statistics, analytics, student management
/api/assignments/*	Assignment details, submission
/api/webhooks/*	Payment processing (Stripe)

#### Service Layer:

Service	Responsibilities
Executor	Code sanitization, sandbox invocation, JDoodle fallback, rate limiting

<b>Auth</b>	NextAuth configuration, session management
<b>Progression</b>	XP calculation, level management, streak tracking
<b>Mentor</b>	Cognitive profiling, mistake classification, mission generation
<b>Security</b>	Rate limiting, code validation, event logging

## 5.3 Database Model Summary

### Core Entities:

Domain	Tables
<b>User</b>	User, UserProgress, UserEnrollment, UserTopicStats, CognitiveProfile, ProProfile
<b>Curriculum</b>	Course, Week, Topic, Lesson, Question (with Tests JSON)
<b>Assessment</b>	Assignment, AssignmentQuestion, AssignmentSubmission
<b>Attempts</b>	Attempt (with status, ExecutionMetrics, TestResults JSON)
<b>Gamification</b>	PointsLedger, Achievement, UserAchievement, SkillNode, SkillUnlock, League
<b>Mistakes</b>	MistakeLog (type, severity, isRecurring)

## 5.4 Security Model

### Multi-Layer Security Architecture:

Layer	Implementation
<b>Authentication</b>	NextAuth.js, bcrypt hashing (cost factor 10), JWT sessions, OAuth
<b>Authorization</b>	Role-based access control (USER, ADMIN), route-level middleware
<b>Code Sanitization</b>	Block Runtime.exec, ProcessBuilder, System.exit, Reflection APIs, File/Network I/O, Thread.sleep >10s
<b>Execution Isolation</b>	Docker containerization, non-root user, network isolation, 25s timeout, 256MB memory
<b>Rate Limiting</b>	Redis-backed: 20 exec/min, 10 hints/min, 100 API/min
<b>Security Headers</b>	X-Frame-Options: DENY, X-Content-Type-Options: nosniff, X-XSS-Protection
<b>Audit &amp; Monitoring</b>	30+ event types logged, Sentry error tracking, IP/session tracking

## 5.5 Execution Sandbox (Docker Isolation)

### Sandbox Configuration:

- Base Image:** Alpine Linux + Eclipse Temurin JDK 17
- Security:** Non-root "sandbox" user, network utilities removed
- Limits:** Single CPU core, 256MB memory, 25 second timeout
- Fallback:** JDoodle API when Docker unavailable (Vercel deployment)

**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

## 6. Data & Privacy

---

### 6.1 User Data Separation

---

#### 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)

#### 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

---

#### Data Minimization:

- Collect only data necessary for educational function
- No tracking beyond learning analytics
- No third-party data sharing for marketing

#### Purpose Limitation:

- Learning progress tracking
- Academic assessment
- System improvement
- No secondary commercial use

#### 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**

---

#### **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

## **6.4 Academic Integrity Enforcement**

---

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:

The platform supports multiple courses with shared and isolated components:

- **Content Sharing:** Question bank shared across courses, topics reusable
- **Content Versioning:** Course-specific variations supported
- **Isolation:** Student enrollments and grades are course-specific
- **Progress:** Can be global or course-scoped

### 7.2 Multi-Class Readiness

---

#### Section Support:

- Multiple sections per course (Morning, Afternoon, Evening)
- TA assignment per section
- Section-specific administrative views
- Same assignments with optional due date adjustments
- 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

---

#### 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 Integration Points:

Area	Current State	AI Enhancement
Hints	Static sequence	Context-aware based on specific mistakes
Explanations	Generic content	Tailored to understanding level
Error Analysis	Pattern-based	Natural language explanation

<b>Questions</b>	Algorithm selection	Dynamic generation for skill gaps
<b>Coaching</b>	Automated missions	Conversational progress reviews

## 8. Why This Is Institutional Infrastructure

---

### 8.1 Academic Operating System Characteristics

---

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

#### Infrastructure Characteristics:

Characteristic	Implementation
<b>Abstraction</b>	Hides complexity of code execution, grading, analytics
<b>Resource Management</b>	Manages compute for thousands of executions
<b>Process Scheduling</b>	Queues and processes submissions asynchronously
<b>Security Kernel</b>	Isolates untrusted code execution
<b>User Management</b>	Authentication, authorization, sessions
<b>Data Management</b>	Stores, retrieves, analyzes learning data
<b>System Services</b>	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

---

**Student Information System (SIS):**

- User provisioning via external ID
- Enrollment synchronization
- Grade export for transcript recording

**Learning Management System (LMS):**

- LTI integration capability
- Deep linking to specific assignments
- Grade passback via LTI

**Identity Provider (IdP):**

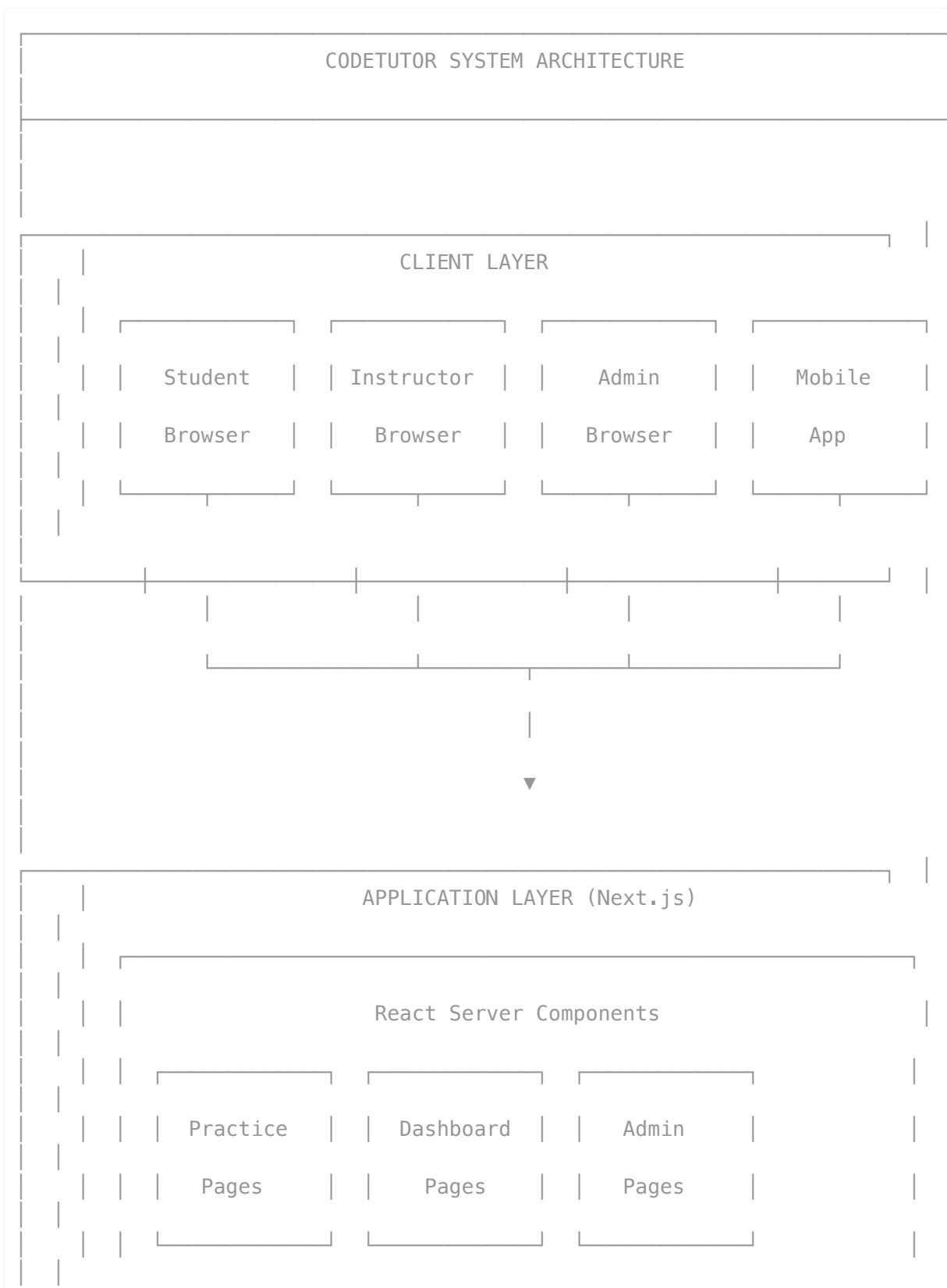
- SAML/OAuth SSO integration
- Institutional login support
- Role mapping from directory

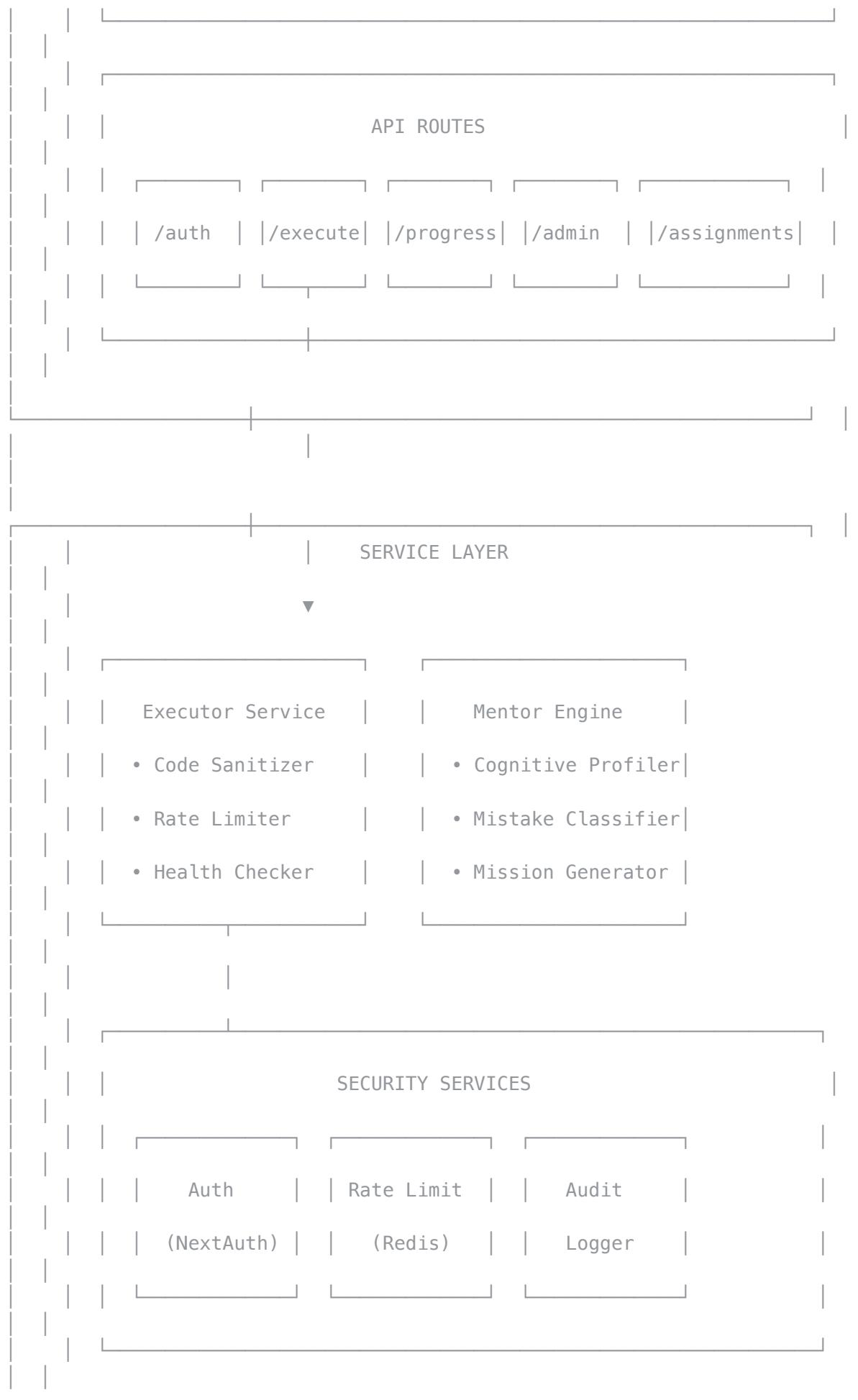
**Analytics Platforms:**

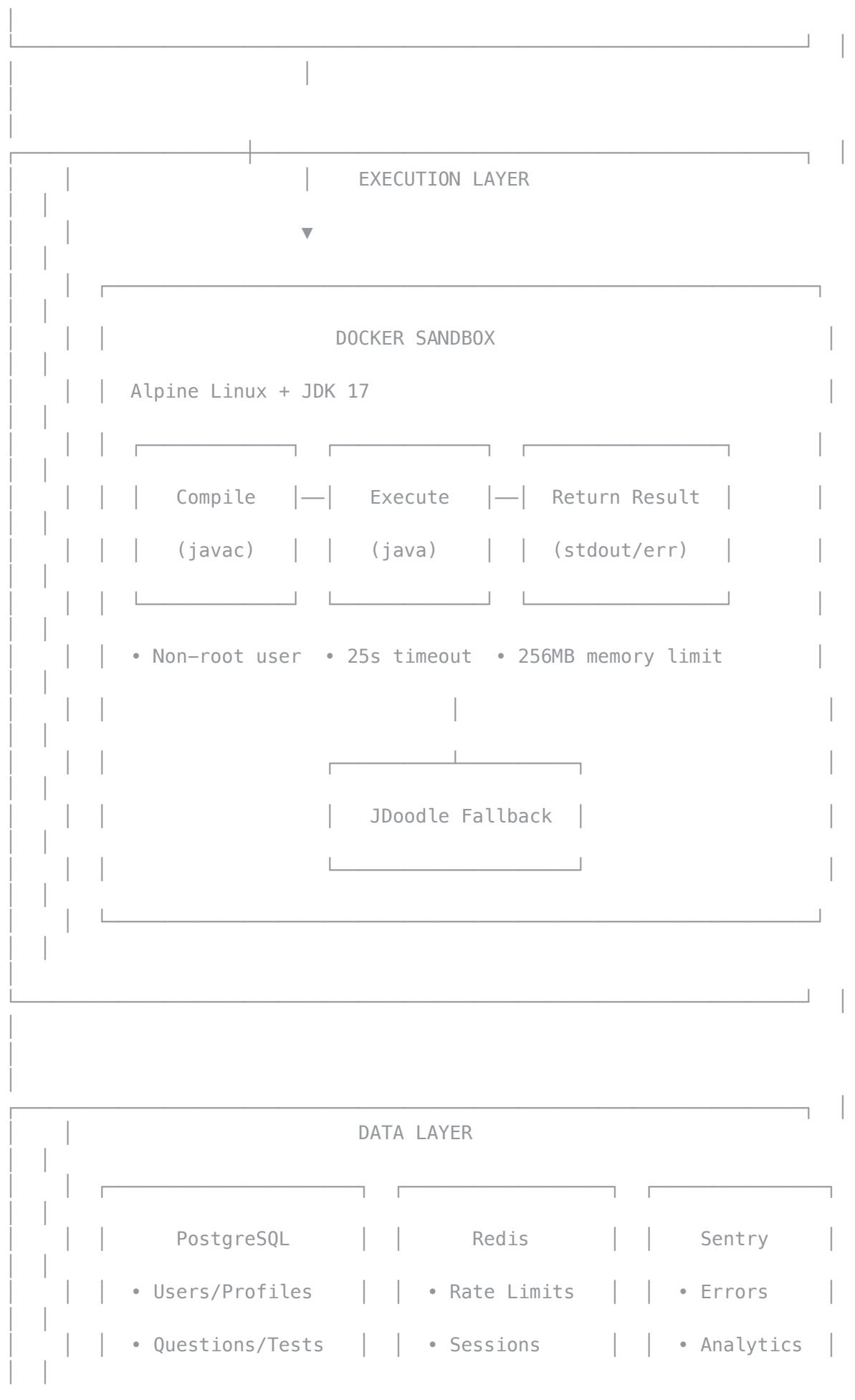
- Data export for institutional analytics
- Learning analytics dashboards
- Predictive modeling data feeds

## 9. Architecture Diagrams

### 9.1 System Architecture Diagram

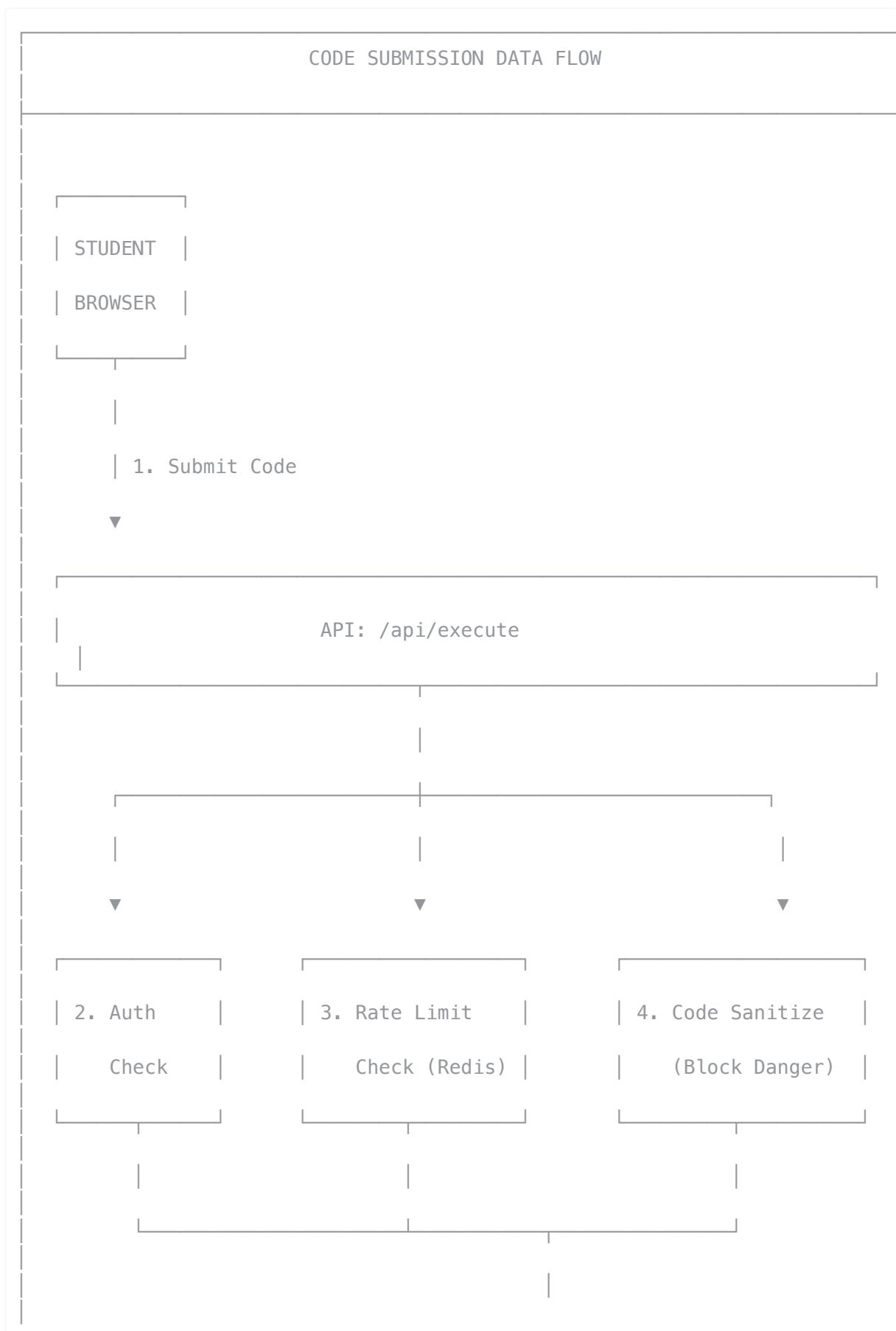








## 9.2 Data Flow Diagram



## 5. EXECUTOR SERVICE

### DOCKER SANDBOX

Compile → Execute → Evaluate Tests  
(javac)      (java)      (compare output)

6. Create | 7. Update | 8. Update | 9. Classify | 10. Update  
Attempt | Progress | Topic Stats | Mistakes | Cognitive  
Record | XP/Level | | | | Profile

(Parallel async updates – non-blocking)

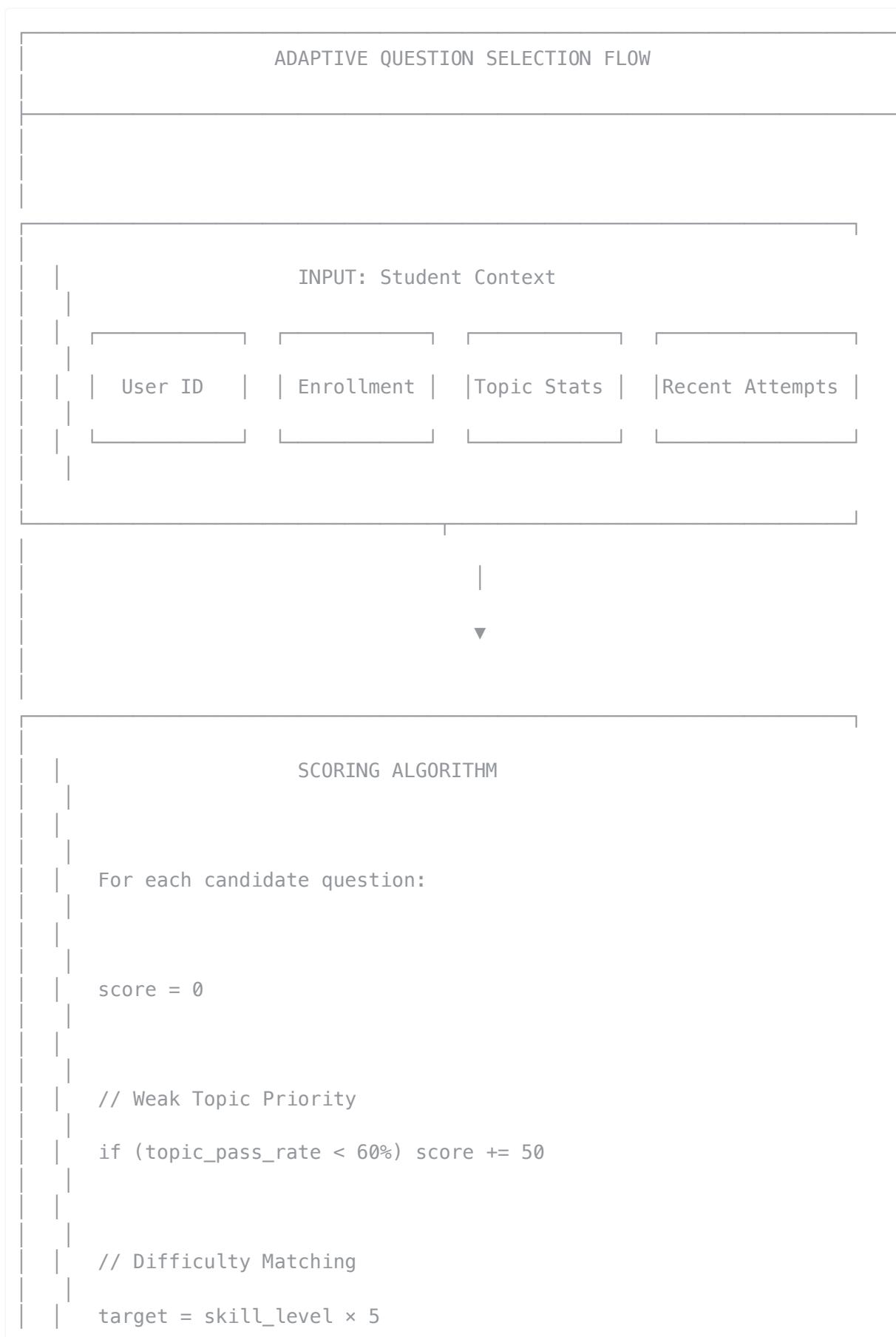
```
11. RESPONSE  
{  
    status: "PASS" | "FAIL" | "ERROR",  
    testResults: [{ passed, input, expected, actual }],  
    executionTime: 234,  
    pointsEarned: 25,  
    newLevel: 5,  
    achievements: ["First Try Bonus"],  
    streakBonus: 15  
}
```



```
STUDENT | ← Immediate feedback (< 5 seconds)
```

```
BROWSER |
```

### 9.3 Adaptive Learning Flow



```
    score += 30 - |question_difficulty - target| × 3

    // Recent Failure Retry
    if (failed_in_last_24h) score += 40

    // Topic Sequence
    if (first_in_new_topic) score += 25

    // Streak Momentum
    if (current_streak >= 3) score += 20

    // Struggle Adjustment
    if (pass_rate < 40%) score -= 30 // easier questions
    if (pass_rate > 80%) score += 30 // harder questions

    // Spaced Repetition (20% chance after 7 days)
    if (passed && days_since > 7 && random() < 0.2) score += 20

    // Randomization
    score += random(0, 20)
```

OUTPUT: Ranked Question List

1. Question A (score: 145) —————> SELECTED
2. Question B (score: 132)
3. Question C (score: 128)
4. Question D (score: 115)

...

## Appendix A: Glossary

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

