CodeTutor System Overview

Version 1.0 Date: January 2026

Executive Summary

What CodeTutor Is

CodeTutor is a web-based educational platform for teaching introductory programming. It provides students with a structured environment to write, execute, and test code, receiving immediate automated feedback. For instructors, it offers tools to create assignments, monitor student progress, and export grades.

Why It Exists

Traditional programming education faces several challenges:

- **Delayed Feedback**: Students submit code and wait days for manual grading, losing the connection between their work and the feedback.
- **Inconsistent Evaluation**: Manual grading introduces subjectivity and variability between graders.
- **Limited Practice Opportunities**: Without immediate feedback, students cannot iterate quickly on their learning.
- **Instructor Workload**: Grading hundreds of programming submissions manually is time-intensive and unsustainable.

CodeTutor addresses these problems by providing automated, immediate, and objective evaluation of student code.

What Academic Problem It Solves

The system provides:

1. **Immediate Feedback**: Students know within seconds whether their code is correct.
2. **Objective Grading**: All submissions are evaluated against the same test cases with no human bias.
3. **Detailed Progress Tracking**: Instructors can identify struggling students and problematic topics through data.

4. **Reduced Administrative Burden**: Automated grading frees instructors to focus on teaching and student support.

Who It Is Designed For

Stakeholder	Primary Benefit
Students	Immediate feedback, unlimited practice, clear progress visibility
Instructors	Automated grading, student analytics, assignment management
Institution	Scalable assessment, audit trail, grade export capability

---

System Scope: What Exists Today

Currently Implemented

The following features are fully operational in the current system:

Programming Environment

- **Language Support**: Java (single language)
- **Code Editor**: Full-featured editor with syntax highlighting, auto-completion, and error indicators
- **Code Execution**: Secure server-side compilation and execution
- **Test-Based Evaluation**: Automated comparison of program output against expected results

Assessment System

- **Question Bank**: Structured repository of programming problems organized by topic
- **Automated Grading**: Deterministic pass/fail evaluation based on test cases
- **Assignments**: Grouping of questions into homework assignments with due dates
- **Submission Tracking**: Complete history of all student attempts

Student Features

- **Practice Mode**: Free practice on any published question
- **Homework Mode**: Structured assignment completion
- **Progress Dashboard**: XP, level, streak, and topic completion tracking
- **Achievement System**: Badges for milestones and accomplishments
- **Immediate Feedback**: Test results shown after each submission

Instructor Features

- **Admin Dashboard**: Platform-wide statistics and activity monitoring
- **Student List**: Overview of all enrolled students with key metrics

- **Individual Student View**: Detailed performance analysis per student
- **Gradebook**: Assignment grades with filtering, sorting, and CSV export
- **Curriculum Management**: Create and organize courses, weeks, topics, and questions
- **Assignment Management**: Create, edit, and publish homework assignments

Data and Reporting

- **Attempt History**: Every code submission is permanently recorded
- **Grade Export**: CSV download of gradebook data
- **Analytics**: Pass rates, attempt counts, and topic performance metrics

Not Yet Implemented

The following features do not exist in the current system:

Feature	Status
Plagiarism Detection	Not implemented
AI-Powered Feedback	Infrastructure exists; not active
LMS Integration (Canvas, Moodle, etc.)	Not implemented
SSO / LDAP Authentication	Not implemented
Multi-Language Support	Java only
Peer Review	Not implemented
Real-Time Collaboration	Not implemented
Partial Credit Grading	Not implemented (binary pass/fail)

Sitemap and Page Structure

Student Pages

/login

Purpose: User authentication entry point.

Data Shown: Login form (email, password).

Decisions Supported: Access control to the platform.

/dashboard

Purpose: Central hub for student learning activity.

Data Shown:

- Current level and total XP earned
- Active streak (consecutive practice days)
- Daily challenge status
- Progress through course topics
- Recent activity summary

Decisions Supported: What to practice next, whether streak is at risk, overall progress assessment.

/practice

Purpose: Browse and select practice questions.

Data Shown:

- List of all topics

- Questions within each topic
- Difficulty indicators
- Completion status per question

Decisions Supported: Which topic or question to practice.

/practice/[questionId]

Purpose: Individual question practice environment.

Data Shown:

- Problem statement and requirements
- Code editor with starter code
- Test execution results (pass/fail per test)
- Hints (optional, with point deduction)
- Solution (optional reveal)

Decisions Supported: How to solve the problem, whether to use hints, when to move on.

/homework/[assignmentId]

Purpose: Complete assigned homework.

Data Shown:

- Assignment title and due date
- Ordered list of questions
- Completion status per question
- Submit button when ready

Decisions Supported: Which question to work on, when to submit.

/achievements

Purpose: View earned badges and milestones.

Data Shown:

- Earned achievements with descriptions
- Locked achievements (goals to work toward)
- Achievement criteria

Decisions Supported: What accomplishments have been made, what goals remain.

/profile

Purpose: Personal statistics and account information.

Data Shown:

- Total questions solved
- Pass rate
- Streak history
- Level and XP breakdown
- Topic-by-topic performance

Decisions Supported: Self-assessment of strengths and weaknesses.

Admin / Instructor Pages

/admin

Purpose: Platform overview and health monitoring.

Data Shown:

- Active users (today, this week)
- Total submissions
- Overall pass rate
- Recent activity feed
- System health indicators

Decisions Supported: Is the platform being used? Are there concerning trends?

/admin/students

Purpose: View all enrolled students.

Data Shown:

- Student list with name, email
- Per-student metrics: attempts, pass rate, streak, level
- Registration date
- Search and filter capabilities

Decisions Supported: Which students need attention? Who is excelling?

/admin/students/[studentId]

Purpose: Deep dive into individual student performance.

Data Shown:

- Student profile information
- Topic-by-topic performance breakdown
- Recent attempt history
- Achievement list
- Mistake patterns (if logged)

Decisions Supported: Why is this student struggling? What topics need reinforcement?

/admin/gradebook

Purpose: Central grade management interface.

Data Shown:

- All assignments with submission status matrix
- Per-assignment statistics (submitted, missing, average grade)
- Filtering by assignment, status, grade range

- Individual submission details (question results, test outcomes)
- CSV export button

Decisions Supported: Who has not submitted? What are class-wide performance patterns? Grade export for records.

/admin/assignments

Purpose: Create and manage homework assignments.

Data Shown:

- List of all assignments
- Due dates and publication status
- Questions included in each assignment
- Submission counts

Decisions Supported: What assignments exist? What needs to be created or modified?

/admin/curriculum

Purpose: Manage course content structure.

Data Shown:

- Course hierarchy: Course > Week > Topic > Question
- Question details: prompt, tests, difficulty, hints
- Publication status controls

Decisions Supported: What content exists? What needs to be added or edited?

Learning and Assessment Model

How Students Practice

Students access the practice environment through two paths:

1. **Free Practice**: Browse topics and select any question to attempt
2. **Homework**: Work through assigned questions in order

For each question, students:

1. Read the problem statement
2. Write code in the editor (starter code is provided)
3. Click "Run" or "Check" to execute their code
4. View test results immediately
5. Iterate until all tests pass or choose to move on

Students may attempt each question unlimited times. All attempts are recorded.

How Submissions Work

When a student submits code:

1. The code is sent to the server
2. Security validation checks for prohibited operations
3. The code is compiled (Java)
4. If compilation fails, the error is returned immediately
5. If compilation succeeds, the code is executed against each test case
6. Each test case provides input via stdin; output is captured from stdout
7. Output is compared character-by-character to expected output
8. Results are returned: pass/fail per test, with actual vs expected output shown

The entire process takes 1-5 seconds for typical submissions.

How Grading Works

Test Case Evaluation

Each question has a set of test cases. A test case consists of:

- **Input**: Data provided to the program
- **Expected Output**: The exact output the program should produce

The student's output must match the expected output exactly (with whitespace normalization) for the test to pass.

Question-Level Grading

A question is marked **PASS** only if all test cases pass. If any test fails, the question is marked **FAIL**.

Possible statuses:

- **PASS**: All tests passed
- **FAIL**: One or more tests failed
- **COMPILE_ERROR**: Code did not compile
- **RUNTIME_ERROR**: Exception during execution
- **TIMEOUT**: Execution exceeded time limit
- **MEMORY_EXCEEDED**: Execution exceeded memory limit

Assignment-Level Grading

Assignment grade is calculated as:

```
Grade = (Questions Passed / Total Questions) x 100
```

Example: An assignment has 10 questions. A student passes 8. Grade = 80%.

Why Grading Is Objective and Deterministic

Every submission is evaluated by the same automated process:

- Same test cases for all students

- Same comparison logic
- No human judgment involved

Two students who submit identical code will always receive identical results.

Why Formatting and Style Are Not Graded

The system evaluates **functional correctness** only:

- Does the program produce the correct output for each input?

Code style, variable naming, comments, and formatting are not evaluated. This is intentional:

- Objective evaluation requires measurable criteria
- Style is subjective and difficult to automate fairly
- Functional correctness is the foundational skill for beginners

How Retrying Supports Learning

Students can retry any question unlimited times. Each attempt:

- Provides immediate feedback
- Shows which tests passed and failed
- For visible tests, shows expected vs actual output
- Allows students to identify and fix their mistakes

This rapid feedback loop supports iterative learning, where students learn from errors and improve incrementally.

Error Handling and Feedback: Current State

Types of Errors

Compile Errors

What Happens: Java compiler fails to compile the code.

What Student Sees: Full compiler error message, including line number and error description.

Example: "error: ';' expected at line 5"

Runtime Errors

What Happens: Code compiles but throws an exception during execution.

What Student Sees: Exception type and message.

Example: "ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 5"

Timeout

What Happens: Code execution exceeds the time limit (typically 5 seconds per test).

What Student Sees: Message indicating the code took too long, suggesting possible infinite loop.

Failed Test Cases

What Happens: Code runs but produces incorrect output.

What Student Sees:

- Which test(s) failed
- For visible tests: the input, expected output, and actual output
- For hidden tests: only pass/fail status (details revealed after passing)

Current Feedback Limitations

The system currently provides:

- Technical error messages (from compiler or runtime)
- Input/output comparison for failed tests

The system does **not** currently provide:

- Natural language explanation of errors
- Pedagogical guidance on how to fix mistakes
- Hints specific to the error type
- Links to relevant learning materials

Planned Enhancement

The database schema includes a `PedagogicalFeedback` model designed to provide:

- Conceptual classification of mistakes (e.g., "loop boundary error", "off-by-one")
- Natural language explanations
- Guiding questions to prompt student thinking
- Suggested topics to review

This feature is **not yet active**. Currently, error classification is recorded but semantic explanations are not generated.

Admin Visibility and Academic Control

Real-Time Instructor Visibility

Instructors can see:

Data	Location	Update Frequency
Active users today	Admin Dashboard	Real-time
Total submissions	Admin Dashboard	Real-time
Pass rate trends	Admin Dashboard	Real-time
Individual student attempts	Student Detail Page	Real-time
Assignment submission status	Gradebook	Real-time
Question-by-question results	Submission Detail	Real-time

Identifying Student Difficulty

The system surfaces struggling students through:

1. **Pass Rate**: Students with low overall pass rates are visible in the student list
2. **Attempt Counts**: High attempt counts on specific questions indicate difficulty
3. **Topic Performance**: Per-topic statistics show where students struggle
4. **Missing Submissions**: Gradebook highlights students who have not submitted
5. **Streak Breaks**: Dashboard shows students whose activity has dropped

Identifying Problematic Topics

Instructors can identify difficult content through:

1. **Topic-Level Pass Rates**: Analytics show which topics have lowest success rates
2. **Question-Level Statistics**: Individual questions can be sorted by pass rate

3. **Class-Wide Patterns**: If many students fail the same question, the question may be too hard or unclear

Grade Export Process

1. Navigate to Gradebook (`/admin/gradebook`)
2. Optionally filter by assignment, status, or grade range
3. Click "Export CSV"
4. Download file contains:
    - Student external ID (if set)
    - Student name
    - Student email
    - Assignment title
    - Week number
    - Submission status
    - Numeric grade (0-100)
    - Submission timestamp

This CSV can be imported into institutional grading systems.

Reduction of Manual Grading Load

Traditional Process	CodeTutor Process
Collect submissions manually	Automatic submission capture
Run each submission to test	Automated execution
Compare output, assign grade	Deterministic test-based grading
Record grade in spreadsheet	Automatic grade recording
Provide individual feedback	Immediate automated feedback

Estimated time savings: For a class of 100 students with 10 assignments of 10 questions each, CodeTutor eliminates approximately 100+ hours of manual grading per semester.

Technical Foundations

Architecture Overview

CodeTutor is a web application with the following components:

Frontend

- **Framework**: Next.js (React-based)
- **Code Editor**: Monaco Editor (the same editor used in Visual Studio Code)
- **Styling**: Tailwind CSS with accessible component library
- **Data Fetching**: React Query for efficient server communication

Backend

- **API Layer**: Next.js API routes (Node.js runtime)
- **Authentication**: NextAuth.js with JWT sessions
- **Database ORM**: Prisma (type-safe database access)
- **Input Validation**: Zod schema validation

Database

- **Engine**: PostgreSQL
- **Data Stored**: User accounts, submissions, grades, progress, content, analytics

Code Execution

- **Primary**: Remote executor service running in Docker container
- **Fallback**: Third-party code execution API (JDoodle)
- **Isolation**: Each submission runs in isolated environment

Security Principles

Code Execution Security

Student code is executed in a sandboxed environment with:

1. **Prohibited Operations**: Code is scanned and blocked from:
   - Executing system commands
   - Accessing the file system
   - Making network connections
   - Using reflection
   - Spawning processes
2. **Resource Limits**:
   - Time limit: 5 seconds per test (25 seconds total)
   - Memory limit: 256 MB
   - No persistent storage access
3. **Isolation**: Each execution runs in a fresh container with no access to other submissions or system resources.

Data Security

- Passwords are hashed using bcrypt
- Sessions use signed JWT tokens
- Database access is parameterized (no SQL injection)
- Admin routes require role verification

Access Control

- Two roles: USER (student) and ADMIN (instructor)
- Middleware enforces authentication on protected routes
- Admin pages verify admin role before rendering

Academic Readiness and Limitations

What the System Is Ready For Today

CodeTutor is suitable for:

1. **Single-Course Deployment**: Running an introductory Java programming course
2. **Automated Assessment**: Replacing manual grading for code correctness
3. **Student Practice**: Providing unlimited practice opportunities with feedback
4. **Progress Monitoring**: Tracking student engagement and performance
5. **Grade Recording**: Maintaining audit trail and exporting for institutional records

The system has been designed with academic integrity in mind:

- All submissions are permanently recorded
- Timestamps are captured
- Attempt history is preserved

Requirements for Full Institutional Deployment

For institution-wide adoption, the following would be needed:

Requirement	Current State	Needed
Multi-course support	Data model supports it	Configuration and testing
Concurrent users	Moderate scale	Load testing, executor scaling
SSO integration	Not implemented	SAML/LDAP integration
LMS grade sync	CSV export only	LTI or direct API integration
Multi-language support	Java only	Additional language executors
Backup and recovery	Standard DB backup	Institutional backup policies

Known Limitations

Technical Limitations

Limitation	Impact	Mitigation
Java only	Cannot teach Python, C++, etc.	Architecture supports adding languages
No SSO	Students need separate accounts	Can set external student ID for matching
No LMS integration	Manual grade export/import	CSV export available
Binary grading	No partial credit	Each question is pass/fail

Pedagogical Limitations

Limitation	Impact	Mitigation
No style grading	Students may write poor style	Style can be taught separately
No plagiarism detection	Cannot detect copying	External tools can be used
Limited error explanation	Students see raw errors	Pedagogical feedback is planned
No peer review	Students cannot review each other	Not in scope for automated tool

Operational Limitations

Limitation	Impact	Mitigation
Single deployment	All courses share instance	Can deploy separate instances
No academic calendar	No term/semester management	Use due dates on assignments
English only	Interface not localized	Content can be in any language

Conclusion

CodeTutor provides a functional, secure, and scalable platform for automated programming assessment. It is ready for deployment in introductory programming courses where objective, test-based grading is appropriate.

The system prioritizes:

- Immediate student feedback
- Objective evaluation
- Instructor visibility
- Data integrity

Known limitations are documented and can be addressed through future development or integration with complementary tools.

Document generated: January 2026 Version: 1.0