# RunitIn - Online Judge Platform

RunitIn is a comprehensive online coding platform that allows users to:

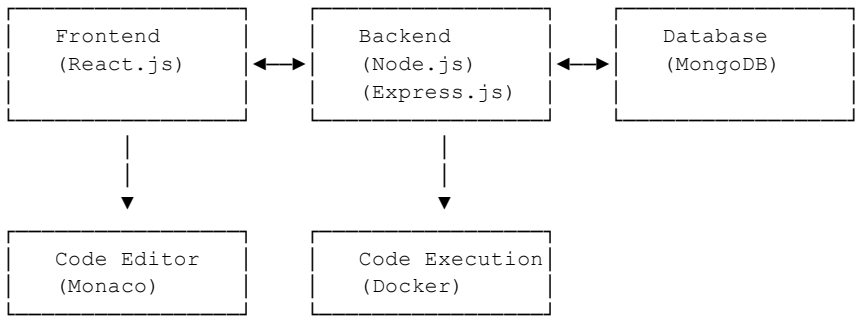
- 🖥️ Solve Coding Questions – Practice a variety of programming problems
- ✍️ Create Custom Questions – Design and publish your own challenges
- 🚀 Run and Judge Code – Execute code in multiple languages (C++, Python, Java)
- 📊 Track Progress – View submissions, judge performance, and analyze success

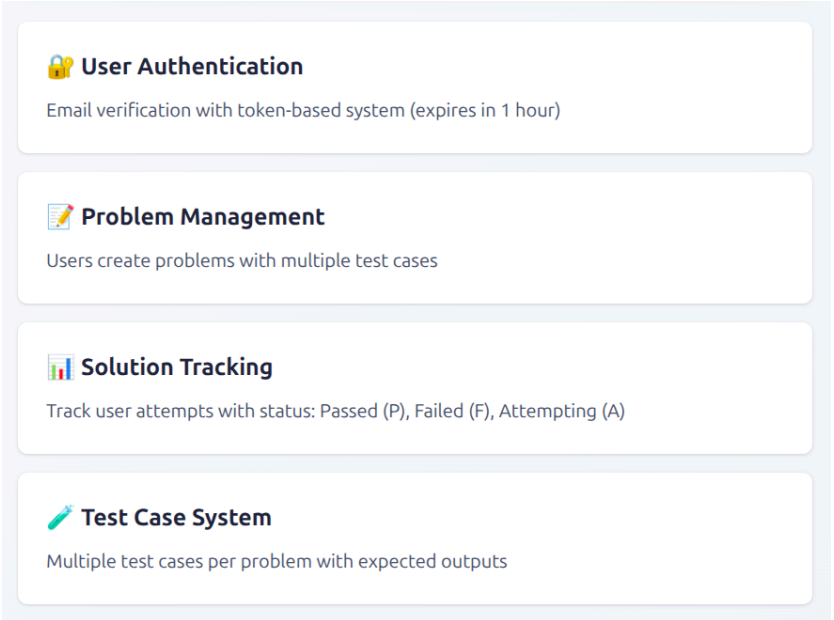
## Key Features

- Multi-language code execution (C++, Python, Java)
- Real-time code compilation and execution
- Test case validation and scoring
- User authentication with email verification
- Problem creation and management
- Progress tracking and statistics
- Responsive web interface

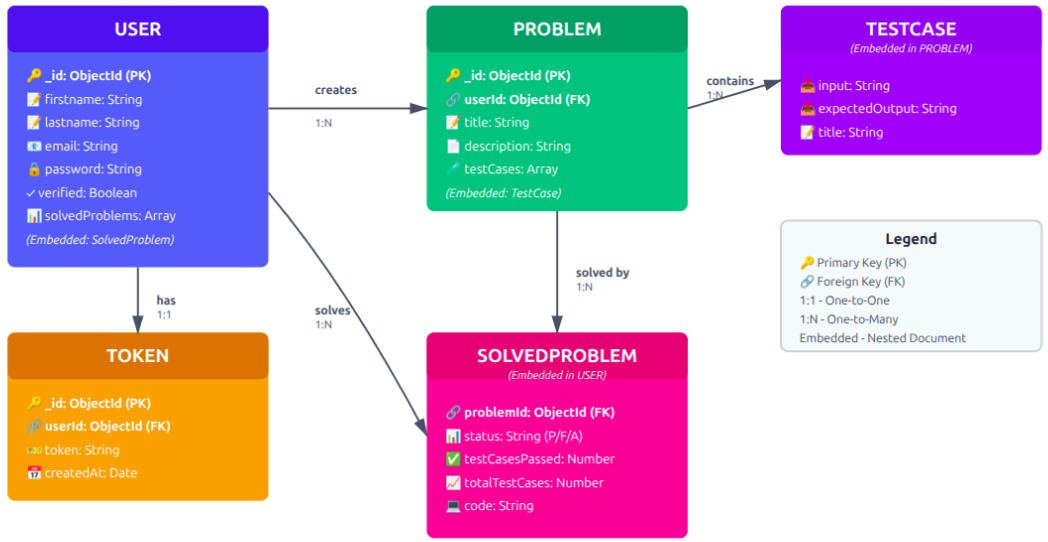
## Architecture

The project follows a full-stack architecture with clear separation of concerns:



# Online Judge - Entity Relationship Diagram




# Backend Documentation

## Key Controllers

1. **Authentication Controller (auth.js)**: Handles user registration (with email verification), email verification, login (JWT), logout, and authentication status.
2. **Problem Controller (problem.js)**: Manages fetching all problems, specific problems (with user code), user-created problems, and operations (create, update, delete) on problems, as well as fetching solved problems.
3. **Code Execution Controller (run.js)**: Executes C++, Python, and Java code with custom input, returning output and errors.
4. **Judge Controller (judge.js)**: Runs code against test cases, compares results, and updates user progress/stats.
5. **Middleware: Authentication Middleware (authMiddleware.js)**: Validates JWT tokens from cookies, protects authenticated routes, and adds user info to requests.

# Frontend Documentation

## Key Components

- **Authentication:** User login, User registration, Email verification.
- **Problem:** All problems, Individual problem with editor, User's created problems, User's solved problems
- **Layout:** App wrapper, navigation, Authenticated user route protection.

**State ManagementAuthentication Context (AuthContext.jsx)**

- Manages user authentication
- Persists authentication data (localStorage)
- Provides authentication methods

**Custom Hooks**

- useAuth: Authentication state and methods (login, logout, status)

**ServicesAuthentication Service (authService.js)**

- API calls: user registration, login/logout
- Handles authentication responses

**Problem Service (problemService.js)**

- API calls: problem CRUD
- Code execution, judging, saving requests

# API Documentation

### Authentication Endpoints

1. **POST** /auth/register
2. **POST** /auth/login
3. **GET** /auth/verify-email/:id/:token

**Problem Endpoints**

1. **GET** /problems
2. **GET** /problems/:id
3. **POST** /problems

**Code Execution Endpoints**

1. **POST** /run
2. **POST** /judge

# Code Execution Engine

### Supported Languages

1. C++ - Compiled with g++
2. Python - Interpreted with python
3. Java - Compiled with javac, executed with java

### Execution Process

1. File Generation: Create temporary files for code and input
2. Compilation: Compile code (for C++ and Java)
3. Execution: Run code with input redirection
4. Output Capture: Capture stdout and stderr
5. Cleanup: Remove temporary files

### File Management

- Temporary Files: Generated in `helpers/codes/` directory
- Input Files: Generated in `helpers/inputs/` directory
- Output Files: Generated in `helpers/outputs/` directory
- Cleanup: Automatic cleanup after execution

# Authentication & Authorization

### Authentication Flow

1. Registration: User creates account with email verification
2. Email Verification: User clicks verification link
3. Login: User authenticates with email/password
4. JWT Token: Server issues JWT token stored in HTTP-only cookie
5. Protected Routes: Middleware validates JWT for protected endpoints

### Security Features

- Password Hashing: bcryptjs for secure password storage
- JWT Tokens: Secure token-based authentication
- HTTP-Only Cookies: Prevents XSS attacks
- Email Verification: Ensures valid email addresses
- CORS Protection: Cross-origin request security

### Authorization Levels

- Public Routes: Home, login, register, email verification
- Protected Routes: All problem-related operations
- Owner-Only: Problem creation, editing, deletion