Coding Standards Document – ELO Learning Platform

Overview

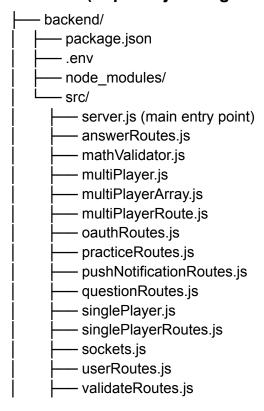
This document defines the coding standards and best practices for the **ELO Learning Platform**, built using:

- Backend: Express.js with NestJS structure
- Frontend: React.js with Next.js (PWA)
 - **Database**: PostgreSQL (relational), InfluxDB (time-series)
- Communication: REST APIs & WebSockets
- Language: JavaScript (both frontend and backend), intended to transition to TypeScript

All developers must follow these standards to ensure code is consistent, readable, testable, and secure.

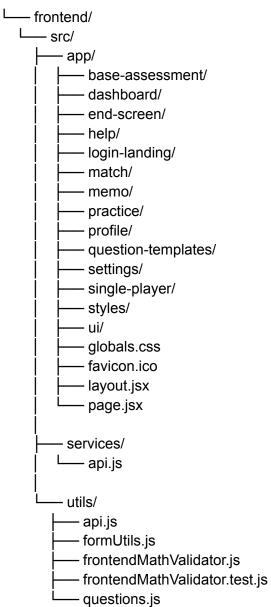
1. Project Structure & Architecture

Backend (Express.js using NestJS-style layering)



```
utils/ (utility files
```

Frontend (Next.js with modular components)



2. Language and Syntax

Language

- Use TypeScript for all frontend and backend code.
- Strict typing (strict: true in tsconfig.json).

Formatting

- Enforce **Prettier** for code formatting (2 spaces, semicolons, trailing commas).
- Enforce **ESLint** with recommended and security rules.

3. Modules and Imports

- Use absolute imports (@/components/...) in Next.js.
- Avoid deeply nested files; group by domain (e.g., /auth, /leaderboard).
- No circular dependencies.

4. Naming Conventions

Element	Convention	Example
Files	kebab-case	user-profile.service .ts
Classes/Interfaces	PascalCase	UserService, IUserData
Variables	camelCase	userId, isAuthenticated
Constants	UPPER_SNAKE_CASE	MAX_ATTEMPTS, TOKEN_EXPIRY
DTOs	Suffix with Dto	CreateUserDto
React Components	PascalCase	MathInputField.tsx

5. Testing

Backend:

- Use Jest for unit/integration tests.
- Minimum 80% code coverage (fail CI otherwise).

Frontend:

- Use Cypress for E2E tests.
- o Include tests for login, problem solving, and leaderboard.

6. Security Practices

- Always hash passwords using bcrypt (12 or more salt rounds).
- All communication must be over HTTPS (TLS 1.2+).
- Use JWT & OAuth 2.0 for token-based authentication.
- Enforce RBAC (Role-Based Access Control) on backend endpoints.
- Sanitize and validate all user inputs (e.g., using class-validator or Zod).

7. Database Standards

PostgreSQL

- Table names: snake_case plural (users, user_profiles)
- Column names: snake_case (first_name, elo_rating)
- Use **UUIDs** for primary keys.
- Normalize where appropriate; use foreign keys.
- Use migrations (e.g., **Prisma Migrate** or **TypeORM**) no raw schema changes.

8. API Design Standards

- Use **RESTful** principles for endpoints.
- Prefix routes with /api/v1/....
- Use **DTOs** for all request/response shapes.
- Use OpenAPI/Swagger for documentation.
- Response shape:

```
{
  "success": true,
  "data": { ... },
  "message": "Optional descriptive message"
}
```

9. WebSocket Communication

- All socket events follow camelCase convention.
- Use @WebSocketGateway() decorators for event handling in NestJS.
- Authenticate users on connection using JWT token validation.

10. Frontend Development Standards (React/Next.js)

- Functional components with **React Hooks** (useState, useEffect, useReducer)
- Avoid anonymous functions in JSX.
- Use PropTypes or TypeScript interfaces for props validation.
- Prefer useContext or state management libraries over prop drilling.
- CSS: Use TailwindCSS or modular CSS per component.

11. CI/CD & Deployment

- All changes must pass:
 - o ESLint
 - Prettier formatting
 - Unit + integration tests
- Use **GitHub Actions** to deploy via Docker containers to AWS or Azure.
- Every commit to main must be associated with a pull request and review.

12. Documentation

- All public functions and services must be documented with **JSDoc**.
- Markdown-based documentation stored in /docs or README.md.
- API routes must be documented via Swagger.

13. Code Review & Version Control

All commits follow Conventional Commits:

- Use feature branches: feature/leaderboard-ui, fix/auth-bug
- Pull requests must:
 - Be peer-reviewed
 - Pass CI
 - Include test coverage