SkillWise Project: Problem, Solution, and Tech Stack

# Problem

Young adults learning new skills (coding, design, languages, public speaking) often lack mentors and a structured path. They waste time figuring out what to practice, how to practice, and whether they’re improving.

# App Definition: SkillWise

SkillWise is a web app that turns a personal learning goal into a sequence of actionable challenges. Students can:  
- Create goals (e.g., “Learn JavaScript basics in 3 weeks”).  
- Generate or write challenges aligned to that goal.  
- Submit work (text/code/links), get AI feedback, and track progress.  
- (Stretch) Do peer reviews and see a leaderboard for motivation.  
  
Each student will build a full vertical slice—frontend, backend, database, tests, DevOps, and AI integration—over four sprints, solo. At the end of each sprint, they will open a PR; the team will review and vote on the best implementation to merge to main.

# Scope at a Glance

- MVP1 (Sprints 1–2): Auth → Goals → Challenges → Progress → Dashboard + Tests + CI  
- MVP2 (Sprints 3–4): AI-generated challenges → AI feedback → Leaderboard & Peer Review → Deploy

# Tech Stack (aligned to User Stories)

## Frontend (Web)

• React 18 (JavaScript) – app UI and routing  
• React Router – page navigation  
• Forms & validation: React Hook Form + Zod  
• UI styling: Tailwind CSS or MUI  
• HTTP client: Axios (or fetch)  
• Charts: Recharts (progress/leaderboard)  
• State: Local state, optional React Query  
• Accessibility: axe DevTools  
  
Stories covered: Signup/Login, dashboard, goals/challenges UI, progress/leaderboard, e2e testing

## Backend (API)

• Node.js (LTS) + Express – REST API  
• Auth: JWT (access + refresh) via httpOnly cookies; bcrypt for password hashing  
• Validation: Zod or custom JS checks  
• Logging: pino (or console)  
• Rate limiting: express-rate-limit  
• Error handling: Express middleware  
  
Stories covered: Auth endpoints, CRUD for goals/challenges/progress, AI endpoints, peer review, leaderboard

## Database

• PostgreSQL (via Docker locally, managed in prod)  
• Schema includes: users, refresh\_tokens, goals, challenges, progress, submissions, ai\_feedback, explanations, peer\_reviews, leaderboard, optional teams  
• DB access options: node-postgres (pg), Knex.js, or Prisma  
  
Stories covered: Persistence for all core entities

## AI Integration

• API: OpenAI (or compatible)  
• Patterns: reusable prompt templates, log prompt+response  
• Testing: snapshot tests for known prompts  
• Rate limiting on endpoints  
  
Stories covered: generate challenges, submit for feedback, explanations

## Testing

• Unit/API tests: Jest + Supertest  
• End-to-end: Cypress  
• Accessibility: axe-core  
  
Stories covered: auth tests, CRUD tests, AI snapshot tests, e2e goal workflow

## DevOps / Tooling

• Version control: Git (branch per student, PR per sprint)  
• Containers: Docker + Docker Compose  
• CI: GitHub Actions (lint + Jest + Cypress)  
• Deploy: Vercel (frontend), Render/Railway (backend), Neon/Supabase (DB)  
• Monitoring: Sentry  
• Docs: Swagger/OpenAPI + Runbook  
  
Stories covered: local dev, CI, error tracking, production deploy, docs

# Milestones (Definition of Done per Sprint)

• Sprint 1: Auth + Dashboard shell, Docker up, auth tests pass  
• Sprint 2: Goals/Challenges/Progress CRUD, Cypress smoke test, CI pipeline green  
• Sprint 3: AI challenge + feedback loop, snapshot tests, Sentry error capture  
• Sprint 4: Leaderboard + peer review, accessibility fixes, deployed app, Swagger docs