# **Problem & App Definition**

## **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.

You will build a full vertical slice—**frontend, backend, database, tests, DevOps, and AI integration**—over four sprints, solo. At the end of each sprint, you open a PR; your team reviews and votes on the best implementation to merge to main.

# **Scope at a Glance (What you’ll build)**

* **MVP1 (Sprints 1–2):** Auth → Goals → Challenges → Progress → Working dashboard + basic tests + CI
* **MVP2 (Sprints 3–4):** AI-generated challenges → AI feedback on submissions → Leaderboard & peer review → Deploy

# **Tech Stack (aligned to the user stories)**

Frontend must be **plain JavaScript (no TypeScript)**. Backend can be JS; use whichever you prefer.

## **Frontend (Web)**

* **React 18 (JavaScript)** – app UI and routing
* **React Router** – page navigation
* **Forms & validation:** **React Hook Form** + **Zod** (use Zod via zod + custom JS types; or validate in code if you prefer)
* **UI styling:** Tailwind CSS *or* MUI (choose one; Tailwind = utility-first, MUI = component library)
* **HTTP client:** Axios (or fetch)
* **Charts:** Recharts (progress/leaderboard)
* **State:** Local component state + React Query *optional* (for cache/fetch); simple apps can skip extra state libs
* **Accessibility:** axe DevTools (run in Cypress test or browser extension)

**Why these map to stories**

* Signup/Login/Forms → React + RHF + (Zod)
* Dashboard/Lists/Modals → React + Router + Tailwind/MUI
* Progress/Leaderboard → Recharts
* E2E tests for flows → Cypress + axe

## 

## **Backend (API)**

* **Node.js (LTS) + Express** – REST API
* **Auth:** JWT (access + refresh) via **httpOnly** cookies; **bcrypt** for password hashing
* **Input validation:** Zod (shared schemas, or simple JS validation if you prefer)
* **Logging:** pino (or console for basic projects)
* **Rate limiting:** express-rate-limit (protect AI endpoints)
* **Error handling:** Express error middleware + structured errors

**Why these map to stories**

* Auth endpoints /signup, /login, /logout → Express + JWT + bcrypt
* Goals/Challenges/Progress CRUD → Express routes + DB layer
* AI endpoints /ai/generateChallenge, /ai/submitForFeedback → Express + LLM API
* Peer review & leaderboard endpoints → Express routes + DB

## **Database**

* **PostgreSQL** (local via Docker; any managed Postgres in prod)
* **Schema:** Use the provided **starter ER diagram** to create your own tables  
   (tables include: users, refresh\_tokens, goals, challenges, progress, submissions, ai\_feedback, explanations, peer\_reviews, leaderboard, *(optional)* teams, team\_members)
* **DB access options (pick one):**
  + **Option A (no ORM):** node-postgres (pg) + SQL files (great for learning SQL)
  + **Option B (query builder):** **Knex.js** (JS migrations + SQL control)
  + **Option C (ORM):** Prisma (fast dev; you’ll still write your own endpoints)

**Why these map to stories**

* Auth persistence → users, refresh\_tokens
* MVP1 data → goals, challenges, progress
* AI feedback loop → submissions, ai\_feedback
* MVP2 collab → peer\_reviews, leaderboard (+ optional teams)

## **AI Integration**

* **LLM API:** OpenAI (or compatible) via REST
* **Patterns:**
  + Reusable **prompt templates**
  + Minimal **safety checks** on inputs/outputs
  + Persist prompt, model, response in ai\_feedback
* **Testing AI:**
  + Snapshot test a few fixed prompts
  + Rate limit endpoints to avoid abuse

**Why these map to stories**

* Generate challenges from goal context → /ai/generateChallenge
* Review user submissions → /ai/submitForFeedback
* Ask “why” explanations (optional stretch) → /ai/explainConcept

## **Testing**

* **Unit & API tests:** **Jest** + **Supertest** (Express routes)
* **End-to-end:** **Cypress** (login → create goal → add challenge → mark complete)
* **Accessibility (a11y):** **axe-core** (via Cypress or browser extension)
* **What to test based on stories**
  + Auth success & failure paths
  + CRUD endpoints happy path + bad inputs
  + AI endpoints with mocked LLM responses (snapshot tests)
  + Core user journey E2E

## **DevOps / Tooling**

* **Version control:** Git (branch per student, PR each sprint)
* **Containers:** **Docker + Docker Compose** (API + Postgres + pgAdmin)
* **CI:** **GitHub Actions** – run lint + Jest + Cypress on PRs
* **Deploy:**
  + Frontend: **Vercel** (static build)
  + Backend: **Render** or **Railway**
  + Database: **Neon** or **Supabase**
* **Monitoring:** **Sentry** (frontend & backend)
* **Docs:** Swagger/OpenAPI at /docs, plus a short **Runbook** (env vars, local dev, deploy steps)

**Why these map to stories**

* Docker Compose → local “one command” start
* GitHub Actions → run tests on PR (the voting branch)
* Sentry → Sprint 3 story “add error tracking”
* Deploy flows → Sprint 4 “launch” stories

## **Environment & Security Basics**

* **Environment variables:** .env for local (NEVER commit), provider secrets in platform dashboards
* **Cookies:** Use **httpOnly**, SameSite=Lax (or Strict), and **Secure** in prod
* **CORS:** allow your frontend origin in dev; lock down in prod
* **Passwords:** bcrypt; never store plaintext
* **Uploads:** store files in object storage (S3/R2) and keep URLs in submissions

## **Suggested Folder Structure (reference)**

/frontend

/src

/components

/pages

/routes

/api (thin wrappers around fetch/axios)

/styles

/backend

/src

/routes

/controllers

/services (AI, business logic)

/db (queries or ORM client)

/middleware

/tests

/docker

docker-compose.yml

/docs

openapi.yaml

RUNBOOK.md

## **What “Done” Looks Like (per milestone)**

* **End Sprint 1:** Local Docker up; Sign up & Login work; JWT refresh; DB has users; basic dashboard route; Jest + Supertest auth tests passing.
* **End Sprint 2 (MVP1):** Goals/Challenges/Progress CRUD + UI; Cypress smoke test (login → goal → challenge → complete); GitHub Actions CI green.
* **End Sprint 3:** AI challenge generation + submission feedback end-to-end; ai\_feedback persisted; snapshot tests; Sentry capturing errors.
* **End Sprint 4 (MVP2):** Leaderboard + peer review working; a11y fixes for top issues; production deploy (Vercel FE, Render/Railway BE); Swagger + Runbook.