# EXECUTION PROMPT — BUILD A REAL, FUNCTIONING WEB APP

<b>Project Name:</b> Vibes

You are a senior staff-level full-stack engineer whose task is to build, not simulate.

You must deliver a fully functioning, production-ready web application named “Vibes”.

■ This is NOT a mock, NOT a prototype, NOT a UI simulation.

■ Every feature must be implemented end-to-end:

- • Database tables
- • API routes
- • Authentication
- • Frontend integration
- • Persistent data

The app must run successfully, accept real user input, store data, and be deployable.

■

## ■ Product Definition

“Vibes” is a social community platform for vibecoders.

Users can:

- • Create accounts and log in
- • Create full profiles
- • Discover and submit vibecoded projects
- • Upvote, comment, and follow
- • Learn from curated resources
- • Submit projects to grant programs
- • Be discovered as builders

Think Product Hunt + GitHub + learning community, but purpose-built for vibecoding.

■

## ■■ MANDATED STACK (NO SUBSTITUTIONS)

- • Frontend: React + TypeScript
- • Styling: Tailwind CSS
- • Backend: Node.js + Express
- • Database: PostgreSQL
- • ORM: Prisma
- • Auth: JWT + OAuth (GitHub + Google)
- • State/Data: React Query
- • Hosting: Replit
- • Version Control: Git + Public GitHub repository

■

## ■ AUTHENTICATION (REAL, WORKING)

- • Signup & login with hashed passwords
- • OAuth login (GitHub required)
- • JWT access + refresh tokens
- • Protected API routes
- • Persistent sessions
- • Password reset (email logic stub acceptable, token logic must work)

■

## ■ USER PROFILES (PERSISTENT)

- • Avatar upload (store URL)
- • Bio
- • Vibecoding skills & tools
- • Social links
- • Public profile pages
- • Follow/unfollow users (stored in DB)

■

## ■ PROJECT SYSTEM (CORE FEATURE)

Users must be able to:

- • Submit projects
- • Edit & delete their projects
- • Attach:
- • Title
- • Description
- • Demo link
- • GitHub repo
- • Tags
- • View project detail pages
- • Upvote projects (1 vote per user)
- • Comment on projects

All actions must persist in PostgreSQL.

■

## ■ LEARNING HUB

- • Admin-seeded resources table
- • Categories & tags
- • Upvotes
- • Comments
- • Bookmark/save resources per user

■

## ■ GRANT SYSTEM (FUNCTIONAL MVP)

- • Create grant programs
- • Submit projects to grants
- • Review submissions
- • Mark winners
- • Grant status tracking
- • No token payments yet, but DB must support rewards later

■

## ■ NOTIFICATIONS (REAL)

- • Stored in database
- • Trigger on:
- • New follower
- • New comment
- • Project upvote
- • Display in UI

■

## ■■ ADMIN CAPABILITIES

- • Role-based access control
- • Feature projects/resources
- • Moderate content
- • Manage grants

■

## ■ GITHUB REQUIREMENTS (MANDATORY)

- • Initialize Git repo
- • Push entire codebase to public GitHub
- • Include:
- • README.md (real setup instructions)
- • Prisma schema
- • .env.example
- • License
- • App must be runnable by cloning the repo

■

## ■ ENGINEERING REQUIREMENTS

- • Prisma migrations
- • Seed script
- • API validation (Zod or Joi)
- • Error handling (backend + frontend)
- • Loading states
- • Empty states
- • No fake data except initial seed
- • No placeholder buttons
- • No commented-out logic

■

## ■ DEPLOYMENT

- • App must run on Replit
- • Provide steps to:
- • Install dependencies
- • Run migrations
- • Start server
- • Ensure frontend and backend communicate correctly

■

## ■ STRICTLY FORBIDDEN

- • UI-only builds
- • Hardcoded arrays pretending to be DB
- • Mock APIs
- • “Imagine this does X” descriptions
- • Skipping features
- • Pseudocode

■

## ■ DELIVERY INSTRUCTIONS

You must:

- 1. Create the full folder structure
- 2. Define Prisma schema
- 3. Build backend APIs
- 4. Implement authentication
- 5. Build frontend pages
- 6. Connect frontend ↔ backend
- 7. Push to GitHub
- 8. Verify app runs

■

## ■ START NOW

Begin by:

- • Designing the database schema
- • Creating the repo structure
- • Initializing Prisma
- • Committing and pushing to GitHub

■

## ■ IMPORTANT FINAL NOTE TO AI

If any feature cannot be completed fully, do not simulate it — explain the limitation and implement the closest real alternative.