Automated Codebase Refactoring and Comprehension

### **Project and Task Description**

#### **(a) Project Summary**

The **Codebase Refactoring and Comprehension Project** is an AI-powered system designed to analyze, refactor, and document messy codebases while generating interview-style questions. This project addresses a common challenge faced by developers, educators, and recruiters: navigating poorly documented code repositories and evaluating technical understanding.

**Codebase:** A codebase is a collection of source codes for a software project, that may include program files, configurations, documentation, and tests. It serves as the source of truth that developers collaborate on, usually managed with version control systems like Git.

**Problems with unorganized/raw codebases developed by students/programmers:**

* **Poor Documentation** – Developers struggle to understand or extend the code.
* **High Complexity** – Messy, unstructured code increases bugs and slows progress.
* **Low Readability** – Inconsistent style, unclear variable names, and lack of comments reduce clarity.
* **Technical Debt** – Quick fixes and outdated practices lower long-term quality.
* **Scalability Issues** – Large codebases become difficult to test, refactor, and extend.
* **Security Vulnerabilities** – Weak coding practices or outdated dependencies expose the system to risks.
* **Knowledge Transfer & Assessment Challenges** – New developers or students find it hard to revise and memorize logic and concepts, making evaluation and learning difficult, especially for interviews.

**The problem statement our project aims to solve:**

* Automate refactoring of messy, unstructured code to reduce complexity.
* Improve readability and documentation with inline comments and summaries.
* Support learning and assessment by generating interview-style technical questions.

**Purpose**

The purpose of this project is to save time and effort by using AI (Artificial Intelligence) to understand large codebases and improve them. Instead of manually reading every file, our project will:

1. Analyze the project’s structure.
2. Clean and reorganize the code.
3. Write simple explanations and documentation.
4. Generate useful interview-style questions about the project.

This tool benefits:

* **Students:** Learn better by seeing clean, documented code and practice with code-related questions.
* **Developers:** Quickly understand large repositories and maintain them.
* **Recruiters/Educators:** Automatically create technical questions for interviews or exams.

**Specifications:**

* **Input Options:** Users upload a **zip file** of their code, paste raw code, or provide a **GitHub repository link** (an online platform where developers store and share code).
* **Output:** Refactored code (A cleaned-up version of the code with better structure) with inline comments, documentation, and 5 tailored interview questions.

#### **Core Features (Explained)**

1. **Code Understanding & Refactoring**:
   * The system reads and analyzes the project structure (folders, files, and functions).
   * It reorganizes messy code without changing its functionality.
   * Adds helpful comments and improves formatting.
2. **Question Generation**:
   * AI creates interview-style questions based on the code logic, helping students and interviewees.
3. **Automated Documentation**:
   * Generates human-readable summaries for files and functions.

* **Tech Stack:**
  + **Frontend:** React/Next.js web interface.
  + **Backend:** Python with FastAPIs.
  + **AI Core:** arge Language Models (LLMs) like Gemini Pro, Groq API, or OpenRouter – AI models used to understand and rewrite code. Model selection will be done by analysing their performance matrix.
  + **Integrations:** GitHub API and vector databases for embeddings.
  + **Deployment:** Vercel (for frontend) and Render (for backend).

**Summary of Approach:** The system follows a three-phase workflow:

1. **Input Phase:** User uploads a repository or topic.
2. **Processing Phase:** File Tree Walker → Code Analyzer → Refactorer → Question Generator.
3. **Output Phase:** Provide refactored, documented code and an interview preparation module.

### **(b) Individual Role and Tasks**

 **Sahil Sharma – 22BCE7921 (Backend Engineer & AI Architect)**

* Builds backend services that analyze and refactor code.
* Integrates AI models for code understanding and question generation.
* Designs APIs (ways for frontend and backend to communicate).

 **Harsh Raj – 22BCE7591 (Frontend Developer & Deployment Lead)**

* Creates evaluation metrics to test system accuracy.
* Designs and builds a simple, clean user interface with Next.js.
* Ensures frontend and backend integration.
* Deploys the app on Vercel and Render for public access.
* Writes documentation for frontend workflows.

**Deliverables:**

1. A functional and responsive frontend.
2. Smooth integration between frontend and backend APIs.
3. Deployment setup on Vercel (frontend) and Render (backend).
4. Performance metrics, test cases and debugging reports to ensure system stability.
5. Clear documentation of frontend workflows and deployment processes.

#### **(c) Detailed Approach**

1. **Frontend Development:**
   * Build a clean, user-friendly interface with Next.js/React.
   * Ensure proper integration with backend APIs for seamless user experience.
2. **Deployment (Vercel & Render):**
   * Deploy frontend on Vercel with global edge delivery for performance.
   * Deploy backend services on Render and configure secure communication.
3. **Testing & Debugging:**
   * Create automated test cases for frontend modules.
   * Debug integration issues between frontend and backend.
   * Evaluate different combination of approaches and models based on developed performance measures.
4. **Documentation:**
   * Write user-friendly guides for deployment steps and frontend workflows.
   * Maintain technical notes for future scalability.

#### **(d) Phases of Design Process**

|  |  |  |
| --- | --- | --- |
| Phase/Review | Objective | Deadline |
| Review 1 | submit a detailed project proposal that explains the problem statement and proposed solution; present a clear approach for implementation with defined steps; outline measurable criteria for evaluating the performance of the Large Language Model (LLM); and create a PowerPoint presentation summarizing the overall project overview.. | 30th August, 2025 |
| Review 2 | To deliver a functional project prototype that integrates a working Large Language Model (LLM) with core backend features, demonstrates key functionalities such as code analysis and question generation, and includes a basic but usable user interface (UI) for testing and feedback. | 30th September 2025 |
| Review 3 | To present a fully deployed version of the project with a refined, visually appealing, and user-friendly interface, ensuring smooth integration between the frontend, backend, and LLM-based functionalities. | 8th November 2025 |
| Review 4 | To submit the final project report along with all required supporting materials, including a project poster, deployment files, and source code ensuring the project is well-documented and presentation-ready. | 5th December 2025 |

**Performance Measures:**

**Evaluation Framework for Comparing LLM APIs:**

The team will evaluate different APIs such as Grok, Claude, Gemini, and GPT on a set of carefully designed prompts. To make this evaluation systematic, the team will build a matrix of prompts × models, and rated each output based on multiple performance measures.

**Matrix Setup:**

* Rows: Different prompts/tasks relevant to our project, such as *code refactoring, documentation generation, code comprehension, and interview question creation*.
* Columns: Different models (Grok, Claude, Gemini, GPT, etc. and their versions/releases).
* Cells: The objective and subjective scores of each model for the respective prompt.

This structure allows the team to make direct, side-by-side comparisons of performance.

Since the project focuses on automated codebase refactoring and comprehension, we have defined performance measures in two categories:

1. **Automatic Metrics:**

* **BLEU / ROUGE** – For comparing generated documentation against a set reference text and generating a score.
  + BLEU (Bilingual Evaluation Understudy) measures how much the machine-generated text overlaps with a reference text at the *n-gram* (word sequence) level.
  + ROUGE (Recall-Oriented Understudy for Gisting Evaluation) → measures how much of the reference text’s content is captured in the model’s output.
* **Code Quality Linters** – To measure how clean or standardized the refactored code is (using tools like Pylint, ESLint, etc.). They evaluate style, formatting, and adherence to best practices and score.
* **Test Case Accuracy** – To verify correctness of outputs on provided test cases. Validates functional correctness of the refactored code. Scores on how well logic is preserved.

**b) Human Evaluation Rubric (Subjective Quality):** Human volunteers independently rate outputs on a **Likert scale (1–5)** for:

* Correctness
* Readability
* Relevance of the generated comment
* Relevance of the generated interview questions
* Conciseness

An average of this score will be considered for evaluation a particular matrix combination.