Requirement Document

Crowdsourcing Code Comments with ChatGPT (CrowdDoc)

CSE 4101 Senior Project - Fall 2026

Ahmed Alzahrani and Mohammed Zakry

Faculty Advisor / Client: Professor Marius Silaghi

1. Purpose and Scope

CrowdDoc is a website for reading open-source C code with explanations saved beside the code. A user can browse a known source tree, open a function, read an AI Writer explanation, see a second AI Checker / Reviewer result, and add human corrections or votes. The original source code stays read-only.

The current Milestone 1 prototype uses a small snapshot from Linux v6.12. It contains three C files (lib/bcd.c, lib/string.c, and lib/math/int_sqrt.c) and eleven selected functions. The goal later is to support a larger source tree, but the current documents should match what we really have now.

Terminology note: in the current website, the second AI agent is labeled Reviewer. In these documents it is called Checker / Reviewer, meaning the same second AI agent.

2. Users

- Reader: browses files, opens a function, and reads explanations.
- Contributor: adds a human correction or addition and can vote.
- Reviewer or expert: checks explanations and may be identified as trusted in a later version.
- Project administrator: maintains the source snapshot, website, AI connection, and database.

3. Functional Requirements

FR-1: The system shall let the user browse the supported Linux v6.12 source snapshot by folders and files.

FR-2: The system shall show supported C source files as read-only code and shall not write comments into the original files.

FR-3: The system shall identify the selected C functions and allow the user to open one function at a time.

FR-4: Each function shall be tied to its repository, Linux version, revision, file path, function name, and function-content hash so comments stay with the correct source version.

FR-5: For a selected function, the Writer AI shall create the first explanation using the fields purpose, inputs, outputs, and side effects.

FR-6: After the Writer result is saved, the Checker / Reviewer shall receive the same function and that exact Writer result. The Checker / Reviewer shall say whether it finds a problem and may suggest an improved explanation.

FR-7: If live AI is unavailable, CrowdDoc may show prepared stand-ins, but the page shall clearly say that they are stand-ins and not live model output.

3. Functional Requirements (continued)

FR-8: A user shall be able to submit a human correction or addition to an AI explanation.

FR-9: A human correction shall stay linked to the same function and to the AI explanation it is answering.

FR-10: A user shall be able to vote for an explanation on the selected function.

FR-11: The system shall show when a function has review activity such as a human comment or votes. A vote count shall not be treated as proof that an explanation is correct.

FR-12: Functions, explanations, and votes shall be saved so they can be loaded again later.

FR-13: The function page shall provide an external Elixir/Bootlin link to the matching Linux v6.12 source when available.

4. User Interface Requirements

IR-1: The home page shall identify CrowdDoc and provide a clear way to start browsing the connected source snapshot.

IR-2: Folder and file pages shall show the current path and breadcrumbs so the user knows where they are.

IR-3: The function page shall show the read-only source together with the Writer, Checker / Reviewer, and human explanations for that function.

IR-4: AI Writer, Checker / Reviewer, stand-in, and human explanations shall have clear text labels. The meaning shall not depend only on color.

IR-5: The page shall provide a simple form for adding a correction or addition.

IR-6: The page shall provide a clear voting control.

IR-7: Errors shall be shown in simple language instead of a server stack trace.

5. System Interface Requirements

The current project uses Next.js and TypeScript for the web application. Tree-sitter C is used to find C functions, SQLite is used for saved data, and the OpenAI API can be used for live Writer and Checker / Reviewer calls.

SI-1: The server shall read the saved Linux v6.12 source snapshot from the application data.

SI-2: The C parser shall extract the function name, source text, signature, line range, and content hash for supported functions.

SI-3: SQLite shall store the current prototype's function records, explanations, and votes.

SI-4: When live AI is enabled, OpenAI requests shall be made from the server. The API key shall not be sent to the browser.

SI-5: External source links shall point to Linux v6.12 in Elixir/Bootlin.

6. Performance and Quality Requirements

PR-1: Browsing the small source snapshot and opening a saved function should feel responsive on a normal laptop. Actual response time shall be recorded during testing instead of being guessed.

PR-2: A live Writer or Checker request may take longer than a normal page load, so the interface should show that work is in progress.

PR-3: AI accuracy shall be checked against the source code with human review. The Checker / Reviewer agreeing with the Writer does not automatically prove correctness.

PR-4: The saved Linux source files shall remain unchanged when explanations or votes are saved.

PR-5: Source browsing, parsing, AI calls, database work, and user actions should stay separated into understandable modules or functions.

PR-6: User-visible pages and errors shall not reveal the OpenAI API key or other private server values.

7. Error and Unusual Input Requirements

- Unsafe path parts such as '..' or backslashes shall not be allowed to leave the saved source tree.
- A human correction must point to an AI explanation from the same function.
- Purpose, inputs, outputs, and side effects must all contain text before a human explanation is saved.
- The current prototype limits each human explanation field to 4000 characters.
- A vote with an invalid function/explanation relationship shall not be saved.
- If the Writer succeeds but the Checker / Reviewer fails, the saved Writer result should remain available.

8. Data That Must Be Kept

- Function data: repository, revision, tag, path, function name, content hash, source text, line range, and source link.
- Explanation data: function ID, AI or human type, Writer and Checker / Reviewer role, parent explanation when used, four explanation fields, author label, and creation time.
- Vote data: function ID, explanation ID, and creation time.

9. Current Scope and Planned Work

The current working demo contains three Linux v6.12 C files and eleven selected functions. It already has source browsing, function pages, Writer and Checker logic, human corrections/additions, voting, SQLite storage, and external source links. When no OpenAI key is available, the demo uses labeled stand-in examples.

The current demo does not yet import an arbitrary full repository. It also does not use sign-in for Milestone 1 and does not yet have reliable user identity, expert verification, or one-vote-per-user protection. These are planned improvements and should not be described as completed features.

10. Questions for Client Review

- Are these requirements reasonable for the final project?
- Which requirements should be changed, added, or removed?
- Which requirements should have the highest priority next?