Use Case Discription

Sprint 1 version 1

Use Case 1: Submit PR

Actor: Developer

Goal: To submit a Pull Request for code changes.

Pre-conditions:

- Developer has a valid GitHub repository.
- Code changes are committed locally.
- Internet connection is active.

Main Flow:

1. Developer selects the option to create a Pull Request.
2. System collects branch details and commit information.
3. System creates a Pull Request on GitHub.
4. Confirmation is shown to the Developer that PR is successfully submitted.

Post-conditions:

- A new Pull Request is created on GitHub.
- PR ID and metadata are stored in the system.

Alternate Flow:

- **1A. Missing commit / branch conflict:**
    - System shows an error message.
    - Developer resolves conflicts and retries submission.

Use Case 2: Fetch PR Diff

Actor: GitHub

Goal: Provide the code difference (diff) for the submitted Pull Request.

Pre-conditions:

- A valid PR ID exists.
- GitHub API is reachable.

Main Flow:

1. System requests diff details from GitHub using PR ID.
2. GitHub validates the request.
3. GitHub returns the PR diff.
4. System stores the diff for review generation.

Post-conditions:

- Full PR diff is available for further review generation.

Alternate Flow:

- **1A. GitHub server not reachable:**
    - System retries or shows an error message.

- **2A. Invalid PR ID:**
    - GitHub returns a failure response.
    - System logs the issue and informs the user.

Use Case 3: Generate Review

Actor: Groq / LLM

Goal: Automatically generate a code review based on the PR diff.

Pre-conditions:

- PR diff has already been fetched
- LLM API key/configuration is available.

Main Flow:

1. System sends the PR diff to Groq/LLM.
2. LLM processes the code and analyzes changes.
3. LLM generates structured review comments.
4. System receives and temporarily stores the generated review.

Post-conditions:

- Review text is generated and ready to be posted to GitHub.

Alternate Flow:

- **1A. LLM API rate limit / timeout:**
    - System retries after waiting.
- **2A. LLM returns incomplete response:**
    - System asks LLM again or shows error.

- **3A. Diff too large:**
    - System chunks the diff and retries generation.

Use Case 4: Post Review

Actors: GitHub, Groq/LLM

Goal: Post the automatically generated review back to the GitHub PR.

Pre-conditions:

- Review is generated using LLM.
- Valid GitHub access token exists.
- PR is still open.

Main Flow:

1. System triggers <<include>> Generate Review.
2. System prepares the final review payload.
3. System calls GitHub API to post the review.
4. GitHub validates the request.
5. Review is successfully added to PR.
6. System notifies the Developer and Senior Developer.

Post-conditions:

- Review comment is visible on GitHub PR.

Alternate Flow:

- **3A. GitHub API authentication fails:**
    - System asks user to re-authenticate.
- **4A. GitHub PR is closed:**
    - System aborts posting.
- **5A. Review posting partially fails:**
    - System retries posting or logs an error.

Use Case 5: View Review

Actors: Developer, Senior Developer

Goal: To view the automatically generated review for the PR.

Pre-conditions:

- A review exists in the system or on GitHub.
- User is authenticated.

Main Flow:

1. Developer/Senior Developer opens the PR review section.
2. System fetches and displays the review.
3. User reads, evaluates, and may take action on suggestions.

Post-conditions:

- Review is successfully viewed.

Alternate Flow:

- **1A. No review available:**
  - System shows “Review not available yet.”
- **2A. Review fetch fails:**
  - System retries or displays an error.

Sprint 2 version_1.1

Use Case 1: Submit PR

Actor: Developer

Goal: Submit a Pull Request for review.

Pre-conditions:

- Developer has prepared code and created a branch.
- GitHub repository is configured.

Main Flow:

1. Developer initiates “Submit PR”.
2. System collects branch & commit information.
3. System sends request to GitHub API.
4. GitHub creates the PR.
5. Confirmation is shown to the Developer.

Post-conditions:

- A new PR exists on GitHub and its PR ID is stored.

Alternate Flow:

- **GitHub rejects PR creation** → Show error and retry.

Use Case 2: Fetch PR Diff

Actor: GitHub

Goal: Provide the diff for the submitted PR.

Pre-conditions:

- A valid PR ID exists.
- GitHub API is reachable.

Main Flow:

1. System requests PR diff from GitHub.
2. GitHub validates the request.
3. GitHub returns the code diff.
4. System stores the diff for prompt selection & review generation.

Post-conditions:

- PR diff is stored in the system

Alternate Flow:

- **Invalid PR ID** → System logs and shows error.
- **API unreachable** → System retries or reports failure.

Use Case 3: Select Best Prompt (avg. score)

Actor: System (version_1.1)

Goal: Automatically select the most effective prompt based on average scoring metrics (e.g., accuracy, completeness, coherence).

Pre-conditions:

- PR diff is already fetched.
- Multiple candidate prompts exist.
- System has evaluation metrics (stored from previous runs or ratings).

Main Flow:

1. System loads all available prompts.
2. System retrieves evaluation scores (avg. performance) for each prompt.
3. System compares and ranks the prompts based on average score.
4. System selects the highest-scoring prompt.
5. Selected prompt is passed to “Generate Review”.

Post-conditions:

- The best-performing prompt is selected for review generation.

Alternate Flow:

- **No scoring data exists:**
  - Use default fallback prompt.
- **Tie between prompts:**
  - Select the most recent or system-preferred prompt.
- **Prompt configuration missing:**
  - System logs error and stops review generation.

Use Case 4: Generate Review

Actor: Groq / LLM

Goal: Generate AI-based review comments using the selected best prompt.

Pre-conditions:

- PR diff is fetched.
- Best prompt is selected.
- LLM is online and API credentials valid.

Main Flow:

1. System triggers <<include>> Select Best Prompt.
2. System sends PR diff + selected prompt to Groq/LLM.
3. LLM processes the code and creates review comments.
4. System stores the generated review.

Post-conditions:

- Review text is ready for posting.

Alternate Flow:

- **LLM timeout / failure:** Retry or show error.
- **LLM produces incomplete review:** Ask LLM again.
- **Diff too large:** System splits diff and retries.

Use Case 5: Post Review

Actors: GitHub, System(version_1.1)

Goal: Post the generated review back to the PR on GitHub.

Pre-conditions:

- Review is generated using <<include>> Generate Review.
- GitHub access token exists.

- PR is open.

Main Flow:

1. System triggers <<include>> Select Best Prompt (avg. score).
2. System triggers <<include>> Generate Review using the best prompt.
3. System prepares review payload
4. System submits the review comment to GitHub.
5. GitHub validates and posts the comment.
6. Developer and Senior Developer are notified.

Post-conditions:

- Review appears on GitHub PR.

Alternate Flow:

- **GitHub authentication fails:** Ask user to re-authenticate.
- **PR closed:** Abort posting.
- **Partial posting failure:** Retry or notify error.

Use Case 6: View Review

Actors: Developer, Senior Developer

Goal: View the posted AI-generated review.

Pre-conditions:

- Review is posted to GitHub or stored in the system.
- User is authenticated.

Main Flow:

1. The Developer/Senior Developer opens the PR review section.
2. System fetches review details.
3. Review comments are displayed.

Post-conditions:

- Review is successfully viewed.

Alternate Flow:

- **Review not yet generated:** Show “Review not available”.
- **System unable to fetch review:** Retry or show error.

Sprint 2 version 1.2

Use Case 1: Submit PR

Actor: Developer

Goal: Submit a Pull Request for code review**.**

Pre-conditions:

- The developer has committed changes.
- GitHub repo and branch are valid.

Main Flow:

1. The developer chooses “Submit PR”.
2. System gathers branch/commit info.
3. The system sends a PR creation request to GitHub.
4. GitHub creates a PR and returns the PR ID.
5. Confirmation is shown to the Developer.

Post-conditions:

- PR successfully created on GitHub.
- PR ID stored in system.

Alternate Flow:

- GitHub rejects request → Show error and retry.

Use Case 2: Fetch PR Diff

Actor: GitHub

Goal: Provide PR diff so system can perform automatic review**.**

Pre-conditions:

- Valid PR ID exists.
- GitHub API reachable.

Main Flow:

1. System requests diff using PR ID.
2. GitHub validates request.
3. GitHub sends PR diff.

4. System stores diff.

Post-conditions:

- Diff is available for prompt selection and review generation.

Alternate Flow:

- Invalid PR ID → Error shown.
- GitHub down → Retry or notify user.

Use Case 3: Select Best Prompt (Iterative)

Actor: System (version_1.2)

Goal: Iteratively evaluate prompts and select the best-performing prompt.

Pre-conditions:

- PR diff is already fetched.
- System has a set of candidate prompts.
- Evaluation metrics exist (e.g., relevance, correctness, completeness).

Main Flow:

1. System loads the list of available prompts.
2. For each prompt:
   a. System sends diff + prompt to LLM.
   b. LLM generates a trial review.
   c. System evaluates the trial review using scoring metrics.
3. System compares all prompt scores.
4. System selects the prompt with highest performance.
5. Selected prompt passed to Generate Review.

Post-conditions:

- Best prompt (after iterative testing) is selected.

Alternate Flow:

- No prompts available → Use default prompt.
- LLM fails for a specific prompt → Skip that prompt and continue.
- All prompts fail → Abort review generation and notify user.

Use Case 4: Generate Review

Actor: Groq/LLM

Goal: Generate final AI review using the selected best prompt.

Pre-conditions:

- Diff fetched.
- Best prompt selected via Select Best Prompt (Iterative).
- LLM API credentials valid.

Main Flow:

1. System triggers <<include>> Select Best Prompt (Iterative).
2. System sends diff + chosen prompt to Groq/LLM.
3. LLM analyzes code changes.
4. LLM returns final structured review.
5. System stores the review.

Post-conditions:

- Review is ready to be posted.

Alternate Flow:

- LLM timeout → Retry.
- LLM returns incomplete review → Regenerate.
- Diff too large → Split and process iteratively.

Use Case 5: Post Review

Actors: GitHub, System (version_1.2)

Goal: Automatically post the AI-generated review on the GitHub PR.

Pre-conditions:

- Review generated via LLM.
- GitHub token valid.
- PR is open.

Main Flow:

1. System calls <<include>> Select Best Prompt (Iterative).
2. System calls <<include>> Generate Review with chosen prompt.
3. System prepares the final comment payload.
4. System sends the review payload to GitHub.
5. GitHub validates and posts the comment.
6. System notifies Developer and Senior Developer.

Post-conditions:

- Review appears on GitHub PR.

Alternate Flow:

- GitHub authentication failure → Prompt for re-authentication.
- PR closed → Abort posting.
- Partial failure → Log error or retry**.**

Use Case 6: View Review

Actors: Developer, Senior Developer

Goal: View the review posted on the PR.

Pre-conditions:

- Review exists in GitHub or in the system.

Main Flow:

1. Developer/Senior Developer selects “View Review”.
2. System fetches review comments.
3. System displays all review details.

Post-conditions:

- Review is successfully shown to the user.

Alternate Flow:

- Review not generated yet → Show “Review pending”.
- Fetch failure → Retry or show error.

For sprint 2 version_2

Use Case 1: Submit PR

Actor: Developer

Goal: Submit a Pull Request for code review.

Pre-Conditions:

- Developer has valid GitHub repository access.
- Code changes are committed to a branch.

Main Flow:

1. Developer selects “Submit PR”.
2. System collects branch & commit info.
3. System sends PR creation request to GitHub.
4. GitHub creates the PR and assigns an ID.
5. System notifies Developer that PR is submitted.

Post-Conditions:

- New PR exists on GitHub.
- PR ID stored.

Alternate Flows:

- GitHub unavailable → retry / show error.

Use Case 2: Fetch PR Diff

Actor: GitHub

Goal: Provide PR diff for further analysis.

Pre-Conditions:

- Valid PR ID exists.
- GitHub API reachable.

Main Flow:

1. System requests diff from GitHub.
2. GitHub verifies the request.
3. GitHub returns the PR diff.
4. System stores the diff.

Post-Conditions:

- PR diff stored for next steps (Static Analysis + Review Generation).

Alternate Flows:

- Invalid PR ID → return error.
- GitHub API fails → retry or abort.

Use Case 3: Static Analysis & Linters

Actor: System (version 2)

Goal: Automatically analyze PR code using static analysis tools and linters to detect stylistic, syntactic, and logical issues.

Pre-Conditions:

- PR diff is fetched.
- Static analysis tools/linter configurations exist.

Main Flow:

1. System loads PR diff.
2. System runs configured static analysis tools (e.g., flake8, eslint).
3. System collects warnings, errors, violations.
4. System formats results into structured output.
5. Output becomes part of review generation and posting.

Post-Conditions:

- Analysis output (warnings/errors/lints) is stored.
- These results can **extend** the View Review use case.

Alternate Flows:

- Linter configuration missing → skip analysis.
- Tool crashes → log error and proceed with partial results.

Use Case 4: Generate Review

Actor: Groq/LLM

Goal: Generate AI review based on PR diff and analysis outputs.

Pre-Conditions:

- PR diff fetched.
- Static Analysis results available (optional).
- LLM credentials valid.

Main Flow:

1. System sends PR diff + any static analysis summary to Groq/LLM.
2. LLM analyzes code and comments.
3. LLM generates structured review text.
4. System stores generated review.

Post-Conditions:

- Review text is ready to be posted.

Alternate Flows:

- LLM timeout → retry.
- LLM produces incomplete response → regenerate.

Use Case 5: Post Review

Actors: GitHub, System (version 2)

Goal: Post both static analysis results and AI-generated review to the PR.

Pre-Conditions:

- Review generated.
- GitHub token valid.
- PR open.

Main Flow:

1. System runs <<include>> Static Analysis & Linters.
2. System runs <<include>> Generate Review.
3. System prepares final review payload (AI review + analysis warnings).
4. System sends payload to GitHub.
5. GitHub posts review comment.
6. Developer/Senior Developer notified.

Post-Conditions:

- Review + static analysis comments appear on GitHub.

Alternate Flows:

- Auth failures → request new token.
- PR closed → abort.

Use Case 6: View Review

Actors: Developer, Senior Developer

Goal: View the full review including AI comments and static analysis results.

Relationship:

- **Extends:** Static Analysis & Linters (because viewing may display linter/analysis output)

Pre-Conditions:

- Review exists on GitHub or in system storage.

Main Flow:

1. Developer/Senior Developer selects “View Review”.
2. System fetches review details.
3. System displays AI review comments.
4. System displays static analysis results (if available).

Post-Conditions:

- User views full review summary.

Alternate Flows:

- Review not generated yet → Show message “Review pending.”
- Fetch failure → retry or show error.

Sprint 3

Use Case 1: Submit PR

Actor: Developer

Goal: Submit pull request for code review.

Pre-conditions:

- Code is committed to a branch.
- Developer has permission to create PRs.
- GitHub is reachable.

Main Flow:

1. Developer selects “Submit PR.”
2. System collects branch/commit info.
3. System requests GitHub to create PR.
4. GitHub creates PR and returns PR ID.
5. System confirms PR submission.

Post-conditions:

- PR is created on GitHub.
- PR ID stored.

Alternate Flow:

- GitHub API unreachable → show error and retry.

Use Case 2: Fetch PR Diff

Actor: GitHub

Goal: Provide code diff for PR.

Pre-conditions:

- Valid PR exists.

- GitHub API accessible.

Main Flow:

1. System requests PR diff from GitHub.
2. GitHub validates the request.
3. GitHub returns the diff.
4. System stores diff.

Post-conditions:

- PR diff available for RAG pipeline.

Alternate Flow:

- Invalid PR ID → error.
- PR closed → abort diff fetching.

Use Case 3: Extract Repo

Actors: GitHub, System (Version 1)

Goal: Extract or clone the repository for RAG processing.

Pre-conditions:

- Repository URL exists.
- System has permissions to access repo.
- PR diff fetched.

Main Flow:

1. System requests repository content from GitHub.
2. GitHub sends repo contents (or clone link).
3. System downloads/extracts repo files.
4. System prepares files for indexing.

Post-conditions:

- Repository content is available locally.

Alternate Flow:

- GitHub rate limit → retry later.
- Repo too large → partial extraction.

Use Case 4: Repo Indexing

Actor: RAG Engine

Goal: Convert repository into searchable vector embeddings for context retrieval.

Pre-conditions:

- Repo extracted successfully.
- File formats supported by indexer.

Main Flow:

1. System sends repo files to RAG Engine.
2. RAG Engine chunks source files.
3. RAG Engine generates embeddings.
4. Engine stores embeddings in vector DB.

Post-conditions:

- Repository fully indexed for context retrieval.

Alternate Flow:

- Unsupported file → skip file and continue.
- Indexer fails → retry or abort.

Use Case 5: Retrieve Code Context

Actors: RAG Engine, LLM

Goal: Retrieve relevant code sections to provide contextual review to LLM**.**

Pre-conditions:

- Repo is indexed.
- PR diff fetched
- RAG Engine is online.

Main Flow:

1. System provides PR diff to RAG Engine.
2. RAG Engine searches vector DB.
3. RAG Engine returns relevant code context.
4. System forwards context to LLM for use in review generation.

Post-conditions:

- Relevant contextual code is available.

Alternate Flow:

- No relevant context found → fallback to diff only.
- RAG Engine unreachable → retry or notify.

Use Case 6: Generate Review

Actors: LLM

Goal: Generate a high-quality review using PR diff + retrieved context.

Pre-conditions:

- PR diff available.
- Code context retrieved via RAG.
- LLM credentials valid.

Main Flow:

1. System triggers <<include>> Retrieve Code Context.
2. System sends PR diff + context to LLM.
3. LLM analyses combined inputs.
4. LLM generates detailed review comments.
5. System stores generated review.

Post-conditions:

- Review text ready for posting.

Alternate Flow:

- LLM timeout → retry.
- Context too large → send summarized context.

Use Case 7: Post Review

Actors: GitHub, System (Version 1)

Goal: Post RAG-enhanced review to GitHub PR.

Pre-conditions:

- Review generated.
- GitHub token valid.
- PR open.

Main Flow:

1. System triggers <<include>> Extract Repo (if needed).
2. System triggers <<include>> Generate Review.
3. System prepares final review payload.
4. System sends review to GitHub.
5. GitHub posts review.
6. System notifies developers.

Post-conditions:

- Review appears on GitHub PR.

Alternate Flow:

- PR closed → abort.
- GitHub posting fails → retry.

Use Case 8: View Review

Actors: Developer, Senior Developer

Goal: Allow user to view AI-generated + RAG-enhanced review.

Pre-conditions:

- Review already posted or generated.

Main Flow:

1. Developer/Senior Developer selects “View Review.”
2. System fetches review from storage/GitHub.
3. System displays LLM-generated review.
4. (If available) System also shows any contextual references retrieved via RAG.

Post-conditions:

- User sees complete review output.

Alternate Flow:

- Review missing → show “No review available.”
- GitHub fetch error → retry.

Sprint 4 version 1.3

Use Case 1: Submit PR

Actor: Developer

Goal: Submit pull request for review.

Pre-conditions:

- Developer has committed code to a branch.
- Developer has GitHub repo access & network availability.

Main Flow:

1. Developer selects “Submit PR”.
2. System gathers branch & commit details.
3. System requests GitHub to create the PR.
4. GitHub creates the PR and returns PR ID.
5. System confirms PR submission.

Post-conditions:

- PR is created and stored in the system for further actions.

Alternate Flow:

- GitHub unreachable → system notifies developer.

Use Case 2: Fetch PR Diff

Actor: GitHub

Goal: Retrieve PR diff for downstream review generation.

Pre-conditions:n

- PR ID exists.
- GitHub API is accessible

Main Flow:

1. System sends request to GitHub for PR diff.
2. GitHub validates request.
3. GitHub returns diff content.
4. System stores diff for prompt selection and review generation.

Post-conditions:

- PR diff is stored locally

Alternate Flow:

- Invalid PR ID → show error.
- GitHub fails → retry or abort.

Use Case 3: Select Best Prompt (online estimation)

(NEW in version_1.3)
(Included by Generate Review and Post Review)

Actor: System (version 1.3)

Goal:

Dynamically estimate the best prompt in real-time based on **online scoring metrics** (e.g., response quality, latency, coherence).

Pre-conditions:

- PR diff is available.
- A set of prompts is pre-configured.
- Online scoring mechanism exists.

Main Flow:

1. System loads all prompts.
2. System sends sample queries or partial PR diff to LLM for testing.
3. System evaluates each prompt's performance **in real time** using online metrics.
4. Metrics include:
    - relevance
    - correctness
    - LLM confidence
    - runtime/latency
5. System selects the prompt with the best **online estimated score**.
6. Selected prompt is passed to Generate Review.

Post-conditions:

- Best-performing prompt identified using real-time estimation.

Alternate Flow:

- LLM fails for some prompts → skip those prompts.
- No prompt exceeds threshold → fallback to default prompt.
- Online estimator fails → use average historical score

Use Case 4: Generate Review

Actors: Groq/LLM

Goal: Generate review comments for PR using selected online-estimated prompt.

Pre-conditions:

- Diff fetched.
- Best prompt selected.
- LLM reachable.

Main Flow:

1. System triggers Select Best Prompt (online estimation).
2. System sends PR diff + chosen prompt to LLM.
3. LLM analyzes code changes.
4. LLM generates structured review comments.

5. System stores the generated review.

Post-conditions:

- Review is ready to be posted.

Alternate Flow:

- LLM timeout → retry automatically.
- Review quality too low → regenerate with second-best prompt.
- Request exceeds token limits → send chunked diff.

Use Case 5: Post Review

Actors: GitHub, System (version 1.3)

Goal: Post the AI-generated review onto GitHub PR.

Pre-conditions:

- Review text generated.
- GitHub token valid.
- PR open.

Main Flow:

1. System calls Select Best Prompt (online estimation).
2. System generates the review text.
3. System formats the review payload.
4. System calls GitHub API to post the review.
5. GitHub validates and posts the comment.
6. System notifies Developer and Senior Developer.

Post-conditions:

- AI review is posted on GitHub successfully.

Alternate Flow:

- GitHub API fails → retry.
- PR closed → abort posting.

Use Case 6: View Review

Actors: Developer, Senior Developer

Includes: <<include>> Generate Review (optional on-demand)

Goal: Allow users to view the review generated for the PR.

Pre-conditions:

- Review is available on GitHub or system.

Main Flow:

1. Developer/Senior Developer selects “View Review.”
2. System fetches review from GitHub or local storage.
3. System displays the AI-generated review comments.

Post-conditions:

- User views the latest review content.

Alternate Flow:

- No review available → show “Review pending.”
- GitHub fetch error → retry.

Sprint 4 version_1.2_rag

Use Case 1: Submit PR

Actor: Developer

Goal: Submit a pull request for review.

Pre-conditions:

- Code committed to branch.
- GitHub access available.

Main Flow:

1. Developer selects “Submit PR”.
2. System collects branch & commit info.
3. System sends PR creation request to GitHub.
4. GitHub creates PR and returns PR ID.
5. System confirms PR submission.

Post-conditions:

- PR exists on GitHub.

Alternate Flow:

- GitHub unreachable → retry or show error.

Use Case 2: Fetch PR Diff

Actor: GitHub

Goal: Provide the PR's code dif**f.**

Pre-conditions:

- Valid PR ID.
- GitHub API accessible.

Main Flow:

1. System requests PR diff.
2. GitHub validates request.
3. GitHub returns the diff.
4. System stores diff.

Post-conditions:

- PR diff stored for analysis + RAG.

Alternate Flow:

- Invalid PR ID → show error.
- API failure → retry.

Use Case 3: Static Analysis

Actor: System (Version 1.2)

Goal: Analyze code with linters/static analysis tools.

Pre-conditions:

- PR diff available.
- Linter configuration exists.

Main Flow:

1. System loads PR diff.
2. System runs static analysis tools.
3. Tools identify warnings/errors.
4. System stores results.

Post-conditions:

- Static analysis report ready for review.

Alternate Flow:

- Linter failure → skip static analysis.
- Partial results → continue with what is available.

Use Case 4: Extract Repo

Actors: GitHub, System (Version 1.2)

Goal: Retrieve repository content for RAG processing.

Pre-conditions:

- Repo exists.
- PR diff fetched.

Main Flow:

1. System sends request to GitHub.
2. GitHub returns repo files/clone link.
3. System extracts repo content.
4. Repo prepared for indexing.

Post-conditions:

- Repo extracted locally.

Alternate Flow:

- Repo too large → partial extraction.

Use Case 5: Context Retrieval

(High-level RAG step before retrieving vectors)

Actor: System (Version 1.2**)**

Goal: Prepare inputs and trigger retrieval from vector DB.

Pre-conditions:

- Repo extracted.
- Indexing available.
- PR diff available.

Main Flow:

1. System prepares retrieval query from PR diff.
2. System sends query to RAG Engine.
3. RAG Engine starts searching relevant sections.

Post-conditions:

- Retrieval query is sent to RAG Engine.

Alternate Flow:

- No repository index → fallback to diff only.

Use Case 6: Retrieve Code Context

Actors: RAG Engine

Goal: Retrieve relevant code embeddings/contexts for LLM.

Pre-conditions:

- Repo indexed by vector database.
- Retrieval query exists.

Main Flow:

1. RAG Engine searches vector DB using embeddings.
2. Engine selects top relevant code chunks.
3. Engine returns contextual code sections to system.
4. System stores context for review generation.

Post-conditions:

- Context retrieved for prompt selection + LLM

Alternate Flow:

- No relevant context → send empty context.

Use Case 7: Select Best Prompt (Iterative)

Actor: System (Version 1.2)

Goal:

Iteratively evaluate prompts and pick the best performer.

Pre-conditions:

- Context retrieved.
- PR diff available.

- Multiple prompts available.

Main Flow:

1. System loads prompt set.
2. For each prompt:
   a. System tests prompt with partial diff or sample.
   b. LLM gives mini-responses.
   c. System scores responses using metrics.
3. System compares all scores.
4. System selects highest scoring prompt.
5. Best prompt passed to Generate Review.

Post-conditions:

- Best prompt identified.

Alternate Flow:

- All prompts fail → use default prompt

Use Case 8: Generate Review

Actor: LLM

Goal:

Generate AI-assisted review using diff + context + best prompt.

Pre-conditions:

- Best prompt selected.
- LLM online.

Main Flow:

1. System triggers Retrieve Code Context.
2. System triggers Select Best Prompt (Iterative).
3. System sends diff + context + best prompt to LLM.
4. LLM generates review.
5. System stores review.

Post-conditions:

- Review ready for posting.

Alternate Flow:

- LLM timeout → retry.

Use Case 9: Post Review

Actors: GitHub, System (Version 1.2)

Goal: Post full review (AI + RAG + static analysis) to GitHub PR.

Pre-conditions:

- Review generated.
- Static analysis results available.
- PR open.

Main Flow:

1. System runs Static Analysis.
2. System selects best prompt.
3. System generates review.
4. System prepares final review output.
5. System sends output to GitHub.
6. GitHub posts the review.

Post-conditions:

- Review visible on GitHub PR.

Alternate Flow:

- GitHub failure → retry.

Use Case 10: View Review

Actors: Developer, Senior Developer

Goal: View the final combined review.

Pre-conditions:

- Review posted or stored.

Main Flow:

1. Developer/Senior Developer selects “View Review.”
2. System fetches the review.
3. System displays:
    - AI review
    - Static analysis results

    - RAG-based context references

4. User reads review.

Post-conditions:

- Review fully displayed.

Alternate Flow:

- Review not available → show message

Version 1.3_rag

Use Case 1: Submit PR

Actor: Developer

Goal: Submit a pull request for review.

Pre-conditions:

- Developer has committed code to a branch
- GitHub reachable and developer authenticated.

Main Flow:

1. Developer selects “Submit PR.”
2. System collects relevant branch & commit information.

3. System sends request to GitHub to create PR.
4. GitHub creates PR and returns PR ID.
5. System confirms PR creation.

Post-conditions:

- A new PR exists on GitHub.

Alternate Flow:

- GitHub unreachable → system displays error.

Use Case 2: Fetch PR Diff

Actor: GitHub

Goal: Provide the PR code diff for analysis and review generation.

Pre-conditions:

- PR ID exists.
- GitHub API reachable.

Main Flow:

1. System requests code diff from GitHub.
2. GitHub validates request.
3. GitHub returns diff.
4. System stores PR diff.

Post-conditions:

- Diff available for RAG + prompt estimation + review generation.

Alternate Flow:

- Invalid PR ID → system shows error.
- API timeout → retry or abort.

Use Case 3: Static Analysis

Actor: System (version 1.3)

Goal: Analyze code using linters or static analysis tools.

Pre-conditions:

- PR diff available.
- Static analysis tools configured.

Main Flow:

1. System loads PR diff.
2. System runs static analysis tools.
3. Tools detect warnings/errors.
4. System stores results.

Post-conditions:

- Static analysis report ready to include in review.

Alternate Flow:

- Tool failure → partial output or skip.

Use Case 4: Extract Repo

Actors: GitHub, System (version 1.3)

Goal: Retrieve complete repository for RAG processing.

Pre-conditions:

- Repo exists and accessible.
- PR diff fetched.

Main Flow:

1. System requests repo content from GitHub.
2. GitHub returns repo data or clone link.
3. System extracts repository files locally.
4. Repo prepared for context retrieval pipeline.

Post-conditions:

- Local repository copy available.

Alternate Flow:

- Repo large → partial extraction.

Use Case 5: Context Retrieval

Actor: System (version 1.3)

Goal: Prepare a query from PR diff and trigger RAG retrieval.

Pre-conditions:

- Repo extracted.
- PR diff available.

Main Flow:

1. System analyzes diff to generate retrieval query.
2. System sends query to RAG Engine.
3. RAG Engine begins retrieval.

Post-conditions:

- Retrieval process triggered.

Alternate Flow:

- Index missing → fallback to diff only.

Use Case 6: Retrieve Code Context

Actor: RAG Engine

Goal: Find relevant code context to help LLM understand PR changes.

Pre-conditions:

- Repo indexed.
- Query prepared.

Main Flow:

1. RAG Engine searches vector DB.
2. Engine retrieves relevant code chunks.
3. Engine returns ranked context list.
4. System stores received context.

Post-conditions:

- Context ready for online prompt selection + LLM.

Alternate Flow:

- No results → default to diff only.

Use Case 7: Select Best Prompt (online)

Actor: System (Version 1.3)

Goal:

Choose best prompt using **online, real-time scoring** based on immediate response quality.

Pre-conditions:

- PR diff available.
- Retrieved context available.
- Prompt library exists.

Main Flow:

1. System loads available prompts.
2. For each prompt:
   a. System tests prompt on sample code or mini-query.
   b. LLM returns mini-response.
   c. System evaluates response using online metrics:
      relevance
      - coherence
      - correctness
      - latency
3. Online estimator calculates score.
4. System selects highest-scoring prompt.
5. Selected prompt sent to Generate Review.

Post-conditions:

- Best prompt selected dynamically.

Alternate Flow:

- All prompts under-perform → use fallback prompt.
- Online estimator fails → use last known best prompt.

Use Case 8: Generate Review

Actor: LLM

Goal:

Generate a complete, context-aware, online-prompt-selected review.

Pre-conditions:

- Best prompt determined.
- Context available
- LLM reachable.

Main Flow:

1. System retrieves code context.
2. System selects best prompt (online).
3. System sends diff + context + best prompt + static analysis summary to LLM.
4. LLM generates comprehensive review output.
5. System stores review.

Post-conditions:

- Review ready for posting.

Alternate Flow:

- LLM timeout → retry.
- Review incoherent → regenerate with second-best prompt.

Use Case 9: Post Review

Actors: GitHub, System (version 1.3)

Goal:

Post final review to GitHub PR.

Pre-conditions:

- Review generated.

- GitHub token valid.
- PR still open.

Main Flow:

1. System invokes static analysis.
2. System selects best prompt.
3. System generates review.
4. System creates final review payload.
5. System sends review to GitHub.
6. GitHub posts the review.

Post-conditions:

- Review appears on GitHub.

Alternate Flow:

- GitHub API error → retry.
- PR closed → abort.

Use Case 10: View Review

Actors: Developer, Senior Developer

Goal:

View full review including static analysis + AI comments + RAG context references.

Pre-conditions:

- Review exists.

Main Flow:

1. User selects “View Review”.
2. System displays:
    - AI review
    - Static analysis results
    - Context references
3. If Senior Developer interacts:
    - Trigger extended “Select Best Prompt (online)” for inspection.

Post-conditions:

- Review fully visible to user.

Alternate Flow:

- Review not generated → show “Not available”.

SPRINT 5

Use Case 1: Login

Actor: Developer

Secondary Actor: GitHub Authentication

Goal:

Authenticate Developer using GitHub login to access dashboard features.

Pre-Conditions:

- Developer has a valid GitHub account.
- Internet connection is active.

Main Flow:

1. Developer selects “Login”.
2. System redirects to GitHub OAuth authentication.
3. Developer approves access.
4. GitHub Authentication verifies credentials.
5. System logs developer into dashboard.

Post-Conditions:

- Developer is authenticated and session is created.

Alternate Flow:

- Invalid GitHub credentials → show error.
- User cancels login → return to login page.

Use Case 2: View Dashboard

Actor: Developer

Goal:

Display summary overview of repository and PR statistics.

Pre-Conditions:

- Developer is logged in.

Main Flow:

1. Developer selects “View Dashboard.”

2. System loads summary metrics.
3. System gathers:
    - Total PR
    - Acceptance Rate
    - Active Repo count
    - Currently Open PRs
4. System displays dashboard overview.

Post-Conditions:

- Developer sees summarized project status.

Alternate Flow:

- API rate limit → partial dashboard loaded.

Use Case: Total PR

Included By: View Dashboard

Goal:

Retrieve total number of PRs across repos.

Main Flow:

1. System requests total PR count from GitHub.
2. GitHub returns the count.
3. Dashboard displays the number.

Use Case: Acceptance Rate

Ided By: View Dashboard

Goal: Calculate PR acceptance rate (merged / total).

Main Flow:

1. System retrieves merged PRs count.
2. System retrieves total PR count.
3. System calculates acceptance rate.
4. Dashboard displays the rate.

Use Case: Active Repo

Included By: View Dashboard

Goal:

Show list/count of repositories with activity.

Main Flow:

1. System checks repos with recent PRs.
2. System marks “active” repos.
3. Dashboard displays results.

Use Case: Open PR

Included By: View Dashboard

Goal:

Show number of currently open PRs.

Main Flow:

1. System queries GitHub for open PRs.
2. GitHub returns list of open PRs.
3. Dashboard displays the count.

Use Case: Filter PR

Included By: View All PRs

Goal:

Filter PRs based on selected criteria.

Main Flow:

1. Developer selects filter criteria (open/closed/repo/date).
2. System applies filters.
   System displays filtered list.

Use Case 3: View All PRs

Actor: Developer

Goal: Display all PRs across all repositories.

Pre-Conditions:

- Developer logged in.

Main Flow:

1. Developer selects “View All PRs.”
2. System retrieves PRs from all repos.
3. Includes:
    - All Repo (list of repos)

    - Filter PR (e.g., open/closed, repo-wise, date-wise)

4. System displays complete PR list.

Post-Conditions:

- Developer sees all PRs with filtering options.

Alternate Flow:

- GitHub API limit hit → System loads partial data.

Use Case: All Repo

Included By: View All PRs

Goal: Retrieve all repositories.

Main Flow:

1. System requests repo list.
2. GitHub returns repository list.
3. System displays repo names for PR selection.

Use Case 4: View AI Review

Actor: Developer

Goal:

Allow developer to view AI-generated reviews for a specific PR.

Pre-Conditions:

- Review exists for the selected PR.

Main Flow:

1. Developer selects “View AI Review.”
2. System fetches stored AI review comments.
3. System displays review details.

Post-Conditions:

- Developer sees AI-generated review.

Alternate Flow:

- Review not generated yet → system shows “No AI Review available.”

Use Case 5: View Analytics

Actor: Developer

Goal:

Display analytics charts for PR performance, repository activity, and contributor performance.

Pre-Conditions:

- Developer logged in.

Main Flow:

1. Developer selects “View Analytics.”
2. System loads analytics data.
3. System generates:
    - Charts (PR trends, merge rate trends, etc.)
    - Best performing repo (highest merge rate or fastest merges)
    - Most active contributors (PR frequency, commits)
4. System displays analytics visuals.

Post-Conditions:

- Developer sees graphical analytics.

Alternate Flow:

- Insufficient data → show “Not enough data for analytics.”

Use Case: Charts

Goal:

Visualize data in chart formats.

Main Flow:

1. System processes PR data into chart form.
2. Dashboard shows charts.

Use Case: Best Performing Repo

Included By: View Analytics

Goal: Identify repo with best performance metrics.

Main Flow:

1. System analyzes repos (merge speed, acceptance rate).
2. System selects the best performing repo.

3. Dashboard displays it.

Use Case: Most Active Contributors

Included By: View Analytics

Goal:

Identify contributors creating most PRs or merges.

Main Flow:

1. System retrieves contributor activity logs.
2. System ranks contributors by activity.
3. Dashboard displays the top list.