

User Story 1

As a developer, I want AI to have awareness of my entire folder (repo) of code, so that I can ask AI for a repo summary or change code without needing to manually append context.

Development Specification 1:

1. Header

Version: v0.1 (2025-09-24)

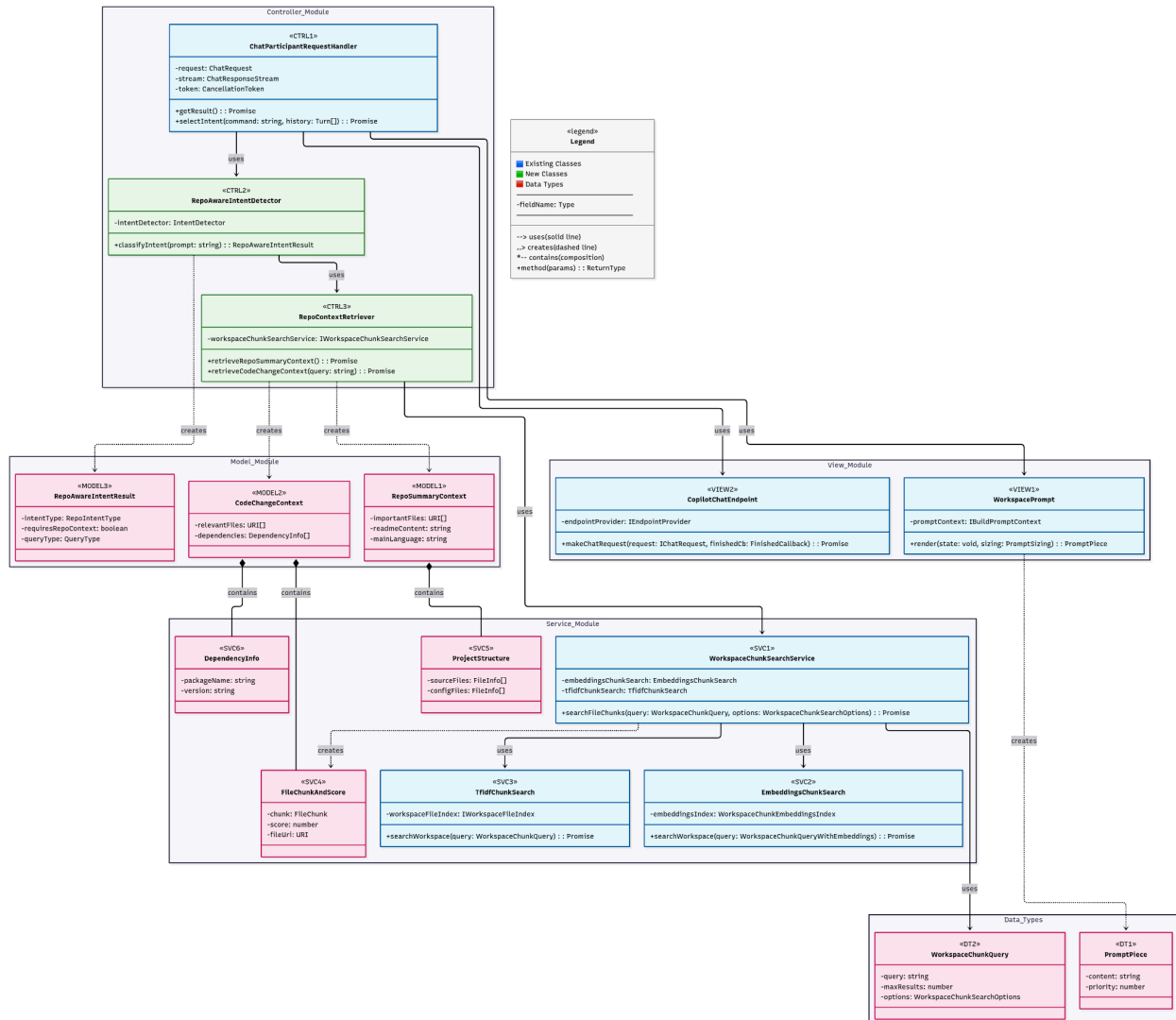
Authors & Roles:

- Christina Ma - Initial draft for the dev spec
- Baolin Shi - validate the initial draft
- Dejun Yang - validate the initial draft

Rationale: first draft, we just randomly volunteered for who to draft.

Chat Log Link: <https://chatgpt.com/share/68d1aaf8-32e4-800a-81b6-79c8e864392f>

2. Architecture Diagram



Architecture Diagram Mermaid Code

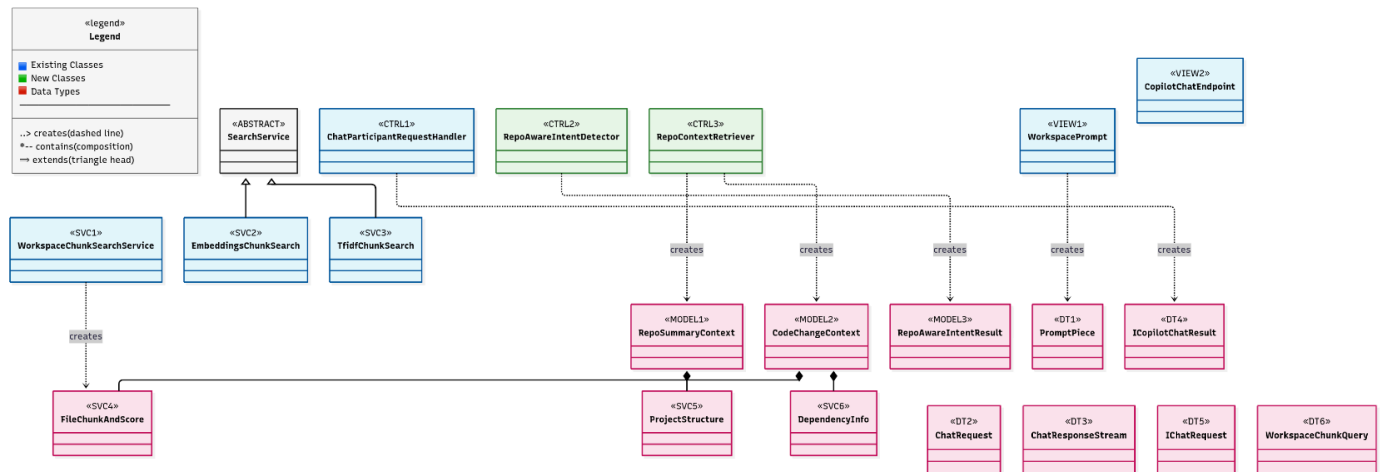
Rationale: We build on top of `vscode` and `vscode-copilot-chat` since this functionality doesn't exist there. Since we plan to implement based on the current open-sourced IDE, we need to reuse their classes/methods whenever possible. Cursor was used to summarize existing repo and.

Chat Log Links:

- <https://chatgpt.com/share/68d1aaf8-32e4-800a-81b6-79c8e864392f>
- <https://chatgpt.com/share/68d22970-7618-8009-8957-1f5f78fa4127>
- <https://chatgpt.com/share/68d238db-8688-800a-9c36-768c8cbfc968>

- 📁 P2-cursor-chat-histories

3. Class Diagram



Rationale: The class diagram follows the architecture diagram but presents an object-oriented inheritance design instead of the component-based grouping. This diagram emphasizes shared behaviors and extensibility over functional separation, making relationships clearer through inheritance instead of usage dependencies.

Chat Log Link:

- <https://chatgpt.com/share/68d1aaf8-32e4-800a-81b6-79c8e864392f>
- <https://chatgpt.com/share/68d41b56-b6dc-800a-8128-e181ba19a95f>
- 📁 P2-cursor-chat-histories

4. List of Classes

Got it. I've reviewed your **Architecture Diagram** and converted it into both a **class diagram** (object-oriented inheritance-oriented, per your spec) and a **detailed list of classes grouped by module**.

Controller Module

- **CTRL1 – ChatParticipantRequestHandler**

Purpose: Main entrypoint for handling user chat requests. Orchestrates intent detection, context retrieval, and passing results to view.

- **CTRL2 – RepoAwareIntentDetector**

Purpose: Detects repo-related user intent. Produces `RepoAwareIntentResult` for downstream use.

- **CTRL3 – RepoContextRetriever**

Purpose: Retrieves appropriate repo context (`RepoSummaryContext` or `CodeChangeContext`) based on detected intent.

Model Module

- **MODEL1 – RepoSummaryContext**

Purpose: Holds summary of repository (important files, readme content, project structure, language).

- **MODEL2 – CodeChangeContext**

Purpose: Stores context for code modifications (relevant files, code chunks, dependencies).

- **MODEL3 – RepoAwareIntentResult**

Purpose: Encapsulates classification results from intent detection (intent type, whether repo context is needed, query type).

View Module

- **VIEW1 – WorkspacePrompt**

Purpose: Renders prompts with repo-aware context for the user. Produces `PromptPiece`.

- **VIEW2 – CopilotChatEndpoint**

Purpose: Sends chat requests to AI backend and delivers results to client via callback.

Service Module

- **SVC1 – WorkspaceChunkSearchService**

Purpose: Orchestrates code search using embeddings and TF-IDF strategies.

- **SVC2 – EmbeddingsChunkSearch**

Purpose: Performs semantic code search using embeddings index.


- **SVC3 – TfidfChunkSearch**
Purpose: Performs lexical/TF-IDF-based search of repo files.
- **SVC4 – FileChunkAndScore (Struct)**
Purpose: Data structure for storing code chunks with associated relevance scores.
- **SVC5 – ProjectStructure (Struct)**
Purpose: Data structure for describing repo structure (source files, config files).
- **SVC6 – DependencyInfo (Struct)**
Purpose: Data structure for dependency metadata (package + version)

Data Types (Structs)

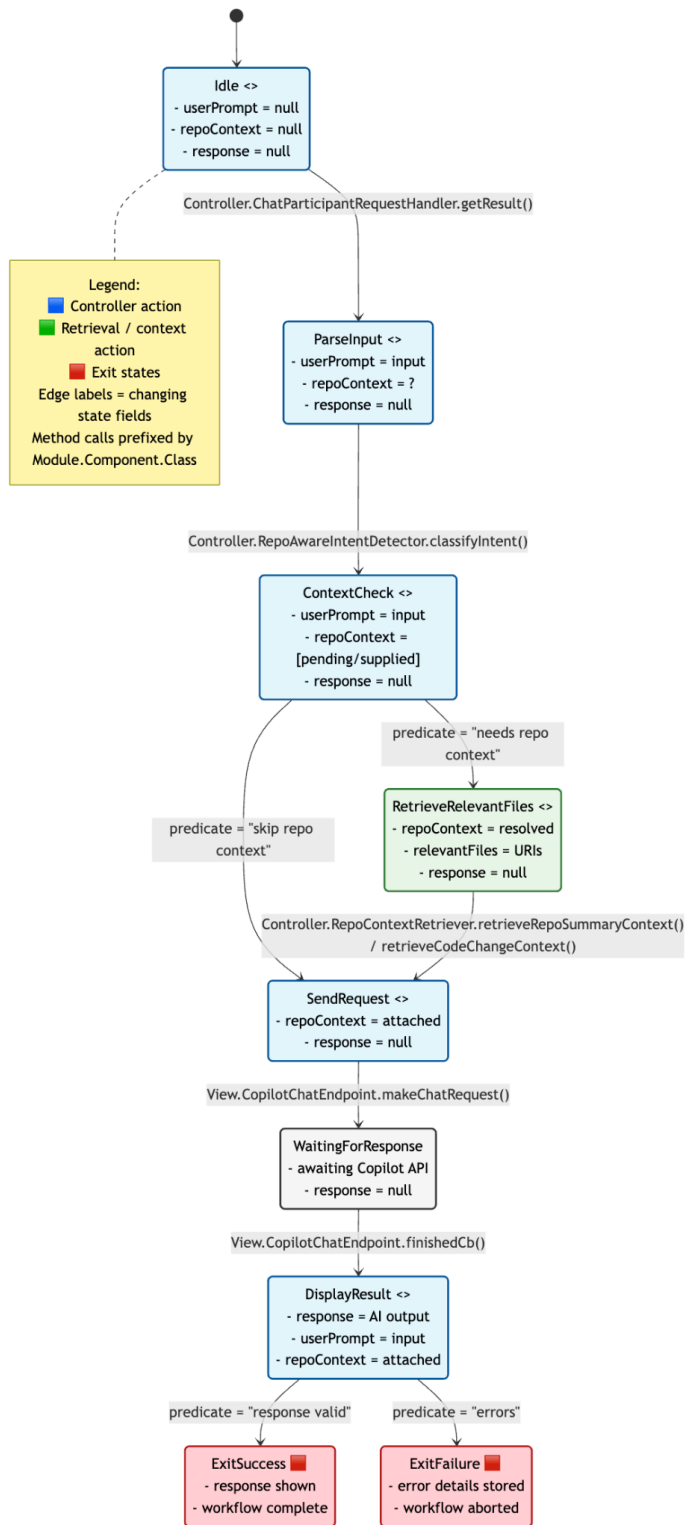
- **DT1 – PromptPiece**
Purpose: Unit of rendered prompt (content + priority).
- **DT2 – ChatRequest**
Purpose: Request object containing prompt, references, and command.
- **DT3 – ChatResponseStream**
Purpose: Stream of chat responses, supports receiving partial messages.
- **DT4 – ICopilotChatResult**
Purpose: Container for AI's response, with metadata + message parts.
- **DT5 – IChatRequest**
Purpose: Input request format to send to AI endpoint.
- **DT6 – WorkspaceChunkQuery**
Purpose: Encapsulates query for chunk-based search (query string, max results, options).

Rationale: The classes were grouped by module boundaries (Controller, Model, View, Service, Data Types) to preserve the original architecture's intent while making each class's role and responsibility explicit. Struct-like classes were separated as data storage entities, ensuring a clear distinction between behavior-driven components and pure data carriers, which aligns with object-oriented design best practices.

Chat Log Link:

- <https://chatgpt.com/share/68d41b56-b6dc-800a-8128-e181ba19a95f>
- <https://chatgpt.com/share/68d1aaf8-32e4-800a-81b6-79c8e864392f>
-  P2-cursor-chat-histories

5. State Diagrams

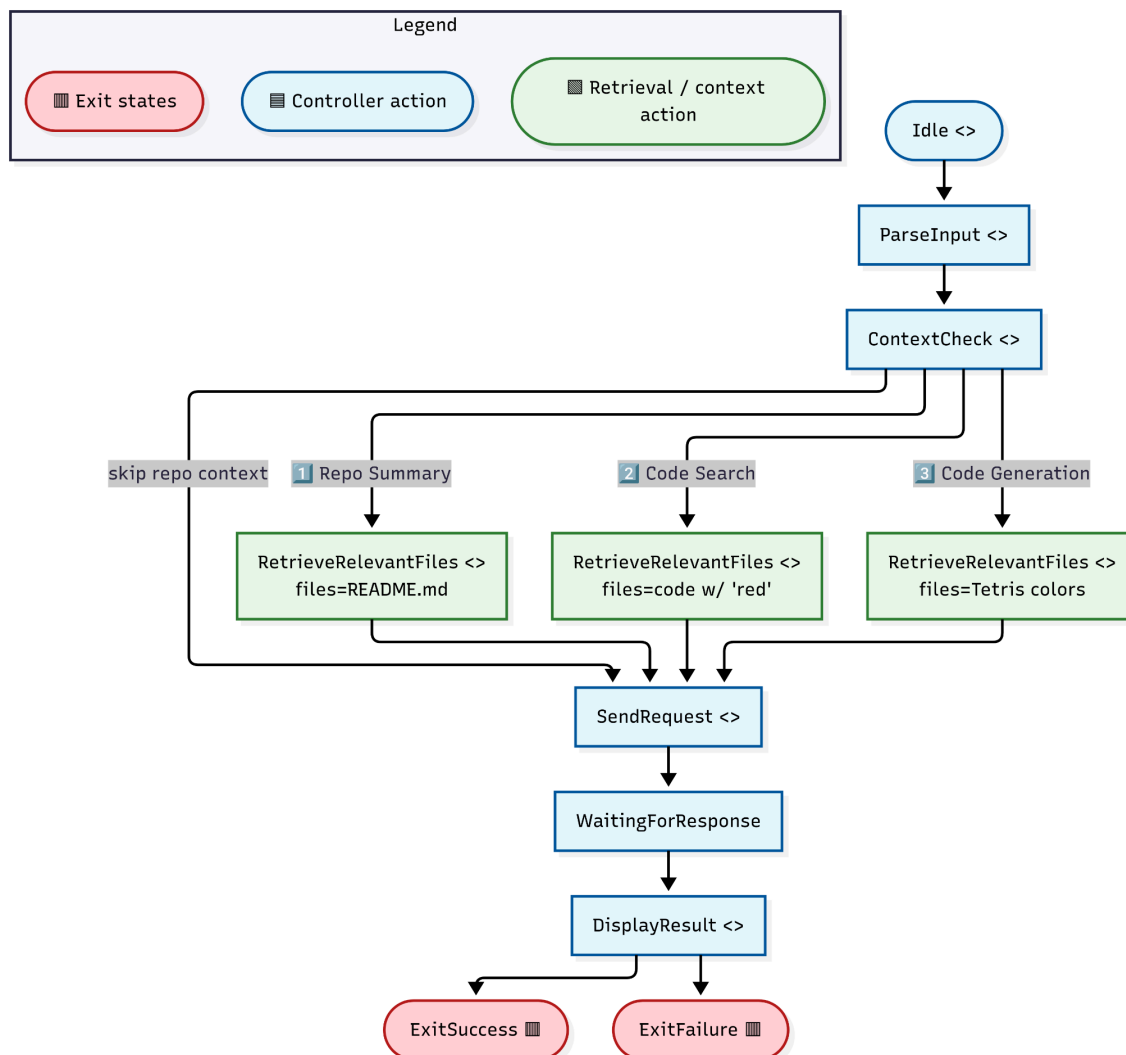


Rationale: The state diagram captures the full lifecycle of a user query across all scenarios. It tracks system state via four key fields (userPrompt, repoContext, relevantFiles, response) and shows how these evolve as the workflow progresses. Controller states (■) handle orchestration, retrieval states (■) resolve repo context, and terminal states (■) capture either successful completion (ExitSuccess) or failure (ExitFailure).

Chat Log Link:

- <https://chatgpt.com/share/68d41b56-b6dc-800a-8128-e181ba19a95f>
- <https://chatgpt.com/share/68d1aaf8-32e4-800a-81b6-79c8e864392f>

6. Flow Chart



Rationale: The flow charts illustrate three user scenarios (repo summary, code search, code generation) branching from the same ContextCheck decision point. Each scenario specifies what context is retrieved (README.md, files with “red”, or Tetris color files), but all converge on a shared path through SendRequest → WaitingForResponse → DisplayResult. This design emphasizes the reuse of the same pipeline with variations only in context retrieval.

Chat Log Link:

- <https://chatgpt.com/share/68d1aaf8-32e4-800a-81b6-79c8e864392f>
- <https://chatgpt.com/share/68d41b56-b6dc-800a-8128-e181ba19a95f>

7. Development Risks and Failures

1. Controller Module

ChatParticipantRequestHandler (CTRL1), RepoAwareIntentDetector (CTRL2), RepoContextRetriever (CTRL3)

- **FM01 – Controller crash**
 - *Effect:* User request dropped, no response.
 - *Recovery:* Restart controller process, replay unprocessed requests from log.
 - *Diagnostics:* Detect via process health monitor (cross-ref test).
 - *Likelihood:* M / *Impact:* H
- **FM02 – Lost runtime state**
 - *Effect:* Intent classification history and cancellation tokens reset.
 - *Recovery:* Reconstruct session state from persisted user inputs; re-run classification.
 - *Likelihood:* L / *Impact:* M
- **FM03 – Jumped to wrong state**
 - *Effect:* Skips retrieval → wrong or empty response.
 - *Recovery:* Validate transitions; auto-retry from last known valid state.
 - *Likelihood:* M / *Impact:* H

2. Model Module

RepoSummaryContext (MODEL1), CodeChangeContext (MODEL2), RepoAwareIntentResult (MODEL3)

- **FM04 – Model object corruption**

- *Effect:* Wrong context (e.g., missing files, bad dependencies). User sees irrelevant answers.
- *Recovery:* Rebuild model objects from repo scan or re-run intent detection.
- *Likelihood:* M / *Impact:* H

- **FM05 – Erased stored model data**

- *Effect:* Cannot provide summaries or code change context.
- *Recovery:* Recompute from repo files on demand.
- *Likelihood:* L / *Impact:* M

3. View Module

WorkspacePrompt (VIEW1), CopilotChatEndpoint (VIEW2)

- **FM06 – Endpoint RPC failed**

- *Effect:* User gets timeout or no response.
- *Recovery:* Retry with exponential backoff; fall back to cached AI endpoint.
- *Likelihood:* H / *Impact:* H

- **FM07 – DisplayResult failure**

- *Effect:* Response not shown, UI hangs.
- *Recovery:* Reload UI, fetch last successful result from log.
- *Likelihood:* M / *Impact:* M

4. Service Module

WorkspaceChunkSearchService (SVC1), EmbeddingsChunkSearch (SVC2), TfidfChunkSearch (SVC3), FileChunkAndScore, ProjectStructure, DependencyInfo (SVC4–6)

- **FM08 – Search service overloaded**

- *Effect:* Slow retrieval, degraded UX.
- *Recovery:* Throttle incoming requests; scale horizontally.
- *Likelihood:* H / *Impact:* H

- **FM09 – Out of RAM**

- *Effect:* Search service crashes; cannot build embeddings.

- *Recovery*: Restart service; shard indexes to multiple machines.
- *Likelihood*: M / *Impact*: H
- **FM10 – Database out of space**
 - *Effect*: Cannot store embeddings or TF-IDF indexes.
 - *Recovery*: Free disk, archive old indexes, expand storage.
 - *Likelihood*: M / *Impact*: H
- **FM11 – Data corruption (index mismatch)**
 - *Effect*: Inaccurate results returned.
 - *Recovery*: Recompute indexes from repo files.
 - *Likelihood*: L / *Impact*: M

5. Connectivity Failures

(Affect **all modules** relying on Copilot API and storage backends)

- **FM12 – Lost network connectivity**
 - *Effect*: No responses; user sees error.
 - *Recovery*: Detect via ping; retry once connectivity restored.
 - *Likelihood*: M / *Impact*: H
- **FM13 – Traffic spikes**
 - *Effect*: Increased latency or throttling.
 - *Recovery*: Rate limit per user; auto-scale endpoints.
 - Likelihood*: H / *Impact*: M

6. Hardware Failures

- **FM14 – Server down**
 - *Effect*: Entire module unavailable.
 - *Recovery*: Failover to replica.
 - *Likelihood*: M / *Impact*: H
- **FM15 – Bad configuration file**
 - *Effect*: Services fail to start or misbehave.
 - *Recovery*: Roll back to last known good config.
 - *Likelihood*: L / *Impact*: H
- **FM16 – Relocation / new URI**
 - *Effect*: Services unreachable due to DNS/URI mismatch.

- *Recovery*: Update config, propagate new endpoint.
- *Likelihood*: L / *Impact*: M

7. Intruder Threats

- **FM17 – Denial of Service (DoS)**
 - *Effect*: Users unable to access service.
Recovery: Rate limit, firewall rules, upstream filtering.
 - *Likelihood*: M / *Impact*: H
- **FM18 – OS trashed / project code rewritten**
 - *Effect*: Service integrity destroyed.
 - *Recovery*: Restore from backups, re-deploy clean images.
 - *Likelihood*: L / *Impact*: H
- **FM19 – Database copied**
 - *Effect*: Repo context exposed → confidentiality breach.
 - *Recovery*: Revoke compromised creds; notify users; re-encrypt data.
 - *Likelihood*: L / *Impact*: H
- **FM20 – Bot signup spam**
 - *Effect*: System flooded with bogus queries.
Recovery: CAPTCHA, rate limit, block offending IPs.
 - *Likelihood*: H / *Impact*: M
- **FM21 – HTTP session hijack**
 - *Effect*: Unauthorized user can issue queries as another.
 - *Recovery*: Invalidate sessions; enforce HTTPS and token expiry.
 - *Likelihood*: M / *Impact*: H

Likelihood vs. Impact Summary

Failure Mode	Likelihood	Impact	Priority
FM06 – Endpoint RPC failed	H	H	● Critical
FM08 – Search service overloaded	H	H	● Critical
FM12 – Lost network connectivity	M	H	● Critical

FM17 – DoS attack	M	H	● Critical
FM21 – HTTP session hijack	M	H	● Critical
FM01, FM04, FM09, FM10, FM14	M	H	● High
FM02, FM05, FM07, FM11, FM13, FM15, FM16, FM18, FM19, FM20	L–M	M–H	● Medium

Rationale: The failure modes were enumerated **per module and component** to ensure both user-visible errors (e.g., failed responses, corrupted outputs) and internal issues (e.g., lost state, bad indexes) are covered systematically. Each mode includes a **clear recovery path** to restore the system to a sane state, and failures are prioritized by **likelihood and impact**.

Chat Log Link: <https://chatgpt.com/share/68d41b56-b6dc-800a-8128-e181ba19a95f>

8. Technology Stack

T01 – TypeScript (v5.3)

- Use: Main implementation language for controllers, services, and models.
- Why: Type safety, excellent IDE support, ecosystem maturity.
- URL: <https://www.typescriptlang.org/>
- Author: Microsoft
- Docs: <https://www.typescriptlang.org/docs/>

T02 – Node.js (v20 LTS)

- Use: Runtime environment for backend services (controllers, retrieval, API calls).
- Why: Widely supported, async-friendly, ecosystem of packages.
- URL: <https://nodejs.org/>
- Author: OpenJS Foundation
- Docs: <https://nodejs.org/en/docs/>

T03 – Express.js (v4.18)

- Use: Lightweight web framework for exposing APIs (CopilotChatEndpoint integration).
- Why: Simplicity, strong middleware ecosystem, widely adopted.
- URL: <https://expressjs.com/>
- Author: TJ Holowaychuk, OpenJS Foundation
- Docs: <https://expressjs.com/en/4x/api.html>

T04 – OpenAI API (v1.12)

- Use: Provides large language model responses for repo summaries and code edits.
- Why: State-of-the-art model quality, strong SDK support.
- URL: <https://platform.openai.com/>
- Author: OpenAI
- Docs: <https://platform.openai.com/docs/>

T05 – Claude API (Anthropic, v2024-01-15)

- Use: Alternative LLM provider for conversational repo-aware queries.
- Why: Strong reasoning & coding ability and complements OpenAI models.
- URL: <https://www.anthropic.com/>
- Author: Anthropic
- Docs: <https://docs.anthropic.com/>

T06 – PostgreSQL (v15)

- Use: Database for storing repo embeddings, TF-IDF indexes, and metadata.
- Why: Reliable, ACID-compliant, strong ecosystem, easy integration.
- URL: <https://www.postgresql.org/>
- Author: PostgreSQL Global Development Group
- Docs: <https://www.postgresql.org/docs/>

T07 – Redis (v7)

- Use: Caching layer for quick access to frequently used embeddings and query results.
- Why: Extremely fast in-memory store, simple pub/sub model.
- URL: <https://redis.io/>
- Author: Redis Ltd.

- Docs: <https://redis.io/docs/>

T08 – TensorFlow.js (v4.12)

- Use: Embeddings computation and model execution for repo-aware search.
- Why: JS/TS-native ML library, integrates well with Node.js stack.
- URL: <https://www.tensorflow.org/js>
- Author: Google Brain Team
- Docs: <https://www.tensorflow.org/js/guide>

T09 – Elasticsearch (v8.12)

- Use: Search backend for TF-IDF queries over repo files.
- Why: High-performance full-text search, mature ecosystem.
- URL: <https://www.elastic.co/elasticsearch/>
- Author: Elastic N.V.
- Docs: <https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html>

T10 – Docker (v24.0)

- Use: Containerization for consistent deployment of services (controllers, search, DB).
- Why: Industry standard, portable environments.
- URL: <https://www.docker.com/>
- Author: Docker, Inc.
- Docs: <https://docs.docker.com/>

T11 – Kubernetes (v1.29)

- Use: Orchestration of containers, scaling for search services and API endpoints.
- Why: De facto standard for container orchestration, scaling, self-healing.
- URL: <https://kubernetes.io/>
- Author: Cloud Native Computing Foundation
- Docs: <https://kubernetes.io/docs/>

T12 – Git (v2.43)

- Use: Version control for codebase and repo indexing.
- Why: Industry standard for versioning, integration with CI/CD.
- URL: <https://git-scm.com/>

- Author: Linus Torvalds, Git contributors
- Docs: <https://git-scm.com/doc>

T13 – VS Code Extension API (v1.85)

- Use: Extension development interface used for WorkspacePrompt and Copilot integrations.
- Why: Provides seamless integration into VS Code UI.
- URL: <https://code.visualstudio.com/api>
- Author: Microsoft
- Docs: <https://code.visualstudio.com/api>

T14 – VS Code Copilot Chat (v0.12)

- Use: Microsoft's extension for Copilot chat integration.
- Why: Provides direct in-IDE conversational workflows, baseline for repo-aware enhancement.
- URL: <https://marketplace.visualstudio.com/items?itemName=GitHub.copilot-chat>
- Author: GitHub / Microsoft
- Docs: <https://code.visualstudio.com/docs/copilot/copilot-chat>

Rationale: This stack combines **core development technologies** (TypeScript, Node.js, Git, VS Code) with **infrastructure tools** (Docker, Kubernetes, PostgreSQL, Redis, Elasticsearch) and **AI/ML providers** (OpenAI, Claude, TensorFlow.js). VS Code and its Copilot Chat extension were explicitly included to anchor repo-aware workflows inside the developer's IDE, ensuring **high usability and adoption**. Each technology was selected for **maturity, ecosystem strength, and seamless integration** into a repo-centric AI workflow.

Chat Log Link: <https://chatgpt.com/share/68d41b56-b6dc-800a-8128-e181ba19a95f>

9. APIs

Controller Module

CTRL1 – ChatParticipantRequestHandler

Public Methods

- `getResult(prompt: string, userId: string): Promise<Response>`
 - Entry point for handling user queries.

Private Methods

- `sanitizeInput(raw: string): string`
- `logRequest(userId: string, prompt: string): void`

CTRL2 – RepoAwareIntentDetector

Public Methods

- `classifyIntent(prompt: string): IntentResult`
 - Detects whether repo context is needed.

Private Methods

- `loadModel(path: string): void`
`validateClassification(result: IntentResult): boolean`

CTRL3 – RepoContextRetriever

Public Methods

- `retrieveRepoSummaryContext(repoPath: string): Promise<FileContext[]>`
- `retrieveCodeChangeContext(repoPath: string, query: string): Promise<FileContext[]>`
 - **Overload:** Both methods return `FileContext[]` but differ by parameters.

Private Methods

- `filterRelevantFiles(files: File[]): FileContext[]`

Model Module

MODEL1 – RepoSummaryContext

Public Methods

- `toJSON(): object`
- `fromFiles(files: File[]): RepoSummaryContext`

MODEL2 – CodeChangeContext

Public Methods

- `toJSON(): object`
- `fromFiles(files: File[], query: string): CodeChangeContext`

MODEL3 – RepoAwareIntentResult

Public Methods

- `isRepoAware(): boolean`
- `getConfidence(): number`

View Module

VIEW1 – WorkspacePrompt

Public Methods

- `render(prompt: string, context: FileContext[]): void`
- `clear(): void`

VIEW2 – CopilotChatEndpoint

Public Methods

- `makeChatRequest(prompt: string, context: FileContext[]): Promise<Response>`
- `finishedCb(response: Response): void`

Private Methods

- `formatPayload(prompt: string, context: FileContext[]): object`

Service Module

SVC1 – WorkspaceChunkSearchService

Public Methods

- `search(query: string): Promise<FileChunkAndScore[]>`

SVC2 – EmbeddingsChunkSearch

Public Methods

- `buildEmbeddings(file: File): number[]`
- `searchByEmbedding(vector: number[]): FileChunkAndScore[]`

SVC3 – TfidfChunkSearch

Public Methods

- `search(query: string): FileChunkAndScore[]`

SVC4 – FileChunkAndScore (Data class)

Public Methods

- `getChunk(): string`
- `getScore(): number`

SVC5 – ProjectStructure (Data class)

Public Methods

- `getModules(): string[]`

SVC6 – DependencyInfo (Data class)

Public Methods

- `getDependencies(): string[]`

Data Types

DT1 – FileContext (struct)

Public Fields

- `path: string`
 `content: string`

DT2 – IntentResult (struct)

Public Fields

- `intent: string`
- `confidence: number`

Rationale: APIs are organized by **module boundaries** from the architecture diagram to maintain traceability. Public methods represent **external contracts** that other modules can rely on, while private methods encapsulate **internal helpers**. Overloads in `RepoContextRetriever` are called out explicitly since they affect testing and type safety. This grouping clarifies **separation of concerns**: controllers orchestrate workflows, models encapsulate data, views render/dispatch, and services implement search logic.

Chat Log Link: <https://chatgpt.com/share/68d41b56-b6dc-800a-8128-e181ba19a95f>

10. Public Interfaces

Controller Module

CTRL1 – ChatParticipantRequestHandler

- *Used within same component:*
 - `sanitizeInput` (private, not included in public list).
- *Used across components (same module):* None.
Used across modules:
 - `getResult(prompt: string, userId: string): Promise<Response>`
 - Called by **VIEW2.CopilotChatEndpoint** to handle user queries.

CTRL2 – RepoAwareIntentDetector

- *Used within same component:* None.
Used across components (same module):
 - `classifyIntent(prompt: string): IntentResult`
 - Used by **CTRL1.ChatParticipantRequestHandler**.
- *Used across modules:* None.

CTRL3 – RepoContextRetriever

- *Used within same component:* None.
- *Used across components (same module):*
 - `retrieveRepoSummaryContext(repoPath: string): Promise<FileContext[]>`

- `retrieveCodeChangeContext(repoPath: string, query: string): Promise<FileContext[]>`
 - Used by **CTRL1.ChatParticipantRequestHandler**.
- *Used across modules:* None.

Model Module

MODEL1 – RepoSummaryContext

- *Used across modules:*
 - `fromFiles(files: File[]): RepoSummaryContext`
 - Used by **CTRL3.RepoContextRetriever**.
 - `toJSON(): object`
 - Used by **VIEW2.CopilotChatEndpoint** for serialization.

MODEL2 – CodeChangeContext

- *Used across modules:*
 - `fromFiles(files: File[], query: string): CodeChangeContext`
 - Used by **CTRL3.RepoContextRetriever**.
 - `toJSON(): object`
 - Used by **VIEW2.CopilotChatEndpoint**.

MODEL3 – RepoAwareIntentResult

- *Used across modules:*
 - `isRepoAware(): boolean`
 - `getConfidence(): number`
 - Used by **CTRL2.RepoAwareIntentDetector** and **CTRL1.ChatParticipantRequestHandler** for decision-making.

View Module

VIEW1 – WorkspacePrompt

- *Used within same module:*
 - `render(prompt: string, context: FileContext[]): void`
 - `clear(): void`
 - Used by **VIEW2.CopilotChatEndpoint** to manage display.

VIEW2 – CopilotChatEndpoint

- *Used across modules:*
 - `makeChatRequest(prompt: string, context: FileContext[]): Promise<Response>`
 - Called by **CTRL1.ChatParticipantRequestHandler**.
 - `finishedCb(response: Response): void`
 - Called back from Copilot API responses, uses **VIEW1.WorkspacePrompt** for rendering.

Service Module

SVC1 – WorkspaceChunkSearchService

- *Used across modules:*
 - `search(query: string): Promise<FileChunkAndScore[]>`
 - Used by **CTRL3.RepoContextRetriever**.

SVC2 – EmbeddingsChunkSearch

- *Used across modules:*
 - `buildEmbeddings(file: File): number[]`
 - `searchByEmbedding(vector: number[]): FileChunkAndScore[]`
 - Used by **SVC1.WorkspaceChunkSearchService**.

SVC3 – TfidfChunkSearch

- *Used across modules:*
 - `search(query: string): FileChunkAndScore[]`
 - Used by **SVC1.WorkspaceChunkSearchService**.

SVC4 – FileChunkAndScore

- *Used across modules:*
 - `getChunk(): string`
 - `getScore(): number`
 - Returned to **CTRL3.RepoContextRetriever** for ranking.

SVC5 – ProjectStructure

- *Used across modules:*
 - `getModules(): string[]`
 - Used by **CTRL3.RepoContextRetriever** for repo-wide context.

SVC6 – DependencyInfo

- *Used across modules:*
 - `getDependencies(): string[]`
 - Used by **CTRL3.RepoContextRetriever** for dependency resolution.

Data Types

DT1 – FileContext

- *Used across modules:*
 - Fields: `path: string, content: string`
 - Used by **CTRL3.RepoContextRetriever**, **VIEW2.CopilotChatEndpoint**, and **MODEL1/MODEL2**.

DT2 – IntentResult

- *Used across modules:*
 - Fields: `intent: string, confidence: number`
 - Produced by **CTRL2.RepoAwareIntentDetector** and consumed by **CTRL1.ChatParticipantRequestHandler**.

Cross-Module Usage (Summary)

- **Controller** → **Model**: Uses `RepoSummaryContext`, `CodeChangeContext`, `RepoAwareIntentResult`.
- **Controller** → **Service**: Uses `WorkspaceChunkSearchService` for retrieval.
- **Controller** → **View**: Feeds queries into `CopilotChatEndpoint`.
- **View** → **Model**: Serializes contexts (`toJSON`).
- **Service** → **Data Types**: Returns `FileChunkAndScore`, `ProjectStructure`, `DependencyInfo`.

Multi-language / Interface Access

- **REST / HTTP:**
 - `CopilotChatEndpoint.makeChatRequest()` exposed via JSON POST `/chat/request`.
 - `CopilotChatEndpoint.finishedCb()` exposed as callback response endpoint `/chat/result`.
- **TypeScript API:**
 - Direct imports (`import { RepoContextRetriever } from "controller";`) for integration in Node.js projects.
- **VS Code Extension API:**
 - `WorkspacePrompt.render()` and `WorkspacePrompt.clear()` exposed to extension host environment.

Rationale: This interface spec emphasizes **how public methods are actually used across boundaries**. Grouping by *within component*, *across component*, and *across module* highlights dependencies clearly and avoids exposing internals unnecessarily. Mapping **cross-module usage** ensures traceability back to the architecture diagram. Multi-language interfaces (TypeScript imports, REST endpoints, VS Code API) ensure the system can integrate with developer workflows both programmatically and via the IDE.

Chat Log Link: <https://chatgpt.com/share/68d41b56-b6dc-800a-8128-e181ba19a95f>

11. Data Schemas

DS01 – FileContext Table

- **Runtime Class:** `DATA1.FileContext`
- **Purpose:** Stores source file metadata and contents for repo-aware queries.
- **Columns:**
 - `id` SERIAL PRIMARY KEY
 - `path` VARCHAR(512) – relative path of file in repo
 - `content` TEXT – full file content
 - `repo_id` UUID – foreign key to repo
- **Notes:** `content` stored as TEXT for variable-length files.
- **Storage:** $\sim \text{len}(\text{path}) + \text{len}(\text{content})$ bytes per record. For N files with avg 20KB content: $\approx 20\text{KB} * N$.

DS02 – IntentResult Table

- **Runtime Class:** `DATA2.IntentResult`
- **Purpose:** Records results of intent classification for auditing.
- **Columns:**
 - `id` SERIAL PRIMARY KEY
 - `intent` VARCHAR(128) – intent label
 - `confidence` FLOAT – confidence score (0–1)
 - `created_at` TIMESTAMP – classification time
- **Notes:** Confidence stored as `FLOAT` (4 bytes).
- **Storage:** ~200 bytes per row.

DS03 – RepoSummaryContext Table

- **Runtime Class:** `MODEL1.RepoSummaryContext`
- **Purpose:** Caches computed repo summaries for fast reuse.
- **Columns:**
 - `id` SERIAL PRIMARY KEY
 - `repo_id` UUID
 - `summary` TEXT
 - `files` JSONB – list of file paths included in summary
- **Notes:** JSONB allows flexible storage of file sets.
- **Storage:** $\approx \text{len}(\text{summary}) + \text{len}(\text{files})$ per record. Typical 2–4 KB.

DS04 – CodeChangeContext Table

- **Runtime Class:** `MODEL2.CodeChangeContext`
- **Purpose:** Stores context around code edits (retrieved files + query).
- **Columns:**
 - `id` SERIAL PRIMARY KEY
 - `repo_id` UUID
 - `query` VARCHAR(1024) – user query text
 - `files` JSONB – list of file paths and extracted chunks
- **Storage:** $\approx \text{len}(\text{query}) + \text{len}(\text{files})$ per record. For queries touching 10 files: ~3–5 KB.

DS05 – FileChunkAndScore Table

- **Runtime Class:** `SVC4.FileChunkAndScore`
- **Purpose:** Stores chunked file data with retrieval scores.
- **Columns:**
 - `id` SERIAL PRIMARY KEY
 - `file_id` INT REFERENCES FileContext(id)
 - `chunk` TEXT – code/text snippet
 - `score` FLOAT – retrieval relevance score
- **Storage:** $\approx \text{len}(\text{chunk}) + 8$ bytes. For 1KB chunks, ≈ 1 KB/row.

DS06 – EmbeddingVector Table

- **Runtime Class:** `SVC2.EmbeddingsChunkSearch`
- **Purpose:** Stores vector embeddings for repo search.
- **Columns:**
 - `id` SERIAL PRIMARY KEY
 - `file_id` INT REFERENCES FileContext(id)
 - `vector` FLOAT[1536] – embedding array (OpenAI default size)
- **Notes:** $1536 * 4$ bytes ≈ 6 KB per vector.
- **Storage:** For N chunks: $\approx 6 \text{ KB} * N$.

DS07 – ProjectStructure Table

- **Runtime Class:** `SVC5.ProjectStructure`
- **Purpose:** Stores logical module breakdown of repo.
- **Columns:**
 - `id` SERIAL PRIMARY KEY
 - `repo_id` UUID
 - `modules` JSONB – array of module names
- **Storage:** ~ 500 bytes per repo (small).

DS08 – DependencyInfo Table

- **Runtime Class:** `SVC6.DependencyInfo`
- **Purpose:** Stores dependency graph for repo.
- **Columns:**
 - `id` SERIAL PRIMARY KEY
 - `repo_id` UUID
 - `dependencies` JSONB – adjacency list of dependencies

- **Storage:** Depends on repo size; ~1–2 KB typical.

Rationale: Data schemas map directly from **runtime classes (Data + Model + Service)** to persistent structures in PostgreSQL. Using **JSONB** for variable lists (files, modules, dependencies) balances flexibility and queryability. Embeddings require the largest storage footprint (~6 KB per chunk), making them the main scaling factor. By estimating storage, we ensure system design can handle repos from small projects to large enterprise codebases.

Chat Log Link: <https://chatgpt.com/share/68d41b56-b6dc-800a-8128-e181ba19a95f>

12. Security and Privacy

1. PII Temporarily Stored

- **PII Types:**
 - **userId** (internal user identifier, not email).
 - **userPrompt** (may contain incidental PII if typed by user).
- **Justification:** Needed to associate requests with users, enforce rate limits, and personalize retrieval context.
- **Source:** Entered by users in VS Code Copilot Chat extension.
- **Flow Through System:**
 - **VIEW1.WorkspacePrompt.render()** → captures user input.
 - **CTRL1.ChatParticipantRequestHandler.getResult()** → binds **prompt** + **userId**.
 - **CTRL2.RepoAwareIntentDetector.classifyIntent()** → analyzes prompt text (may contain PII).
 - **CTRL3.RepoContextRetriever.retrieve*Context()** → pairs user query with repo files.
 - **VIEW2.CopilotChatEndpoint.makeChatRequest()** → dispatches to Copilot API.
- **Active Use:** Input validation, intent detection, search context retrieval.
- **After Use:** Sent to Copilot/LLM API as part of request payload.
- **Disposal:** Prompt discarded from memory once response is returned or stored in logs if user permitted.
- **Protections:**

- In-memory only, zeroized after response lifecycle.
- TLS (HTTPS) enforced for all network traffic.
- Access controls on VS Code extension → Node backend link.

2. PII in Long-Term Storage

- **Stored PII:**
 - `userId` (hashed UUID).
 - **No names, emails, or sensitive identifiers** stored.
- **Justification:** Required for tracking request quotas, audits, and user-specific caching (optional).
- **Storage:**
 - PostgreSQL (DS02.IntentResult table → `userId` foreign key).
 - Stored hashed + salted (SHA-256) for anonymity.
- **Entry Path:**
 - **CTRL1.ChatParticipantRequestHandler** → persists classification/audit results.
- **Exit Path:** Queried only for analytics or rate limiting.

3. Responsibility for Securing Long-Term Storage

- **DB Admins:** Responsible for Postgres/Redis hardening.
- **Service Owners:** Developers maintaining retrieval and context services.
- **DevOps/SRE:** Backup/restore, access rotation, infra patching.

4. Customer-Visible Privacy Policy

- **Policy:** “We do not retain prompts or repo data beyond the active session unless you want to save it for your logging purpose, except anonymized request metadata (hashed `userId`, request type) for audit and quota enforcement.”
- **Location/Acceptance:**
 - Shown during extension installation.
 - Displayed in extension settings (“Data & Privacy”).

5. Access Control Policies

- **Temporary PII:** Access restricted to active service processes.

- **Long-Term PII:** DB-level RBAC (role-based access control).
 - Developers: no direct DB access.
 - Service processes: least-privilege accounts.
 - Auditors: read-only audit logs.

6. Auditing Procedures

- **Routine:** Automatic logging of all access to **IntentResult** and related PII tables.
- **Non-Routine:** Manual review + escalation when anomalies detected (e.g., mass export attempt).
- **Data Structures:**
 - Append-only audit log.
 - Cross-referenced by Failure Mode IDs from Threats/Failures.

7. Minors and Sensitive Access

- **Policy:** System does not solicit or knowingly store PII of minors (<18).
- **Safeguards:**
 - Enforced by Terms of Service.
 - System access limited by GitHub/VS Code account auth (age screening externalized).

Rationale: This design ensures **minimal PII exposure**: only **userId** and transient prompts may contain identifying info. Prompt lifecycle is ephemeral (disposed after request unless user want to keep it). Long-term PII is anonymized and hashed, stored only for quota and audit. Strict access controls, TLS, and audit logging provide strong protection against misuse. The privacy policy is transparent to users at install time. Compliance is enforced via **role separation** (DB admin vs. security officer vs. developers). For this class project's sake, we will just keep an eye on the security aspect of this, but if this is scaled up to a company level a security officer will be needed.

Chat Log Link: <https://chatgpt.com/share/68d41b56-b6dc-800a-8128-e181ba19a95f>

13. Risks to Completion

Controller Module

- `CTRL1.ChatParticipantRequestHandler`
- `CTRL2.RepoAwareIntentDetector`
- `CTRL3.RepoContextRetriever`

Risks: Medium — correctness depends heavily on intent detection accuracy.

- **Learning Curve:** Moderate — requires solid understanding of async request handling and state machine flow.
- **Design/Implementation:** Medium complexity; orchestration logic must be resilient to failures.
- **Verification:** High testing effort, since branching logic (intent detection vs. repo retrieval) is critical.
Maintenance: Moderate — as repo features evolve, must keep classifiers and retrievers updated.
- **Updates:** Internal codebase, so updates are at our discretion; no external dependency.
- **Support:** No vendor support — requires in-house expertise.

Model Module

- `MODEL1.RepoSummaryContext`, `MODEL2.CodeChangeContext`,
`MODEL3.RepoAwareIntentResult`

Risks: Low

- **Learning Curve:** Low — mainly data encapsulation and serialization.
- **Design/Implementation:** Straightforward, but ensuring consistency across serialization/deserialization is essential.
- **Verification:** Easy — unit tests for `toJSON`, `fromFiles`.
- **Maintenance:** Low — only changes if data schema evolves.
- **Updates:** Internal.
- **Support:** In-house only.

View Module

- `VIEW1.WorkspacePrompt`, `VIEW2.CopilotChatEndpoint`

Risk: Medium

- **Learning Curve:** Moderate — requires knowledge of VS Code extension API and Copilot Chat integration.
- **Design/Implementation:** Medium — UI event handling is straightforward but error handling must be polished.
- **Verification:** Requires end-to-end testing inside IDE.
- **Maintenance:** Higher risk if VS Code extension API changes frequently.
- **Updates:** Must track VS Code release cadence; potential breaking changes.
- **Support:** Documentation and open-source community. No vendor support for custom extensions.

Service Module

- `SVC1.WorkspaceChunkSearchService`
- `SVC2.EmbeddingsChunkSearch`
- `SVC3.TfidfChunkSearch`
- `SVC4.FileChunkAndScore, SVC5.ProjectStructure, SVC6.DependencyInfo`

Risks: High — most complexity concentrated here.

- **Learning Curve:** High — developers need background in IR (information retrieval) and ML embeddings.
- **Design/Implementation:** Challenging — embedding retrieval, indexing, and scoring logic must scale.
- **Verification:** Hard — requires synthetic test repos and golden queries to ensure precision/recall.
- **Maintenance:** Medium — as repos grow, performance tuning may be required.
- **Updates:** Depends on underlying libraries (TensorFlow.js, Elasticsearch). Rollbacks possible only if indexes remain backward-compatible.
- **Support:** Vendor/community for off-the-shelf libs. In-house for custom logic.

Data Schemas: DS01–DS08 (`FileContext`, `IntentResult`, `RepoSummaryContext`, `CodeChangeContext`, `FileChunkAndScore`, `EmbeddingVector`, `ProjectStructure`, `DependencyInfo`)

Risks: Medium — mostly operational, tied to scaling data.

- **Learning Curve:** Low — relational and JSONB schemas are standard.
Design/Implementation: Moderate — must balance normalization vs. JSON flexibility.
- **Verification:** Moderate — need schema migration tests and integrity checks.
- **Maintenance:** Higher for embeddings (DS06) due to storage footprint.
- **Updates:** Postgres version updates must be tracked carefully. Rollbacks possible but costly for corrupted indexes.
- **Support:** Postgres community, managed hosting vendors (if used).

Technologies

T01 TypeScript, T02 Node.js, T03 Express.js

- Risks: Low learning curve, huge ecosystem support. Risk lies in dependency churn.
- Updates: Track LTS releases. Roll back if type-breaking changes occur.
- Support: Open-source, active community.

T04 OpenAI API, T05 Claude API

- Risks: Medium – Low learning curve (SDKs provided); Vendor lock-in and cost sensitivity.
- Design/Implementation: Easy — but usage cost and API rate limits are external risks.
Maintenance: Must track API versioning; sudden deprecations possible.
- Support: Vendor support via docs and tickets.

T06 PostgreSQL, T07 Redis

- Risks: Moderate — need operational knowledge (backup, scaling), tied to operational reliability.
- Maintenance: Requires DBA-level skills.
Support: Strong open-source community, enterprise support available.

T08 TensorFlow.js, T09 Elasticsearch

- Risks: High — specialized domain knowledge required. Correctness and performance sensitive.
- Maintenance: Must track breaking changes; performance tuning complex.
- Support: Community-based, paid support available for Elastic.

T10 Docker, T11 Kubernetes

- Risks: Moderate — infrastructure complexity.
- Maintenance: Requires DevOps expertise.
- Updates: Must follow security patches; rolling updates common.
- Support: Community + cloud vendor support.

T12 Git

- Risks: Low — ubiquitous tool.
- Support: Strong community.

T13 VS Code, T14 VS Code Copilot Chat

- Risks: Moderate — extension API evolves quickly.
- Maintenance: Must track VS Code releases to avoid breaking integration.
- Support: Docs, GitHub issues, community.

Update Criteria

- **Take Update:** Security fixes, critical bug fixes, LTS upgrades, performance improvements.
- **Rollback Update:** Breaking changes in APIs or data formats, regressions detected in regression test suite.
- **Source/Binary Availability:**
 - Open-source stack (TypeScript, Node.js, Express, PostgreSQL, Redis, TensorFlow.js, Elasticsearch, VS Code) provides source.
 - SaaS APIs (OpenAI, Claude, GitHub Copilot Chat) only expose SDKs/binaries — vendor-dependent.
- **Support Model:**
 - Open-source: community forums, GitHub issues.
 - Vendors: paid support (e.g., Elastic, Redis Enterprise, Anthropic, OpenAI).
- **Costs:**
 - Open-source: free, but higher in-house maintenance cost.
 - SaaS APIs: recurring usage fees.
 - Enterprise contracts: typically 10–20% of original license/subscription annually.

Rationale: Risks were evaluated by difficulty to **learn, design, implement, verify, maintain, and update**. The heaviest risks concentrate in the **Service module (retrieval, embeddings, indexing)** and **external APIs (OpenAI, Claude)** due to dependency on third parties and scaling complexity. Update strategy prioritizes **security and compatibility**, with rollbacks when regressions are found. Open-source technologies lower cost but shift maintenance burden to developers, while SaaS APIs trade flexibility for vendor-managed reliability.

Chat Log Link: <https://chatgpt.com/share/68d41b56-b6dc-800a-8128-e181ba19a95f>