

SOFTWARE REQUIREMENTS SPECIFICATION

kratai

Prepared by Kratai Engineering Team Client Developer Community

ABOUT THIS DOCUMENT

This Software Requirements Specification (SRS) describes what this system does, who uses it, and how its parts fit together - its actors and use cases, its data model, and the non-functional requirements it must satisfy. It's meant to give anyone unfamiliar with the codebase a clear, shared reference for what the system is and does, independent of implementation detail.

Overview

Kratai is a desktop workspace that parses local codebases to render interactive architectural class diagrams overlaid with live Git diffs, allowing developers to explore system structure and track uncommitted changes. By securely integrating multi-provider LLMs and interactive chat, it analyzes source code to auto-generate and export fully-dressed Use Case Models, Data Models, and PDF software specifications while managing AI token usage via PKCE-authenticated web services.

1. Use Case Diagram

This section shows how each actor interacts with the system through its key use cases.

The developer securely logs into the application via Kratai Web Services. They proceed through onboarding by selecting a local project and evaluating AI token costs. Developers can view, interact with, and analyze the auto-generated codebase architecture through the Class Diagram view, review uncommitted structural code changes, and chat with an AI assistant to explore the architecture. They can generate and view specification models (Use Case and Data Models) utilizing the multi-provider LLM integration, and export these as PDF reports.

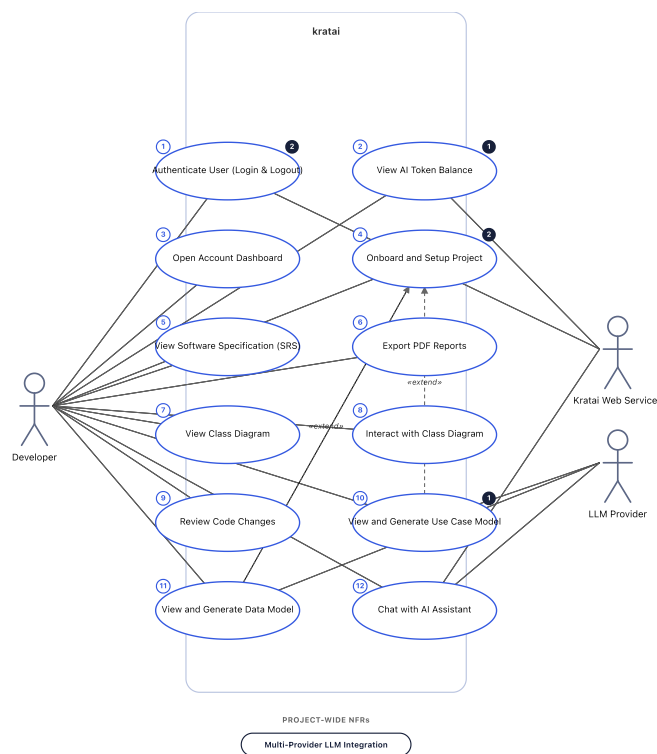


Figure 1: Use Case Diagram

2. Actors & roles

The following actors were identified as participants in the system, along with the use cases each one is involved in.

ACTOR	ROLE	DESCRIPTION	USE CASES
Developer	User	The developer using the desktop application.	Authenticate User (Login & Logout), View AI Token Balance, Open Account Dashboard, Onboard and Setup Project, View Software Specification (SRS), Export PDF Reports, View Class Diagram, Interact with Class Diagram, Review Code Changes, View and Generate Use Case Model, View and Generate Data Model, Chat with AI Assistant
Kratai Web Services	External System	The external web service managing accounts, tokens, and proxying AI requests.	Authenticate User (Login & Logout), View AI Token Balance, Chat with AI Assistant
LLM Provider	External System	The external AI backend processing generation requests.	View and Generate Use Case Model, View and Generate Data Model, Chat with AI Assistant

3. Use cases

Each use case below describes one capability the system provides, along with any requirements specific to it.

UC-1: Authenticate User (Login & Logout)

Log in and log out of the Kratai desktop application securely via PKCE browser flow.

Goal: *Establish a secure, authenticated local session to allow access to core application features and token usage.*

PRECONDITIONS

- Developer has downloaded and launched the desktop app.

MAIN FLOW

1. Developer opens the desktop application and is presented with a sign-in gate.
2. Developer chooses to sign in, which initiates a PKCE challenge.
3. System opens the default OS web browser to the Kratai Web Services login page.
4. Developer authenticates on the web.
5. Kratai Web Services redirects to a `kratai://` deep-link with an authorization code.
6. System intercepts the deep-link, exchanges the code for a token, and stores it in local device persistence.
7. System clears the token from local storage upon Developer requesting logout.

POSTCONDITIONS

- Developer has a valid local session token.
- The main application UI and project selection screens are unlocked.
- **NFR-1 PKCE OAuth Flow:** Authentication must utilize a PKCE challenge/verifier flow, opening the system browser for sign-in and capturing the result via a `kratai://` deep link callback.
- **NFR-2 Local Token Storage:** Device tokens retrieved from the exchange endpoint must be stored locally. Signing out only clears the local credential.

UC-2: View AI Token Balance

View remaining AI tokens directly from the desktop shell.

Goal: *Allow the developer to track their AI usage and remaining budget transparently.*

PRECONDITIONS

- Developer is authenticated.

MAIN FLOW

1. System reads the local session token.
2. System's main process securely queries the device-balance API at Kratai Web Services.
3. System broadcasts the fetched balance over IPC.
4. System renders the updated credit amount in the UI top bar meter.

POSTCONDITIONS

- The correct token balance is visible on-screen.
- **NFR-3 Desktop IPC Sync:** Token balance must be securely synchronized via IPC polling between the main process and renderer.

UC-3: Open Account Dashboard

Open the Kratai account dashboard in the default browser.

Goal: *Provide quick access to web-based account management.*

PRECONDITIONS

- None.

MAIN FLOW

1. Developer clicks the Manage Account or Dashboard button.
2. System intercepts the `kratai-action://open-dashboard` event.
3. System launches the default OS browser navigated to the external user dashboard.

POSTCONDITIONS

- The web dashboard is open in the user's browser.

UC-4: Onboard and Setup Project

Select a codebase folder and configure it for analysis.

Goal: *Import a local repository and safely prepare it for architectural extraction.*

PRECONDITIONS

- Developer is authenticated.

MAIN FLOW

1. Developer selects a new folder via the OS file picker or clicks a recent workspace.
2. System analyzes the raw directory size and file count.
3. System checks if Spec models already exist for the project.
4. If no specs exist, System prompts Developer with the projected AI cost versus their current balance.
5. Developer explicitly consents to generation (or skips).
6. System displays a real-time loading checklist while performing extraction or opening the shell.

POSTCONDITIONS

- The project is loaded and the main architecture shell is active.
- **NFR-4 Mandatory Auth Gate:** Sign-in is a hard gate; users cannot proceed to folder selection or the application UI without an active session.
- **NFR-5 Explicit AI Cost Consent:** First-open auto-generation of specs must display the current AI credit balance and project size, prompting for explicit user consent before proceeding.

UC-5: View Software Specification (SRS)

Read the generated Use Case Model and Data Model documents in a readable format.

Goal: *Provide a centralized, text-and-visual view of all extracted requirements and data schemas.*

PRECONDITIONS

- A project is onboarded and Spec models have been generated.

MAIN FLOW

1. Developer navigates to the Spec Document view in the desktop shell.
2. System renders the textual narrative, actor list, full use case details, and NFRs from the parsed spec data.
3. System renders the Data Model schemas and entity relationships.

POSTCONDITIONS

- The Spec document is displayed for reading.

UC-6: Export PDF Reports

Export the SRS or diagrams as a PDF file.

Goal: *Allow the developer to share offline architectural documentation with clients or team members.*

PRECONDITIONS

- Developer has an active workspace and a view open to export.

MAIN FLOW

1. Developer triggers a PDF export for a specific diagram or the full SRS view.
2. System orchestrates a background render of the active SVG or document.
3. System opens a native OS save dialog to request a file destination.
4. System writes the generated PDF file to the selected disk path.

POSTCONDITIONS

- A PDF file is successfully saved to the local file system.

UC-7: View Class Diagram

View the hierarchical Class Diagram of the parsed codebase.

Goal: *Allow the developer to visualize class structures, properties, methods, and folder groupings.*

PRECONDITIONS

- A project is onboarded and analyzed.

MAIN FLOW

1. Developer navigates to the Class Diagram view.
2. System parses local code files, extracting classes, interfaces, and methods.
3. System maps these nodes into their directory folder structures.
4. System renders the nodes, grouping them visually inside folder boxes.
5. System applies git diff statuses (added, modified, deleted) to nodes if changes exist.

POSTCONDITIONS

- The complete architectural hierarchy is visible.

UC-8: Interact with Class Diagram

Interact with Class Diagram elements such as folding folders, focusing classes, and toggling relationship lines.

Goal: *Enable navigation and decluttering of complex dependency graphs.*

PRECONDITIONS

- The Class Diagram is actively rendered.

MAIN FLOW

1. Developer clicks a folder node header to collapse or expand its internal classes.
2. Developer toggles global dependency lines on or off to reduce visual noise.
3. Developer clicks a specific class node to enter Focus Mode, dimming unrelated components.
4. Developer clicks a property or method to open the raw source code in the editor.

POSTCONDITIONS

- The diagram's visual state is updated based on the interaction.

UC-9: Review Code Changes

Review the structural impact of uncommitted git changes in the architecture diagrams.

Goal: *Provide a visual diff to understand how current work affects the overall architecture before committing.*

PRECONDITIONS

- The project has git tracking enabled and local changes exist.

MAIN FLOW

1. Developer modifies files locally in their IDE.
2. System detects uncommitted git modifications in the workspace.
3. System analyzes the differences to find added, modified, or deleted structural members (classes, methods).
4. System overlays color-coded highlights (e.g., green for added, yellow for modified, red for deleted) directly onto the Class Diagram nodes.
5. Developer visually reviews these highlights.

POSTCONDITIONS

- The architectural impact of the local diff is understood.

UC-10: View and Generate Use Case Model

Analyze codebase and extract actors and use cases into the Spec document.

Goal: *Automate the creation of a Use Case Model by having the AI analyze the codebase's behavior and routing.*

PRECONDITIONS

- Developer has sufficient AI tokens.

MAIN FLOW

1. Developer prompts the system to generate or regenerate the Use Case Model.
2. System gathers architectural context (routes, entry points, class data).
3. System securely sends the context to the LLM Provider.
4. LLM Provider identifies actors, use cases, and relationships, returning structured schema data.
5. System validates the schema, defensively dropping dangling references.
6. System saves the resulting Use Case Model to disk and renders it in the UI.

POSTCONDITIONS

- The Use Case Model is populated and visible.
- **NFR-7 Use Case Schema & Validation:** Functional Requirements are captured as fully-dressed use cases. Untrusted LLM schema generation must be safely validated by dropping dangling relations instead of failing. NFRs support a short name and detail text.

UC-11: View and Generate Data Model

Analyze codebase and extract data entities into the Spec document.

Goal: *Automate the creation of a Domain/Data Model by having the AI inspect ORM schemas, interfaces, and types.*

PRECONDITIONS

- Developer has sufficient AI tokens.

MAIN FLOW

1. Developer prompts the system to generate or regenerate the Data Model.
2. System gathers context around data access, schemas, and types.
3. System securely sends the context to the LLM Provider.
4. LLM Provider identifies entities, attributes, and relationships.
5. System validates and writes the resulting Data Model to disk.
6. System updates the Data Model view UI.

POSTCONDITIONS

- The Data Model is populated and visible.

UC-12: Chat with AI Assistant

Engage in an interactive conversation with the AI to ask questions about the architecture or instruct it to update specifications.

Goal: *Allow the developer to explore codebase architecture iteratively and trigger local tool actions through conversational AI.*

PRECONDITIONS

- Developer is authenticated and has an active workspace.

MAIN FLOW

1. Developer enters a query or instruction in the AI Chat panel.
2. System packages the current conversation history and workspace summary.
3. System sends a stateless request to Kratai Web Services via `/api/chat`.
4. Kratai Web Services proxies the request to the LLM Provider.
5. LLM Provider returns either a text response or a tool call request (e.g., `read_file`, `update_use_case_model`).
6. If a tool call is received, the local System executes it securely against the local codebase and loops the result back to the LLM.
7. System displays the final AI text response in the chat panel.

POSTCONDITIONS

- The developer's question is answered or the requested action is completed.
- AI token balance is decremented.

4. Project-wide non-functional requirements

The following non-functional requirements apply across the whole system, rather than to any single use case.

- **NFR-6 Multi-Provider LLM Integration:** The system must abstract LLM integration to support multiple configurable AI backends (e.g., Anthropic Claude and Google Gemini) for generation tasks.

5. Data model

This section describes the system's core data entities and how they relate to one another.

The conceptual domain model outlines the application's workspace and specification hierarchy. A User manages multiple Projects, each containing exactly one Spec. This Spec serves as a unified container composed of exactly one Use Case Model, Data Model, and Class Diagram. Requirements are tracked within the Use Case Model as individual Use Case Items. Non-Functional Requirements (NFRs) can be scoped globally directly to the Use Case Model, or locally to a specific Use Case Item.

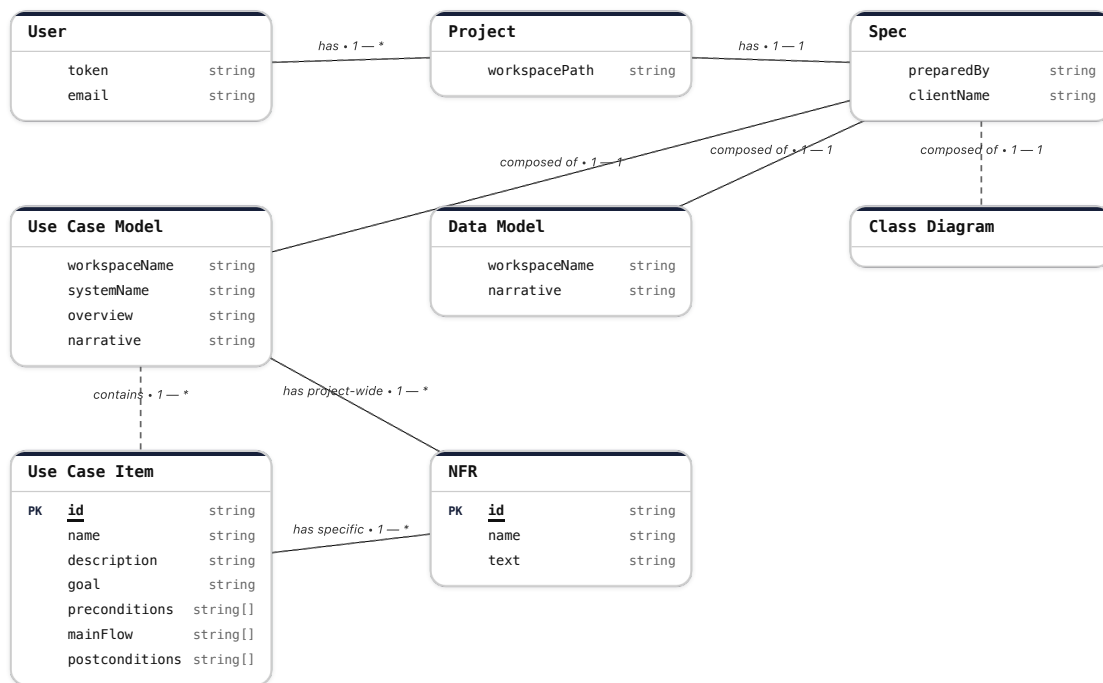


Figure 2: Data Model Entity-Relationship Diagram

User

ATTRIBUTE	TYPE	KEY
token	string	
email	string	

Project

ATTRIBUTE	TYPE	KEY
workspacePath	string	

Spec

ATTRIBUTE	TYPE	KEY
preparedBy	string	
clientName	string	

Use Case Model

ATTRIBUTE	TYPE	KEY
workspaceName	string	
systemName	string	
overview	string	
narrative	string	

Data Model

ATTRIBUTE	TYPE	KEY
workspaceName	string	
narrative	string	

Class Diagram

ATTRIBUTE	TYPE	KEY
-----------	------	-----

Use Case Item

ATTRIBUTE	TYPE	KEY
id	string	PK
name	string	
description	string	
goal	string	
preconditions	string[]	
mainFlow	string[]	
postconditions	string[]	

NFR

ATTRIBUTE	TYPE	KEY
id	string	PK
name	string	
text	string	

- User → Project (one-to-many - has)
- Project → Spec (one-to-one - has)

- Spec → Use Case Model (one-to-one - composed of)
- Spec → Data Model (one-to-one - composed of)
- Spec → Class Diagram (one-to-one - composed of)
- Use Case Model → Use Case Item (one-to-many - contains)
- Use Case Model → NFR (one-to-many - has project-wide)
- Use Case Item → NFR (one-to-many - has specific)