*Ryan A. Snell*

*Keiser University*

*Capstone, CSSE*

Inventory App

**Index**

[Requirements Questionnaire](#RequirementsQuestionaire)

[Markdown](#Markdown)

[Pandoc](#Pandoc)

[Agile (SCRUM)](#Agile)

[Design Documents](#DesignDoc)

[Dev Resources](#DevResources)

**Requirements Questionnaire**

This questionnaire is intended to guide stakeholders through defining and clarifying the functional and non-functional requirements for the Inventory Management App.

1. General Information

1.1 What is the primary goal of this inventory system?

1.2 Who are the intended users of the system (roles, technical ability)?

1.3 Will this application be used internally, externally, or both?

1.4 Is there a preferred platform for deployment (e.g., cloud provider, local server)?

2. Functional Requirements

2.1 What operations should users be able to perform on inventory items?

  (e.g., Create, Read, Update, Delete)

2.2 What data fields should each item include?

  (e.g., name, quantity, location, description, tags)

2.3 Should there be different access levels (e.g., admin vs. viewer)?

2.4 Do users need to search, filter, or sort items? If so, by which fields?

2.5 Will there be any item categories or tags?

2.6 Should the system support file/image attachments (e.g., product photos)?

2.7 Is version history or item change tracking needed?

3. Web & Mobile UI

3.1 What is the preferred look and feel of the app (minimal, detailed, themed)?

3.2 Are there accessibility or language requirements?

3.3 Should web and mobile UIs offer identical features?

3.4 What platforms should the mobile app support (iOS, Android, both)?

3.5 Is offline functionality required on mobile?

4. API and Integration

4.1 Will the API be public or restricted to internal use?

4.2 Are there other systems this API must integrate with (e.g., ERP, CRM)?

4.3 Should the API support batch operations (bulk updates)?

4.4 Will data import/export (CSV, Excel) be needed?

5. Security & Authentication

5.1 Will users need to authenticate (e.g., login via email, SSO)?

5.2 What roles or permissions are required?

5.3 Are audit logs or access logs required?

5.4 Is data encryption needed at rest and/or in transit?

6. Non-Functional Requirements

6.1 What performance expectations should be met (e.g., response time < 500ms)?

6.2 What is the expected number of concurrent users?

6.3 Should the system be scalable (multi-tenant, sharded)?

6.4 Are there reliability or uptime requirements (e.g., 99.9%)?

6.5 What backup and disaster recovery processes are needed?

7. Constraints & Assumptions

7.1 What are the project timelines or deadlines?

7.2 Are there budget constraints?

7.3 Is there existing infrastructure or technologies to consider?

7.4 Will training or documentation be required for end users?

Respondent:

Date:

Reviewed by:

**Markdown Manpages:**

* Common:

[**https://spec.commonmark.org/0.31.2/**](https://spec.commonmark.org/0.31.2/)

* GitHub:

[**https://github.github.com/gfm/**](https://github.github.com/gfm/)

* Markdown Syntax Summary:

[**https://daringfireball.net/projects/markdown/syntax**](https://daringfireball.net/projects/markdown/syntax)

* JetBrains Markdown:

<https://www.jetbrains.com/help/webstorm/markdown.html>

* Terminal Manual (if installed via markdown or pandoc):

If you have a markdown tool installed like markdown from the discount package or pandoc, you can try:

man markdown

Or for pandoc’s markdown:

man pandoc\_markdown

**PANDOC CONVERSION**

Pandoc is installed, but it’s missing the PDF engine it uses by default: pdflatex.

Install a LaTeX Engine

To generate PDFs with pandoc, you need a LaTeX engine. Here are two good options:

Option 1: Install MacTeX (Recommended for Full LaTeX Support)

brew install --cask mactex

• It’s about 4GB, but gives full compatibility.

• After installing, make sure /Library/TeX/texbin is in your $PATH.

Add this to your ~/.zshrc if needed:

export PATH="/Library/TeX/texbin:$PATH"

Then restart your terminal and try:

1. Check if pdflatex is available:

which pdflatex

If it returns something like /Library/TeX/texbin/pdflatex, you’re good to go.

2. Convert your Markdown files to PDF using pandoc: (inside the folder that the md files are located)

pandoc <md file names> -o <pdf.pdf file name desired>

Option 2: Use a Lighter PDF Engine like weasyprint

Alternatively, skip LaTeX entirely by using weasyprint:

brew install weasyprint

Then convert your Markdown to HTML:

pandoc full\_inventory\_design\_doc.md -o temp.html

And convert HTML to PDF:

weasyprint temp.html full\_inventory\_design\_doc.pdf

Optional (Make it Permanent):

If you want /Library/TeX/texbin to always be in your PATH, add this to your ~/.zshrc file:

export PATH="/Library/TeX/texbin:$PATH"

Then reload your terminal or run:

source ~/.zshrc

**AGILE**

Scrum:

<https://www.scrum.org/resources/scrum-guide>

**How to Create Agile Scrum Documents**

Agile Scrum emphasizes "working software over comprehensive documentation," so Scrum documents are concise, collaborative, and evolve with the project. This guide covers key artifacts and how to create them effectively.

**1. Product Vision Statement**

***Purpose:*** Defines the project’s big-picture goal and value.

***How to Create:*** State the product’s purpose (e.g., "A task app to boost team productivity via real-time collaboration"). Identify target users and key benefits. Keep it short (1-2 sentences), inspiring, and aligned with stakeholder goals.

***Tools:*** Google Docs, Confluence, or a simple text file.

***Example:*** "For teams who need efficient task management, our app provides real-time updates and intuitive tracking, delivering better coordination and faster results."

**2. Product Backlog**

***Purpose:*** A prioritized list of all desired features, fixes, and tasks.

***How to Create:*** List items (e.g., features, bugs, tech debt) as user stories, tasks, or epics. ***Format user stories:*** "As a [user], I want [action] so that [benefit]." Prioritize by value, urgency, or risk (e.g., use MoSCoW: Must, Should, Could, Won’t). Refine regularly with the team and Product Owner.

***Tools:*** Jira, Trello, Excel, or Notion.

***Example Entry:*** "As a user, I want to assign tasks so that my team knows their responsibilities."

**3. Sprint Backlog**

***Purpose:*** A subset of the product backlog selected for a sprint (e.g., 2-4 weeks).

***How to Create:*** During sprint planning, the team selects items from the product backlog. Break items into tasks (e.g., "Design UI," "Code API endpoint"). Estimate effort (e.g., story points, hours) collaboratively. Keep it flexible—adjust as needed during the sprint.

***Tools:*** Jira, Trello, physical Kanban board.

***Example Task:*** "Code task creation API – 3 story points."

**4. User Stories**

***Purpose:*** Describes features from the user’s perspective to focus on value.

***How to Create:*** Use the template: "As a [user type], I want [function] so that [benefit]." Add acceptance criteria (e.g., "Given a task, when I click 'Save,' then it appears in the list"). Keep them small, testable, and independent. Discuss with the team during backlog refinement.

***Tools:*** Jira, Confluence, or index cards.

***Example:*** "As a team lead, I want to see task status so that I can track progress.

***Criteria:*** Updates in real-time, shows 'To Do,' 'In Progress,' 'Done.'"

**5. Sprint Goal**

***Purpose:*** A short statement of the sprint’s focus and outcome.

***How to Create:*** Define a clear, achievable objective during sprint planning. Align it with the product vision and sprint backlog items. Keep it concise and measurable (e.g., "Enable task creation and basic UI for the app").

***Tools:*** Document in Jira, Confluence, or a sprint planning note.

***Example:*** "Deliver a functional task creation feature with a basic dashboard by sprint end."

**6. Burndown Chart**

***Purpose:*** Visualizes work completed versus time remaining in a sprint.

***How to Create:*** Plot remaining story points or tasks (y-axis) against sprint days (x-axis). Update daily based on completed work. Use to spot delays or scope creep.

***Tools:*** Jira, Excel, or draw on a whiteboard.

***Tip:*** A steep downward trend shows good progress; flat lines signal issues.

**7. Definition of Done (DoD)**

***Purpose:*** A shared checklist of criteria for a task or story to be considered complete.

***How to Create:*** Collaborate with the team to define standards (e.g., "Code written, tested, reviewed, deployed"). Ensure it’s specific, measurable, and consistent. Update as the team learns or processes evolve.

***Tools:*** Confluence, wiki, or a shared doc.

***Example:*** "1. Unit tests pass. 2. Code reviewed. 3. Deployed to staging. 4. Docs updated."

**8. Sprint Review and Retrospective Notes**

***Purpose:*** Documents feedback (review) and lessons learned (retrospective) after a sprint.

***How to Create:*** Review: Note what was delivered, stakeholder feedback, and backlog updates. Retrospective: Record "What went well," "What didn’t," and "Action items" (e.g., "Improve testing speed"). Keep it collaborative—use team input.

***Tools:*** Mural, Miro, Confluence, or a simple doc.

***Example:*** "Retrospective: Went well—daily standups; Didn’t—API delays; Action—add mock data."

**Tips for Success**

***Keep It Lightweight:*** Focus on just enough detail to guide the team.

***Collaborate:***Involve the Scrum team (Product Owner, Scrum Master, developers) in creation.

***Iterate:***Update documents as priorities or insights shift.

***Store Accessibly:***Use shared tools (e.g., Jira, Confluence, Git) for visibility.

***Align with Scrum:***Follow Scrum Guide principles.

**Example Outline**

***Title:*** Scrum Documents for Task App

***Team:*** Your Team Name

***Date:*** May 29, 2025

***Sprint:*** Sprint 1 of 10

Follow the sections above, tailoring to your project’s needs.

**DESIGN DOCUMENT**

**How to Create a Design Document**

A design document outlines the architecture, components, and plan for a project, ensuring clarity for developers, designers, and stakeholders. This guide walks you through the process.

**1. Purpose and Scope**

***Purpose:*** Define the goal of the project. What problem does it solve? For example, "Build a user-friendly task management app to boost team productivity."

***Scope:*** Specify what’s included (e.g., features, platforms) and what’s not (e.g., no hardware design).

***Audience:*** Identify readers -- developers, designers, project managers, or clients.

**2. Background and Context**

***Problem Statement:*** Describe the issue or need (e.g., "Current task tools lack real-time collaboration").

***Assumptions:*** List assumptions (e.g., "Users have internet access").

***Constraints:*** Note limitations (e.g., budget, timeline, tech stack).

**3. Requirements**

***Functional Requirements:*** Detail what the system must do (e.g., "Allow users to create, edit, and delete tasks").

***Non-Functional Requirements:*** Cover performance, scalability, security (e.g., "Load 100 tasks in under 2 seconds").

***User Stories:*** Write from the user’s perspective (e.g., "As a team lead, I want to assign tasks, so my team stays organized").

**4. System Overview**

***Architecture:*** Explain the high-level design (e.g., "Client-server model with a React frontend and Node.js backend").

***Components:*** List key parts (e.g., UI, database, API).

***Tech Stack:*** Specify tools (e.g., "React, Express, MongoDB, AWS").

**5. Design Details**

***Diagrams:*** Include visuals:

- Flowcharts for user workflows (e.g., task creation process).

- Architecture diagrams for system structure.

- Wireframes or mockups for UI.

***Data Models:*** Define schemas (e.g., "Task table: ID (int), Title (string), Status (enum: 'todo', 'done')").

***APIs/Interfaces:*** Outline endpoints (e.g., "GET /tasks - Retrieve all tasks, returns JSON").

**6. Implementation Plan**

***Milestones:*** Break into phases (e.g., "Week 1: Set up database, Week 2: Build API").

***Timeline:*** Provide a rough schedule (e.g., "Design: 1 week, Dev: 4 weeks, Testing: 2 weeks").

***Resources:*** List team roles, tools, and budget needs.

**7. Risks and Mitigation**

***Risks:*** Identify potential issues (e.g., "Delays if API integration fails").

***Mitigation:*** Plan solutions (e.g., "Use mock data for testing, allocate buffer time").

**8. Testing and Validation**

***Approach:*** Describe testing (e.g., "Unit tests for API, user testing for UI").

***Success Criteria:*** Define measurable outcomes (e.g., "90% of users complete tasks in <1 minute").

**9. Maintenance and Updates**

***Support:*** Explain post-launch plans (e.g., "Bug fixes via weekly sprints").

***Scalability:*** Note future growth (e.g., "Add Redis for caching if user base >10,000").

**Tips for Success**

***Keep It Concise:*** Focus on clarity and avoid over-documentation.

***Use Tools:*** Write in a text editor, store in Git, or use tools like Confluence, Notion, or Google Docs.

***Iterate:*** Update the document as the project evolves.

***Collaborate:*** Get feedback from stakeholders early.

**Example Outline**

***Title:*** Task Management App Design Document

***Author:*** Your Name

***Date:*** May 29, 2025

***Version:*** 1.0

Follow the sections above, tailoring content to your project.

**DEV RESOURCES**

MongoDB:

<https://www.w3schools.com/mongodb/index.php>

MongoDB manual page:

<https://www.mongodb.com/docs/manual/>

MongoDB Atlas manual page:

<https://www.mongodb.com/docs/atlas/>

MongoDB migration manual page:

<https://www.mongodb.com/docs/tools-and-connectors/>

MongoDB Shell installation instructions:

<https://www.mongodb.com/docs/mongodb-shell/install/?utm_campaign=w3schools_mdb&utm_source=w3schools&utm_medium=referral>

Node.js:

<https://www.w3schools.com/nodejs/default.asp>

Vue.js:

<https://www.w3schools.com/vue/index.php>

<https://vuejs.org/>

Render:

<https://render.com/docs>

Cloudflare:

[https://developers.cloudflare.com/workers/tutorials/?\_gl=1\*p1m2tu\*\_gcl\_au\*NTg5MjEwMzkuMTc0ODM2MzE3Mw..\*\_ga\*ZWNjYTBiNWEtZTBhZC00YmFlLTgyMjgtNTNjNDE3NDY0MTlm\*\_ga\_SQCRB0TXZW\*czE3NDg1NDkwNTkkbzIkZzAkdDE3NDg1NDkwNTkkajYwJGwwJGgw](https://developers.cloudflare.com/workers/tutorials/?_gl=1*p1m2tu*_gcl_au*NTg5MjEwMzkuMTc0ODM2MzE3Mw..*_ga*ZWNjYTBiNWEtZTBhZC00YmFlLTgyMjgtNTNjNDE3NDY0MTlm*_ga_SQCRB0TXZW*czE3NDg1NDkwNTkkbzIkZzAkdDE3NDg1NDkwNTkkajYwJGwwJGgw)

CORS:

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/CORS>

<https://fetch.spec.whatwg.org/>

<https://enable-cors.org/>

<https://web.dev/articles/cross-origin-resource-sharing>

Express.js/Node.js:

<https://expressjs.com/>

Axios:

<https://axios-http.com/docs/intro>

Mongoose.js:

<https://axios-http.com/docs/intro>

Vite:

<https://vite.dev/>

React Native:

<https://reactnative.dev/docs/getting-started>

Nodemon:

<https://nodemon.io/>

Jest.js:

<https://jestjs.io/>

Dotenv:

<https://github.com/motdotla/dotenv>

Turbo repo:

<https://turborepo.com/>

Prettier:

<https://prettier.io/docs/>

PostCSS:

<https://postcss.org/>

Expo/Metro:

<https://docs.expo.dev/>