

A PRACTICAL GUIDE
FOR EVERYONE

Build Software from a Document

A SIMPLE GUIDE FOR NON-TECHNICAL USERS

From a simple idea
to real software
with the help of AI

*Your Ideas
Can Become
Real Software*



A CLEAR
IDEA



A STRUCTURED
DOCUMENT



AI
UNDERSTANDS



WORKING
SOFTWARE

WRITTEN BY

Assem Hijazi

SIMPLE IDEAS. REAL POSSIBILITIES.



Author Introduction

We are entering an amazing new world.

The capabilities of AI assistants such as ChatGPT are developing so quickly that, in some ways, they can feel scary. At the same time, they are incredibly attractive to explore and use.

For this project, I deliberately chose one of the simplest possible ways to demonstrate what is changing in software creation.

I am **not targeting developers**. I am targeting regular users—people who have ideas for software but do not know how to program.

My question is very simple: **What happens when an ordinary user has an idea for software and explains that idea to ChatGPT? Can AI actually turn that idea into working software?**

Why I Chose the Simple Way

There are already more advanced ways to use AI for software development. We can use development environments, AI coding tools, agents, skills, specialized roles, virtual teams, and many other approaches.

- I deliberately ignored all of that for this experiment.
- I wanted to start with something familiar: **the normal web-based ChatGPT experience**.
- I selected a simple Mind Mapping Application as the example because the objective is not to prove that we can create the world's most advanced software.

The objective is to show how a relatively small idea from an ordinary user can become **real, usable software** through a conversation with AI.

Many people may still believe that building software with AI requires moving into a special development environment or understanding programming. Even opening a dedicated AI coding environment may make some non-technical users feel that they have entered the world of developers.

I wanted to remove that psychological barrier.

So the starting point is simply: **A user + ChatGPT + an idea + a document**.

Why a Document Instead of a Long Prompt?

I also deliberately did not take the simplest route of writing one large prompt saying: “Build me a mind mapping application with these features...”

Instead, I wanted to explore something more important.

Can we build software from a document?

I gave ChatGPT the idea, discussed some of the features I wanted, and explained that I wanted to demonstrate building software from a structured document.

I then asked ChatGPT to help divide that document into chapters so that it would be easy for both a **Human and AI** to read and understand.

Interestingly, even the document you are reading was developed with the assistance of ChatGPT.

There is nothing hidden about that.

I provided the idea, direction, decisions, feedback, and requirements. ChatGPT helped me structure, expand, and write them.

I reviewed the result, challenged parts of it, changed what I wanted, and eventually reached a document that describes the software I want to create.

And now I can give that document back to ChatGPT with an incredibly simple instruction:

Read and understand the complete attached document. Then build the software described in it.

That is the experiment.

A Document for Two Readers

There is another idea behind this project that I believe will become increasingly important.

Documents and books of the future should not necessarily be written only for humans.

They can be written for **two readers: Human + AI**.

The Human needs to understand the ideas, requirements, reasoning, and objectives.

AI needs to understand the same document so that it can help the Human analyze it, work with it, or—as in this case—**treat parts of the document as instructions and build something from it.**

This is why this document contains what I call **AI Advisory**.

AI Advisory speaks directly to AI inside a normal human-readable document.

It can tell AI how to interpret a requirement, what it should preserve, where it has freedom to make decisions, and how it should behave when building or modifying the software.

It does not disturb the Human reader. In fact, the Human can read exactly the same instructions and understand what AI is being asked to do.

There is no hidden layer.

Human and AI are reading the same document.

From a Small Experiment to Something Much Bigger

The Mind Mapping Application in this document is deliberately simple.

But the concept does not have to remain simple.

Imagine a much larger software project where different parts of the specification are prepared by real specialists:

Product Managers → Business Analysts → UX Designers → UI Designers → Developers → Architects → Testers → Security Specialists → Domain Experts

Each contributes knowledge to a structured software document.

The document becomes much richer.

AI reads the combined knowledge, understands the relationships between the requirements, and uses that knowledge to help build the software.

In that scenario, the document is no longer merely something produced **after or around software development**.

The document itself becomes part of the software creation process.

For this experiment, however, I intentionally avoided AI agents, complex skills, virtual development teams, and specialized AI roles.

I embedded the required knowledge and guidance into **one document** and asked AI to understand and perform from it.

If the simple version works, we can then imagine how far the concept could go when the document, the AI capabilities, and the people contributing to it become much more sophisticated.

Why I Wrote This

My objective is not to convince everyone that software development has suddenly become easy.

My objective is much simpler.

I want ordinary people to see that the distance between **having a software idea** and **seeing that idea become working software** is becoming dramatically smaller.

- You do not have to begin by asking: “How do I program this?”
- You can begin with: **“What do I want to build?”**
- Then describe it.
- Structure it.
- Give that knowledge to AI.
- And see what happens.

That is exactly what we are going to do in this document.

We are not going to learn how to program a Mind Mapping Application.

We are going to describe one well enough that AI can build it for us.

And perhaps, from this small experiment, people will begin imagining much bigger ones.

Assem Hijazi

Contents

Author Introduction	2
Why I Chose the Simple Way	2
Why a Document Instead of a Long Prompt?.....	3
A Document for Two Readers.....	3
There is no hidden layer.....	4
The document itself becomes part of the software creation process.	4
Why I Wrote This	5
Chapter1 Introduction	12
Chapter 2 How to Use This Document — Human + AI	14
2.1 A Document Written for Two Readers.....	14
2.2 The Human Does Not Need to Become a Programmer	15
2.3 The Role of the Human	16
2.4 The Role of AI	16
2.5 Why the Document Is Important.....	17
2.6 How This Document Is Organized	18
2.7 AI Advisory in This Document	19
2.8 The Document Does Not Need to Define Everything.....	20
2.9 From Building to Changing	20
2.10 The Human + AI Software Creation Cycle	21
Chapter Principle	21
Chapter 3 The Software We Want to Build	22
3.1 Software Overview	22
3.2 Purpose of the Application	22
3.3 Target User	23
3.4 Main Objectives	23
3.5 Application Environment	24
3.6 The Importance of the Standalone HTML Application.....	24
3.7 Scope of Version 1	25

3.8 What We Are Not Building	25
3.9 Expected Final Result	26
3.10 Human + AI Understanding	26
Chapter Principle	27
Chapter 4 Application Structure	28
4.1 Purpose of This Chapter.....	28
4.2 Main Application Structure	28
4.3 AS-01 — Main Canvas	29
4.4 AS-02 — Central Topic	30
4.5 AS-03 — Branches & Nodes	30
4.6 AS-04 — Main Toolbar.....	31
4.7 AS-05 — Node Editing Panel.....	31
4.8 AS-06 — Navigation Controls	32
4.9 AS-07 — Search	32
4.10 AS-08 — Save & Export	32
4.11 AS-09 — Application Information.....	33
4.12 Overall Screen Organization	33
4.13 Separation of Structure, Features and Design.....	34
4.14 AI Advisory — Application Structure.....	34
AI ADVISORY.....	34
Chapter Principle	35
Chapter 5 Software Features	36
5.1 Purpose of This Chapter.....	36
5.2 Version 1 Feature Overview.....	36
5.3 SF-01 — Create a Mind Map	37
5.4 SF-02 — Create Branches and Nodes	37
5.5 SF-03 — Edit Nodes.....	38
5.6 SF-04 — Delete Nodes and Branches	38
5.7 SF-05 — Move and Reorganize	38

5.8 SF-06 — Expand and Collapse	39
5.9 SF-07 — Notes	39
5.10 SF-08 — Status	39
5.11 SF-09 — Priority	40
5.12 SF-10 — Colors	40
5.13 SF-11 — Search	41
5.14 SF-12 — Zoom and Navigation	41
5.15 SF-13 — Undo and Redo	41
5.16 SF-14 — Map Information	42
5.17 Interaction Between Features	42
5.18 Features Intentionally Left for Future Versions	42
5.19 Human + AI Feature Understanding	43
5.20 AI Advisory — Software Features	43
AI ADVISORY	43
Chapter 6 Design & User Experience	45
6.1 Purpose of This Chapter	45
6.2 Design Objectives	45
6.3 Overall Visual Style	46
6.4 Visual Hierarchy of the Mind Map	47
6.5 Branch Color System	47
6.6 User-Controlled Colors	48
6.7 Node Appearance	48
6.8 Connections Between Nodes	49
6.9 Typography	49
6.10 Toolbar Design	49
6.11 Node Editing Panel	50
6.12 Interaction Feedback	50
6.13 Simplicity Before Decoration	51
6.14 Responsive Behavior	51

6.15 AI Design Freedom	51
6.16 Human + AI Design Understanding	52
6.17 AI Advisory — Design & User Experience	53
AI ADVISORY	53
Chapter Principle	53
Chapter 7 Save, Data & Export	54
7.1 Purpose of This Chapter.....	54
7.2 Save, Data & Export Overview.....	54
7.3 SD-01 — Automatic Local Save	55
7.4 SD-02 — New / Reset Map	55
7.5 SD-03 — JSON Backup	56
7.6 SD-04 — JSON Import.....	56
7.7 SD-05 — Standalone HTML	57
7.8 SD-06 — PDF Export	57
7.9 SD-07 — Image Export.....	58
7.10 SD-08 — Print	58
7.11 SD-09 — Data Integrity.....	59
7.12 What Should Be Stored	59
7.13 Export Is Different from Save	60
7.14 Keep File Handling Simple.....	61
7.15 Human + AI Understanding	61
7.16 AI Advisory — Save, Data & Export	62
AI ADVISORY	62
Chapter Principle	62
Chapter 8 AI Build Advisory	64
8.1 Purpose of This Chapter.....	64
8.2 Read the Complete Document Before Building.....	64
8.3 What the AI Must Preserve	65
8.4 What the AI May Decide	65

8.5 When Information Is Not Specified	66
8.6 Build Working Software, Not a Demonstration of Software	67
8.7 Keep the Technical Complexity Away from the Human	68
8.8 Do Not Overbuild Version 1	68
8.9 Build for Future AI Modification	69
8.10 Validate Before Delivering the Application	69
8.11 Handling Technical Limitations.....	70
8.12 Final AI Build Instruction	70
8.13 Human Understanding	71
Chapter Principle	71
Chapter 9 Changing the Software with AI	73
9.1 Purpose of This Chapter.....	73
9.2 The Generated Software Becomes the Next Source	73
9.3 Changes Can Be Described in Normal Language	73
9.4 Types of Changes	74
9.5 Understand Before Changing	75
9.6 Preserve What Is Already Working.....	75
9.7 Preserve Existing User Data.....	75
9.8 The Human Can Evaluate Visually	76
9.9 Modification Can Continue Iteratively	76
9.10 The Document Can Also Evolve	77
9.11 AI Change Advisory.....	78
AI CHANGE ADVISORY	78
9.12 A Simple Change Request.....	78
9.13 Human + AI Modification Cycle.....	79
Chapter Principle	79
Chapter 10 Build & Acceptance Checklist.....	80
10.1 Purpose of This Chapter.....	80
10.2 AI Build Verification	80

10.3 Human Acceptance Check	82
10.4 When Something Does Not Work.....	82
10.5 Version 1 Acceptance	83
10.6 Final AI Instruction Before Delivery	83
FINAL VALIDATION.....	83
10.7 The Complete Process.....	84
Chapter Principle	84
Conclusion	84

Chapter1 Introduction

What if creating software could begin with a **document instead of code**?

Millions of people understand their work, their business problems, and the tools they wish they had. They may know what a software application should do, what features it should contain, how users should interact with it, and even how they would like it to look.

What they often do not know is how to program it.

Traditionally, this creates a barrier between **having a software idea** and **turning that idea into working software**. Programming languages, development environments, frameworks, APIs, databases, hosting, and other technical requirements mean that software creation has largely remained the domain of developers.

Generative Artificial Intelligence is beginning to change this relationship.

A user can describe the required software in a structured document written primarily in normal human language. The document can explain the purpose of the software, its features, behavior, design, user interactions, information requirements, and expected outputs.

Artificial Intelligence can then work from that document as a knowledge and instruction source and translate the described application into functional software.

The relationship becomes: **SOFTWARE IDEA → SOFTWARE DOCUMENT → CHATGPT → WORKING SOFTWARE**

The user does not need to know how to write HTML, JavaScript, CSS, or other programming languages. The user does not need to decide how functions should be coded or how the internal technical structure should be implemented.

- The user's responsibility is primarily to explain **what should be built**.
- The AI's role is to determine **how to build it**.

This document demonstrates that approach through the creation of a simple browser-based **Mind Mapping Application**. The objective is not to reproduce every capability of existing mind-mapping products. Instead, the application is intentionally kept relatively simple so that it can demonstrate the larger idea clearly:

A person without software-development knowledge can describe software through a document, provide that document to ChatGPT, and receive a usable first version of a web-based application.

The resulting application should require no API, no development environment, and no programming knowledge from the user. It should be possible to open the generated HTML application directly in a web browser and begin using it.

The relationship does not have to end after the first version. The user can use the application, identify improvements, describe those changes in normal language, update the document where appropriate, and ask AI to generate the next version.

This creates a new and accessible software creation cycle: **DESCRIBE → BUILD → USE → REVIEW → CHANGE → BUILD AGAIN**

The purpose of *Build Software from a Document* is therefore larger than creating one mind-mapping application.

It demonstrates how the combination of **human knowledge, a well-structured software document, and Artificial Intelligence** can allow people who are not programmers to participate directly in creating the software they need.

Chapter 2 How to Use This Document — Human + AI

Yes. Since **2.1 already establishes the two-reader concept**, I would integrate the crucial “AI must actually build from this document” idea directly into 2.1 rather than creating a separate 2.2. It will be stronger and avoid repetition.

2.1 A Document Written for Two Readers

Build Software from a Document is designed differently from a traditional software specification.

It is written for **two readers at the same time**:

1. **The Human** — the person who understands the idea, need, business problem, desired features, and expected experience.
2. **The Artificial Intelligence** — the system that reads and interprets the document and transforms its specifications into working software.

The human is not expected to understand programming. The AI is not expected to independently decide what software the human needs.

Instead, this document creates a **shared knowledge foundation** between them.

This follows the Human + AI principle introduced in *Educate AI Before You Ask*: the human and AI can work from the same knowledge while performing different responsibilities. The human provides purpose, context, requirements, judgment, and validation, while AI provides interpretation, technical implementation, and software generation.

In this document, that relationship is applied directly to software creation.

Importantly, **this document is not intended only to explain the software to a human reader**. Once completed, the document is intended to be uploaded in its complete form to an Artificial Intelligence system such as ChatGPT.

The AI should first **read and understand the complete document** before beginning software development. The document collectively defines the software purpose, application structure, features, behavior, design, data handling, saving, export requirements, and other expectations.

The AI should therefore treat this document as both:

- **A document for the human to understand and define the software**

- **A primary build specification for AI to create the software.**

Where the document defines what is required but intentionally does not specify the technical implementation, AI may determine an appropriate implementation while remaining consistent with the objectives, requirements, and boundaries established throughout the document.

The principle is simple: **The human defines what software is needed. The document establishes that knowledge. AI determines how to technically build it.**

The relationship therefore becomes: **HUMAN IDEA → SHARED DOCUMENT → AI UNDERSTANDING → AI BUILDS → WORKING SOFTWARE**

2.2 The Human Does Not Need to Become a Programmer

The purpose of this approach is not to teach people how to code.

A user should not need to learn HTML, CSS, JavaScript, programming frameworks, development environments, or software architecture simply to create a relatively simple application. Instead, the human concentrates on something they already understand:

What do I want the software to do?

For example, when defining the Mind Mapping Application used in this document, the human can say:

- I want to create parent and child topics.
- I want to move topics around.
- I want branches to have different colors.
- I want to change those colors myself.
- I want to add notes and status information.
- I want to save my work.
- I want to export the result as PDF or an image.
- I want the application to be easy to use.

The human does not need to explain which JavaScript function should create a node, how browser events should be handled, how the data should be represented internally, or which CSS rules should produce the interface.

Those are implementation decisions that AI can make.

The principle is simple: **The Human defines what should be built. AI determines how it can be built.**

2.3 The Role of the Human

The human remains the owner of the software idea. AI may build the application, but the human defines its purpose and determines whether the result is useful.

Human Responsibility	Meaning
Define the Idea	Explain what software is needed and why.
Define the Features	Describe what the application should be able to do.
Define Important Behavior	Explain how the user expects the application to respond.
Define Preferences	Specify important design, color, layout, or usability preferences.
Review the Result	Use the generated application and determine whether it meets expectations.
Identify Problems	Tell AI what is missing, incorrect, difficult, or unnecessary.
Request Changes	Describe improvements in normal human language.
Approve the Software	Decide when the application is suitable for its intended use.

The human therefore remains actively involved throughout the process. AI does not replace the human's understanding of the problem. It reduces the technical barrier between **understanding the problem** and **building something that helps solve it**.

2.4 The Role of AI

The AI has a different responsibility. It should first understand the document before attempting to build the application.

The document becomes the AI's primary source for understanding: **Purpose → Structure → Features → Behavior → Design → Data → Output**

AI then translates that understanding into the technical components required to make the software work.

For this demonstration, AI may determine technical matters such as the HTML structure, CSS implementation, JavaScript functions, internal data structures, browser interactions, rendering methods, local storage mechanisms, and other implementation details.

These decisions do not need to be explained by the human unless the human specifically wants to control them.

This is an important distinction: **The document describes the software rather than attempting to describe the code.**

2.5 Why the Document Is Important

A simple instruction such as: “Build me a mind mapping application.”

may allow AI to generate something.

But it leaves many questions unanswered.

- What features should it contain?
- How should nodes behave?
- Should information be saved?
- Can colors be changed?
- What should be exported?
- Should branches collapse?
- Should the application work offline?
- What information belongs to a node?

AI would have to make many assumptions. This document reduces those assumptions by establishing the required knowledge **before asking AI to build**.

That reflects the central methodology of *Educate AI Before You Ask*: important AI work benefits from establishing the relevant knowledge and context rather than assuming that the AI already has everything required for the task.

For software creation, we can express the same idea very simply:

- **Don't begin with: ASK AI → BUILD SOFTWARE**

- Instead: **DEFINE → DOCUMENT → EDUCATE AI → BUILD SOFTWARE**

2.6 How This Document Is Organized

The following chapters progressively educate both readers about the application.

Document Area	Human Learns	AI Learns
Software Purpose	What we are trying to create	What the application is intended to achieve
Application Structure	What the main parts of the software are	What major interface components need to exist
Software Features	What the application can do	What functionality must be implemented
Design & User Experience	How the application should look and feel	What visual and interaction principles to follow
Save, Data & Export	What happens to the user's information	How information must be stored, restored, and exported
AI Build Advisory	What freedom AI has when building	How to interpret unspecified technical decisions
Changing the Software	How to request future improvements	How to modify the existing application without unnecessarily changing other parts
Acceptance Checklist	How to review the result	What should be verified before presenting the application

This structure intentionally separates **features from design**.

For example:

- **Feature:** The user can change the color of a node.
- **Design:** Main branches should initially use clearly distinguishable colors.
- **AI Build Guidance:** If the human has not specified an exact color, AI may choose an appropriate professional default.

Keeping these concepts separate makes the document easier for both readers to understand.

2.7 AI Advisory in This Document

This document also introduces a limited **AI Advisory** layer.

The concept is adapted from *Educate AI Before You Ask*, where AI Advisory provides practical guidance about how established knowledge should influence AI assistance rather than turning every section into a collection of prompts.

For this project, AI Advisory will be deliberately simple.

There are two primary situations where it is needed.

AI Build Advisory

- This guides AI when creating the first version of the software.

AI BUILD ADVISORY

- Read and understand the complete document before beginning implementation. Treat the document as the primary specification for the application. Preserve all required features and behaviors. Where technical implementation is not defined, make reasonable decisions that support simplicity, usability, portability, and the overall purpose of the application.

AI Change Advisory

- This guides AI when modifying software that has already been generated.

AI CHANGE ADVISORY

- When the human requests a change, first understand the existing application and the requested modification. Preserve existing functionality, data, hierarchy, and unrelated design elements unless the requested change requires otherwise. Modify what is necessary without unnecessarily rebuilding or removing working features.

AI Advisory therefore provides **guidance rather than programming instructions**.

2.8 The Document Does Not Need to Define Everything

An important part of this approach is knowing what **not** to specify. If every technical detail had to be documented, the human would once again need software-development knowledge.

That would defeat the purpose. The document should therefore define what matters to the human and allow AI reasonable freedom over technical implementation.

For example:

- The human may specify: “The application must remember the user's mind map when the browser is closed and reopened.” The document does not necessarily need to explain the JavaScript code or browser-storage mechanism required to accomplish that. AI can determine an appropriate implementation.
- Similarly, the human can specify: “Each main branch should automatically receive a different color.” AI can determine how those colors are generated and applied unless the document specifies particular colors.

The objective is therefore **sufficient definition, not maximum definition**.

2.9 From Building to Changing

Creating Version 1 is only the beginning. After using the application, the human may discover new ideas.

- Perhaps the user decides: “Make the Work branch dark blue.”
- or: “Add a Budget field to every topic.”
- or: “Move the toolbar to the left.”
- or: “When I double-click a topic, create a child topic.”

The human should not need to open the source code and find the correct programming function. Instead, the existing application can be provided back to ChatGPT together with the requested change.

The cycle becomes: **DOCUMENT → CHATGPT → VERSION 1 → HUMAN USES THE SOFTWARE → HUMAN DESCRIBES A CHANGE → CHATGPT → VERSION 2**

This is an important part of the concept. AI is not only being used to **generate** software.

It becomes a conversational way to **evolve** software.

2.10 The Human + AI Software Creation Cycle

The complete approach presented in this document can therefore be summarized as:

- 1. DEFINE** The human defines the software idea.
- 2. DOCUMENT** The idea is transformed into a structured description that both Human and AI can understand.
- 3. EDUCATE** AI receives the complete document as the knowledge foundation for the application.
- 4. BUILD** AI translates the document into working software.
- 5. USE** The human opens and uses the generated application.
- 6. EVALUATE** The human determines what works, what does not, and what could be improved.
- 7. CHANGE** The human describes desired changes in normal language.
- 8. BUILD AGAIN** AI modifies the application and produces the next version.

The cycle can continue for as long as the software remains useful.

Chapter Principle

The human does not need to tell AI how to program the software. The human needs to establish sufficient knowledge for AI to understand what software should be built.

The document becomes the bridge between the two.

Human Knowledge + Structured Document + AI Capability → Working Software

And that is the foundation of **Build Software from a Document**.

Chapter 3 The Software We Want to Build

3.1 Software Overview

The first application created using the **Build Software from a Document** approach will be a browser-based **Mind Mapping Application**.

A mind map provides a simple visual method for organizing information around a central subject. Ideas can be divided into main branches, sub-branches, and additional levels, allowing information to be understood as a visual hierarchy rather than only as paragraphs, lists, or tables.

The application will allow a user to create and manage this structure visually through a normal web browser.

The purpose of selecting a Mind Mapping Application is not to create the most advanced mind-mapping software available.

It has been selected because it provides a clear demonstration of what AI can build from a document. It combines several common elements of software development: structured information; interactive visual elements; user input; editing; navigation; colors and design; data storage; import and export; user preferences.

This makes it a useful first demonstration of the **Build Software from a Document** concept.

3.2 Purpose of the Application

The purpose of the application is to allow users to **visually create, organize, modify, and save information in a mind-map structure**.

- A user should be able to begin with a central topic and progressively create branches and sub-branches.
- The application should make it easy to reorganize ideas, change their appearance, add basic information, navigate larger maps, and save the result.
- Most importantly, the application should feel like **software**, not simply a static webpage displaying a diagram.
- The user should be able to interact with it, make changes, save information, close it, reopen it, and continue working.

3.3 Target User

The application is designed primarily for an ordinary computer user. The user may understand how to use a browser and common productivity applications but should not be expected to understand: **HTML, CSS, JavaScript, programming, software frameworks, APIs, databases, or development environments.**

The application should therefore be intuitive enough that a user can understand its basic operation without technical knowledge.

This principle also applies to future modification of the software.

If the user wants something changed, the preferred approach should be to **describe the requested change to ChatGPT**, rather than requiring the user to modify the program code.

3.4 Main Objectives

The first version of the application has a deliberately limited number of objectives.

Objective	Description
Create	Allow users to create a central topic, branches, and sub-branches.
Organize	Allow information to be moved, edited, expanded, collapsed, and reorganized.
Visualize	Present information through a clear and attractive visual hierarchy.
Personalize	Allow basic control over colors and appearance.
Add Information	Allow useful information such as notes, status, and priority to be associated with topics.
Navigate	Allow users to zoom, pan, search, and move around the map easily.
Save	Preserve the user's work so that it can be reopened and continued.
Export	Allow the user to produce useful outputs such as HTML, PDF, image, or data backup where supported by the application.
Change with AI	Allow the generated software to be returned to ChatGPT for future changes without requiring programming knowledge.

These objectives establish what the software should achieve. Detailed functionality will be defined later in the **Software Features** chapter.

3.5 Application Environment

The first version should be designed as a **browser-based application**. The preferred deliverable is a standalone HTML file that contains the components necessary for the application to operate.

For example: **MyMindMap.html**

The user should be able to open the file using a normal modern web browser and begin using the application.

For the first version, the application should aim to operate without requiring: a dedicated server; a separate software installation; an API connection; a development environment; programming tools; programming knowledge from the user.

Where possible, the application should keep its core functionality contained within the generated application itself. This makes the software highly portable.

The HTML file can potentially be stored, copied, moved, backed up, or shared like another normal computer file.

3.6 The Importance of the Standalone HTML Application

The standalone HTML approach is particularly important to this demonstration. Normally, the word **software** may make a non-technical user think about installation packages, servers, databases, cloud infrastructure, development environments, or application stores.

For many lightweight applications, this does not necessarily need to be the starting point. Modern browsers are already capable application environments.

HTML, CSS, and JavaScript can provide the interface, design, interaction, logic, and local data handling required for many useful applications.

The human does not need to understand these technologies. ChatGPT can use them as implementation technologies while the human continues to work at the level of the software idea.

Therefore:

- **The human describes the application.**
- **ChatGPT creates the technical implementation.**
- **The browser runs the application.**

This is one of the reasons the first demonstration focuses on a standalone web application.

3.7 Scope of Version 1

Version 1 should remain intentionally simple. The objective of this project is **not to measure success by the number of features generated**.

Adding dozens of advanced capabilities could make the document unnecessarily complicated and hide the central idea being demonstrated. Instead, Version 1 should contain enough functionality to demonstrate that ChatGPT can transform a structured software document into a useful interactive application.

The application should therefore focus on a small number of well-defined capabilities:

Create → Organize → Design → Navigate → Save → Export → Modify

If those capabilities work well, the demonstration has achieved its purpose.

3.8 What We Are Not Building

This project is not intended to reproduce every capability of established commercial mind-mapping products.

It is also not intended, in its first version, to become: a large enterprise application; a multi-user collaboration platform; a cloud service; a complex database system; an AI-integrated application requiring external APIs; a replacement for professional mind-mapping platforms.

Those capabilities may be possible in future versions, but they are outside the purpose of this demonstration. Keeping the scope controlled is intentional.

- The question being tested is not: **How much software can ChatGPT build?**
 - The more important question is: **Can a non-programmer define useful software in a document and have ChatGPT turn that document into a working application?**
-

3.9 Expected Final Result

At the end of the process, the human should be able to provide this completed document to ChatGPT and request that the described application be built.

The expected result is a **working browser-based Mind Mapping Application** that reflects the structure, functionality, behavior, and design established in this document.

The human should then be able to: **Open it → Use it → Save work → Review it → Request changes**

The human should not need to understand the source code in order to accomplish these activities.

The generated software itself also becomes part of the continuing Human + AI relationship.

After Version 1 has been created, the human may provide the application back to ChatGPT and request further changes in normal language.

For example:

- “I want a different color for each main branch.”
- “Add a new field called Progress.”
- “Make the nodes larger.”
- “Add another export option.”
- “I don't like the toolbar. Make it simpler.”

The human describes the desired result. AI modifies the technical implementation. This means the same approach used to **create** the software can also be used to **evolve** it.

3.10 Human + AI Understanding

Before moving into the detailed structure and features of the application, both readers should have the same understanding of what is being created.

Human Understanding	AI Understanding
I am defining a simple Mind Mapping Application.	I am expected to build a functional Mind Mapping Application.
I do not need to define the programming implementation.	I may determine appropriate technical implementation where it is not specified.
The application should be easy for ordinary users.	Technical choices should prioritize usability and simplicity.

Version 1 does not need every possible feature.	Do not unnecessarily expand the scope beyond the specification.
The application should work as real usable software.	Build interactive functionality rather than a static visual mockup.
I should be able to use it from a browser.	The application should be browser-based and preferably standalone.
I can later ask AI to change it.	The implementation should remain understandable and maintainable enough for future AI modification.

Chapter Principle

Build enough software to prove the idea—not enough features to hide it.

The Mind Mapping Application is the first demonstration.

The larger objective is to show that the distance between **having a software idea** and **having working software** can become dramatically smaller when a human can describe the required application clearly and AI can handle the technical implementation.

IDEA → DOCUMENT → CHATGPT → SOFTWARE

The next chapter defines the **Application Structure**—the main parts that ChatGPT should create before we define exactly what each part can do.

Chapter 4 Application Structure

4.1 Purpose of This Chapter

The previous chapters established **why the application is being created and what it is intended to achieve**. This chapter begins defining the actual structure of the software that Artificial Intelligence will later build.

The application should be organized into a small number of clear areas. Each area has a specific purpose and will later contain the features defined in the following chapters.

This chapter does not define every function in detail. Its purpose is to establish the **main building blocks of the application** so that both the Human and AI share the same understanding of its overall structure.

When this complete document is later provided to AI, the application areas defined in this chapter should be treated as components of the software to be created.

4.2 Main Application Structure

The Mind Mapping Application should contain the following primary areas:

ID	Application Area	Purpose
AS-01	Main Canvas	The primary visual workspace where the mind map is displayed and manipulated.
AS-02	Central Topic	The root topic from which the entire mind map begins.
AS-03	Branches & Nodes	The visual hierarchy used to organize main topics, subtopics, and additional levels.
AS-04	Main Toolbar	Provides quick access to the most important application actions.
AS-05	Node Editing Panel	Allows the user to edit information and properties associated with the selected node.
AS-06	Navigation Controls	Allows the user to zoom, fit, pan, and navigate around the mind map.

AS-07	Search	Allows the user to locate information within the map.
AS-08	Save & Export	Provides access to saving, backup, import, and export functions.
AS-09	Application Information	Provides useful information and feedback about the current map and application state.

These areas define the overall application structure. Their detailed functionality will be established in later chapters.

4.3 AS-01 — Main Canvas

The **Main Canvas** is the central working area of the application.

- It should occupy the majority of the available screen because the mind map itself is the primary focus of the software.
- The canvas displays the central topic, branches, nodes, connections, and the visual relationships between them.
- The user should be able to interact directly with the canvas rather than depending entirely on menus or forms.

The canvas will support actions such as:

- viewing the complete mind map;
- selecting nodes;
- moving around the map;
- zooming;
- expanding and collapsing branches;
- interacting with topics and branches.

The detailed interaction behavior will be defined later. The canvas should visually remain the **main workspace of the application**.

4.4 AS-02 — Central Topic

- Every mind map begins with a **Central Topic**.
- The Central Topic represents the main subject of the map and acts as the root from which all primary branches originate.

For example, a user creating a planning map might begin with: **My Business Plan**. The user can then create primary branches such as: **Marketing — Finance — Operations — Technology**

- The Central Topic should be visually distinguishable from other nodes so that the user can immediately understand where the hierarchy begins.
 - The user should be able to edit its title and basic appearance.
-

4.5 AS-03 — Branches & Nodes

- The information within the mind map will be organized through **Branches and Nodes**.
- A primary branch connects directly to the Central Topic.
- Each branch may then contain child nodes, and those nodes may contain additional child nodes.

This creates a hierarchical structure:

Central Topic

→ **Main Branch**

→ **Child Node**

→ **Sub-Child Node**

→ **Additional Levels**

- The application should not visually restrict the user to only one or two levels.
 - Branches and nodes are the main information objects of the application and will therefore support many of the features defined later in this document.
-

4.6 AS-04 — Main Toolbar

- The application should provide a **Main Toolbar** containing quick access to commonly used actions.
 - The toolbar should remain simple and should not dominate the screen.
 - Typical actions may include:
 - **Add → Edit → Delete → Undo → Redo → Search → Save → Export**
 - Additional controls may be included where they are required by features defined later in this document.
 - The toolbar should use understandable labels, icons, or a combination of both.
 - The objective is to allow an ordinary user to discover the application's primary actions without requiring technical knowledge or extensive instructions.
-

4.7 AS-05 — Node Editing Panel

- When a node is selected, the user should have access to a **Node Editing Panel**.
- This area provides a convenient location for viewing and modifying information associated with the selected node.
- Depending on the features defined later, the panel may include information such as:

Property	Purpose
Title	The text displayed on the node
Notes	Additional information associated with the node
Status	Basic status of the topic or task
Priority	Importance of the node
Color	User-selected visual color
Other Properties	Additional properties defined by later requirements

- The editing panel should appear only when useful and should not unnecessarily reduce the available mind-map workspace.
- AI may determine the most appropriate technical implementation—for example, a side panel, floating panel, or another simple interface—provided that the resulting design remains easy to understand and use.

4.8 AS-06 — Navigation Controls

- Mind maps can become larger than the visible screen.
 - The application should therefore provide basic **Navigation Controls**.
 - These controls should allow the user to comfortably explore both small and larger maps.
 - Navigation should include access to functions such as:
 - **Zoom In — Zoom Out — Fit Map to Screen — Reset View**
 - The user should also be able to move around the canvas naturally.
 - The exact technical implementation is not prescribed by this document. AI may select appropriate browser interaction methods while maintaining intuitive behavior.
-

4.9 AS-07 — Search

- The application should provide a simple **Search** capability.
 - As a mind map becomes larger, manually locating a particular topic may become difficult.
 - Search provides a way for the user to enter text and locate matching nodes.
 - The interface should remain simple and should allow the user to understand where the matching information exists within the map.
 - Detailed search behavior will be defined in the Software Features chapter.
-

4.10 AS-08 — Save & Export

- The application should contain a clearly identifiable **Save & Export** area.
 - This area provides access to functions related to preserving, transferring, backing up, and exporting the user's work.
 - The application may provide functions such as:
 - **Save → Open/Import → Backup → Export → Print**
 - The specific formats and behavior will be defined in the **Save, Data & Export** chapter.
 - This area is particularly important because the user should be able to create useful information without being concerned about losing the work.
-

4.11 AS-09 — Application Information

- The application may provide a small **Application Information** area for useful feedback.
- This should not become a complex dashboard.
- Its purpose is simply to communicate information that helps the user understand the current state of the application.

Examples may include:

- total number of nodes;
- selected node information;
- save status;
- basic application messages;
- current map name.

Only information that provides practical value should be displayed.

4.12 Overall Screen Organization

- The application should prioritize the mind map itself.
- A suggested conceptual arrangement is:

Screen Area	Content
Top / Primary Control Area	Main Toolbar, Search, Save & Export
Center	Main Canvas and Mind Map
Side / Context Area	Node Editing Panel when required
Canvas Controls	Zoom and navigation
Small Information Area	Application status and useful map information

- This arrangement is **guidance rather than a fixed visual design**.
- The detailed appearance of these areas belongs to the later **Design & User Experience** chapter.
- AI may improve the exact positioning where doing so creates a cleaner and more usable interface, provided that all required application areas remain available.

4.13 Separation of Structure, Features and Design

It is important for both the Human and AI to distinguish between three concepts.

Concept	Question It Answers	Example
Application Structure	What parts exist?	The application contains a Node Editing Panel.
Software Feature	What can it do?	The user can change the color of a selected node.
Design	How should it look and feel?	The color selector should be simple and visually clear.

- This chapter defines primarily **what parts exist**.
 - The next chapter defines **what those parts can do**.
 - A later chapter defines **how the application should look and feel**.
 - This separation allows the Human to define the desired software clearly while giving AI sufficient information to implement it correctly.
-

4.14 AI Advisory — Application Structure

AI ADVISORY

- Treat the application areas identified as **AS-01 through AS-09** as part of the software build specification.
- When building the application, create a coherent interface that provides all required areas while keeping the Main Canvas as the primary workspace.
- The exact HTML structure, CSS layout, JavaScript architecture, positioning methods, and technical implementation are not prescribed unless specifically defined elsewhere in this document.
- You may make appropriate implementation and layout decisions where necessary, but **do not remove a required application area simply to simplify development**.
- Where this chapter provides examples of possible controls or properties, verify them against the detailed requirements in later chapters before determining the final implementation.

- This last sentence is important because the AI will be reading the **whole document**, not building Chapter 4 in isolation.
-

Chapter Principle

- **Define the parts of the software clearly. Let AI determine the technical structure required to make those parts work together.**
- The application now has its basic structure.
- The next chapter, **Software Features**, defines exactly what the user should be able to do with it.

Chapter 5 Software Features

5.1 Purpose of This Chapter

This chapter defines the **functional capabilities** of the Mind Mapping Application.

Chapter 4 established the main parts of the application. This chapter tells both the Human and AI **what the user must be able to do with those parts**.

Because this document will later be provided directly to AI to build the software, the features defined here should be treated as **functional requirements for Version 1**, not merely as examples or suggestions.

Version 1 is intentionally kept focused. The objective is not to create the largest possible mind-mapping application, but to demonstrate that a relatively simple document can provide AI with enough knowledge to create useful working software.

5.2 Version 1 Feature Overview

The application should provide the following core capabilities:

ID	Feature Area	Main Capability
SF-01	Create Map	Create a central topic and build a hierarchy from it
SF-02	Create Nodes	Add main branches and child nodes
SF-03	Edit Nodes	Change node titles and information
SF-04	Delete Nodes	Remove individual nodes or complete branches
SF-05	Move & Reorganize	Reposition and reorganize nodes
SF-06	Expand & Collapse	Hide or reveal branches

SF-07	Notes	Add additional information to a node
SF-08	Status	Assign a basic status to a node
SF-09	Priority	Assign a priority to a node
SF-10	Colors	Automatically color branches and allow user color changes
SF-11	Search	Find nodes within the map
SF-12	Zoom & Navigation	Navigate comfortably around the canvas
SF-13	Undo & Redo	Reverse or restore recent actions
SF-14	Map Information	Display useful basic information about the map

Saving, importing, exporting, and file handling are defined separately in **Chapter 7 — Save, Data & Export**.

5.3 SF-01 — Create a Mind Map

- The application should allow the user to create a mind map beginning with a **Central Topic**.
 - The Central Topic becomes the root of the complete information hierarchy.
 - When starting a new map, the user should be able to give the Central Topic a name and begin adding information immediately.
 - The application should present the map visually rather than requiring the user to manually position every element before beginning.
 - The objective is to make starting a new map quick and simple.
-

5.4 SF-02 — Create Branches and Nodes

- The user should be able to add:
- **Central Topic → Main Branch → Child Node → Additional Child Levels**

- A new node should be visually connected to its parent.
 - The application should automatically position newly created nodes in a logical location. The user should not be required to manually draw connecting lines or calculate positioning.
 - The user should be able to continue creating additional levels where needed.
 - Common creation actions should be easy to access through the interface.
-

5.5 SF-03 — Edit Nodes

- The user should be able to select a node and change its title.
 - Editing should be simple and direct. The user should not need to open a complex form simply to rename a topic.
 - Where practical, common interactions such as double-clicking a node may be used to begin editing.
 - Changes should immediately appear on the map.
 - Additional information associated with a node should be editable through the Node Editing Panel defined in Chapter 4.
-

5.6 SF-04 — Delete Nodes and Branches

- The user should be able to delete a node.
 - If the selected node contains child nodes, deleting it may also remove the complete branch beneath it.
 - Because this action could remove multiple pieces of information, the application should provide appropriate protection against accidental deletion where necessary.
 - The Central Topic should not be accidentally removable in a way that leaves the map without a valid root.
-

5.7 SF-05 — Move and Reorganize

- One of the main benefits of mind mapping is the ability to reorganize ideas as thinking develops.
- The user should therefore be able to move nodes and reorganize the hierarchy.
- Where practical, the application should support intuitive interaction such as dragging a node to another appropriate location or parent.

- When a node with children is moved, its child structure should remain connected to it.
 - The user should not lose the contents of a branch simply because its position or parent has changed.
-

5.8 SF-06 — Expand and Collapse

- Branches should be expandable and collapsible.
 - Collapsing a node hides its child structure while keeping the parent visible.
 - Expanding the node restores the hidden branch.
 - This allows users to simplify a large map temporarily and focus on the information that is currently important.
 - The application should provide a clear visual indication when a node contains hidden children.
 - Where practical, the application may also provide:
 - **Expand All** and **Collapse All**
 - for quickly controlling the complete map.
-

5.9 SF-07 — Notes

- A node should be able to contain additional **Notes** that do not need to appear directly inside the visual node.
 - This allows the mind map itself to remain concise while still supporting additional information.
 - When the node is selected, its notes should be available through the Node Editing Panel.
 - The application should provide a simple indication when a node contains notes so that the user knows additional information is available.
-

5.10 SF-08 — Status

- A node should support a basic **Status**.
- For Version 1, the following statuses are sufficient:

Status	Meaning
Not Started	No work has begun

In Progress	Work has started
Waiting	The item is temporarily waiting
Completed	The item has been completed

- Status information should be easy to change.
 - The application may use a small visual indicator, icon, label, or other simple method to communicate the current status without overwhelming the map.
-

5.11 SF-09 — Priority

- A node should also support **Priority**.
 - For Version 1, the following levels are sufficient:
 - **Low — Normal — High — Critical**
 - Priority should be editable from the Node Editing Panel.
 - The application may visually communicate important priorities, but priority indicators should not make the map unnecessarily complicated.
-

5.12 SF-10 — Colors

- Color is an important part of understanding and organizing a mind map.
 - When primary branches are created, the application should automatically assign **different visually distinguishable colors** to them.
 - Child nodes should visually relate to the color of their main branch so that the hierarchy remains easy to understand.
 - The user should also be able to select a node and manually change its color.
 - Where practical, the user should be able to decide whether a color change applies to:**This Node** or **This Branch**
 - The detailed color palette, visual style, and appearance rules are defined in **Chapter 6 — Design & User Experience**.
 - This distinction is important: **The ability to change color is a software feature. The choice of how those colors should look is a design decision.**
-

5.13 SF-11 — Search

- The user should be able to search the mind map using text.
 - Search should identify nodes whose titles match the entered text.
 - Where practical, notes may also be included in the search.
 - Matching nodes should be visually identifiable, and the application should help the user locate the selected result within the map.
 - Search should remain simple and should not require a complex search interface.
-

5.14 SF-12 — Zoom and Navigation

- The user should be able to comfortably navigate maps that are larger than the visible screen.
 - The application should support: **Zoom In — Zoom Out — Fit to Screen — Pan/Move Around Canvas**
 - Mouse-wheel zoom or similar natural browser interaction may be supported where appropriate.
 - A **Fit to Screen** action should automatically position and scale the map so that the visible structure can be viewed conveniently.
 - Navigation should feel natural and should not interfere with selecting or editing nodes.
-

5.15 SF-13 — Undo and Redo

- The application should provide **Undo** and **Redo** for common editing actions.
- This gives the user confidence to experiment with the map without being concerned that every change is permanent.
- Typical actions that should be recoverable where practical include:
 - adding a node;
 - deleting a node;
 - moving a node;
 - editing text;
 - changing important node properties.
- AI may determine an appropriate history mechanism for implementing this capability.

5.16 SF-14 — Map Information

- The application should provide a small amount of useful information about the current map.
 - For Version 1, this may include:
 - **Map Name — Number of Nodes — Number Completed — Save Status**
 - This information should remain secondary to the mind map itself.
 - The purpose is to provide useful feedback, not to create a separate analytics dashboard.
-

5.17 Interaction Between Features

- The features defined in this chapter should work together rather than behave as isolated functions.
 - For example: When a user creates a new branch, it should appear on the canvas, receive an appropriate visual appearance, become searchable, be included in the node count, and become part of the saved map data.
 - When a branch is moved, its children, notes, status, priority, color, and other associated information should move with it.
 - When a branch is deleted and the action is undone, its associated information should also be restored.
 - This is important because the application should behave as **coherent software**, not as a collection of unrelated demonstrations.
-

5.18 Features Intentionally Left for Future Versions

- Version 1 does not need advanced functionality such as real-time collaboration, user accounts, cloud synchronization, online sharing platforms, advanced project management, AI APIs, enterprise permissions, or complex integrations.
 - These capabilities may be considered later.
 - They should **not be added automatically by AI simply because they are technically possible**.
 - The objective remains to create a focused and understandable first application.
-

5.19 Human + AI Feature Understanding

Human Understanding	AI Understanding
These are the functions I expect to use.	These features are requirements to be implemented.
I describe what each feature should achieve.	I determine the appropriate technical implementation.
I do not need to explain the code behind each function.	I should implement the functions and their relationships.
The application should remain simple.	Do not add unnecessary functionality merely because it can be built.
Features should work together naturally.	Maintain consistent state and behavior across related features.
Design is handled separately.	Do not confuse functional requirements with visual design preferences.

5.20 AI Advisory — Software Features

AI ADVISORY

- Treat **SF-01 through SF-14** as the core functional requirements for Version 1.
- Implement the features as working, interconnected software functionality rather than visual demonstrations or inactive interface elements.
- Read the remaining chapters before implementation because design, data persistence, saving, export, and other requirements may affect how these features should be built.
- Where a technical detail is not specified, choose a practical implementation appropriate for a standalone browser-based application.
- Do not add major functionality outside the defined scope merely to make the application appear more advanced. Likewise, do not remove required functionality merely to simplify implementation.

Chapter Principle

- **Every feature should have a purpose. Version 1 needs enough functionality to become useful software, while remaining simple enough to demonstrate how effectively AI can build from a document.**
- At this stage, the AI knows **what the application is, how it is structured, and what it must do.**
- The next chapter is **Chapter 6 — Design & User Experience.**
- That chapter is important because we now tell AI something completely different:
- **“You know what to build. Now this is how we want the software to look, feel, and behave visually.”**

Chapter 6 Design & User Experience

6.1 Purpose of This Chapter

The previous chapters defined **what the application is and what it should do**. This chapter defines **how the application should look, feel, and behave from the user's perspective**.

Design is intentionally separated from software features.

For example:

- **Feature:** The user can change the color of a node.
- **Design:** The application should provide an attractive and easy-to-use color selection experience.
- **AI implementation:** AI determines the CSS, interface controls, and technical method required to make this work.

Because this complete document will later be uploaded to AI to build the application, the design principles in this chapter should be treated as part of the **software build specification**, not merely as visual suggestions. This continues the Human + AI approach already established in the document.

The objective is not to define every pixel. The human establishes the desired experience and important visual principles, while AI is given reasonable freedom to create a professional result.

6.2 Design Objectives

The application should follow six primary design objectives:

ID	Design Objective	Meaning
DS-01	Simple	An ordinary user should understand the application quickly.
DS-02	Visual	The map should communicate structure clearly through position, connections, and color.

DS-03	Attractive	The application should look like modern usable software rather than a technical demonstration.
DS-04	Spacious	The mind map should remain the visual focus and should not be crowded by controls.
DS-05	Consistent	Buttons, panels, nodes, colors, and interactions should follow a coherent visual language.
DS-06	Responsive	The interface should adapt reasonably to different browser window sizes.

6.3 Overall Visual Style

- The application should have a **clean, modern, and professional appearance**.
- It should not look like a plain HTML page containing buttons around a diagram.
- At the same time, excessive visual decoration should be avoided. The mind map itself should remain the most visually important element.

The overall interface should use:

- clear typography;
- comfortable spacing;
- visually understandable controls;
- subtle borders and shadows where useful;
- rounded elements where appropriate;
- a clean background;
- consistent icons and controls;
- clear selected and active states.

AI should use its design capability to create a polished overall interface while respecting the simplicity required for Version 1.

6.4 Visual Hierarchy of the Mind Map

- The mind map should visually communicate its hierarchy without requiring explanation.
- The three most important visual levels are: **Central Topic** → **Main Branch** → **Child Nodes**
- The **Central Topic** should be the most visually prominent node.
- **Main Branches** should be clearly identifiable as the first level beneath the Central Topic.
- **Child Nodes** should remain visually related to their parent branch while becoming progressively simpler as the hierarchy becomes deeper.

The user should be able to look at the map and quickly understand:

- where the map begins;
- which topics are primary branches;
- which nodes belong together;
- which nodes contain children;
- which node is currently selected.

6.5 Branch Color System

- Color should be one of the main visual characteristics of the application.
- Each **main branch** should automatically receive a clearly distinguishable color.
- For example, if a map contains several primary branches, they should not all appear in the same color unless the user chooses that design.
- Child nodes should visually inherit or relate to the color of their main branch.
- This creates visual families within the map.
- Conceptually:

Central Topic

- **White Branch** → related **White** child nodes
- **Green Branch** → related green child nodes
- **Orange Branch** → related orange child nodes
- **Purple Branch** → related purple child nodes

These colors are examples only. They do not prescribe the actual palette.

- AI should choose a balanced default palette with sufficient visual distinction between main branches.
-

6.6 User-Controlled Colors

- Automatic colors provide a useful starting point, but the human should remain in control.
 - The user should be able to select a node and choose another color.
 - Where supported by the feature definition, the user should be able to apply the selected color to: **This Node** or **This Branch**
 - The color selector should provide an easy visual selection of common colors.
 - Where practical, a custom color selector may also be provided so that the user is not limited only to predefined colors.
 - User-selected colors should take precedence over automatically selected colors.
 - The application should remember those selections when the map is saved and reopened.
-

6.7 Node Appearance

- Nodes should be visually attractive while remaining easy to read.
- AI may determine appropriate:
 - node shapes;
 - corner radius;
 - padding;
 - borders;
 - shadows;
 - text sizing;
 - spacing;
 - connection styles.
- However, the visual hierarchy established earlier should remain clear.
- The Central Topic may be larger or more prominent.
- Main Branch nodes may have stronger visual treatment.
- Lower-level nodes may use simpler styling.

- The selected node should have a clearly visible selected state without dramatically changing its normal appearance.
-

6.8 Connections Between Nodes

- Connections between parent and child nodes should clearly communicate the hierarchy.
 - Lines should be visually connected to the appropriate nodes and should move correctly when the map structure changes.
 - Connections should not unnecessarily dominate the map.
 - Where appropriate, AI may use curved or smooth connection lines rather than rigid technical-looking connectors.
 - Connection appearance may relate visually to the color of the associated branch.
 - The exact rendering method is left to AI.
-

6.9 Typography

- Text is the primary information contained within the map, so readability is more important than decorative typography.
 - The application should use a clean and readable font suitable for modern browser interfaces.
 - Typography should help distinguish hierarchy where useful.
 - For example, the Central Topic may use larger or stronger text than lower-level nodes.
 - Text should remain readable against the selected node background.
 - If the user selects a darker node color, the application should use an appropriate text color to maintain readability.
 - AI may automatically determine whether light or dark text provides better contrast.
-

6.10 Toolbar Design

- The Main Toolbar should provide quick access to important actions without making the interface feel complicated.
- Frequently used functions should be easier to reach than less frequently used functions.
- Icons may be used, but unfamiliar icons should have labels or tooltips where appropriate.

- Controls should provide visual feedback when the user interacts with them.
 - The toolbar should remain visually secondary to the Main Canvas.
 - The user should feel that they are working primarily **with the mind map**, not with the toolbar.
-

6.11 Node Editing Panel

- The Node Editing Panel should provide a clean and organized interface for editing the selected node.
- Properties should be grouped logically.
- For example:

CONTENT

Title

Notes

TASK INFORMATION

Status

Priority

APPEARANCE

Color

- The panel should clearly indicate which node is currently being edited.
 - Changes should appear on the mind map as directly as practical.
 - The panel should not permanently consume unnecessary screen space when it is not needed.
 - AI may determine whether the panel should be collapsible, contextual, floating, or otherwise integrated into the interface.
-

6.12 Interaction Feedback

- The application should clearly communicate when the user interacts with something.
- For example, the user should be able to recognize:
 - which node is selected;
 - which button is being used;
 - when a branch is collapsed;

- when search has found a result;
 - when information has been saved;
 - when an action cannot be completed.
 - Feedback should be clear but unobtrusive.
 - The application should avoid unnecessary pop-up messages for normal actions.
-

6.13 Simplicity Before Decoration

- A visually impressive interface is desirable, but appearance should never make the application more difficult to use.
 - When choosing between a decorative design and a simpler interaction, preference should normally be given to the design that is easier to understand.
 - This is particularly important because the application is intended for ordinary users rather than software specialists.
 - The user should not need instructions to understand basic actions such as: **Add → Edit → Move → Delete → Save**
-

6.14 Responsive Behavior

- The application should work reasonably across common desktop and laptop browser sizes.
 - When the browser window becomes smaller, controls should adapt where practical rather than simply disappearing or overlapping the mind map.
 - The Main Canvas should continue to receive the largest practical amount of available space.
 - Version 1 is primarily intended for desktop and laptop use.
 - Full mobile optimization is not required for this demonstration, although the application should avoid unnecessary behavior that prevents future mobile adaptation.
-

6.15 AI Design Freedom

- Unlike functional requirements, visual design should intentionally give AI a reasonable degree of freedom.
- The document does not need to define exact:

- **pixel dimensions, RGB values, CSS classes, margins, border widths, shadows, animation timings, or implementation techniques.**
- Doing so would begin turning the human into a software designer or developer.
- Instead, the human establishes the desired direction: **Clean + Modern + Colorful + Professional + Simple**
- AI determines an appropriate implementation.
- The human can then evaluate the generated result.
- If the human does not like part of the design, it can be changed conversationally after the first version has been generated.

For example:

“Use softer colors.”

“Make the nodes more rounded.”

“Make the Central Topic larger.”

“Use a darker background.”

“Make the interface more corporate.”

“I don't like the toolbar design. Change it.”

- This is an important part of the overall **Build Software from a Document** approach.
- The document does not need to anticipate every possible design preference because **AI remains available after the software has been created.**

6.16 Human + AI Design Understanding

Human Understanding	AI Understanding
I define the visual direction I want.	I translate that direction into an appropriate interface.
I do not need to define every pixel or CSS property.	I may make detailed visual decisions where they are unspecified.
Different main branches should be visually distinguishable.	Create a coherent automatic branch-color system.
I can manually change colors.	User-selected appearance should override automatic defaults.

The map should remain the main focus.	Do not allow toolbars and panels to dominate the canvas.
I want the application to look professional.	Do not deliver an unfinished or purely technical-looking interface.
I can request design changes later.	Keep the implementation suitable for future modification.

6.17 AI Advisory — Design & User Experience

AI ADVISORY

- Treat the design principles in this chapter as part of the software build specification while retaining reasonable freedom over detailed visual implementation.
 - Do not leave unspecified design elements unfinished simply because exact colors, dimensions, fonts, or styling values have not been provided. Use appropriate professional defaults.
 - Prioritize **clarity, usability, visual hierarchy, consistency, and the mind map itself**.
 - Where several valid design solutions exist, choose the solution that provides the simplest and most intuitive experience for a non-technical user.
 - Read the complete document before finalizing the design so that visual decisions support the required functionality, saving behavior, and user interactions.
-

Chapter Principle

- **The Human defines the desired experience. AI transforms that direction into the detailed visual design.**
- This chapter deliberately leaves room for AI creativity.
- That is important to the experiment: if we specify every color, pixel, button position, and visual property ourselves, we are partly designing the software for AI.
- Instead, we want to demonstrate that the human can say **what experience is wanted**, while AI can make many of the detailed decisions necessary to produce it.
- The next chapter, **Chapter 7 — Save, Data & Export**, defines one of the most important parts of making this demonstration feel like real software:
- **How the user's work survives, moves, backs up, and leaves the application.**

Chapter 7 Save, Data & Export

7.1 Purpose of This Chapter

A useful software application must do more than display information while it is open.

The user needs confidence that their work can be **preserved, reopened, backed up, transferred, and exported**.

This chapter defines how the Mind Mapping Application should handle its information and files. The objective is to keep this simple for the user. The human should not need to understand databases, browser storage, file structures, serialization, or other technical concepts.

The human needs to understand only: **I create my map → I save it → I can return to it → I can back it up → I can export it.**

AI determines the technical implementation required to make this possible.

7.2 Save, Data & Export Overview

Version 1 should provide the following capabilities:

ID	Capability	Purpose
SD-01	Automatic Local Save	Preserve the user's current work while using the application
SD-02	New / Reset Map	Allow the user to begin another map
SD-03	JSON Backup	Save the complete editable map data as a backup
SD-04	JSON Import	Restore a previously saved map backup
SD-05	Standalone HTML	Allow the application/map to exist as a portable browser-based file where technically practical

SD-06	PDF Export	Produce a document version suitable for viewing or printing
SD-07	Image Export	Produce an image of the mind map
SD-08	Print	Allow the user to print the map using normal browser capabilities
SD-09	Data Integrity	Preserve the hierarchy and information associated with the map

These capabilities should work together as a simple information-management system.

7.3 SD-01 — Automatic Local Save

- The application should automatically preserve the user's current map while the application is being used.
 - If the user closes the browser and later reopens the application on the same device and browser environment, the application should restore the most recently saved state where technically supported.
 - The user should not need to manually press **Save** after every change.
 - The application should provide a simple indication of save status where useful, such as: **Saved** or **Saving...**
 - The exact technical mechanism is not specified by the human.
 - AI may select an appropriate browser-based persistence method suitable for a standalone application.
-

7.4 SD-02 — New / Reset Map

- The user should be able to begin a new mind map.
- Because this action may replace existing work, the application should protect the user from accidentally losing the current map.
- Before clearing existing information, the application should provide an appropriate confirmation or encourage the user to create a backup where necessary.
- The objective is simple: **Starting a new map should be easy, but losing an existing map accidentally should not be easy.**

7.5 SD-03 — JSON Backup

- The application should allow the user to export the complete editable map information as a **JSON backup file**.
- The JSON file is primarily a data backup rather than a visual document.
- It should contain sufficient information to reconstruct the map, including where applicable:
 - map name;
 - central topic;
 - hierarchy;
 - nodes;
 - parent/child relationships;
 - notes;
 - status;
 - priority;
 - colors;
 - collapsed/expanded states;
 - other required user-defined properties.

The human does not need to understand or manually edit the JSON file.

For the user, it is simply: **Backup My Map**

The technical structure of the JSON may be determined by AI.

7.6 SD-04 — JSON Import

- The application should allow a previously exported JSON backup to be imported.
- After import, the application should reconstruct the saved map and restore its information as accurately as practical.
- The imported map should remain fully editable.

- If the selected file cannot be understood or does not contain valid map data, the application should provide a clear message rather than failing silently or damaging the current map.
 - Where appropriate, the application should protect the existing map before replacing it with imported data.
-

7.7 SD-05 — Standalone HTML

- One of the distinctive objectives of this project is the ability to work with the software as a **standalone HTML application**.
 - The original software generated by AI should therefore be provided as a browser-openable HTML file, for example: **MyMindMap.html**
 - Where technically practical within the selected implementation, the application may also provide a way to create or save a portable HTML representation containing the current map.
 - This capability is particularly valuable because an HTML file can potentially be: **Opened → Copied → Stored → Shared → Opened Again in a Browser**. without requiring the recipient to install specialized mind-mapping software.
 - However, AI should distinguish between: **Application HTML** — the actual software application. And **Map Data** — the information created by the user inside that application.
 - The JSON backup remains the primary portable editable data format unless AI can reliably create a self-contained HTML file containing both the application and the current map.
-

7.8 SD-06 — PDF Export

The user should be able to create a **PDF representation of the mind map** where technically practical in the standalone browser environment.

The PDF should aim to capture the visible map clearly and make it suitable for:

- sharing;
- reviewing;
- archiving;
- presentation;

- printing.

The user should not need to understand how the PDF is technically generated.

If direct PDF generation introduces unnecessary complexity or dependency, AI may use an appropriate browser-based print-to-PDF approach, provided the user can still achieve the intended result easily.

The map should be presented as cleanly as practical without unnecessary interface controls appearing in the output.

7.9 SD-07 — Image Export

- The user should be able to export the mind map as an **image**.
- For Version 1, **PNG** is sufficient.
- The exported image should focus on the mind map itself rather than including unnecessary toolbars, editing panels, or browser interface elements.
- Where practical, the complete map should be included even if parts of it extend beyond the currently visible canvas.
- This allows the user to place the resulting map in presentations, documents, emails, or other material.

7.10 SD-08 — Print

- The application should support printing.
 - Print output should prioritize the mind map and avoid unnecessary application controls.
 - AI may use standard browser printing capabilities where appropriate.
 - The application should provide a simple **Print** action rather than requiring the user to understand the underlying browser implementation.
 - Where possible, the map should automatically fit the printable area in a readable manner.
-

7.11 SD-09 — Data Integrity

- Saving a map means more than saving the text displayed on the screen.
- The relationships and properties that make the map functional must also be preserved.

For example, if a user creates:

Business Plan

→ **Marketing**

→ Social Media

→ Advertising

→ **Finance**

→ Budget

and adds notes, colors, statuses, and priorities, those relationships and properties should remain associated with the correct nodes after saving and reopening the map.

- The application should therefore treat each node as a structured information object rather than only as visual text.
- AI may determine the internal data structure required to accomplish this.

7.12 What Should Be Stored

At minimum, the application data should preserve information required to reconstruct the user's map.

Data	Preserve
Map name	Yes
Central topic	Yes
Node titles	Yes
Parent/child hierarchy	Yes
Notes	Yes
Status	Yes
Priority	Yes
User-selected colors	Yes

Expanded/collapsed state	Yes
Information required for correct reconstruction	Yes

- AI may store additional internal information where necessary for reliable operation.
- The user should not be required to manage that internal information manually.

7.13 Export Is Different from Save

- Both Human and AI should understand an important distinction.
- **Saving** preserves information so that the user can continue editing it.
- **Exporting** creates another representation of the information for viewing, sharing, presenting, or printing.

For example:

Action	Primary Purpose
Automatic Local Save	Continue working later
JSON Backup	Backup and restore editable data
HTML	Portable browser-based application/map where supported
PDF	Share or print a document representation
PNG	Use the mind map as an image
Print	Produce a physical or PDF output through the browser

- An exported PDF or PNG does not need to remain editable as a mind map.
 - The JSON backup does.
-

7.14 Keep File Handling Simple

- The application should use terminology understandable to an ordinary user.
- Where appropriate, interface actions may therefore use labels such as:

New Map

Backup Map

Restore Map

Export as PDF

Export as Image

Print

- The interface does not need to expose technical concepts such as serialization, DOM rendering, Blob objects, local storage keys, data URLs, or JavaScript objects.
 - Those belong to the implementation layer handled by AI.
 - This reflects one of the central principles of this document: **Technical complexity may exist inside the software without becoming complexity for the human using it.**
-

7.15 Human + AI Understanding

Human Understanding	AI Understanding
My work should not disappear when I close the application.	Implement appropriate local persistence where supported.
I want a backup of my editable map.	Provide structured JSON export and import.
I want to share my map visually.	Provide appropriate PDF/PNG output.
I want the software as an HTML application.	Deliver the primary application as browser-openable HTML.
I don't need to understand the internal data structure.	Create a reliable internal representation of the map.
Saving and exporting are different.	Preserve editable data separately from visual export formats.

I should be protected from accidental data loss.	Add reasonable confirmation, validation, and recovery behavior.
---	---

7.16 AI Advisory — Save, Data & Export

AI ADVISORY

Treat **SD-01 through SD-09** as part of the Version 1 build specification.

The primary objective is to provide reliable and understandable data handling without requiring a server, external API, database service, or development environment.

Select technical methods appropriate for a standalone browser-based application.

Preserve the complete logical map structure and its user-defined properties, not merely the visible text.

Where browser security or technical limitations prevent a requested file behavior from operating exactly as described, implement the closest reliable browser-based solution and make the behavior clear to the user rather than creating a non-working control.

Do not create buttons for PDF, PNG, HTML, backup, restore, or other functions unless those controls perform their intended actions.

Prioritize **data integrity and working functionality over the appearance of having more features**.

That final point is especially important when AI eventually builds directly from this document.

Chapter Principle

- **Software becomes useful when the user's work survives beyond the current session.**
- The application should allow the human to create information without needing to understand how that information is technically stored.
- **CREATE → SAVE → CLOSE → REOPEN → CONTINUE**

and when required: **BACKUP → RESTORE** or: **EXPORT → SHARE**

- At this point, the document has defined the application's **purpose, structure, features, design, and data behavior**.
- The next chapter should be **Chapter 8 — AI Build Advisory**.
- That one should be **much shorter** than Chapters 4–7. It is where we stop describing the application and speak directly to the AI that will receive this entire document:
- **“You have now learned what we want. Here is how you should approach building it.”**

Chapter 8 AI Build Advisory

8.1 Purpose of This Chapter

The previous chapters have established the knowledge required to build the Mind Mapping Application.

The AI has been provided with the application's: **Purpose → Structure → Features → Design → User Experience → Data → Save & Export Requirements**

This chapter does not redefine those requirements. Instead, it provides **AI Build Advisory**: guidance for how Artificial Intelligence should interpret the complete document and transform it into working software.

This chapter is written particularly for the AI that will receive this completed document.

The objective is simple: **Understand the complete document first. Then build the software it describes.**

8.2 Read the Complete Document Before Building

The AI should treat this document as a **complete software knowledge and build specification**.

Individual chapters should not be interpreted in isolation.

A feature described in one chapter may depend on design requirements, data requirements, interaction behavior, or other instructions defined elsewhere.

The AI should therefore:

1. Read the complete document.
2. Understand the purpose of the application.
3. Understand the required application structure.
4. Identify all required features.
5. Understand the design and user experience principles.
6. Understand the saving, data, and export requirements.
7. Understand the boundaries of Version 1.

8. Only then begin building the application.

The objective is to build the **complete application described by the document**, rather than independently implementing each chapter.

8.3 What the AI Must Preserve

Some aspects of this document represent requirements rather than suggestions.

The AI should preserve:

Requirement Area	AI Responsibility
Application Purpose	Build the application for the purpose defined in the document
Application Structure	Include the required application areas
Core Features	Implement the defined Version 1 functionality
User Experience	Keep the application understandable for non-technical users
Data	Preserve the user's map structure and information
Save & Export	Implement the required working data and output capabilities
Application Environment	Keep the application browser-based and standalone where specified
V1 Scope	Do not unnecessarily expand the software beyond the intended demonstration

AI should not remove a requirement simply because another implementation would be easier.

8.4 What the AI May Decide

This document intentionally does not define every technical detail.

AI is expected to make many implementation decisions independently.

These may include:

- HTML structure;

- CSS architecture;
- JavaScript architecture;
- functions and internal logic;
- internal data structures;
- rendering methods;
- node-positioning methods;
- browser event handling;
- local persistence implementation;
- undo and redo mechanisms;
- file-generation techniques;
- detailed spacing and dimensions;
- exact colors where they have not been specified;
- appropriate icons;
- technical error handling;
- performance improvements.

The human does not need to make these decisions unless a particular technical choice has explicitly been specified elsewhere in the document.

This freedom is intentional.

The document defines the desired software. AI determines much of the technical implementation.

8.5 When Information Is Not Specified

The absence of a technical detail should not automatically prevent AI from building the application.

If the document establishes the desired outcome but does not explain exactly how to achieve it, AI should make a reasonable implementation decision.

For example:

- If the document says: “The application should remember the current map.”

AI should determine an appropriate browser-based persistence method.

- If the document says: “Main branches should have different colors.”

AI should create an appropriate default color system.

- If the document says: “The interface should look professional.”

AI should make appropriate detailed visual decisions rather than leaving the interface unfinished because exact CSS values were not provided.

The AI should ask the human for clarification only when a missing decision is important enough that making an assumption could significantly change the intended software.

8.6 Build Working Software, Not a Demonstration of Software

This requirement is particularly important.

The final result should be a **functional application**.

It should not merely be:

- a screenshot;
- a visual mockup;
- a static webpage;
- a design concept;
- an explanation of how the software could be built;
- a collection of code fragments;
- an interface containing buttons that do nothing.

If the application contains a control labeled **Export PNG**, that control should export the map as PNG.

If it contains **Undo**, Undo should work.

If it contains **Backup Map**, the user should receive a usable backup.

If a function cannot be implemented reliably within the selected standalone environment, AI should not create a misleading control that only appears functional.

Working functionality is more important than the appearance of having more functionality.

8.7 Keep the Technical Complexity Away from the Human

The generated software may contain significant technical logic internally.

That complexity should not unnecessarily appear in the user experience.

The human should interact with concepts such as: **Add Node — Change Color — Search — Save — Backup — Export**

rather than technical concepts such as: **DOM — JSON Serialization — Event Listeners — Local Storage Keys — Rendering Functions**

The purpose of AI handling the implementation is partly to remove this technical barrier from the human.

The application should therefore translate technical complexity into understandable user actions.

8.8 Do Not Overbuild Version 1

AI should resist the temptation to add functionality simply because it is capable of doing so.

The purpose of Version 1 is to demonstrate **Build Software from a Document**.

The AI should not automatically add:

- user accounts;
- cloud databases;
- collaboration systems;
- AI APIs;
- complex project management;
- enterprise permissions;
- external integrations;
- unnecessary frameworks or services.

Additional functionality can be introduced later through the Human + AI modification process.

The first application should remain: **Simple → Useful → Functional → Portable → Understandable**

8.9 Build for Future AI Modification

Version 1 is not necessarily the final version.

The human may later provide the generated application back to AI and request changes.

The technical implementation should therefore remain reasonably organized and maintainable.

Where practical, AI should:

- organize code logically;
- use understandable internal naming;
- avoid unnecessary duplication;
- keep data structures consistent;
- separate major responsibilities where practical;
- avoid unnecessary complexity.

The human does not need to understand this organization.

Its purpose is to make future AI-assisted modification more reliable.

8.10 Validate Before Delivering the Application

Before presenting the generated application to the human, AI should review the result against this document.

At minimum, AI should verify:

- **Does the application open correctly?**
- **Does the mind map display correctly?**
- **Can nodes be created, edited, moved, and deleted?**
- **Do hierarchy and branches behave correctly?**

- **Do colors and visual hierarchy work?**
- **Do notes, status, and priority work?**
- **Do search and navigation work?**
- **Does Undo/Redo work where required?**
- **Is the user's information preserved correctly?**
- **Do backup and restore work?**
- **Do the required export functions actually work?**
- **Does the interface remain understandable and visually coherent?**
- **Does the application remain within the Version 1 scope?**

The AI should correct identifiable problems before delivering the software where possible.

8.11 Handling Technical Limitations

A standalone browser-based application may have technical or browser-security limitations.

AI should not ignore such limitations or pretend they do not exist.

Where a requirement cannot be implemented exactly as described within the selected environment, AI should: **Preserve the intended user objective → Choose the closest reliable implementation → Keep the experience simple**

The AI should not introduce unnecessary servers, APIs, accounts, or external infrastructure merely to solve a minor limitation unless the document specifically allows it.

Where an important limitation remains, it should be communicated clearly to the human.

8.12 Final AI Build Instruction

AI BUILD INSTRUCTION

You have now been provided with the knowledge and specification required to build the application described in this document.

Read and understand the complete document before beginning implementation.

Treat the application's purpose, structure, features, behavior, design principles, data requirements, saving and export requirements, scope boundaries, and AI Advisory as a unified build specification.

Build the described **Mind Mapping Application as a functional standalone browser-based application.**

Where technical implementation details have not been specified, make appropriate technical decisions consistent with the objectives and principles of this document.

Preserve all required functionality while keeping the application simple, intuitive, visually professional, and suitable for a non-technical user.

Validate the application against the complete document before delivery.

Do not respond only with an explanation, development plan, visual concept, or code example. Produce the working software application described by this document.

8.13 Human Understanding

The human does not need to execute the instructions in this chapter manually.

The purpose of the chapter is to communicate directly with the AI when the completed document is uploaded.

The human's action can therefore remain remarkably simple: **“Read the attached document completely and build the software described in it.”**

The document provides the knowledge.

The AI Build Advisory provides the interpretation guidance.

AI performs the technical implementation.

Chapter Principle

- **Educate first. Build second.**
- At this point, the document has done what it was designed to do:
- **HUMAN IDEA → STRUCTURED DOCUMENT → AI KNOWLEDGE → BUILD INSTRUCTION**
- The next step is the creation of the software.

- But the relationship between the Human and AI does not end when Version 1 is delivered.
- **Chapter 9 — Changing the Software with AI** explains how the same non-technical user can continue developing the application simply by describing what should change.

Chapter 9 Changing the Software with AI

9.1 Purpose of This Chapter

Creating Version 1 is not the end of the **Build Software from a Document** process.

After using the application, the human will naturally discover things they want to change.

A traditional software process may require the user to return to a developer, explain the change, wait for development, test the result, and repeat the process.

In this approach, the human can continue the development conversation directly with AI.

The process becomes: **BUILD → USE → EVALUATE → DESCRIBE CHANGE → AI MODIFIES → USE AGAIN**

The human still does not need to know how the software was programmed.

9.2 The Generated Software Becomes the Next Source

When AI creates Version 1, the resulting HTML application becomes an important source of knowledge for future changes.

To request a modification, the human can provide AI with:

1. **The existing HTML application**
2. **A description of the requested change**

AI should first understand the existing application before modifying it.

This is important because the objective is normally to **change the existing software**, not rebuild a different application from the beginning.

The original software document may also be provided again when needed, particularly when the requested change should be checked against the original requirements.

9.3 Changes Can Be Described in Normal Language

The human should not need to describe changes technically.

For example:

“Make the Work branch dark blue.”

“Add a Progress field to each node.”

“Move the toolbar to the left side.”

“When I double-click a node, create a child node.”

“Make the Central Topic larger.”

“The notes panel takes too much space. Make it collapsible.”

“I don't like the colors. Make them softer.”

These are **software change requests**, even though they are written in ordinary language.

AI translates the requested outcome into the technical modifications required in the existing application.

9.4 Types of Changes

The human may request different kinds of modifications.

Change Type	Example
Feature Change	Add a Progress field
Behavior Change	Double-click creates a child node
Design Change	Make the nodes more rounded
Layout Change	Move the toolbar to the left
Data Change	Store an additional property for every node
Export Change	Improve the appearance of PNG output
Correction	Fix a function that is not behaving correctly
Simplification	Remove an unnecessary control

The human does not need to classify the request before asking AI. The categories simply demonstrate that conversational modification can affect different layers of the software.

9.5 Understand Before Changing

Before modifying the software, AI should understand: **What currently exists → What the human wants changed → What parts of the application are affected**

For a small visual change, only a small part of the application may need modification.

For a new data property, several areas may be affected.

For example, adding a new field called **Progress** could affect: **Node Data → Editing Panel → Display → Saving → JSON Backup → JSON Restore**

The human should not need to identify all of these dependencies.

AI should reason about the consequences of the requested change and update the necessary parts of the software.

This is one of the important benefits of AI-assisted modification.

9.6 Preserve What Is Already Working

A change request should not unnecessarily damage or replace existing functionality.

If the human asks: “Change the color of the toolbar.”

AI should not redesign the entire application.

If the human asks: “Add a Progress field.”

AI should not remove Status or Priority unless that is part of the request.

If the human asks: “Make the nodes larger.”

The map hierarchy, notes, saved information, backup system, and other unrelated functionality should remain intact.

The default principle should be: **Change what is required. Preserve what is not.**

9.7 Preserve Existing User Data

Software changes should also consider information already created by the user.

If the human has an existing map and AI modifies the application, the new version should preserve compatibility with existing data where reasonably possible.

For example, if a new **Progress** field is introduced, existing nodes may simply begin with an empty or default Progress value.

The application should not unnecessarily require the user to recreate the complete mind map because a new feature was added.

Where a requested change requires changing the internal data structure, AI should consider how existing information can remain usable.

9.8 The Human Can Evaluate Visually

The human does not need to inspect the source code to determine whether a modification is successful.

The human can evaluate the software by using it.

For example:

- **Did the new field appear?**
- **Does the new interaction work?**
- **Is the design better?**
- **Did my existing information remain intact?**
- **Did anything that previously worked stop working?**
- This keeps validation at the level where the human has the strongest knowledge:
- **Does the software now behave the way I wanted?**

9.9 Modification Can Continue Iteratively

There is no requirement that every desired improvement be predicted in the original document.

The application can evolve through multiple versions.

For example:

VERSION 1

Initial application generated from the document.

↓

VERSION 2

Human requests several usability improvements.

↓

VERSION 3

Human adds a new field.

↓

VERSION 4

Human changes the design.

↓

VERSION 5

Human adds another capability.

The development process becomes an ongoing conversation between Human and AI.

This also means the original document does not need to become excessively detailed merely to anticipate every future possibility.

9.10 The Document Can Also Evolve

When a change becomes an important permanent part of the software, the human may also update the software document.

For example, if **Progress** becomes an important standard property of every node, it can later be added to the documented feature specification.

The document can therefore remain the human-readable description of what the software has become.

This creates a useful relationship: **DOCUMENT ↔ SOFTWARE**

The document defines the software.

The software is built from the document.

Human experience produces changes.

Important changes can then be reflected back into the document.

This keeps the human-readable knowledge and the working application aligned.

9.11 AI Change Advisory

AI CHANGE ADVISORY

When the human provides an existing application and requests a modification, first read and understand the existing application and the requested change before editing it.

Determine which parts of the application are affected directly or indirectly by the requested modification.

Preserve existing functionality, data, hierarchy, behavior, and unrelated design elements unless the requested change requires otherwise.

Modify only what is necessary while ensuring that related functions continue to work together correctly.

Where the change affects the application's data structure, saving, backup, restore, export, or other dependent functionality, update those areas as necessary.

Preserve compatibility with existing user data where reasonably possible.

After making the modification, verify both the requested change and the important existing functionality that could have been affected.

Return the **complete updated working application**, not only instructions explaining how the human could modify it.

9.12 A Simple Change Request

The human does not need a complicated prompt template.

A change request can be as simple as:

“This is the current version of my application. Please modify it so that each node also has a Progress field from 0% to 100%. Keep the existing functionality and design unless the change requires otherwise. Return the updated working application.”

AI is responsible for understanding the technical implications.

This is important to the overall philosophy of **Build Software from a Document**.

We are not replacing programming with complicated prompt engineering.

We are allowing the human to communicate software requirements in a language they already understand.

9.13 Human + AI Modification Cycle

The complete process now becomes:

DEFINE → DOCUMENT → EDUCATE AI → BUILD → USE → EVALUATE → REQUEST CHANGE → AI MODIFIES → USE AGAIN

This cycle may continue for as long as the software continues to evolve.

Chapter Principle

The human does not need to know how to modify the code. The human needs to know what should change.

The first part of this document demonstrated: **Document → AI → Software**

This chapter completes the idea: **Software → Human Experience → Natural-Language Change → AI → Better Software**

That leaves us with **one final practical chapter: Chapter 10 — Build & Acceptance Checklist.**

It should be short and checklist-driven. Its purpose is simply to answer: **“Before we call Version 1 complete, did AI actually build what the document asked for?”**

Chapter 10 Build & Acceptance Checklist

10.1 Purpose of This Chapter

The application has now been defined through its purpose, structure, features, design, data requirements, build instructions, and future modification process.

Before Version 1 is considered complete, both **Human and AI need a simple way to verify that the software described in this document has actually been built.**

This chapter provides that final check. It is not another software specification.

It is the point where we ask: **Does the working application match the document?**

10.2 AI Build Verification

Before delivering the application, AI should review the completed software against the following checklist.

ID	Acceptance Check	Required
AC-01	Application opens and runs correctly in a modern browser	☐
AC-02	Application operates as a standalone browser-based application as defined	☐
AC-03	Central Topic is displayed and editable	☐
AC-04	Main branches and child nodes can be created	☐
AC-05	Nodes can be edited	☐
AC-06	Nodes and branches can be deleted safely	☐
AC-07	Nodes can be moved and reorganized	☐

AC-08	Branches can be expanded and collapsed	☐
AC-09	Notes can be added and edited	☐
AC-10	Status can be assigned and changed	☐
AC-11	Priority can be assigned and changed	☐
AC-12	Main branches receive distinguishable colors	☐
AC-13	User can change node/branch colors	☐
AC-14	Search works correctly	☐
AC-15	Zoom, pan, and Fit to Screen work	☐
AC-16	Undo and Redo work for required actions	☐
AC-17	Map information is displayed where required	☐
AC-18	Current work is preserved locally where supported	☐
AC-19	JSON backup works	☐
AC-20	JSON restore/import works	☐
AC-21	PDF output works as defined	☐
AC-22	PNG image export works	☐

AC-23	Print functionality works	☐
AC-24	Saved/restored data preserves hierarchy and node information	☐
AC-25	Interface is clean, understandable, and professional	☐
AC-26	No required control is merely decorative or non-functional	☐
AC-27	No unnecessary major functionality has been added outside V1 scope	☐

10.3 Human Acceptance Check

The Human does not need to inspect the code.

The Human should simply **use the application**.

The Human should create a small real mind map and test the normal workflow: **Create → Add → Edit → Move → Organize → Color → Add Information → Search → Save → Close → Reopen → Backup → Restore → Export**

The important questions are:

- **Can I understand how to use it?**
- **Can I create the mind map I expected?**
- **Does it behave naturally?**
- **Does my information remain intact?**
- **Can I return to my work later?**
- **Can I export and back up my map?**
- **Does the application feel like usable software rather than a demonstration?**

If the answer is yes, Version 1 has achieved its purpose.

10.4 When Something Does Not Work

Failure of one acceptance item does not require rewriting the entire software specification.

The Human can provide the application back to AI and describe the problem.

For example: “Everything is working, but when I restore the JSON backup, the branch colors are not restored correctly. Please fix this without changing the other functionality.”

AI should then follow the **AI Change Advisory** defined in Chapter 9.

The cycle continues: **TEST → IDENTIFY → DESCRIBE → AI FIXES → TEST AGAIN**

10.5 Version 1 Acceptance

Version 1 should be considered accepted when:

The core requirements defined in this document are working, the application is usable by the intended non-technical user, important information is preserved correctly, and no critical function prevents normal use.

Acceptance does not mean the software can never improve.

It means: **The document has successfully become working software.**

10.6 Final AI Instruction Before Delivery

FINAL VALIDATION

Before delivering the application, compare the completed software against the full document and the acceptance checklist in this chapter.

Correct identifiable failures where possible.

Do not mark a requirement as completed merely because its interface exists. Verify that the underlying function works.

Where a requirement cannot be completed because of a genuine technical limitation, clearly identify that limitation rather than representing the function as working.

When validation is complete, provide the Human with the **working application file**.

10.7 The Complete Process

We can now see the complete methodology: **HUMAN IDEA STRUCTURED DOCUMENT → AI READS & UNDERSTANDS → AI BUILDS → WORKING SOFTWARE → HUMAN USES & EVALUATES → HUMAN REQUESTS HANGES → AI MODIFIES → SOFTWARE EVOLVES**

The document is no longer simply documentation **about software**. It becomes part of the process **for creating the software**.

Chapter Principle

The final test of the document is not whether it explains the software well. The final test is whether AI can turn it into software that works.

Conclusion

- **From Document to Software**
- Software creation has traditionally required the human idea to pass through a technical layer before it could become a working application.
- The person with the idea explained what was needed.
- A software developer translated that idea into technical architecture and code.
- The computer executed the result.
- AI introduces another possibility.
- A human can now describe software through a structured document written in normal language. That document can establish the purpose, structure, functionality, behavior, design, data requirements, and boundaries of the application.
- AI can read that knowledge and perform much of the technical translation.
- The relationship becomes: **HUMAN KNOWLEDGE + STRUCTURED DOCUMENT + AI CAPABILITY = WORKING SOFTWARE**
- This does not mean that programming, software engineering, or professional developers are no longer important.
- It means the **entry point into software creation is changing**.

A person who understands a problem but does not understand programming can participate much more directly in creating a solution.

And the process does not stop after Version 1.

The Human can use the software, learn from it, describe what should change, and allow AI to modify it again.

DEFINE → DOCUMENT → BUILD → USE → LEARN → CHANGE → BUILD AGAIN

That is the central idea behind **Build Software from a Document**.

The document becomes more than something people read.

The document becomes something AI can build from.

And now the experiment is ready:

Complete the document.

Give it to AI.

Ask AI to read it.

Build the software.

Then see if the document works.