

Design and discovery

This chapter covers

- Using ChatGPT in the design phase
- Breaking down a problem with ChatGPT
- Crafting prompts
- Using ChatGPT to generate a design document
- Generating user stories

Design is key in software development, yet programmers often rush it. I've done this, too. Taking time to plan an app's architecture leads to happy users and lower maintenance costs. How can generative AI speed up the design process?

Our goal here is to design a web application that solves a common problem: studying for a certification exam. The app will provide randomized practice tests for exam preparation. First, we'll analyze the problem and create a design document with ChatGPT's help. Then, we'll use ChatGPT to improve the document and develop user stories to guide development.

By collaborating with ChatGPT in the design phase, we can quickly outline architecture for our app. We'll also ask ChatGPT-targeted questions to uncover ideas and

details we may not have considered on our own. The result is a solid foundation to build and iterate upon as we progress through the project.

Let's learn how AI can boost creativity and productivity, even early in the software life cycle. We'll set ourselves up for easier development by using these generative tools.

3.1 *Getting to know ChatGPT*

ChatGPT is an advanced large language model (LLM) from OpenAI. It aims to be useful, safe, and truthful. It has been trained on a wide range of texts, such as internet content and books. This training gives ChatGPT a broad knowledge base. However, there are some limitations to this AI system:

- ChatGPT cannot independently authenticate or validate information. Its responses are generated stochastically. It's a "best guess" by the model and can contain inaccuracies. You should always verify the output.
- While ChatGPT's responses can seem convincing, they may lack depth. The model doesn't have a genuine comprehension of the topics it discusses.
- ChatGPT has no personal identity, opinions, or memory. This limits its ability to engage in prolonged conversations or exhibit a consistent personality.
- ChatGPT doesn't understand real-world context. For this reason, always review its output carefully, especially for sensitive or critical tasks.
- ChatGPT's skill set is more restricted than the broad intelligence of humans. It is particularly adept at tasks such as text generation, but it doesn't match the overall versatility of human cognition. Don't be fooled! It's not a super-intelligent human.

Even with these limitations, ChatGPT is an invaluable asset for developers. It enables you to generate high-quality text and code. It also fosters creativity and aids in idea development.

The key to using ChatGPT effectively is crafting thoughtful prompts and evaluating responses. We'll explore some techniques to maximize the benefits of using ChatGPT for software development, focusing on design and documentation.

3.2 *The problem*

As software developers, we love building cool stuff. We overhear someone say, "I'd love to have an app for that," and our ears perk up. Instantly, we want to crack open the IDE and get to work. When starting a new project, one of my favorite questions is glaringly simple but often overlooked: What problem are we trying to solve?

It's a question you should ask first and continue to ask throughout the process. There should be only one answer at the beginning. Let's look at a problem and then build a web application to solve it.

Imagine you want to get a HAM radio license. With this license, you can communicate with other radio operators around the world. To get the license, you must pass a test that assesses your knowledge of radio theory, electronics, and rules and regulations related to radio communication. The test consists of 35 multiple-choice questions. So,

rather than finding out how well you know the material by taking the test for the first time, you want to try some practice tests to see how well you do.

The application should use a pool of questions, similar to those on the test, and randomly select 35 to display on the screen. This way, you can practice and see how well you are doing and what areas you need to study.

This application needs to be implemented as a web-based interface for seamless use on any computer. You don't want to download and install software on each of your devices. We will write it in Python because we're Python developers and have built many applications with it. It's familiar. This approach will allow us to focus on the problem rather than learning a new language or platform.

We now have enough information to open the IDE and get started. But let's be smart about this. What if you want to expand the application to all three levels of HAM radio testing? What if you want to make it public? There are things to shake out in the design process before we start.

Our problem so far is simple. We want to

- Study for the test by taking practice tests
- Have questions similar to those on the test
- Have those questions randomly delivered in 35-question sessions

We need to

- Build a web application
- Have a set of test questions with answers
- Store them somewhere
- Have an interface to view them
- Select 35 of them for a test randomly

This is an excellent start. Let's see if we can use ChatGPT to break down this problem. It will help us reveal any blind spots or alternative solutions worth pursuing. We can't run blindly into ChatGPT and hope for the best results. We need to craft our prompt to get exactly what we need. Here are some prompts to help you understand the problem domain:

- "What are common edge cases when building {type of application}?"
- "What regulatory considerations exist for {domain}?"
- "What nonfunctional requirements are typically important for {application type}?"

3.3 Creating the right prompt

Let's talk about prompt engineering. It's a complex topic, but it becomes intuitive after a while. There are some important things to think about when creating prompts. We'll start with a *role prompt*. A role prompt instructs the model to act as someone performing a role. The resulting text is (hopefully) what someone in that role might say. A prompt includes the following components:

- *Introduction*—Give ChatGPT a role to fill: “Act as a software design expert.”
- *Task*—Give it something to do: “Help me break down a problem.”
- *Contextual information*—“I want it to be a web app written in Python.”
- *Instructions*—“Include details about structure and outline some potential pitfalls.”
- *Closing*—“Craft the output in the form of a software design document.”

Your prompt isn’t required to have all these components, but if you aren’t getting the desired results, including these things is helpful.

Using this example, our prompt will look as follows:

JM Act as a software design expert. Help me break down a problem. I want to create practice tests to study for a HAM radio license. I will take a pool of questions from the test. I want to draw 35 questions randomly from that pool. I want to show the possible answers and select an answer. At the end of the test, I would like to grade it with a percentage. Include details about the structure and outline some potential pitfalls. Craft the output in the form of a software design document

This type of prompt—often referred to as a *persona prompt* or a *role prompt*—helps shape the response style, and it should be effective in getting us started.

In addition to the role prompt, other basic prompt types include

- *Chain of thought prompting*—Ask the AI to “think step by step” when breaking down complex design problems.
- *Few-shot prompting*—Show examples of what good output looks like before asking for your own.
- *Persona prompting*—Beyond just “Act as a software design expert,” try “Act as a software architect with 15 years of experience in scalable systems” for more specific expertise.

Now, let’s learn more about these techniques and how to refine them for our workflow.

3.4 *Measuring the effect on the design process*

To understand the real benefits of using ChatGPT in the design process, let’s look at how much time you can save. Your results may vary, but overall, it’s almost always a net gain. As table 3.1 shows, we save significant amounts of time generating documents, while they still retain our original intent.

Table 3.1 Time saved by using ChatGPT

Documentation task	Traditional method	ChatGPT-assisted	Time savings
System overview	90–120 minutes	15 minutes	~83%
Technical stack	60 minutes	10 minutes	83%
User stories	120 minutes	25 minutes	79%
Complete design document	6–8 hours	1–2 hours	~75%

The biggest gains come from not starting with a blank page. While traditional methods require drafting all content from scratch, ChatGPT generates a comprehensive first draft in seconds. This draft then needs refinement. The shift from creation to curation dramatically accelerates the documentation process.

In my experience, a team of developers can spend up to 70% less time in design meetings, with no reduction in documentation quality. We've never just generated a document and run with it. The documents always require editing and revisions; however, that overhead is still present in any documentation project, and the brainstorming and initial assembly costs are now gone.

ChatGPT and other LLMs are great for “removing the boring stuff” so you can work on more creative endeavors. This principle is a recurring theme throughout this book. The biggest reason to adopt this approach is to save valuable time for more interesting work.

3.5 A design document created with ChatGPT

The software development life cycle (SDLC) is a framework that defines the stages software goes through, from initial planning to deployment and maintenance. The SDLC brings structure to an otherwise chaotic process. You are likely already familiar with this framework, so we only cover it briefly here.

Understanding the SDLC can help us align our workflow to industry best practices. It can also show the types of documents we need to create for each step of the way. I don't want to get buried in documentation here, but a simple guide will help us stay on track and know what to expect.

We will use ChatGPT to create that guide with a simple design document. The term “design document” is intentionally abstract. This won't be a complex set of documents you would create for a large enterprise project. I aim to draft a simple outline document to structure my ideas for this application. We'll create it with the SDLC in mind. Remember, this model varies depending on the organization or person creating it. This is my best understanding of the model.

The SDLC process usually follows these steps:

- 1 Planning
- 2 Analysis
- 3 Design
- 4 Implementation
- 5 Testing
- 6 Deployment
- 7 Maintenance

Each stage builds on the last, with some overlap and iteration as needed. Knowing what is expected to happen at each phase sets expectations and helps surface any gaps early on (figure 3.1).

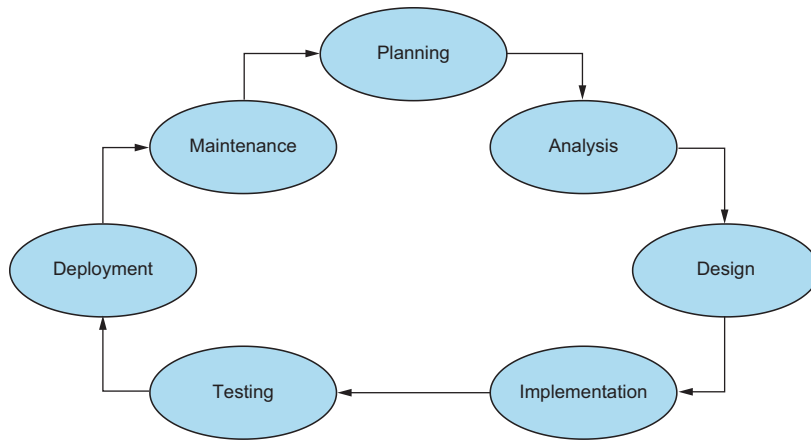


Figure 3.1 An example of the software development life cycle

As we work with ChatGPT to design our app, we'll try to address the key elements presented here and plan for them. Our goal is to produce documentation that captures requirements and gives us guidance.

We start this process with a prompt and then work within a single context to refine and build on the document. As we've discussed, the ChatGPT *context* is like a real-life conversation. The thread of conversation you have with ChatGPT lives in the side panel. Figure 3.2 shows the section where your conversations are stored, as indicated by the arrow.

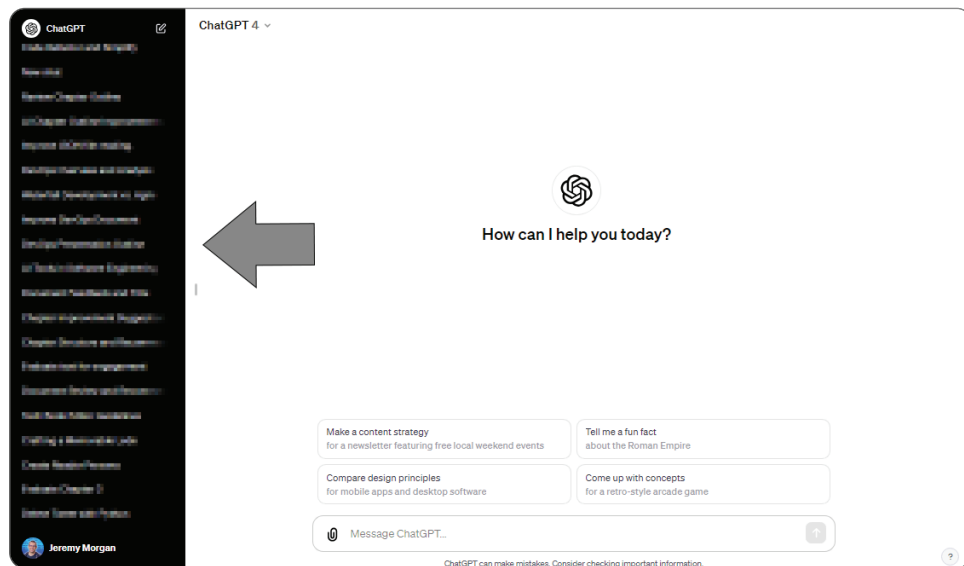


Figure 3.2 The arrow shows where to find conversations in ChatGPT.

This is important to remember because information you've provided is the context, and ChatGPT uses it for future generations. It includes the current conversation history, specific instructions given, and some assumptions based on the dialogue.

To start out our conversation, we'll use the prompt we've just created:

JM

Act as a software design expert. Help me break down a problem. I want to create practice tests to study for the HAM radio license. I will take a pool of questions from the test. I want to draw 35 questions randomly from that pool. I want to show the possible answers and select an answer. At the end of the test, I would like to grade it with a percentage. Include details about structure and outline some potential pitfalls. Craft the output in the form of a software design document.

Note that I asked for a software design document to keep the output abstract. Had I wanted to be more specific, I would have asked for process documentation, budget, or planning documents. But I just want a single, simple document to start out. I want to keep the goal in mind when crafting a prompt.

Red flags in AI-generated design documents

There are things to watch out for with AI-generated design documents, such as

- Overly generic architectural patterns without justification
- Technology recommendations without considering your specific constraints
- Missing error handling or security considerations
- Ignoring cross-platform compatibility when it's critical for your app

I sent the prompt we crafted to ChatGPT 4.0, and the next section contains the output. It's the direct result of my query in ChatGPT. Note that at the time of this writing, ChatGPT 4.0 is only available with a premium account. The free account only allows ChatGPT 3.5; however, if you're following along, ChatGPT 3.5 will work for this, too. Just remember that the output will be different from what you see here, regardless of the version you use. Next, we'll see what ChatGPT produced.

3.6 Software design document: HAM radio license practice test application

Let's jump in and see what ChatGPT produced. We will examine each section of the document in turn, beginning with the introduction.



1. Introduction

The purpose of this document is to provide a detailed and structured plan for the creation of a HAM radio license practice test application. The primary function of this application will be to generate randomized 35-question tests from a predetermined question pool, present possible answers for each question, and finally provide a score percentage at the end of each test.

The introduction dives right into the purpose of the document. It's mostly rewording the input we gave it and writing it in a formal way. It immediately describes the problem and the application's function. It could use a little more detail about what the program should do, but we'll address that.



2. System Overview

The system can be broken down into several main components:

Question Pool Database: This is where all potential questions and corresponding answers will be stored.

Question Selector: This component will handle the random selection of questions from the question pool.

User Interface: This is where the user will interact with the system to take the practice tests and view their results.

Grading Component: This will grade the test upon completion and provide a score.

The system overview gives us a general idea of the parts of the application. ChatGPT breaks it down into four main components. These abstract components help us think about the problem a layer deeper. While this may seem obvious to even beginner developers, these suggestions help create a model in your mind about how this system should work.

The next section digs into the details. This is the next layer to think about. One thing to notice is the inclusion of pitfalls, which we asked for in our prompt.



3. Detailed System Design

3.1 Question Pool Database

The question pool database will store all the possible questions, along with their multiple-choice options and correct answers. This could be implemented using a SQL database or a NoSQL database like MongoDB, depending upon the structure and volume of data.

Structure

A basic structure would have a table/collection with the following fields:

`question_id`: A unique identifier for each question.

`question_text`: The question text.

`options`: The possible answers for each question. This could be an array of strings.

`correct_option`: The correct answer for the question.

Pitfalls

- If the questions, options or answers change over time, there needs to be a system for updating the question pool database to ensure accuracy.
- Considerations for data validation and cleansing are also needed to ensure that incorrect or poorly formed data do not enter the system.

3.2 Question Selector

The question selector module will use a randomizing algorithm to select 35 questions from the question pool database for each test.

Pitfalls

- Care should be taken to ensure true randomness in question selection.
- The same question should not appear more than once in a single test.

Even a quick glimpse of our description tells you the database for this application is simple. The document outlines a data structure and gives us some suggestions for a database server. We have a single table that contains some basic fields. In the pitfalls, we have a couple of considerations for scaling and maintaining the system.

**3.3 User Interface**

The User Interface (UI) will present the questions and answer choices to the user, accept their answers, and present their score at the end of the test. The UI could be implemented as a web application, a desktop application, or a mobile app, depending on requirements.

Pitfalls

- The UI needs to be intuitive and user-friendly. A poor UI could frustrate users and hinder the effectiveness of their study sessions.
- The design should also consider accessibility needs to cater to a wider audience.

This section is sparse. We need a UI, and it suggests a simple workflow. It also tells us we can implement it as a web, desktop, or mobile application, which is fairly obvious but good to note.

**3.4 Grading Component**

The grading component will compare the user's answers with the correct answers from the question pool database to calculate a final score, which will be expressed as a percentage.

Pitfalls

- This component must handle cases where the user doesn't answer some questions. Those should be marked as incorrect for scoring purposes.
- Potential inconsistencies or inaccuracies in grading due to any discrepancies in the database need to be handled.

This section is also sparse, and there isn't much to discuss. We need something that will compare the user's answers to the correct ones and calculate a score. The pitfalls are helpful. ChatGPT mentions things to think about, such as the user not answering questions. What should we do about that? Furthermore, it also includes a reminder about potential database discrepancies.

**4. Future Considerations**

The initial system design as outlined above is minimal and function specific. However, there are several additional features that could be added over time to increase its utility:

- Tracking user performance over time and providing performance analytics

- Allowing users to focus on specific areas or question types
- Integrating with a larger learning platform

By breaking down the application into these components and understanding potential challenges, we can ensure a robust and efficient design that provides a user-friendly tool to help users prepare for their HAM radio license test.

ChatGPT suggests a few features that might be useful. I'm always leery of these types of things in software documentation. Thinking too far ahead leads to stubbing out things we'll never build or falling into the YAGNI (you aren't gonna need it) trap. However, there's no harm in looking at the suggestions.

Tracking user performance over time is a great idea. This helps us as well as the students. It mentions allowing users to focus on specific areas or question types, which is also handy. These are good things to think about, so I'll leave this section in the document.



5. Technical stack

Based on the functionality and design considerations, the following technical stack could be utilized:

Backend: A Node.js server running Express.js for handling API requests and server-side operations.

It recommends a Node.js backend for our project. We've already decided we're using Python for this because that's where our expertise lies. We will change this by altering our context with additional prompts soon. Here's what it recommends. Though we won't use it, I'm happy with the fact that ChatGPT suggested a tech solution with great potential like Node.

The technical stack is where most developers get excited. We don't want to exclude this section, although we will make some changes to it.



Database: MongoDB or a SQL database like PostgreSQL for storing and managing the question pool.

Again, we have some great choices. MongoDB and PostgreSQL are solid products. In my opinion, we don't need a database that complex. This is a very small dataset that doesn't require complex operations or data safety, so I'll use SQLite instead. However, MongoDB and PostgreSQL are excellent alternatives.



Frontend: A React.js application for the user interface would allow for a dynamic, user-friendly experience.

ChatGPT suggests React.js for the frontend—a solid choice and one likely influenced by the number of people using it. However, we're going to use Flask. Using Flask on the frontend allows for easy integration with the Python backend. We don't have the overhead of introducing and setting up a React system for a small, simple UI. Instead, we can stick with Python.



Hosting/Deployment: The application could be hosted on AWS or any other cloud platform, providing flexibility and scalability.

I'm relieved to see ChatGPT isn't suggesting building out some server. It suggests AWS or another cloud platform, which is the best choice for an application like this. It doesn't provide great detail, but we can work with it.



6. Development Plan

6.1 Requirements Gathering

Conduct a detailed study to gather all possible features, requirements, and constraints for the application. Define user roles and permissions if needed.

6.2 Database Design and Setup

Set up the database as per the design mentioned in section 3.1. Import the initial set of questions into the database.

6.3 Backend Development

Implement the server-side functionality, such as the random question selector and grading system. Set up appropriate API endpoints for fetching questions, submitting answers, and receiving grades.

6.4 Frontend Development

Develop the user interface following the best practices for UX/UI. Ensure it is easy to navigate, aesthetically pleasing, and functional across different device types.

6.5 Testing

Carry out rigorous testing to identify and fix any bugs or issues. Both unit testing and end-to-end testing should be included.

6.6 Deployment

After testing, deploy the application to the chosen cloud platform.

In this section, ChatGPT lays out what you need to do once you've started the process. It goes through a familiar SDLC process.



7. Possible Challenges

7.1 Data Security

Ensure that appropriate security measures are in place to protect user data and to prevent unauthorized access to the question pool.

7.2 Scalability

The system should be designed to handle an increasing number of users and questions. This could involve planning for database sharding and considering horizontal scaling of the server setup.

7.3 Accessibility

Ensure that the software complies with accessibility standards and best practices, so that it can be used by as wide an audience as possible.

8. Conclusion

By following the above detailed design and development plan, it would be feasible to develop an efficient and user-friendly HAM radio license practice test application. This application would help users prepare for their exams and potentially enhance their chance of success.

However, the design and development of software applications require careful attention to detail, thorough testing, and continuous refinement based on user feedback. These factors, along with a solid foundational design as provided in this document, will guide the successful development and deployment of the application.

Overall, I'm satisfied with the document that ChatGPT produced so far. It's thorough and provides some real-world suggestions that pass the sniff test. Let's see if we can get some additional details from ChatGPT.

To conclude the document, we have some possible challenges, which include data security, scalability, and accessibility. There's nothing new or groundbreaking here. But it's good to see it's included for us.

3.7 Digging deeper

The document we produced is a good start to our project's journey. We could work from this document and improvise or modify the plan as we go. This is how most of my software projects have gone in the past. We work on a plan, agree on it, and then spend the rest of the time building and modifying it. We can prevent this excessive churn in this process by deciding on some key details upfront. Let's examine some sections of our document in more depth and try to bring more clarity.

3.7.1 System overview (section 2)

Let's look at our system overview section. For now, we have the following:



2. System Overview

The system can be broken down into several main components:

Question Pool Database: This is where all potential questions and corresponding answers will be stored.

Question Selector: This component will handle the random selection of questions from the question pool.

User Interface: This is where the user will interact with the system to take the practice tests and view their results.

Grading Component: This will grade the test upon completion and provide a score.

This is a reasonable overview of the system. It's not too long or detailed. It outlines the main system components. But we'd like just a bit more detail here. If we present this document to executives, it will work. If we send it to engineers, they'll want far more detail. Let's see if ChatGPT can help with this. I'll add the following prompt to the same chat instance as before.



Give more details in “2. System Overview.” Elaborate on the key subjects.

With this prompt, I’m keeping it simple. I asked for more details in the specified section. Then, I mentioned elaborating on key subjects. This is to direct the prompt to the exact detail I want. I run this script, and this is what ChatGPT came up with.



2. System Overview (Detailed)

The system for the HAM radio license practice test application can be divided into several key components, each playing a crucial role in achieving the desired functionality. These components interact with each other to form a cohesive system. Here are the elaborations on each one:

Question Pool Database: This database serves as the primary store of information from which the system operates. It houses all potential questions that can be asked on the HAM radio license test, their corresponding answers, and the multiple-choice options that will be presented to users. The database will be set up to facilitate efficient retrieval of question sets, and secure updating of question-answer pairs as required. The system’s efficiency and effectiveness will heavily rely on the organization and structuring of this database.

Question Selector: Once the question pool is established, the question selector component becomes the key element in generating each practice test. This module will use algorithms to select 35 questions at random from the question pool for each test a user takes. The randomness is an essential factor in mimicking the unpredictability of the actual test, thus better preparing the user. The complexity of these algorithms can range from simple random sampling to more complex methods to avoid question repetition and to ensure an even coverage of topics.

User Interface: This is the point of interaction between the users and the system. The user interface needs to be well-designed, not just aesthetically, but also functionally, to facilitate ease of use. It will present the randomly selected questions and corresponding answer choices to the user, collect their responses, and finally present their score upon test completion. It also needs to handle and display system or user-generated errors appropriately.

Grading Component: The grading component is integral to providing feedback to users. After a user completes a test, this module will take the user’s responses, compare them against the correct answers from the question pool database, and calculate the user’s score. This score will then be converted into a percentage for easy understanding and standardization. This immediate feedback mechanism is crucial for effective learning and preparation.

These components work together to provide the full functionality of the system. The database acts as the primary information source, the question selector uses this information to generate unique tests, the user interface presents these tests to users and collects their responses, and the grading component uses these responses to provide immediate feedback to the user. The careful design and interaction of these components will determine the overall success and utility of the HAM radio license practice test application.

This section is now far more detailed, with important information added. Rather than a single sentence describing each component, we now have a paragraph. There's more detail and more to consider when thinking about our application. The value here isn't in the ready-to-go text it generates. Sure, we have a nice full document, but it could be loaded with inaccuracies. We must review and double-check all the information. Don't rely on implied accuracy!


The primary advantage of using ChatGPT in this case is outlining concepts and exploring ideas you may not have thought about. It helps you start the documentation process and assists you in writing documentation.

3.7.2 *Technical stack (section 5)*

Let's dive into another section and extract more details. Again, we're doing this to get a general outline and suggestions. Because LLMs are trained on vast sets of similar texts, the structure they come up with will likely resemble formats you've encountered or created yourself. It can also bring up topics and ideas you haven't thought of yet.

I want to get additional details from the technical stack section. I also want to influence the model a bit. I don't want to use React, Node.js, or Mongo. We're Python developers! One of the most popular web frameworks for Python is Flask, and it will allow us to use Python to build a web application.

We can steer the model for different results by adding to the existing context. We can do that with a single prompt.

 This web-based application will be a typical Flask application.

Now, the context of our chat will involve Flask and the technologies around it. This helps the model know our intentions clearly.

To get the additional details, I decided to use a very simple and concise prompt. Here's the prompt I used:

 Give more details for section "5. Technical stack."

Notice that this time, I gave a very terse prompt. No need to add "Elaborate on the key subjects" because we're still in the same chat context. ChatGPT will know I'm looking for the same thing as before, but with a different section. Let's see what the model created for us.

5. Technical Stack (Detailed)

This section describes the selection of tools and technologies (referred to as the "tech stack") for our HAM Radio License Practice Test Application.

Backend: A server-side web framework is necessary to handle HTTP requests and serve web pages. In our case, we'll be using Flask. Flask is a lightweight Python web framework that is easy to use and provides a lot of flexibility for developers. It doesn't include many of the tools that heavier frameworks like Django come with, but this means that developers have more freedom to choose the tools they want to use.

Database: Given the structure of our data (questions, multiple-choice options, and correct answers), both SQL and NoSQL databases could be used. PostgreSQL or MySQL could be appropriate choices if we choose SQL. Both are robust, reliable, and widely used, offering strong performance, powerful features, and good support for various data types and index strategies. If we go with NoSQL, MongoDB could be a good choice due to its flexibility, scalability, and performance. MongoDB is a document-oriented database that stores data in flexible, JSON-like documents.

Frontend: For the user interface, we will use React.js, a JavaScript library for building user interfaces. React allows for the creation of reusable UI components, which can greatly speed up development and result in code that's easier to maintain. React is maintained by Facebook and is widely used, so it has a large community and a lot of resources for developers.

Hosting/Deployment: Deployment and hosting could be done on Amazon Web Services (AWS). AWS offers a robust, secure, and scalable environment for deploying web applications. With AWS, you can leverage various services like EC2 for running your application, RDS for database management, and S3 for storage. AWS also offers excellent scalability, which is beneficial for the future growth of the application.

Version Control: We will use Git for version control, which allows us to track changes in our code over time and collaborate effectively. GitHub can be used to host our Git repositories.

Testing: Python's unittest module can be used for back-end testing, while Jest and React Testing Library can test React components on the front end.

Containerization: We could use Docker for creating a container for our application. This helps to eliminate issues related to differences in local development environments.

Continuous Integration/Continuous Deployment (CI/CD): We could use tools like Jenkins or GitHub Actions for setting up a CI/CD pipeline. This would enable us to automatically build, test, and deploy our application whenever we push changes to our codebase.

This proposed tech stack is designed to provide a balance between performance, development speed, and ease of use. However, the exact tools and technologies used may vary depending on specific project requirements, developer familiarity, and other factors.

This result generates some additional details that seem accurate. The document lays out our application very nicely. I will modify many of these details for the final document and trim some things. ChatGPT has saved me a considerable amount of time here. While I won't just copy and print this text, I have an outline and guidance to help me. ChatGPT provides tremendous value for outlining sample ideas.

Why is ChatGPT suggesting React.js?

ChatGPT suggests React.js instead of Flask for the frontend, even though we've specified using Flask for the frontend. There's a good reason for this, and it's not a mistake. Developers frequently use React.js as a frontend in addition to Flask and the Flask API. It's likely that the model has seen this combination so many times it assumes we

(continued)

will do the same. There's nothing wrong with this pairing. I would certainly consider it if we were to build a larger, more complex application. However, the simplicity of this application will allow us to use Flask for the frontend as well. This will save us time since we won't have to configure React.js and set it up. We'll just use HTML templates.

So far, ChatGPT has acted as a software design expert and created a basic document outlining the architecture for our Python web application. The document includes important sections such as

- System overview
- Technical considerations
- Development roadmap

This is a good rough draft for our initial design. However, this documentation doesn't cover something very important—the user's point of view.

We can expand the design by having ChatGPT take on another role—that of the product owner. The product owner is the person who represents the user of the application and ensures their needs are met. By asking ChatGPT to act as one, we can generate user stories to capture requirements for the application from the user's point of view. Let's see what ChatGPT comes up with.

Using AI to help your tech stack

Here are some additional techniques to help you evaluate tech choices:

- Pros/cons analysis of different frameworks for your specific use case
- Specifying questions to determine if the proposed technology is appropriate
- Prompts for getting migration paths if technologies need to be changed later


3.8 *Generating user stories for our project*

Organizing tasks in an optimal way is an art. This isn't a book about project management, so we won't explore all the possibilities here. We will, however, see what ChatGPT can do to help us organize our project. We'll start by having it generate some user stories for our project.

I want to go down a level deeper into the project. The projects I've been working on for many years have had a product owner involved. Product owners manage the project and control the backlog of tasks. They try to assign tasks in an orderly manner to fit the flow of the project. It's a tough job, and ChatGPT can help with it.

ChatGPT will change from a *software design expert* to an *experienced product owner*. We'll still be in the same context (chat) we used to generate the previous documentation. ChatGPT is aware of the project, so we won't have to add additional background information to our prompt.

The next prompt I send is

 Act as an experienced product owner. Create user stories for this software project.

This is the prompt I used with GPT-4, and we'll look at the output. I intentionally kept it short and sweet, so the request was clear.

3.8.1 Defining roles in prompts

ChatGPT generates content based on what it thinks someone in that role will say. By providing a role such as “software designer” or “product owner,” we set the context for the information we want to generate.

Roles give ChatGPT a persona, guiding the output's style and substance. As a software designer, it will provide technical details. As a product owner, it will focus on user needs. Roles such as marketer or CEO would yield different perspectives.

Clearly defining a role provides a lens through which ChatGPT views the problem. This role-playing mirrors real life. The CEO of the company will see software problems differently from a developer. They will also have entirely different conversations, and using roles helps ChatGPT emulate that. You're creating a persona to discuss the problem with. You wouldn't ask a person in finance which database provider to consider. We can interact with ChatGPT just like a real person in that role.

Specifying roles in prompts is a great way to steer the output of ChatGPT. It saves guesswork and optimizes your time. Otherwise, the model may present less information that you don't care about. Also, this approach allows you to problem-solve from different angles and can uncover things you haven't thought about. Let's see what this prompt from a new role gives us.

3.8.2 The output: User stories document

The following is the output from ChatGPT, unedited, that builds user stories based on our previous prompt and the context of the overall conversation. ChatGPT knows as much about our project as we do, based on our conversation so far. Here's what it generated for my user stories:



User Stories for HAM Radio License Practice Test Application

1. User Registration and Authentication

- 1.1 As a new user, I want to be able to register an account so that I can have a personalized test-taking experience.
- 1.2 As a registered user, I want to log in to access my personal test dashboard and track my progress.
- 1.3 As a registered user, I want to reset my password in case I forget it.

2. Accessing the Test

- 2.1 As a user, I want to start a new practice test so that I can test my knowledge on HAM radio topics.

2.2 As a user, I want to see a clear indication of how many questions are in each test so that I can manage my time effectively.

3. Taking the Test

3.1 As a user, I want to be presented with one question at a time to focus on answering each question without distraction.

3.2 As a user, I want to see multiple-choice options for each question so that I can select the best answer.

3.3 As a user, I want the ability to skip questions and return to them later before submitting the test.

3.4 As a user, I want a timer that shows me how much time I have left for the test to manage my pace.

4. Submitting and Grading

4.1 As a user, once I've answered all questions or the time is up, I want to submit my test for grading.

4.2 As a user, after submission, I want to see my test score as a percentage to understand how well I performed.

4.3 As a user, I'd like to see which questions I got wrong, and what the correct answers are so that I can learn from my mistakes.

5. Tracking Progress

5.1 As a registered user, I want to see a history of my past test scores so that I can track my progress over time.

5.2 As a registered user, I want insights on areas/topics where I need improvement based on my test results.

6. User Interface and Experience

6.1 As a user, I want an intuitive and friendly interface so that I can navigate the application easily.

6.2 As a user, I want to see helpful feedback messages or alerts if I forget to answer a question or if there's an error in the system.

7. Accessibility and Usability

7.1 As a user, I want the application to be accessible, catering to users with disabilities, ensuring everyone can use it without hindrance.

7.2 As a user with a slow internet connection, I want the application to load efficiently without much waiting time.

8. Additional Resources

8.1 As a user, I'd like to have access to additional study materials or resources related to HAM radio to aid my preparation.

8.2 As a user, I'd like a frequently asked questions (FAQ) section in case I have common queries about the test or the platform.

9. Feedback and Support

9.1 As a user, I want the ability to provide feedback about the questions, user interface, or any bugs so that the system can be improved.

9.2 As a user, I want a support or help section where I can get assistance if I face any challenges.

These user stories outline the primary interactions and expectations users might have with the system. As development progresses, new stories might emerge, and existing ones may need to be refined based on user feedback and changing project requirements.

3.8.3 Analyzing the output

Now that we have ChatGPT's initial draft of our user stories, we will scrutinize the content. As I've said before, you must perform due diligence and review all the output from ChatGPT. Don't blindly copy and paste code, and make sure you understand the output. Generative AI provides great value in rapidly producing frameworks and outlines. However, they must be reviewed.

Before diving into the details, we'll take a high-level view. Do the user stories follow a logical progression that mirrors the user's journey? Does it seem accurate? Does the language resonate, or does it need to be tweaked a bit? This critical step allows us to shape the output into documentation that fits our needs. With good reviews, we use ChatGPT's speed without compromising accuracy.

The next section reviews the stories individually to extract the useful parts and determine what needs adjustment. The goal is to retain ChatGPT's framework, while fine-tuning the content to our specifications. With thoughtful examination and editing, we can craft documentation that clearly captures user and technical requirements.

Additional prompts that help with document generation include

- "Create a technical specification template for a REST API."
- "Generate a database schema design document outline."
- "Create an architecture decision record template."

3.8.4 User stories document in detail

This output looks great at a quick glance. I can see areas where I'll likely make changes to our application, but it's a great start. Let's look at each section and see whether it meets our needs.



User Registration and Authentication

This section walks us through the first steps in the process. We can generate tasks from these high-level items: registering an account, a personal dashboard, and resetting the password. It also includes why we'd like to do this. So, the user can have a personalized test-taking experience, track their progress, and reset their password.

Accessing the Test

This section sheds little light on content and should be rolled into another section. It outlines a story for starting a new test and another step to see how many questions are in each test. These are simple requirements but still need to be added to our backlog.

Taking the Test

Section three digs deeper into the process. It shows a workflow of the user being presented one question at a time, with multiple options for each. It also outlines a feature for skipping questions and returning to them later and a timer. These are things I hadn't yet considered for my application. I appreciate the idea generation ChatGPT has provided us.

Submitting and Grading

This section is crucial to letting the student know what their score was. It outlines the process of submitting the test for a grade, converting it to a percentage, and showing what answers were wrong. All great features we can think about adding to our application.

Tracking Progress

This section digs deeper into the process and suggests adding more features.

Namely a history of past scores and areas for improvement. These are a bit complex for an MVP, but we may implement them later.

User Interface and Experience

In section six, we have interface concerns addressed, however they're abstract. It outlines creating an intuitive interface with helpful feedback. These are implied functions regardless, so we should give more detail to this section for our project.

Accessibility and Usability

This section is very important. If we want everyone possible to use this application, we need to accommodate as many people as possible. As a personal note on rewording: I don't like the phrase "cater to users with disabilities." It has a negative connotation so that I wouldn't add this phrase to a user document. I will find another way to say I want my software to be inclusive of others. This is an example of using the tools to outline your ideas while adding your own voice. You should absolutely reword text when needed.

Additional Resources

This section outlines the addition of additional resources. It also outlines a new feature, the FAQ section. Again, this is an idea I hadn't thought much about, but thanks to GPT-4, it's here. This is certainly something I will consider in the plan.

Feedback and Support

Section nine is the final section and adds some features to our application. We're adding the ability for users to provide feedback about the questions or the application itself. And it suggests creating a help section for the application.

This follows our software's "user journey" from start to finish. It's a decent start to our collection of user stories. Remember, generative AI should assist you in your task rather than do the task for you. This applies to documentation just as much as code. The documentation we've generated so far is decent. We will use this as a guide to building our application. In some ways, we'll follow it closely. Most software design documents aren't set in stone, and this document is no exception. We'll follow it closely in some areas and stray from it in others, depending on the problem. In the next chapter, we'll dig into coding the first version of our application.

Summary

- Precise prompts boost AI effectiveness. Well-crafted prompts state the task, context, and specific instructions clearly. Vague prompts lead to guesswork and poor results. Detailed prompts yield more accurate and useful code.
- AI collaboration improves through refinement. Treat AI interaction as a conversation. This means refining prompts based on your initial results. This dialogue helps you guide the AI toward better outputs.
- Evaluate multiple responses for quality. Creating and comparing various AI responses helps find the best solution. This method ensures we don't accept poor initial results. It also boosts code quality.
- Technical verification is key for AI content. While AI tools offer valuable help, all generated code and suggestions need human validation. You must use your expertise to fill knowledge gaps and ensure accuracy.
- Strategic AI integration balances help with independence. Using AI tools well means using them for brainstorming, drafting, and everyday tasks. However, it's important to retain critical thinking and technical judgment. The goal is to enhance human abilities, not replace crucial developer decisions.