Darren Broemmer
TBG Publ;ishing

PROMPT GENIUS

Generate Python Web Applications Using AI

Prompt Genius: Generate Python Web Applications using AI

Darren Broemmer

This book is for sale at
http://leanpub.com/ai-prompt-templates-to-build-apps

This version was published on 2023-06-12

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

Contents

1 AI templates to build your entire application

The world of software development has completely changed. Artificial Intelligence (AI) makes it easy to create entire applications, chatbots, and websites. OpenAI's ChatGPT, Google Bard, Microsoft Bing, and GitHub Copilot can all write and debug code in almost any programming language.

This book contains **step by step prompt templates** you can use to generate your application code using AI services. The approach detailed here follows a proven system design and development methodology. The difference with this approach vs. traditional software development is that the coding portion is accelerated from months to days using AI.

You design your application largely by defining the requirements. The design is comprised of natural language descriptions of the data your application manages and what users can do with it in your system. Software development methodologies have already been doing this. The difference with an AI-assissted approach is that you need to be precise in terms of the details.

Traditional software development leaves room to elaborate on the requirements through each iteration. While this is still true, when it comes time to prompt the AI to generate code, you need to give it all of the relevant information about how your application should work.

The term prompt engineering is commonly used to reference the activity of constructing prompts that create the desired output. This is a fancy term for a simple concept, but an important one. Simply put, you need to be able to define and capture your thoughts in clear, English text that can be used in a prompt.

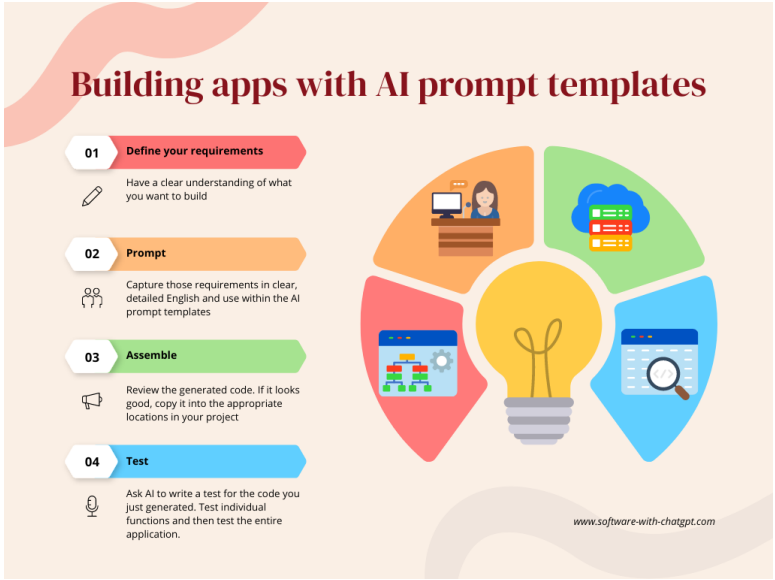
> A problem well-stated is half solved.
>
> —Charles Kettering

Charles Kettering was the head of research at General Motors from 1920 to 1947. This quote has never been more true. Fortunately, AI enables us to deal with the other half of the problem faster than ever before.

You fill in your specific application details in the AI prompt templates provided in this book. These prompts must specify your requirements, so be as detailed as possible. Any detail that you do not provide will be decided by the AI. In some cases, you may be happy with what it generates. In other cases, you may want something different than what it chose. More often tha not, you will have specific constraints and ideas about how your application should work. Be sure to think through these details and include them in the prompt templates.

A series of prompts is needed because you won't be able to generate your entire application in one prompt, unless it is very tiny. AI is amazing, but does not have the ability to manage that much context at a time. Coding assistants are incredibly good at writing isolated or small-scale software functions. However, you need a full-scale software application, website, or service.

You will use an iterative process to create your application as shown in the diagram below.



1.1 The state of the world in AI assisted coding

As of the time of writing (summer 2023), AI can write about 90% of your code completely and accurately. This is not clickbait, I will show you the data. It is quite incredible what it can do.

Even for problems it can't entirely solve, it can shorten the coding time from hours to minutes. Incomplete solutions that it generates provide a great start. They can even be an inspiration for ideas about how to finish or rewrite the solution.

This is not hands-free driving though. AI generated code can still contain subtle bugs. For example, it can generate code to insert data into a database that fails to consider referential integrity checks. Sometimes it fails to properly create or parse JSON data.

Fortunately, there are techniques you can use to quickly remediate these issues. For example, we have found that having AI generate and parse XML data is far more reliable than plain text or JSON formats.

Your role as a software engineer changes with AI. You are a designer. You are a code reviewer. You are an assembler. The good news is, the whole process goes much faster. You can impress your manager or build that startup idea you've been waiting to bring to life. The world of software development has changed, for the better.

1.2 How software engineering roles change when using AI

The table below shows how traditional software engineering tasks change in an AI-powered world. The emphases moves from writing code to specifying requirements, validating output, and assembling the final result. It is critical to prompt AI to generate tests at each step of the process and run those tests.

Traditional Engineering Tasks	**AI-assisted Engineering Tasks**
Obtain specifications	Write specifications
Design use cases	Design use cases
Write code	Prompt to generate code
Perform code reviews	Review AI output
	Integrate code into project
Write test cases	Prompt to generate test cases
Run tests	Run tests

This book walks you through the entire process, step by step. The secret to effectively using AI is to break up the process into smaller steps. This allows the AI to create your software a few components or functions at a time, an amount of code that fits within its context limits.

Following a step-by-step process is beneficial for both you and your AI assistant. It helps you to think through the details of each use case. You start with a high-level business idea. Then you model the data for the application. Then you define the use cases that each actor in the system can perform. Each step along the way requires detailed information in the prompts so that AI can generate the code that meets your requirements. Breaking down the development into a step by step process forces you to think through how you want each step to work.

A little bit of design goes a long way in terms of code generation. So once you have the vision in mind for your app, you are ready to get started.

1.3 Your code may vary

One last thing to note is code you generate may be different than the examples shown here. That is perfectly okay. In software, there are many different ways to accomplish the same thing.

The important thing is that whatever the implementation is, it produces the desired result. The same test cases should pass on an implementation generated by ChatGPT, Bard, or any other AI assistant.

With that introduction out of the way, let's get your environment setup.

2 Setup your environment

Python is the primary programming language used in this book. However, the techniques are applicable to any common programming language.

The primary items you need are:

- An OpenAI ChatGPT account and API key. This is free to obtain from OpenAI.
- A coding environment with Python, an IDE (we use PyCharm), and the CodeGPT plugin installed.
- Optionally, you can create an account on the repl.it[1] website. There is a template created for this book that already has all of the required libraries installed and ready to go.

For beginners, we recommend using the replit.com site that has templates for push-button environment setup. You can use this online programming environment to run examples from the book. The Gradio template[2] has support for Python and the early examples in the book with web interfaces. The Django[3] template allows you quickly build web apps, including examples from this book.

Eventually, you will want to have your own workstation or environment configured with Python and the OpenAI libraries. We use the PyCharm IDE in the examples in this book. However, you can also use Visual Studio Code (VSCode) or any other IDE. We recommend you use an IDE that supports AI plugins, such

[1] https://replit.com/
[2] https://replit.com/@HuggingFace/Gradio
[3] https://replit.com/@replit/django

as CodeGPT. These plugins make working with AI services much easier during the development process.

Even if you are not familiar with the libraries or frameworks used in the examples, that is okay. In fact, that is the whole point. You can use ChatGPT to give you all the setup instructions or explain what certain components do.

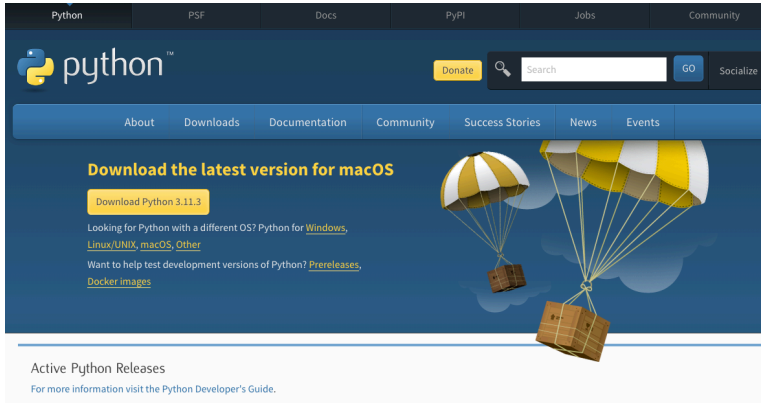
You can also go back and forth between the online replit.com environment and your local workstation. replit.com allows you to download the entire project as a zip file, which you can then easily unzip and open up in your IDE.

You can create code with ChatGPT chatbot on any operating system such as Windows, macOS, Linux, or ChromeOS. You do not need a powerful computer with a high-performance CPU or GPU because the AI processing is handled by OpenAI's API on the cloud. Your computer's hardware is not a limiting factor.

2.1 Install Python

If you do not already have Python installed, browse to downloads[4] to obtain the setup file for your platform.

[4] https://www.python.org/downloads/



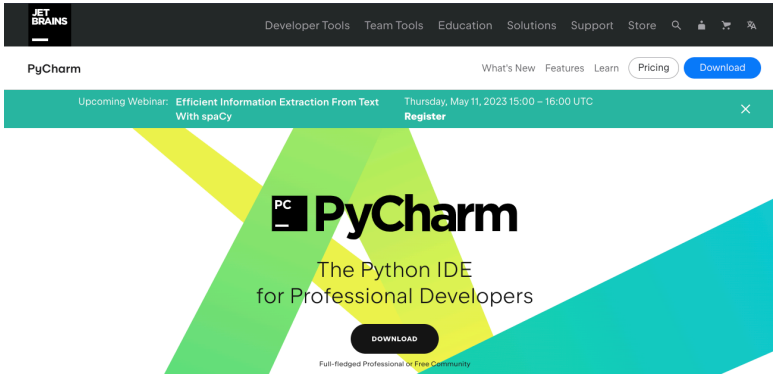
Run the setup file and be sure to enable any checkbox for adding Python to your path. Click "Install Now" and follow the steps to install Python. You can verify that Python is installed using the following command. Depending on your system, you may need to specify python3. We use a virtual environment that is set up by PyCharm (more info on that in a moment).

```
(venv) broemmerd$ python --version
Python 3.9.14
(venv) broemmerd$ python3 --version
Python 3.9.14
```

2.2 Install an IDE (i.e. PyCharm)

We use the PyCharm IDE, although you can use Visual Studio Code or another IDE. If you don't already have an IDE, browse to PyCharm[5] and download the file for your platform. There is a paid version, but the Community (free) edition works very well.

[5] https://www.jetbrains.com/pycharm/



Run the installation file and create a new project in the desired directory. PyCharm will automatically create a virtual environment for your project. After creating your project, go to the terminal to install the required libraries. The icon to open the terminal is in the lower-left corner of the IDE.

Pip, the Python package manager, was also installed when you installed Python. However, it is a good practice to update it first. Run the following command in the terminal window.

```
python -m pip install -U pip
```

Using the python -m prefix ensures you are running the install within your virtual environment.

Now install the OpenAI client library using the following command.

```
python -m pip install openai
```

Also, install the following libraries.

```
python -m pip install python-dotenv
python -m pip install gradio
```

The `python-dotenv` library will be used so that you can safely store your OpenAI API key, without having to insert it directly into your code. The code will read your API key from an environment file.

The gradio library is an exceptionally useful library that quickly makes web interfaces based on Python functions. This allows you to visually interact with your applications quickly. You can easily create an MVP using this library.

2.3 Create your OpenAI account

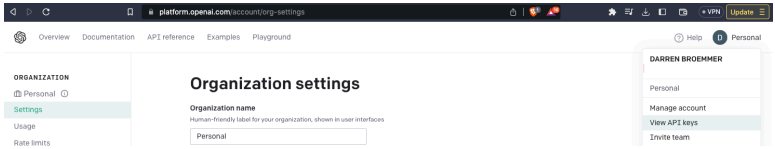
To write software assisted by ChatGPT, you'll need to obtain an API key from OpenAI. This key allows you to use the ChatGPT model within your own interface and display the results in real-time. OpenAI currently offers free API keys, which includes $5 of free credit for the first three months.

After the free credit has been used up, you'll need to pay for continued API access. However, at present, the API access is available to all free users.

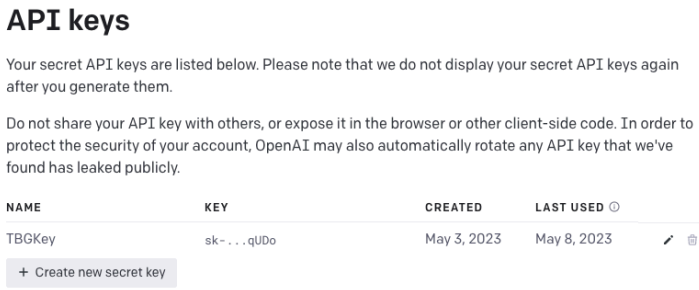
To get started, browse to OpenAI[6] to signup and create a free account. If you already have an account with OpenAI, simply log in to your existing account.

On the OpenAI webpage, click on your profile in the top-right corner and select "View API keys" from the drop-down menu.

[6] https://platform.openai.com/



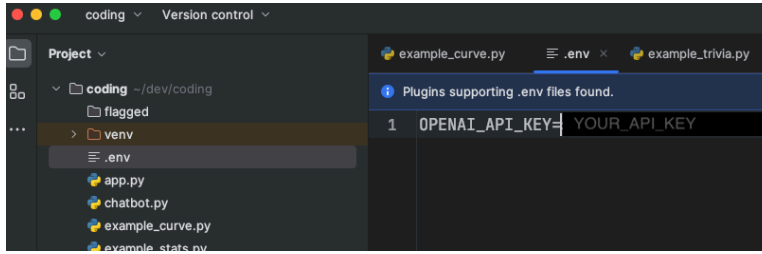
Click on "Create new secret key" and copy the API key. It is important to note that you can't view the full API key later. Save it in a safe place immediately. If something happens, you can always come back and create a new API key or remove old ones.



Your key, the character string, will be used in your code that calls ChatGPT, as well as in the configuration of the CodeGPT plugin. As noted earlier, avoid putting your key in the actual code.

To configure your key in an environment settings file, right-click on your project in PyCharm and select the menu item New | File. Name the file `.env`

In this file, define an environment variable that contains your key. This will be referenced in the code at runtime.



Let's test the configuration by making a call to ChatGPT. Create a file named call_chatgpt.py in your project. Add the following code.

call_chatgpt Python function

```
import openai
import os

# Set the OpenAI key
from dotenv import load_dotenv
load_dotenv()
openai.api_key = os.getenv("OPENAI_API_KEY")

# Define the function to call the API
# The temperature is the amount of randomness and creativ\
ity,
# a value between 0 and 2. Higher values make the output \
more random,
# while lower values are more focused and deterministic.
def generate_code(prompt, temperature = 0.5, max_tokens =\
 256):
    response = openai.Completion.create(
        engine="text-davinci-003",
        prompt=prompt,
        max_tokens=max_tokens,
        temperature=temperature
    )

    # Extract the response text
```

```
    return response.choices[0].text.strip()

# Lets write some code
prompt = (
   "Write a python function that takes a String "
   "as input and returns a counts of the number "
   "of vowels in the string"
)

code = generate_code(prompt)
print(code)
```

This is the basic code to make a call to ChatGPT. Let's examine what is happening here.

First, we import the `openai` library, the ChatGPT client. We also import the `os` library, and the `load_dotenv` function from the `python-dotenv` library. This lets us use the environment file we created with the API key. The `load_dotenv` function is invoked, and that makes our OPENAI_API_KEY accessible to the code through the `os.getenv` function.

The `generate_code` method uses the OpenAI Completion AI. It takes the prompt as a parameter, and optionally, you can specify the temperature and max tokens. As noted in the comments, the temperature tells ChatGPT how creative or consistent to be with its answers. A chatbot may require a higher degree of creativity, whereas a coding assistant is well served using a bit more consistency.

The max tokens is a limit on how many tokens can be used. From the API documentation, "The token count of your prompt plus max_tokens cannot exceed the model's context length. Most models have a context length of 2048 tokens (except for the newest models, which support 4096)."

The context refers to the limit discussed in the first chapter on how

much information ChatGPT can process at a time. Tokens also relate to your usage of the API, which at first uses the free tokens given to you when you create an account. Following that, there is a nominal cost for tokens. See the OpenAI pricing page[7] for more information.

When we run this code asking ChatGPT for a function that counts vowels, we get the following output.

ChatGPT generated function code

```
def countVowels(string):
    vowels = 'aeiou'
    count = 0
    for char in string:
        if char in vowels:
            count += 1
    return count

print(countVowels('Hello World')) #3
```

This function doesn't look for uppercase vowels, but it does work for lowercase letters. As you can see already, coding with ChatGPT can be incredibly productive, but it will also require an iterative process. We can improve upon this implementation. We will learn all about iterative development with ChatGPT starting in the next chapter.

2.4 Install the CodeGPT plugin

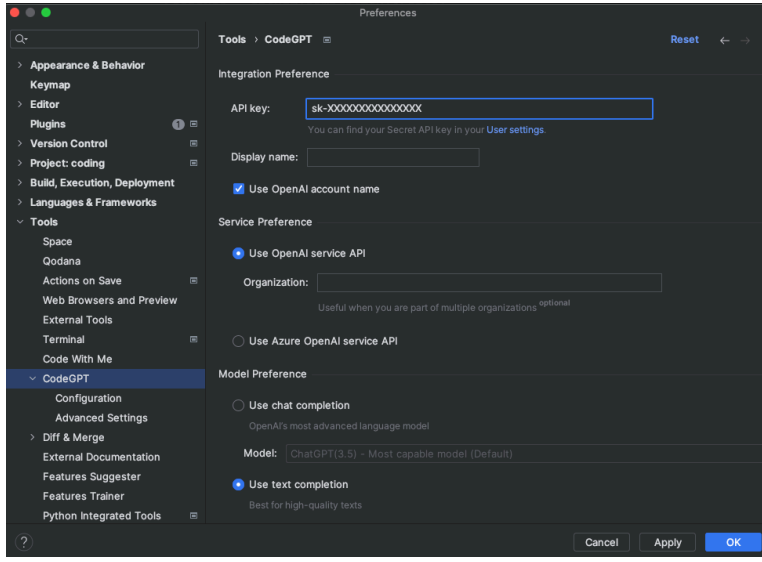
We just demonstrated how you can call ChatGPT from Python code. It will be more convenient during development, however, if you can invoke ChatGPT directly from your IDE. The CodeGPT

[7] https://openai.com/pricing

plugin is available for both PyCharm and Visual Studio Code. Let's install this plugin and see how it is used.

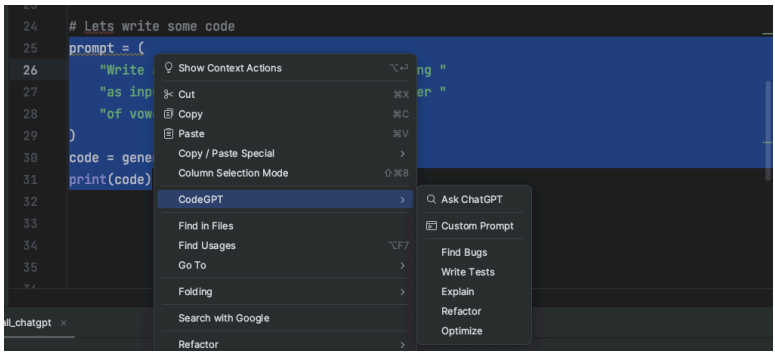
Go to the PyCharm File | Settings, or Preferences on Mac OS. Select Plugins from the list on the left and make sure you are on the Marketplace tab. Search for CodeGPT and once you find it, click the Install button. PyCharm will be restarted for the plugin to take effect.

Now you need to configure the CodeGPT plugin, minimally with your API key so it can make calls to ChatGPT. In the same settings interface, search for CodeGPT. Enter your API key in the appropriate field and click OK. You shouldn't need to change any other configuration options at this time.



Now you can right click in the editor, and see the CodeGPT | Ask ChatGPT option. This allows you to make general queries to ChatGPT, as if you were on the web interface. However, you can also select portions of code and right-click for context-sensitive calls to ChatGPT. We will leverage these features throughout the

book.

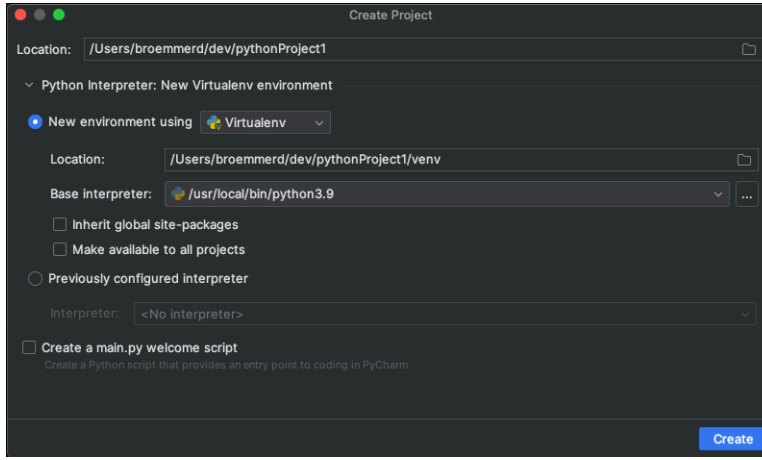


2.5 Idea, set, let's go

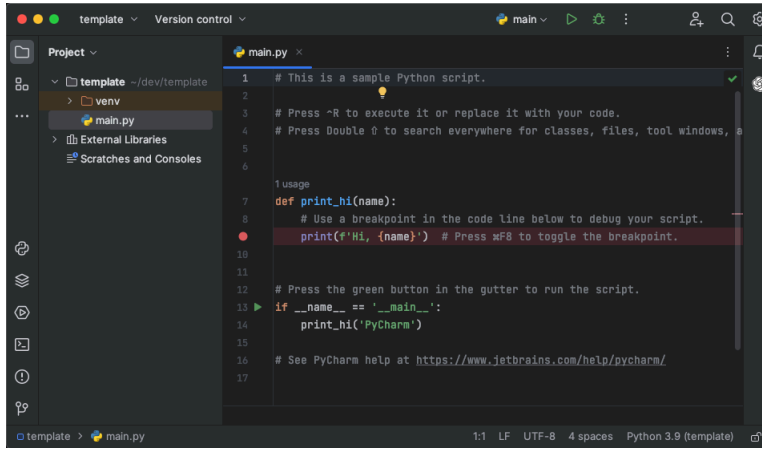
Now that you have everything you need, you can get started.

First, create a directory for your application. Name it based on your application idea. I've called mine "template" but you should name yours according to your application concept.

In PyCharm, select the menu option File | New Project. You will see a popup window similar to the one below.



Change the location folder at the top to the directory you just created. Select the option to create a main.py script at the bottom. Your project will look something like this.



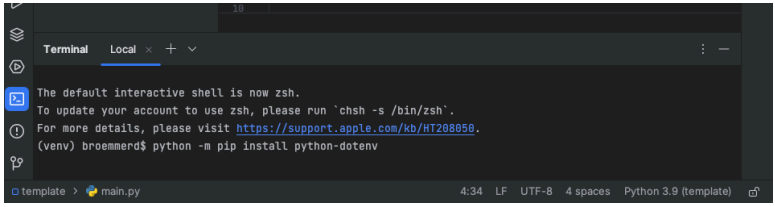
If you are not familiar with PyCharm, take a minute to get used to it. By default, the green arrow icon near the upper right corner will run the main welcome script that was generated. Clicking that will open a shell window at the bottom of the screen and you will see the Hi, PyCharm printed to the terminal.

You have two other basic options to run applications.

1. Open a terminal window and run a command.
2. Right-click a Python file in the left hand side directory tree, and click the Run option.

You will need to use the terminal window to install a few Python packages before we get into building our application. Pip, the Python package manager, was installed when you installed Python. Open a terminal as shown in the image below and run the command:

```
python -m pip install python-dotenv
```



Also install the package for your AI coding service.

```
python -m pip install openai
python -m pip install bardapi
python -m pip install EdgeGPT
```

We will use the web framework Django to build our application. It is a common web framework for Python developers found at https://www.djangoproject.com/ From its website, Django is described as "a high-level Python web framework that encourages rapid development and clean, pragmatic design."

You may not know anything about Django, and that is okay. Your AI coding assistant will help you, and so will we.

By default, Django is configured to use SQLite as its database. This is great for development, but by the time you get to production, you can swap it out for a database such as Postgres.

Our environment is prepared. We are ready for our first prompt.

Note that in this book, generic prompt templates are provided for each step in the process. In order to make these prompts more tangible, we will build an online quiz application and show those specific prompts based on the templates. These are only examples. You can customize the templates to build whatever application you want.

Prompt templates are shown using the following format.

Prompt template
: Write Python code for a **name type** class

Example prompt
: Write Python code for a **Quiz model** class.

Test prompt
: Write a Python unittest for the Quiz model class.

Terms in brackets are named according to what the placeholder should contain. You will substitute in terms specific to your requirements.

3 Prompt One: Start your web project (with Django)

All of the code from this book can be found on GitHub[1].

Use the following prompt to create the basic project structure.

Prompt Template
: Create a small example to say hello world using the **web_framework** framework

Example Prompt
: Create a small example to say hello world using the Django framework

You will get the complete instructions to setup the environment and run a Hello World example. For example, ChatGPT provided the following instructions.

Install Django: If you haven't already, you can install Django by running the following command in your terminal.

```
python -m pip install Django
```

Create a new Django project: Open your terminal or command prompt, navigate to the desired directory, and run the following command. This will create a new Django project named "helloworld" in a directory with the same name.

[1] https://github.com/dbroemme/prompt-genius

```
django-admin startproject helloworld
```

Create a Django app: Change into the project directory by running cd helloworld, and create a new Django app using the following command.

```
python manage.py startapp helloworldapp
```

Define a view: Open the file `helloworldapp/views.py` and replace its contents with the following code:

```
from django.http import HttpResponse

def hello_world(request):
    return HttpResponse("Hello, World!")
```

Create a URL mapping: Open the file `helloworld/urls.py` and replace its contents with the following code:

```
from django.urls import path
from helloworldapp.views import hello_world

urlpatterns = [
    path('', hello_world, name='hello_world'),
]
```

Start the development server: Go back to the project's root directory (helloworld) and run the following command in your terminal.

```
python manage.py runserver
```

Test the application: Open your web browser and visit http://127.0.0.1:8000/. You should see the "Hello, World!" message displayed.

At first, the difference between projects and apps in Django may seem confusing. The Django documentation says that:

- An app is a web application that does something – e.g., a blog system, a database of public records or a small poll app.
- A project is a collection of configuration and apps for a particular website. It can contain multiple apps. Right now, your HelloWorld project has one app called HelloWorldApp.

The database configuration is defined in the `helloworld/settings.py` file. It uses SQLite by default. After you run the first time, you will see a `db.sqlite3` file created in the project root directory.

You can login to this SQLite database using the command `sqlite3 db.sqlite3`. It is empty to begin, as no tables have been created yet. Django uses the concept of database migrations to create tables. There are some built in applications that come with Django, some which use database tables. You likely saw a warning when you started the server that these migrations can be run using the following command. Go ahead and run that command now.

```
(venv) broemmerd$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessio\
ns
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  ...
  Applying sessions.0001_initial... OK
```

Now if you login, you see the tables that are created.

```
(venv) broemmerd$ sqlite3 db.sqlite3
SQLite version 3.28.0 2019-04-15 14:49:49
Enter ".help" for usage hints.
sqlite> .tables
auth_group                  auth_user_user_permissions
auth_group_permissions      django_admin_log
auth_permission             django_content_type
auth_user                   django_migrations
auth_user_groups            django_session
```

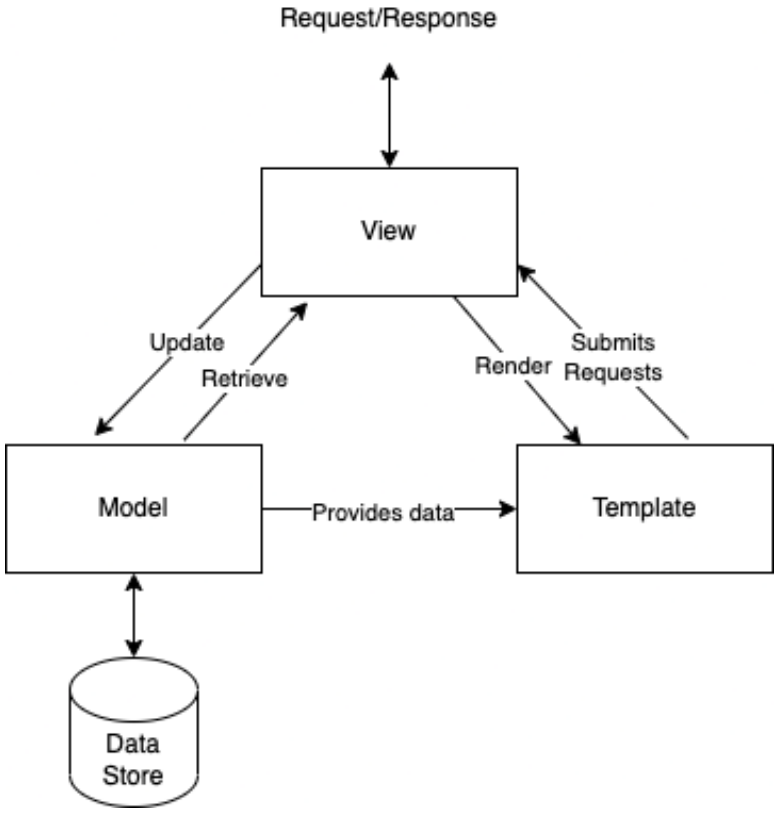
3.1 The Model-View-Controller (MVC) Architectural Pattern

The prompt templates in this book are based on the Model-View-Controller (MVC) architectural pattern. This pattern defines the types of components that make up the application and the way in which they interact. This pattern is used by many web development frameworks such as Django.

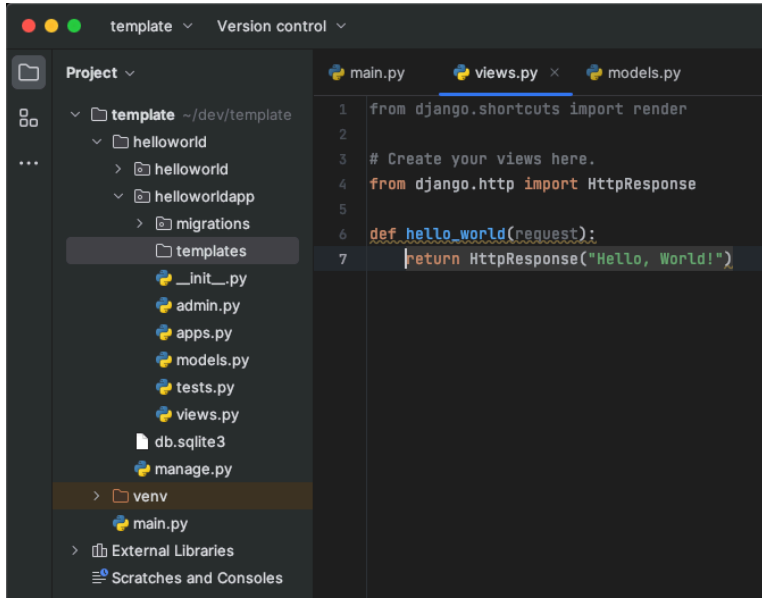
Unfortunately, Django changes the terminology just a bit. It uses a Model-View-Template pattern. Each component type is defined below.

- **Model** classes represent business entities and they implement reading and persisting data to a database.
- **Views** implement the logic used when a user browses to an application URL or submits a form.
- **Templates** contain the HTML and associated presentation logic used to generate the web page and user interface.

The following diagram shows the interaction between these component types in the architecture.



Your directory structure in PyCharm should now look something like the image shown below.



The model classes are stored in the `models.py` file. The views, likewise, are stored in the `views.py` file. Create a templates directory in the helloworldapp folder. This directory will store your html templates.

You will be using prompts to generate code, and then pasting that code into the appropriate location. Be sure to familiarize yourself with the project structure so you can be efficient at integrating code generated by your AI helper.

One recommendation to simplify project organization is having a having a `services.py` file to store reusable functions. This is not required by Django. However, it is a common construct that has been helpful for designing applications for many years.

At the end of the day, it is up to you how you want to organize the code within your project. As long as you know where different types of components are stored and it makes sense to you, then you are in good shape.

4 Prompt Two: Create your models

The best place to start building an application is by modeling your data. If you have a good understanding of your data model, you will also have a solid foundation for designing the rest of your application.

Model classes are a great place to start when building Django web applications. Once you have model classes, you can create the corresponding database tables as well. Model classes typically provide the building blocks for everything else you build in the application.

4.1 Designing Models

If you have software design experience, you are likely able to design your application model classes. Models in Django are analogous to database entity-relationship modeling. These are often captured using an Entity-Relationship Diagram (ERD). After you have the design, you can still have AI write the model code for you.

However, If you are not comfortable doing the entity modeling yourself, you can also have your AI assistant help.

Use a design prompt based on the template below. The requirements in this template are essentially your high-level application requirements. You may have more or less than three requirements. Non-trivial applications will likely have more than three requirements.

You can also break this up into multiple prompts. This becomes more important if you have a large number of requirements and model entities.

Prompt template

You are a software designer building a **project_name** application. Define the model classes needed to implement the following requirements. 1. **Requirement_1** 2. **Requirement_2** 3. **Requirement_3**

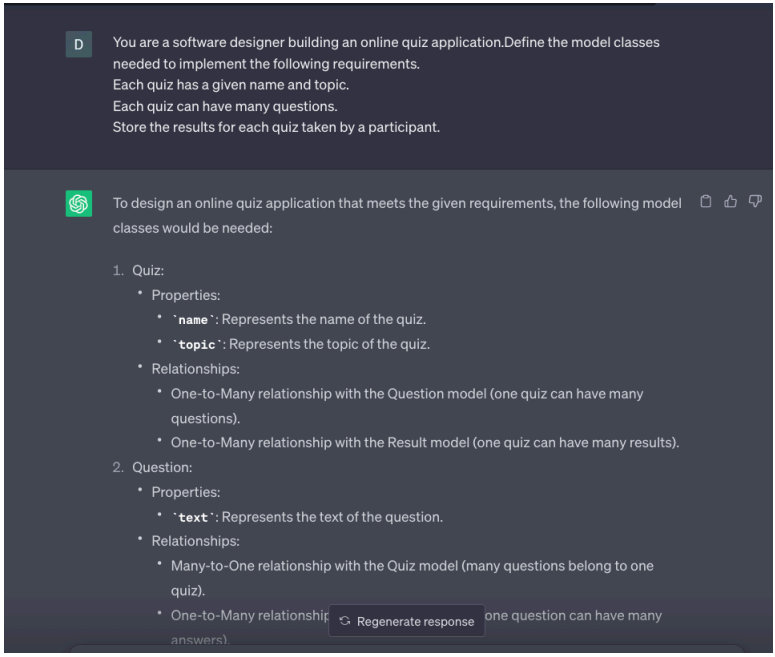
Example prompt

You are a software designer building an **online quiz application**. Define the model classes needed to implement the following requirements. **1. Each quiz has a given name and topic. 2. Each quiz can have many questions. 3. Store the results for each quiz taken by a participant.**

Test prompt

None. Review the AI output.

A portion of the response from the quiz prompt is shown below.



Review the models defined in the AI response. Does it seem like it meets your expectation? If it doesn't, you can use follow up prompts to change the model based on your needs.

> A common problem in this phase is a lack of detail. Be specific whenever possible.

4.2 Err on the side of too much detail in prompts

In the absence of details, AI makes assumptions to fill in the gaps. If you don't provide specifics, AI will choose for you. And it's choices may not be what you had in mind. Providing enough detail and

context is critical to the efficiency of this process. The more detail, the better.

There are context limits to what AI can remember at a given time. Nonetheless, it is better to err on the side of too much context rather than too little. If you do hit a context limit, then you can dial back how much detail you give the AI in a given prompt.

How will you know that you hit a context limit? Typically, the AI response is cut short in the middle of a sentence or a thought.

If this occurs, break the prompt down into multiple prompts. You can split out related requirements into smaller groups and run subsequent prompts to cover all of them.

As AI capabilities grow more advanced, the context (or token) limit will expand. Building the entire application may one day be possible with a single prompt. However, we are not there yet. The process of building the application is, in itself, decomposing the overall problem into a series of smaller, manageable steps.

It is also important to remember that this is an iterative process. It's perfectly okay if the AI doesn't produce exactly what you want on the first try. The great thing about AI assistants is that they retain context and can modify models and code previously generated. You will be continually reviewing the AI output and modifying the results through additional prompts. Once you are satisfied with the output, then you can proceed to the next step in the process.

Use the following prompt to generate the Python Django code for your models.

Prompt template

Write the Python code for these model classes based on the **framework_name** framework

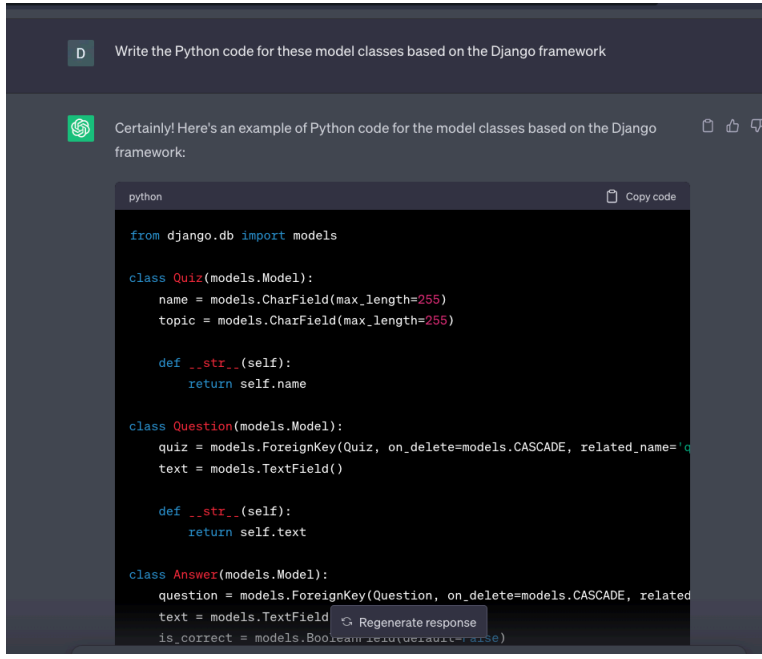
Example prompt

Write the Python code for these model classes based on the Django framework

Test prompt

Write Django test cases for these models

Here is a portion of the sample response from ChatGPT. Copy this code and place it in your `models.py` file.



Anytime you have new or updated model classes, you will want to create migrations and apply them to your database. To do this, you first need to let Django know to look in your applications.

Open the `helloworld/settings.py` file in PyCharm, and add the HelloworldappConfig line to the INSTALLED_APPS section:

```
# Application definition

INSTALLED_APPS = [
    'helloworldapp.apps.HelloworldappConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

Now run the following commands.

```
(venv) broemmerd$ python manage.py makemigrations
Migrations for 'helloworldapp':
  helloworldapp/migrations/0001_initial.py
    - Create model Participant
    - Create model Quiz
    - Create model Result
    - Create model Question
    - Create model Answer

(venv) broemmerd$ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, hellow\
orldapp, sessions
Running migrations:
  Applying helloworldapp.0001_initial... OK
```

The first command created the migration files which define the database tables using a Django DSL, or Domain-Specific Language. The second command runs the file against the database to actually create the tables. You can now login to the sqlite database and see the tables.

```
(venv) broemmerd$ sqlite3 db.sqlite3
SQLite version 3.28.0 2019-04-15 14:49:49
Enter ".help" for usage hints.
sqlite> .tables
auth_group                  django_migrations
auth_group_permissions      django_session
auth_permission             helloworldapp_answer
auth_user                   helloworldapp_participant
auth_user_groups            helloworldapp_question
auth_user_user_permissions  helloworldapp_quiz
django_admin_log            helloworldapp_result
django_content_type
```

Model diagrams are often useful to understand and verify your design. There are a number of tools available to do this. For a simple approach, you can create an ASCII diagram that illustrates the model using the following prompt.

Prompt template

> Create a diagram that shows the relationships between the model classes.

To generate a better model diagram, run the following commands to install Django extensions which has a utility for this purpose.

```
python -m pip install django-extensions
python -m pip install pydotplus
```

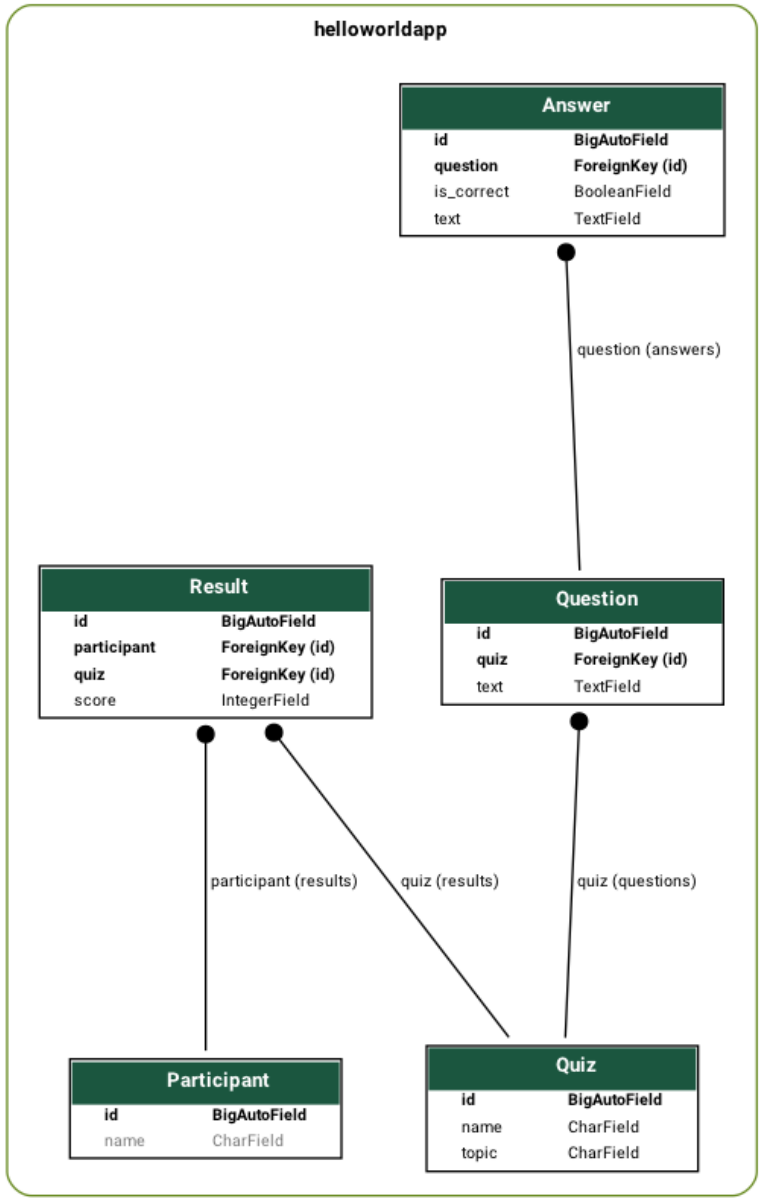
Then add it to the list of INSTALLED_APPS in the settings.py file.

```
'django_extensions',
```

You can now run the following command from the helloworld directory. This will create a model diagram with the name you specify. In this example, the diagram will be created with the filename `myapp_models.png`.

```
python manage.py graph_models -a -g —o myapp_models.png
```

This creates the following image. We show only the relevant app here. The default Django framework models created at the beginning will also be in the diagram unless they are excluded using other command-line options. This model looks pretty good. Each quiz is made up of one or more questions, and each question has a set of possible answers, one of which is correct. Each quiz is taken by participants and has results with a score for that participant.


helloworldapp
Answer
id BigAutoField
question ForeignKey (id)
is_correct BooleanField
text TextField
question (answers)
Result
id BigAutoField
participant ForeignKey (id)
quiz ForeignKey (id)
score IntegerField
Question
id BigAutoField
quiz ForeignKey (id)
text TextField
participant (results)
quiz (results)
quiz (questions)
Participant
id BigAutoField
name CharField
Quiz
id BigAutoField
name CharField
topic CharField

If you want to see more options for the model diagram generation, use the following command.

```
./manage.py graph_models --help
```

4.3 Running Django test cases

Use the test prompt to have AI generate test cases for you. Copy the code and put it into the `tests.py` file in your project. You can run Django tests using the following command.

```
python manage.py test helloworldapp.tests
```

Note that the last token uses whatever name you gave your Django app. Within that app, the tests token that follows refers to the `tests.py` file.

Be in the habit of always using the test prompts and running these tests after each iteration or step in the process. It is significantly easier to identify a problem right away. If the issue doesn't pop up until a few steps later, it will be more challenging to narrow down where the problem was introduced.

5 Prompt Three: Implement your first use case

With the models in place, you are in a position to build user interfaces and services that leverage those model entities.

What user interfaces and services do we need? The answer to that question is driven by your application use cases. The first step in this phase is to identify the user personas.

5.1 Define the user personas

A persona is a type of user that uses your application. You can ask AI for help with this step, but it should be relatively easy to reason through yourself.

Consider the example quiz application. There are two user personas.

- The **quiz creator**, the user who creates quizzes and can view aggregate results and metrics.
- The **quiz participant**, the user who takes the quiz and receives their score.

If we decide that any user can create quizzes, then all users can play both roles.

Given that aggregate quiz results are typically not sensitive data, we can also allow any user to see the overall results for any quiz.

The only restriction then, is that participants are the only user who can see their own individual quiz results. Scores are available to other users only in an aggregated or anonymized form.

5.2 Define what actions each user person can take

Your application may have actions only allowed by specific users. If this is the case in your application, we will revisit the topic of security and authorization in a later chapter.

If you are struggling to identify use cases, try considering how each of your models are created. Are they created by the user taking an action? Does the system generate them based on a certain event? The answers to these questions become your detailed requirements.

For example, here are the requirements about model creation in the quiz application. These were part of the model design prompt.

- A quiz instance is created by a user. The user supplies the quiz name and the topic.
- The quiz questions are generated by the system for the given topic.
- Each question should have four possible answers.

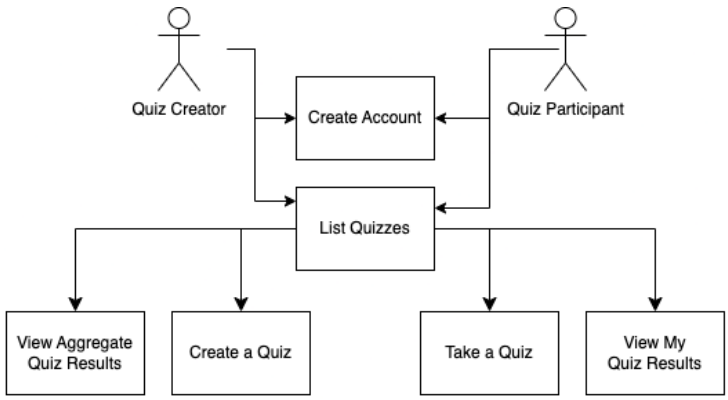
All of this happens in the `Create a Quiz` use case. This use case covers the creation of 3 of the 5 model entities in the system (Quiz, Question, Answer). The creation of Participant occurs in the `Create Account` use case. The creation of Results occurs as a part of the `Take a Quiz` use case.

Additional use cases can be identified by considering what users do with the information. Quiz results are viewed by the participant and also fed into the generation of aggregate quiz results.

After thinking through all of this, list the actions that each user persona can perform in your application. The following table lists the actions, or use cases, for the quiz application.

Action (Use Case)	User Persona
Create account	Creator/Participant
View list of quizzes	Creator/Participant
Create a quiz	Creator/Participant
View aggregate quiz results	Creator/Participant
Take a quiz	Creator/Participant
View individual quiz results	Participant

We can now map out the flow each user takes through these use cases.



After a user has created their account, the start point for any workflow is to view the list of quizzes. Let's implement that use case first.

First, ask yourself, does this use case require any underlying business services? Should we break it down into multiple prompts?

In this case, view the list of quizzes is a fairly simple use case. We

can go straight into the basic use case prompt shown below.

Prompt template

Write the Python Django code to **use_case_description**. Create a custom view in Django with **description_of_page_contents**. Create the corresponding HTML template in Django to **display_details**.

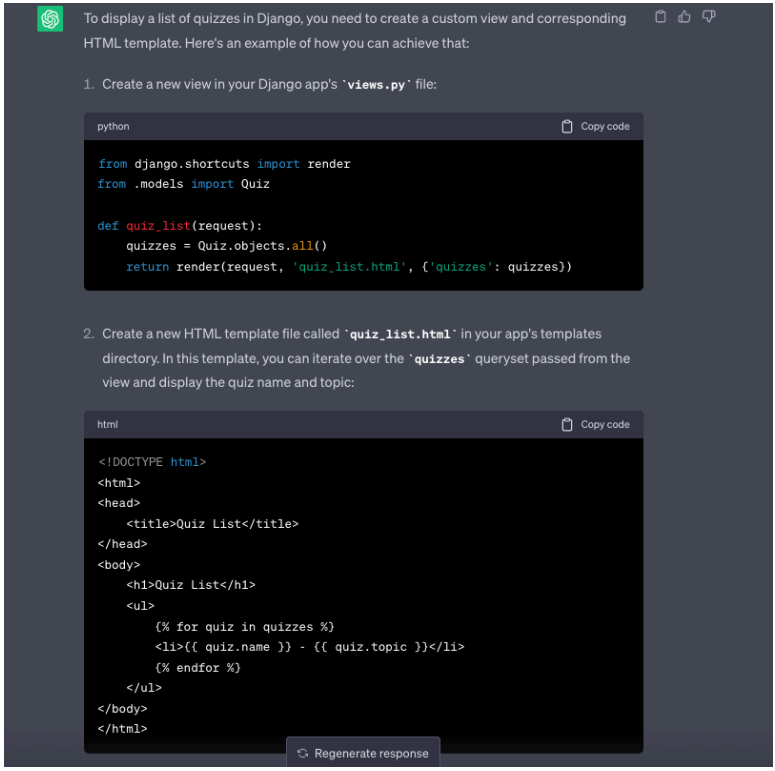
Example prompt

Write the Python Django code to display a list of Quizzes. Create a custom view in Django with the list of quizzes from the database. Create the corresponding HTML template in Django to display each quiz name and topic.

Test prompt

Write a Django test case for this view

The AI produces this output.



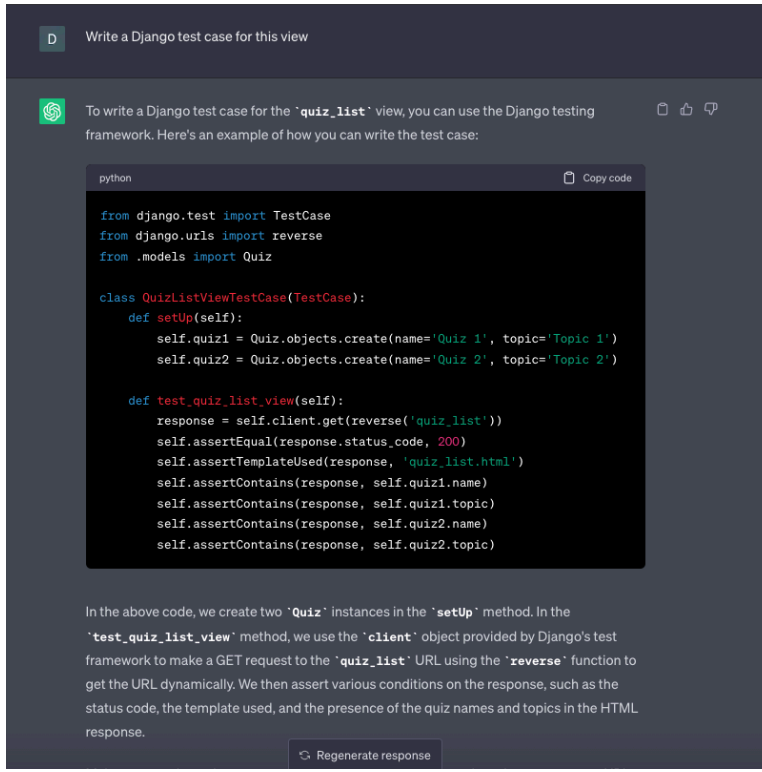
Copy the view code and add it to your `views.py` file. Create a new HTML file in the templates directory called `quiz_list.html` and copy the template code into that file.

ChatGPT will usually remind you to put an entry in the `urls.py` file. You can also prompt it to give you code. For this case, the only new url entry is as shown below.

```
from helloworldapp.views import quiz_list

urlpatterns = [
    . . .
    path('quiz-list/', quiz_list, name='quiz_list'),
]
```

We don't have any data in the database yet, because we haven't implemented the Create a Quiz use case. However, we can still use the test prompt to exercise this code. Use the test prompt from the template above. Here is the output from that prompt.



That test should pass. Now let's create an actual quiz so we can see

some quiz data in the browser.

6 Prompt Four: Implement a business service

Use cases that create data are typically more complex that read-only use cases. The Create a Quiz use case is a great example of a use case that requires additional business logic.

In order to determine what prompts to use, break down the use case into individual steps. These steps can be derived from the detailed requirements you put together during the earlier design phase. You may also need to add requirements in this phase as you go through the specifics of particular use cases.

To create a quiz in our example application, we need to perform the following steps.

1. Create a form for the user to enter the name of the quiz and the topic
2. Create some number of quiz questions. For our application, we decide to use generative AI to create the questions for us. Thus, this is a candidate for a business service.
3. Store the quiz, the list of questions, and the answers in the database.
4. Confirm to the user the quiz was created.

You can see what type of component will be needed for each step, as shown in the table below.

Step	Component type
Data entry form	View, HTML template
Generate quiz questions	Business service
Store the quiz objects	View
Display user confirmation	HTML template

It is best to start at the lowest layer of dependencies and work your way up. The service to generate quiz questions will be used within the view, so generate that first. It is also self-contained and be tested independently.

6.1 Implement a generative AI service

You will need a helper function to invoke the AI service. The code to invoke the OpenAI service is shown below. See the Setup your Environment chapter if you have not already configured and tested the use of OpenAI. You can put this function in the `services.py` file.

call_chatgpt Python function

```
import openai
import os
from dotenv import load_dotenv
load_dotenv()

def call_chatgpt(prompt, temperature = 0.7, max_tokens = \
1024):
    # Set up OpenAI API key
    openai.api_key = os.getenv("OPENAI_API_KEY")

    response = openai.Completion.create(
        engine="text-davinci-003",
        prompt=prompt,
```

```
        max_tokens=max_tokens,
        temperature=temperature,
    )

    return response.choices[0].text
```

The corresponding code to invoke the Bard API is shown below. See the Setup Environment chapter for more information on configuring your Bard account. Note that it has to do a bit of post-processing on the response due to the Bard output format.

call_bard Python function

```
def call_bard(query):
    bard = Bard()
    answer = bard.get_answer(query)
    print(answer)

    # Only return response text within the delimeter,
    # typically triple backticks
    response_text = ""
    lines = answer['content'].split('\n')
    inside_delimeter = False
    for line in lines:
        if line.startswith("```"):
            inside_delimeter = not inside_delimeter
        elif inside_delimeter:
            if len(line) > 0:
                response_text = response_text + line + '\\
n'

    return (response_text)
```

Bard tends to always respond with a first line that says, “Sure, I can do that”. The information specifically requested in a prompt is

often later surrounded by lines with triple backticks. Thus, we want to ignore everything else except the lines between the backticks.

On the trivia question prompt we will see in a minute, Bard also includes an explanation of the answer at the end. While that may be useful later, it causes our parsing logic to fail. So we ignore any lines after the second delimiter line.

Now that you have a function to invoke the generative AI service from your application, you just need to craft a prompt for your use case.

Below is the code and associated prompt to generate a trivia question. Note that the prompt specifies a JSON return format.

> It is a good practice to specify the output format for generative AI prompts used in applications.

This makes it easy for the application software to parse and use response data. In general, we have found that using XML is the most effective format. If you were writing code by hand, you likely would not use XML. Json would be a more logical choice these days.

However, in our testing, XML has been found to be more reliable for AI code generators, both in terms of creating the data and then also parsing it. Bard had trouble correctly creating JSON data as it mixed up quotes and double quotes within the json content. Thus, we specify XML as the response format in the code example below.

Code to use generative AI within Python application

```
def generate_question(topic):

    # Define the prompt to generate trivia questions on t\
he given topic
    prompt = f'''Generate one trivia question on the topi\
c of {topic}.
Provide four possible answers and then indicate the lette\
r of the correct answer.
Format your response using the following XML template.

<response>
<question>[QUESTION_TEXT]</question>
<a>A) [ANSWER_1]</a>
<b>B) [ANSWER_2]</b>
<c>C) [ANSWER_3]</c>
<d>D) [ANSWER_4]</d>
<correct_answer>[CORRECT_ANSWER_LETTER]</correct_answer>
</response>
'''

    return call_chatgpt(prompt)
```

6.2 Parse the data returned from a generative AI function

The `generate_question` function needs to be invoked within the quiz creation use case. We need to be able to create a Question model and corresponding Answer models from the response to this prompt. Let's ask the AI to write this code using the following prompt.

Prompt template

Write a Python function to parse text data and create **model_objects**. The representative format of the text is as follows: **example_data_format**

Example prompt

Write a Python function to parse text data and create a corresponding Question model instance as well as corresponding Answer model instances. The correct answer should set the boolean flag on the corresponding Answer instance. The representative format of the text is as follows:<correct_answer>B</correct_answer>

Test prompt

Write a Django test case for the **generated_function_name** function.

AI produces the following code.

```
def create_question_with_answers_from_xml(xml_text, quiz):
    # Parse the XML
    root = ET.fromstring(xml_text)

    # Extract question text
    question_text = root.find('question').text.strip()

    # Extract answer options
    answer_texts = []
    for child in root:
        if child.tag in ['a', 'b', 'c', 'd']:
            answer_texts.append(child.text.strip())

    # Extract correct answer
    correct_answer = root.find('correct_answer').text.str\
ip()

    # Create the question instance
```

```
    question = Question.objects.create(quiz=quiz, text=qu\
estion_text)

    # Create the answer instances
    for index, answer_text in enumerate(answer_texts):
        option = chr(ord('a') + index).upper()
        is_correct = (option == correct_answer.upper())
        Answer.objects.create(question=question, text=ans\
wer_text, option=option, is_correct=is_correct)

    return question
```

The first code that AI generated had a subtle error which caused our test to fail.

```
django.db.utils.IntegrityError: NOT NULL constraint faile\
d:
helloworldapp_question.quiz_id
```

If you get an error of this form, you can simply tell the AI in a follow up prompt that you got this error. In our case, it was able to rewrite the code to fix the problem. The Question instance in this function has to be created with the reference to the related existing quiz.

This issue highlights the importance of always generating test code using the test prompt from the template, and then running the tests. For the code generated above, the test prompt is shown below.

Test Prompt

> Write a Django test case for the create_question_with_answers_from_xml function.

You can then run the tests using the following command.

```
python manage.py test helloworldapp.tests
```

It is much easier to identify and resolve issues with the code when you can isolate the cause and ask the AI to rewrite the code while all of the information is in context.

You can always rewrite code later on in the development process. You may need to provide the code in question in the prompt. However, if you bundle a series of generated code changes at one time, diagnosing which code caused the problem can become more challenging and time-consuming.

Now that we have the business service needed for the Create a Quiz use case, we can generate the rest of the code.

7 Prompt Five: Implement a complex use case

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/ai-prompt-templates-to-build-apps

7.1 Modify our existing implementation to create multiple quiz questions

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/ai-prompt-templates-to-build-apps

7.2 Modify the existing page style

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/ai-prompt-templates-to-build-apps

8 Prompt Six: Implement a use case that uses generated data

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/ai-prompt-templates-to-build-apps

8.1 Persist the results to the database

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/ai-prompt-templates-to-build-apps

9 Prompt Seven: User authentication

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/ai-prompt-templates-to-build-apps

10 Prompt Eight: Reporting use case

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/ai-prompt-templates-to-build-apps

10.1 Creating a graph based on your data

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/ai-prompt-templates-to-build-apps

11 Prompt Checklist

This content is not available in the sample book. The book can be purchased on Leanpub at http://leanpub.com/ai-prompt-templates-to-build-apps