# Full Explanation of app.py and requirements.txt

## Overview

This application is a **Streamlit-based web app** that serves as a **Gemini-Powered SDLC (Software Development Life Cycle) Assistant**. It integrates with Googles Generative AI via the Gemini API (Text-Bison model) to automate tasks in four main SDLC areas:

- Requirement Analysis
- Code Generation
- Code Review
- Test Case Generation

## File: `requirements.txt`

This file lists Python dependencies:

- `streamlit`: Used to create the interactive web app interface.
- `google-generativeai`: Required to access and interact with Googles Gemini API.

## File: `app.py`

### 1. **Imports and Configuration**

```python
import streamlit as st
```

import google.generativeai as genai

genai.configure(api_key=st.secrets["GEMINI_API_KEY"])

- Loads necessary libraries and configures the Gemini API with a secret key stored in Streamlit's secret management.

### 2. **Model Initialization**

```python
model = genai.GenerativeModel("models/text-bison-001")
```

- Initializes the Gemini model `text-bison-001`.

### 3. **Function: `call_gemini(prompt)`**

Handles all interaction with the model and returns generated text or error message.

### 4. **SDLC-Specific Functions**

Each SDLC function constructs a prompt and sends it to Gemini:

- `summarize_requirements(text)`
- `generate_python_code(req)`
- `review_code(code)`
- `generate_test_cases(desc)`

### 5. **Streamlit UI**

```python
st.set_page_config(...)
st.title(...)
```

tabs = st.tabs([...])

```

- Defines the layout and four functional tabs:

  - **Requirement Analysis**: Accepts a text file upload and summarizes it.

  - **Code Generation**: Generates Python code from natural language input.

  - **Code Review**: Reviews pasted code for bugs or improvements.

  - **Test Case Generation**: Creates test cases from feature descriptions.

---

[Additional pages in the document would explain each function and Streamlit UI component in more detail.]

# Full Explanation of app.py and requirements.txt

## Overview

This application is a **Streamlit-based web app** that serves as a **Gemini-Powered SDLC (Software Development Life Cycle) Assistant**. It integrates with Googles Generative AI via the Gemini API (Text-Bison model) to automate tasks in four main SDLC areas:

- Requirement Analysis

- Code Generation

- Code Review

- Test Case Generation

---
```

## File: `requirements.txt`

This file lists Python dependencies:

- `streamlit`: Used to create the interactive web app interface.

- `google-generativeai`: Required to access and interact with Googles Gemini API.

## File: `app.py`

### 1. **Imports and Configuration**

```python
import streamlit as st

import google.generativeai as genai


genai.configure(api_key=st.secrets["GEMINI_API_KEY"])
```

- Loads necessary libraries and configures the Gemini API with a secret key stored in Streamlit's secret management.

### 2. **Model Initialization**

```python
model = genai.GenerativeModel("models/text-bison-001")
```

- Initializes the Gemini model `text-bison-001`.

### 3. **Function: `call_gemini(prompt)`**

Handles all interaction with the model and returns generated text or error message.

### 4. **SDLC-Specific Functions**

Each SDLC function constructs a prompt and sends it to Gemini:

- `summarize_requirements(text)`

- `generate_python_code(req)`

- `review_code(code)`

- `generate_test_cases(desc)`

### 5. **Streamlit UI**

```python
st.set_page_config(...)

st.title(...)

tabs = st.tabs([...])
```

- Defines the layout and four functional tabs:

  - **Requirement Analysis**: Accepts a text file upload and summarizes it.

  - **Code Generation**: Generates Python code from natural language input.

  - **Code Review**: Reviews pasted code for bugs or improvements.

  - **Test Case Generation**: Creates test cases from feature descriptions.

[Additional pages in the document would explain each function and Streamlit UI component in more detail.]

# Full Explanation of app.py and requirements.txt

## Overview

This application is a **Streamlit-based web app** that serves as a **Gemini-Powered SDLC (Software Development Life Cycle) Assistant**. It integrates with Googles Generative AI via the Gemini API (Text-Bison model) to automate tasks in four main SDLC areas:

- Requirement Analysis
- Code Generation
- Code Review
- Test Case Generation

## File: `requirements.txt`

This file lists Python dependencies:

- `streamlit`: Used to create the interactive web app interface.
- `google-generativeai`: Required to access and interact with Googles Gemini API.

## File: `app.py`

### 1. **Imports and Configuration**

```python
import streamlit as st

import google.generativeai as genai
```

genai.configure(api_key=st.secrets["GEMINI_API_KEY"])

- Loads necessary libraries and configures the Gemini API with a secret key stored in Streamlit's secret management.

### 2. **Model Initialization**

```python

model = genai.GenerativeModel("models/text-bison-001")

```

- Initializes the Gemini model `text-bison-001`.

### 3. **Function: `call_gemini(prompt)`**

Handles all interaction with the model and returns generated text or error message.

### 4. **SDLC-Specific Functions**

Each SDLC function constructs a prompt and sends it to Gemini:

- `summarize_requirements(text)`

- `generate_python_code(req)`

- `review_code(code)`

- `generate_test_cases(desc)`

### 5. **Streamlit UI**

```python

st.set_page_config(...)

st.title(...)

tabs = st.tabs([...])
```

```

- Defines the layout and four functional tabs:

  - **Requirement Analysis**: Accepts a text file upload and summarizes it.

  - **Code Generation**: Generates Python code from natural language input.

  - **Code Review**: Reviews pasted code for bugs or improvements.

  - **Test Case Generation**: Creates test cases from feature descriptions.

---

[Additional pages in the document would explain each function and Streamlit UI component in more detail.]

# Full Explanation of app.py and requirements.txt

## Overview

This application is a **Streamlit-based web app** that serves as a **Gemini-Powered SDLC (Software Development Life Cycle) Assistant**. It integrates with Googles Generative AI via the Gemini API (Text-Bison model) to automate tasks in four main SDLC areas:

- Requirement Analysis

- Code Generation

- Code Review

- Test Case Generation

---

## File: `requirements.txt`
```

This file lists Python dependencies:

- `streamlit`: Used to create the interactive web app interface.

- `google-generativeai`: Required to access and interact with Googles Gemini API.

## File: `app.py`

### 1. **Imports and Configuration**

```python
import streamlit as st
import google.generativeai as genai

genai.configure(api_key=st.secrets["GEMINI_API_KEY"])
```

- Loads necessary libraries and configures the Gemini API with a secret key stored in Streamlit's secret management.

### 2. **Model Initialization**

```python
model = genai.GenerativeModel("models/text-bison-001")
```

- Initializes the Gemini model `text-bison-001`.

### 3. **Function: `call_gemini(prompt)`**

Handles all interaction with the model and returns generated text or error message.

### 4. **SDLC-Specific Functions**

Each SDLC function constructs a prompt and sends it to Gemini:

- `summarize_requirements(text)`

- `generate_python_code(req)`

- `review_code(code)`

- `generate_test_cases(desc)`

### 5. **Streamlit UI**

```python
st.set_page_config(...)
st.title(...)
tabs = st.tabs([...])
```

- Defines the layout and four functional tabs:

  - **Requirement Analysis**: Accepts a text file upload and summarizes it.

  - **Code Generation**: Generates Python code from natural language input.

  - **Code Review**: Reviews pasted code for bugs or improvements.

  - **Test Case Generation**: Creates test cases from feature descriptions.

[Additional pages in the document would explain each function and Streamlit UI component in more detail.]

# Full Explanation of app.py and requirements.txt

## Overview

This application is a **Streamlit-based web app** that serves as a **Gemini-Powered SDLC (Software Development Life Cycle) Assistant**. It integrates with Googles Generative AI via the Gemini API (Text-Bison model) to automate tasks in four main SDLC areas:

- Requirement Analysis

- Code Generation

- Code Review

- Test Case Generation

## File: `requirements.txt`

This file lists Python dependencies:

- `streamlit`: Used to create the interactive web app interface.

- `google-generativeai`: Required to access and interact with Googles Gemini API.

## File: `app.py`

### 1. **Imports and Configuration**

```python

import streamlit as st

import google.generativeai as genai
```

genai.configure(api_key=st.secrets["GEMINI_API_KEY"])

- Loads necessary libraries and configures the Gemini API with a secret key stored in Streamlit's secret management.

### 2. **Model Initialization**

```python

model = genai.GenerativeModel("models/text-bison-001")

```

- Initializes the Gemini model `text-bison-001`.

### 3. **Function: `call_gemini(prompt)`**

Handles all interaction with the model and returns generated text or error message.

### 4. **SDLC-Specific Functions**

Each SDLC function constructs a prompt and sends it to Gemini:

- `summarize_requirements(text)`

- `generate_python_code(req)`

- `review_code(code)`

- `generate_test_cases(desc)`

### 5. **Streamlit UI**

```python

st.set_page_config(...)

st.title(...)

tabs = st.tabs([...])

```

- Defines the layout and four functional tabs:
  - **Requirement Analysis**: Accepts a text file upload and summarizes it.
  - **Code Generation**: Generates Python code from natural language input.
  - **Code Review**: Reviews pasted code for bugs or improvements.
  - **Test Case Generation**: Creates test cases from feature descriptions.

[Additional pages in the document would explain each function and Streamlit UI component in more detail.]

# Full Explanation of app.py and requirements.txt

## Overview

This application is a **Streamlit-based web app** that serves as a **Gemini-Powered SDLC (Software Development Life Cycle) Assistant**. It integrates with Googles Generative AI via the Gemini API (Text-Bison model) to automate tasks in four main SDLC areas:
- Requirement Analysis
- Code Generation
- Code Review
- Test Case Generation

## File: `requirements.txt`

This file lists Python dependencies:

- `streamlit`: Used to create the interactive web app interface.

- `google-generativeai`: Required to access and interact with Googles Gemini API.

## File: `app.py`

### 1. **Imports and Configuration**

```python
import streamlit as st
import google.generativeai as genai

genai.configure(api_key=st.secrets["GEMINI_API_KEY"])
```

- Loads necessary libraries and configures the Gemini API with a secret key stored in Streamlit's secret management.

### 2. **Model Initialization**

```python
model = genai.GenerativeModel("models/text-bison-001")
```

- Initializes the Gemini model `text-bison-001`.

### 3. **Function: `call_gemini(prompt)`**

Handles all interaction with the model and returns generated text or error message.

### 4. **SDLC-Specific Functions**

Each SDLC function constructs a prompt and sends it to Gemini:

- `summarize_requirements(text)`

- `generate_python_code(req)`

- `review_code(code)`

- `generate_test_cases(desc)`

### 5. **Streamlit UI**

```python
st.set_page_config(...)

st.title(...)

tabs = st.tabs([...])
```

- Defines the layout and four functional tabs:

  - **Requirement Analysis**: Accepts a text file upload and summarizes it.

  - **Code Generation**: Generates Python code from natural language input.

  - **Code Review**: Reviews pasted code for bugs or improvements.

  - **Test Case Generation**: Creates test cases from feature descriptions.

[Additional pages in the document would explain each function and Streamlit UI component in more detail.]

# Full Explanation of app.py and requirements.txt

## Overview

This application is a **Streamlit-based web app** that serves as a **Gemini-Powered SDLC (Software Development Life Cycle) Assistant**. It integrates with Googles Generative AI via the Gemini API (Text-Bison model) to automate tasks in four main SDLC areas:

- Requirement Analysis

- Code Generation

- Code Review

- Test Case Generation

## File: `requirements.txt`

This file lists Python dependencies:

- `streamlit`: Used to create the interactive web app interface.

- `google-generativeai`: Required to access and interact with Googles Gemini API.

## File: `app.py`

### 1. **Imports and Configuration**

```python

import streamlit as st

import google.generativeai as genai

genai.configure(api_key=st.secrets["GEMINI_API_KEY"])
```

- Loads necessary libraries and configures the Gemini API with a secret key stored in Streamlit's secret management.

### 2. **Model Initialization**

```python
model = genai.GenerativeModel("models/text-bison-001")
```

- Initializes the Gemini model `text-bison-001`.

### 3. **Function: `call_gemini(prompt)`**

Handles all interaction with the model and returns generated text or error message.

### 4. **SDLC-Specific Functions**

Each SDLC function constructs a prompt and sends it to Gemini:

- `summarize_requirements(text)`

- `generate_python_code(req)`

- `review_code(code)`

- `generate_test_cases(desc)`

### 5. **Streamlit UI**

```python
st.set_page_config(...)

st.title(...)

tabs = st.tabs([...])
```

- Defines the layout and four functional tabs:

  - **Requirement Analysis**: Accepts a text file upload and summarizes it.
  - **Code Generation**: Generates Python code from natural language input.
  - **Code Review**: Reviews pasted code for bugs or improvements.
  - **Test Case Generation**: Creates test cases from feature descriptions.

[Additional pages in the document would explain each function and Streamlit UI component in more detail.]

# Full Explanation of app.py and requirements.txt

## Overview

This application is a **Streamlit-based web app** that serves as a **Gemini-Powered SDLC (Software Development Life Cycle) Assistant**. It integrates with Googles Generative AI via the Gemini API (Text-Bison model) to automate tasks in four main SDLC areas:

- Requirement Analysis
- Code Generation
- Code Review
- Test Case Generation

## File: `requirements.txt`

This file lists Python dependencies:

- `streamlit`: Used to create the interactive web app interface.

- `google-generativeai`: Required to access and interact with Googles Gemini API.

## File: `app.py`

### 1. **Imports and Configuration**

```python
import streamlit as st
import google.generativeai as genai

genai.configure(api_key=st.secrets["GEMINI_API_KEY"])
```

- Loads necessary libraries and configures the Gemini API with a secret key stored in Streamlit's secret management.

### 2. **Model Initialization**

```python
model = genai.GenerativeModel("models/text-bison-001")
```

- Initializes the Gemini model `text-bison-001`.

### 3. **Function: `call_gemini(prompt)`**

Handles all interaction with the model and returns generated text or error message.

### 4. **SDLC-Specific Functions**

Each SDLC function constructs a prompt and sends it to Gemini:

- `summarize_requirements(text)`
- `generate_python_code(req)`
- `review_code(code)`
- `generate_test_cases(desc)`

### 5. **Streamlit UI**

```python
st.set_page_config(...)
st.title(...)
tabs = st.tabs([...])
```

- Defines the layout and four functional tabs:
  - **Requirement Analysis**: Accepts a text file upload and summarizes it.
  - **Code Generation**: Generates Python code from natural language input.
  - **Code Review**: Reviews pasted code for bugs or improvements.
  - **Test Case Generation**: Creates test cases from feature descriptions.

[Additional pages in the document would explain each function and Streamlit UI component in more detail.]

# Full Explanation of app.py and requirements.txt

## Overview

This application is a **Streamlit-based web app** that serves as a **Gemini-Powered SDLC (Software Development Life Cycle) Assistant**. It integrates with Googles Generative AI via the Gemini API (Text-Bison model) to automate tasks in four main SDLC areas:

- Requirement Analysis
- Code Generation
- Code Review
- Test Case Generation

## File: `requirements.txt`

This file lists Python dependencies:

- `streamlit`: Used to create the interactive web app interface.
- `google-generativeai`: Required to access and interact with Googles Gemini API.

## File: `app.py`

### 1. **Imports and Configuration**

```python
import streamlit as st

import google.generativeai as genai

genai.configure(api_key=st.secrets["GEMINI_API_KEY"])
```

- Loads necessary libraries and configures the Gemini API with a secret key stored in Streamlit's secret management.

### 2. **Model Initialization**

```python
model = genai.GenerativeModel("models/text-bison-001")
```

- Initializes the Gemini model `text-bison-001`.

### 3. **Function: `call_gemini(prompt)`**

Handles all interaction with the model and returns generated text or error message.

### 4. **SDLC-Specific Functions**

Each SDLC function constructs a prompt and sends it to Gemini:

- `summarize_requirements(text)`
- `generate_python_code(req)`
- `review_code(code)`
- `generate_test_cases(desc)`

### 5. **Streamlit UI**

```python
st.set_page_config(...)
st.title(...)
tabs = st.tabs([...])
```

- Defines the layout and four functional tabs:
  - **Requirement Analysis**: Accepts a text file upload and summarizes it.

  - **Code Generation**: Generates Python code from natural language input.

  - **Code Review**: Reviews pasted code for bugs or improvements.

  - **Test Case Generation**: Creates test cases from feature descriptions.

[Additional pages in the document would explain each function and Streamlit UI component in more detail.]

# Full Explanation of app.py and requirements.txt

## Overview

This application is a **Streamlit-based web app** that serves as a **Gemini-Powered SDLC (Software Development Life Cycle) Assistant**. It integrates with Googles Generative AI via the Gemini API (Text-Bison model) to automate tasks in four main SDLC areas:

- Requirement Analysis

- Code Generation

- Code Review

- Test Case Generation

## File: `requirements.txt`

This file lists Python dependencies:

- `streamlit`: Used to create the interactive web app interface.

- `google-generativeai`: Required to access and interact with Googles Gemini API.

## File: `app.py`

### 1. **Imports and Configuration**

```python
import streamlit as st
import google.generativeai as genai

genai.configure(api_key=st.secrets["GEMINI_API_KEY"])
```

- Loads necessary libraries and configures the Gemini API with a secret key stored in Streamlit's secret management.

### 2. **Model Initialization**

```python
model = genai.GenerativeModel("models/text-bison-001")
```

- Initializes the Gemini model `text-bison-001`.

### 3. **Function: `call_gemini(prompt)`**

Handles all interaction with the model and returns generated text or error message.

### 4. **SDLC-Specific Functions**

Each SDLC function constructs a prompt and sends it to Gemini:

- `summarize_requirements(text)`

- `generate_python_code(req)`

- `review_code(code)`

- `generate_test_cases(desc)`

### 5. **Streamlit UI**

```python
st.set_page_config(...)
st.title(...)
tabs = st.tabs([...])
```

- Defines the layout and four functional tabs:
  - **Requirement Analysis**: Accepts a text file upload and summarizes it.
  - **Code Generation**: Generates Python code from natural language input.
  - **Code Review**: Reviews pasted code for bugs or improvements.
  - **Test Case Generation**: Creates test cases from feature descriptions.

[Additional pages in the document would explain each function and Streamlit UI component in more detail.]