**[8:09 PM, 4/7/2025] Ali Mehdi: Ok9pm**

**[8:55 PM, 4/7/2025] Ali Mehdi: meeting link sent**

**[9:00 PM, 4/7/2025] Ali Mehdi: Done**

**[9:52 PM, 4/7/2025] Ali Mehdi: https://www.youtube.com/live/83l01nAHG6E?si=s4w7VgnrKHMHRg7T**

**[9:52 PM, 4/7/2025] Ali Mehdi: following is useful as per Sir Saleh**

**https://github.com/panaversity/learn-agentic-ai/tree/main/01\_openai\_agents**

**[9:56 PM, 4/7/2025] Ali Mehdi: https://www.youtube.com/watch?v=e7qvd2bOITc&t=1611s**

**imp.docx**

**requirements.txt**

**v1\_basic\_agent.py**

**v2\_structured\_output.py**

**v3\_tool\_calls.py**

**v4\_handoffs.py**

**v5\_guardrails\_and\_context.py**

**v6\_streamlit\_agent.py**

**agents.py**

**requirements.txt**

**v1\_basic\_agent.py**

**v2\_structured\_output.py**

**v3\_tool\_calls.py**

**v4\_handoffs.py**

**v5\_guardrails\_and\_context.py**

**v6\_streamlit\_agent.py**

**==========================================================================**

**Openai Agentic SDK-Chatbot with openai LLM and streamlit UI**

**===========================================================================**

1. Helping video: <https://www.youtube.com/watch?v=e7qvd2bOITc>
2. Repo: <https://github.com/coleam00/ottomator-agents/tree/main/openai-sdk-agent>
3. Create a new folder travel\_agent and open it with cursor ide
4. Create a .env file with code

OPENAI\_API\_KEY=sk-proj-EqSBSlDEro90zzylHnezT3BlbkFJhZAs73Vi89CIiSNBpoRp

MODEL\_CHOICE=gpt-4o-mini

1. Copy requirements.txt file from repo step 2
2. Install virtual environment creation ability in python by running command in the terminal “pip install virtualenv”
3. Create a virtual environment by command “python -m venv myenv”
4. Activate it by “myenv\Scripts\activate”
5. To install dependencies from requirements.txt command = “pip install -r requirements.txt”
6. Download all files 6 files v1 to v6 from repo and place into travel\_agent folder.
7. Go through all the steps of each file one by one to understand the functionality of Openai-SDK.
8. Especially run and uderstand the last two files v5\_... and v6….
9. Here below is the detailed explanation of these two files by google gemini code assistant:

**==========================================================**

**Explanation of code in File v5\_guraduals\_and\_context.py**

**===========================================================**

Okay, I understand. You want a super-detailed, easy-to-understand explanation of the v5\_guardrails\_and\_context.py file, which is a complex example of using the OpenAI Agents SDK to build a travel planning assistant. I'll go through it line by line, explaining each part in simple terms and providing examples where necessary.

**File: v5\_guardrails\_and\_context.py**

This file builds a sophisticated travel planning assistant that can:

* Get weather forecasts.
* Search for flights and hotels.
* Create travel plans.
* Consider user preferences.
* Use "guardrails" to check if a user's budget is realistic.
* Hand off tasks to specialized agents.

Let's dive into the code:

**1. Imports:**

unfold\_lesspython

content\_copyaddcompare\_arrowsopen\_in\_full

import asyncio

import json

from datetime import datetime

from dataclasses import dataclass

from typing import List, Optional

from pydantic import BaseModel, Field

from agents import Agent, RunContextWrapper, Runner, function\_tool, ModelSettings, InputGuardrail, GuardrailFunctionOutput, InputGuardrailTripwireTriggered

from dotenv import load\_dotenv

import logfire

import os

* **import asyncio**: This is for running asynchronous code. Think of it as a way to do multiple things at the same time without waiting for each one to finish. This is useful for things like making API calls, which can take some time.
  + **Example:** Imagine you're ordering food online. You don't want to just stare at the screen waiting for the restaurant to confirm your order. Asynchronous code lets you continue browsing while the order is being processed in the background.
* **import json**: This is for working with JSON (JavaScript Object Notation) data, a common format for data exchange on the web.
  + **Example:** When you search for flights, the flight search API will likely return the results in JSON format.
* **from datetime import datetime**: This is for working with dates and times.
  + **Example:** You'll use this to store the start time of a user's session or to represent flight departure/arrival times.
* **from dataclasses import dataclass**: This is a decorator that makes it easy to create classes that primarily store data. It automatically generates methods like \_\_init\_\_, \_\_repr\_\_, etc.
  + **Example:** We'll use this to create the UserContext class, which will hold information about a user's preferences.
* **from typing import List, Optional**: These are for type hinting, which makes your code more readable and helps catch errors.
  + **List**: Indicates that a variable will hold a list of items.
    - **Example:** preferred\_airlines: List[str] means preferred\_airlines will be a list of strings (airline names).
  + **Optional**: Indicates that a variable might be None.
    - **Example:** max\_price: Optional[float] means max\_price can either be a floating-point number (a price) or None (if no maximum price is specified).
* **from pydantic import BaseModel, Field**: Pydantic is a library for data validation and settings management.
  + **BaseModel**: Used to create data models with type checking and validation.
  + **Field**: Used to add extra information to model fields, like descriptions.
* **from agents import Agent, RunContextWrapper, Runner, function\_tool, ModelSettings, InputGuardrail, GuardrailFunctionOutput, InputGuardrailTripwireTriggered**: These are the core components of the OpenAI Agents SDK.
  + **Agent**: Represents an AI agent that can perform tasks.
  + **RunContextWrapper**: A wrapper for passing context to tools.
  + **Runner**: Used to run agents and manage the interaction between them.
  + **function\_tool**: A decorator to turn a regular Python function into a tool that an agent can use.
  + **ModelSettings**: Used to configure the model settings.
  + **InputGuardrail**: Used to define a guardrail for the agent's input.
  + **GuardrailFunctionOutput**: Used to define the output of a guardrail function.
  + **InputGuardrailTripwireTriggered**: An exception that is raised when a guardrail is triggered.
* **from dotenv import load\_dotenv**: This is for loading environment variables from a .env file.
  + **Example:** You might store your OpenAI API key in a .env file and load it into your code using this.
* **import logfire**: This is for logging and tracing.
* **import os**: This is for interacting with the operating system, like getting environment variables.

**2. Environment Variables and Logfire:**

unfold\_lesspython

content\_copyaddcompare\_arrowsopen\_in\_full

*# Load environment variables*

load\_dotenv()

*# Comment these lines out if you don't want Logfire tracing*

logfire.configure(send\_to\_logfire='if-token-present')

logfire.instrument\_openai\_agents()

model = os.getenv('MODEL\_CHOICE', 'gpt-4o-mini')

* **load\_dotenv()**: Loads environment variables from a .env file (if it exists) into the os.environ dictionary.
* **logfire.configure(...)**: Configures the Logfire logging system.
* **logfire.instrument\_openai\_agents()**: Sets up Logfire to track the activity of the OpenAI agents.
* **model = os.getenv('MODEL\_CHOICE', 'gpt-4o-mini')**: Gets the MODEL\_CHOICE environment variable. If it's not set, it defaults to 'gpt-4o-mini'. This allows you to easily switch between different language models.

**3. Data Models (Pydantic):**

unfold\_lesspython

content\_copyaddcompare\_arrowsopen\_in\_full

class FlightRecommendation(BaseModel):

airline: str

departure\_time: str

arrival\_time: str

price: float

direct\_flight: bool

recommendation\_reason: str

class HotelRecommendation(BaseModel):

name: str

location: str

price\_per\_night: float

amenities: List[str]

recommendation\_reason: str

class TravelPlan(BaseModel):

destination: str

duration\_days: int

budget: float

activities: List[str] = Field(description="List of recommended activities")

notes: str = Field(description="Additional notes or recommendations")

class BudgetAnalysis(BaseModel):

is\_realistic: bool

reasoning: str

suggested\_budget: Optional[float] = None

* These are Pydantic models, which are like blueprints for data. They define the structure and types of data that will be used in the application.
* **FlightRecommendation**: Represents a flight recommendation.
  + **Example:**

unfold\_lessjson

content\_copyaddcompare\_arrowsopen\_in\_full

{

"airline": "SkyWays",

"departure\_time": "08:00",

"arrival\_time": "10:30",

"price": 350.00,

"direct\_flight": true,

"recommendation\_reason": "This is a direct flight and the price is reasonable."

}

* **HotelRecommendation**: Represents a hotel recommendation.
  + **Example:**

unfold\_lessjson

content\_copyaddcompare\_arrowsopen\_in\_full

{

"name": "City Center Hotel",

"location": "Downtown",

"price\_per\_night": 199.99,

"amenities": ["WiFi", "Pool", "Gym"],

"recommendation\_reason": "This hotel is in a great location and has the amenities you requested."

}

* **TravelPlan**: Represents a complete travel plan.
  + **Example:**

unfold\_lessjson

content\_copyaddcompare\_arrowsopen\_in\_full

{

"destination": "Miami",

"duration\_days": 5,

"budget": 2000.00,

"activities": ["Visit South Beach", "Explore Wynwood Walls", "Go to Everglades National Park"],

"notes": "Remember to bring sunscreen!"

}

* + **Field(description="...")**: This adds a description to the field, which can be helpful for the AI agent to understand what the field is for.
* **BudgetAnalysis**: Represents the result of analyzing a user's budget.
  + **Example:**

unfold\_lessjson

content\_copyaddcompare\_arrowsopen\_in\_full

{

"is\_realistic": false,

"reasoning": "Your budget of $300 is too low for a week in Dubai. You should consider increasing it.",

"suggested\_budget": 1500.00

}

**4. User Context (Dataclass):**

unfold\_lesspython

content\_copyaddcompare\_arrowsopen\_in\_full

@dataclass

class UserContext:

user\_id: str

preferred\_airlines: List[str] = None

hotel\_amenities: List[str] = None

budget\_level: str = None

session\_start: datetime = None

def \_\_post\_init\_\_(self):

if self.preferred\_airlines is None:

self.preferred\_airlines = []

if self.hotel\_amenities is None:

self.hotel\_amenities = []

if self.session\_start is None:

self.session\_start = datetime.now()

* **@dataclass**: This decorator automatically generates the \_\_init\_\_ method, \_\_repr\_\_, and other methods for this class.
* **UserContext**: This class stores information about a user's preferences and session.
  + **user\_id: str**: A unique ID for the user.
  + **preferred\_airlines: List[str] = None**: A list of the user's preferred airlines.
    - **Example:** ["SkyWays", "OceanAir"]
  + **hotel\_amenities: List[str] = None**: A list of the user's preferred hotel amenities.
    - **Example:** ["WiFi", "Pool", "Gym"]
  + **budget\_level: str = None**: The user's budget level (e.g., "budget", "mid-range", "luxury").
  + **session\_start: datetime = None**: The time when the user's session started.
* **\_\_post\_init\_\_(self)**: This method is called *after* the \_\_init\_\_ method. It's used to set default values.
  + If preferred\_airlines or hotel\_amenities is None, it sets them to an empty list ([]).
  + If session\_start is None, it sets it to the current time (datetime.now()).

**5. Tools (Function Tools):**

unfold\_lesspython

content\_copyaddcompare\_arrowsopen\_in\_full

@function\_tool

def get\_weather\_forecast(city: str, date: str) -> str:

"""Get the weather forecast for a city on a specific date."""

*# ... (weather data and logic) ...*

@function\_tool

async def search\_flights(wrapper: RunContextWrapper[UserContext], origin: str, destination: str, date: str) -> str:

"""Search for flights between two cities on a specific date, taking user preferences into account."""

*# ... (flight data and logic) ...*

@function\_tool

async def search\_hotels(wrapper: RunContextWrapper[UserContext], city: str, check\_in: str, check\_out: str, max\_price: Optional[float] = None) -> str:

"""Search for hotels in a city for specific dates within a price range, taking user preferences into account."""

*# ... (hotel data and logic) ...*

* **@function\_tool**: This decorator turns a regular Python function into a tool that an agent can use.
* **get\_weather\_forecast(city: str, date: str) -> str**: This tool gets the weather forecast for a given city and date.
  + It uses a hardcoded weather\_data dictionary for simplicity. In a real application, you would call a weather API here.
  + It returns a string describing the weather.
* **search\_flights(wrapper: RunContextWrapper[UserContext], origin: str, destination: str, date: str) -> str**: This tool searches for flights between two cities on a given date.
  + **wrapper: RunContextWrapper[UserContext]**: This is how you access the UserContext within a tool.
  + It uses a hardcoded flight\_options list for simplicity. In a real application, you would call a flight search API here.
  + It returns a JSON string representing the flight options.
  + **User Preferences:** It checks if the user has preferred\_airlines in their UserContext and, if so, it sorts the flight options to put the preferred airlines first.
* **search\_hotels(wrapper: RunContextWrapper[UserContext], city: str, check\_in: str, check\_out: str, max\_price: Optional[float] = None) -> str**: This tool searches for hotels in a city for given dates.
  + **max\_price: Optional[float] = None**: This parameter is optional. If provided, it filters the hotels to only include those with a price per night less than or equal to max\_price.
  + It uses a hardcoded hotel\_options list for simplicity. In a real application, you would call a hotel search API here.
  + It returns a JSON string representing the hotel options.
  + **User Preferences:** It checks if the user has preferred\_amenities and budget\_level in their UserContext and, if so, it sorts the hotel options to put the preferred hotels first.

**6. Guardrails:**

unfold\_lesspython

content\_copyaddcompare\_arrowsopen\_in\_full

budget\_analysis\_agent = Agent(

name="Budget Analyzer",

instructions="""

... (instructions for budget analysis) ...

""",

output\_type=BudgetAnalysis,

model=model

)

async def budget\_guardrail(ctx, agent, input\_data):

"""Check if the user's travel budget is realistic."""

*# ... (budget analysis logic) ...*

* **budget\_analysis\_agent**: This is a specialized agent that analyzes a user's budget to see if it's realistic.
  + It has its own instructions and output\_type (BudgetAnalysis).
* **budget\_guardrail(ctx, agent, input\_data)**: This is a guardrail function. It's called *before* the main travel\_agent processes the user's input.
  + **ctx**: The context of the agent run.
  + **agent**: The agent that is being run.
  + **input\_data**: The user's input.
  + **Logic:**
    1. It creates an analysis\_prompt based on the user's input.
    2. It runs the budget\_analysis\_agent with this prompt.
    3. It checks if the final\_output from the budget\_analysis\_agent indicates that the budget is not realistic (not final\_output.is\_realistic).
    4. If the budget is not realistic, it prints a message to the console.
    5. It returns a GuardrailFunctionOutput object.
       - **output\_info**: The BudgetAnalysis object.
       - **tripwire\_triggered**: True if the budget is not realistic, False otherwise.

**7. Specialized Agents:**

unfold\_lesspython

content\_copyaddcompare\_arrowsopen\_in\_full

flight\_agent = Agent[UserContext](

*# ... (flight agent definition) ...*

)

hotel\_agent = Agent[UserContext](

*# ... (hotel agent definition) ...*

)

conversational\_agent = Agent[UserContext](

*# ... (conversational agent definition) ...*

)

* These are specialized agents that handle specific tasks.
* **flight\_agent**: This agent is responsible for finding and recommending flights.
  + It uses the search\_flights tool.
  + It has instructions to guide its behavior.
  + It has an output\_type of FlightRecommendation.
* **hotel\_agent**: This agent is responsible for finding and recommending hotels.
  + It uses the search\_hotels tool.
  + It has instructions to guide its behavior.
  + It has an output\_type of HotelRecommendation.
* **conversational\_agent**: This agent is responsible for giving basic responses to the user to carry out a normal conversation.
  + It has instructions to guide its behavior.
  + It does not have any tools.

**8. Main Travel Agent:**

unfold\_lesspython

content\_copyaddcompare\_arrowsopen\_in\_full

travel\_agent = Agent[UserContext](

*# ... (travel agent definition) ...*

)

* **travel\_agent**: This is the main agent that interacts with the user.
  + It has instructions to guide its behavior.
  + It uses the get\_weather\_forecast tool.
  + It has handoffs to the flight\_agent, hotel\_agent, and conversational\_agent. This means it can delegate tasks to these specialized agents when needed.
  + It has an input\_guardrails list containing the budget\_guardrail. This means the budget\_guardrail will be run before the agent processes the user's input.
  + It has an output\_type of TravelPlan.

**9. Main Function:**

unfold\_lesspython

content\_copyaddcompare\_arrowsopen\_in\_full

async def main():

*# ... (user context and queries) ...*

for query in queries:

*# ... (run the agent and format the output) ...*

* **async def main()**: This is the main function that runs the application.
* **user\_context**: It creates a UserContext object with some example preferences.
* **queries**: It defines a list of example queries to test the system.
* **Loop:** It loops through the queries:
  + It runs the travel\_agent with the current query and user\_context.
  + It checks the type of the final\_output and formats the output accordingly (flight, hotel, travel plan, or generic response).
  + It prints the final response.
  + It handles the InputGuardrailTripwireTriggered exception.

**How It All Works Together:**

1. **User Input:** The user provides input (a query) to the travel\_agent.
2. **Guardrail Check:** The budget\_guardrail is run to check if the user's budget is realistic.
3. **Agent Processing:** The travel\_agent processes the user's input.
   * If the user asks about the weather, the travel\_agent uses the get\_weather\_forecast tool.
   * If the user asks about flights, the travel\_agent hands off to the flight\_agent.
   * If the user asks about hotels, the travel\_agent hands off to the hotel\_agent.
   * If the user asks a basic question, the travel\_agent hands off to the conversational\_agent.
   * Otherwise, the travel\_agent tries to create a TravelPlan.
4. **Specialized Agent Work:** If there's a handoff, the specialized agent uses its tools and instructions to generate a response.
5. **Response:** The final response is formatted and printed to the console.

**Key Concepts:**

* **Agents:** AI components that can perform tasks.
* **Tools:** Functions that agents can use to interact with the outside world or perform specific actions.
* **Handoffs:** The ability for one agent to delegate tasks to other specialized agents.
* **Guardrails:** Checks that are run before an agent processes input to ensure safety or quality.
* **Context:** Information that is passed to agents and tools to help them make better decisions.
* **Data Models:** Structures for organizing and validating data.

I hope this detailed explanation is helpful! Let me know if you have any more questions.

Rate this answer: thumb\_upthumb\_downHelp us improve

Context Sources (5)

e:\AI\Programs\openai\_agents\_sdk\travel\_agent\v5\_guardrails\_and\_context.pye:\AI\Programs\openai\_agents\_sdk\travel\_agent\v3\_tool\_calls.pye:\AI\Programs\openai\_agents\_sdk\travel\_agent\v4\_handoffs.pye:\AI\Programs\openai\_agents\_sdk\travel\_agent\v5\_guardrails\_and\_context.pye:\AI\Programs\openai\_agents\_sdk\travel\_agent\v6\_streamlit\_agent.py

ok

Gemini

Okay, I understand. You've provided me with four Python files (v3\_tool\_calls.py, v4\_handoffs.py, v5\_guardrails\_and\_context.py, and v6\_streamlit\_agent.py) and you want me to focus on explaining v5\_guardrails\_and\_context.py in detail, but with the context of the other files available if needed. You've also confirmed that you're ready for the explanation.

Great! Let's proceed with a deep dive into v5\_guardrails\_and\_context.py, keeping in mind the evolution of the code across the other files.

**Overall Goal of v5\_guardrails\_and\_context.py**

This file represents a significant step up in complexity compared to v3\_tool\_calls.py and v4\_handoffs.py. Its primary goals are to:

1. **Introduce User Context:** Store and use user preferences (like preferred airlines and hotel amenities) to personalize recommendations.
2. **Implement Guardrails:** Add a "budget guardrail" to check if a user's stated budget is realistic for their trip.
3. **Enhance Tools:** Improve the search\_flights and search\_hotels tools to consider user preferences.
4. **Add a Conversational Agent:** Add a basic conversational agent to handle basic questions.
5. **Maintain Handoffs:** Keep the handoff mechanism from v4\_handoffs.py to delegate tasks to specialized agents.
6. **Integrate with Streamlit:** This file is used by v6\_streamlit\_agent.py to create a web app.

**Detailed Breakdown of v5\_guardrails\_and\_context.py**

Let's go through the code section by section, explaining each part in detail.

**1. Imports:**

unfold\_lesspython

content\_copyaddcompare\_arrowsopen\_in\_full

import asyncio

import json

from datetime import datetime

from dataclasses import dataclass

from typing import List, Optional

from pydantic import BaseModel, Field

from agents import Agent, RunContextWrapper, Runner, function\_tool, ModelSettings, InputGuardrail, GuardrailFunctionOutput, InputGuardrailTripwireTriggered

from dotenv import load\_dotenv

import logfire

import os

* **import asyncio**: For asynchronous programming (running multiple tasks concurrently).
* **import json**: For working with JSON data (common for APIs).
* **from datetime import datetime**: For working with dates and times.
* **from dataclasses import dataclass**: For creating data classes (classes that primarily store data).
* **from typing import List, Optional**: For type hinting (improving code readability and maintainability).
* **from pydantic import BaseModel, Field**: For creating data models with validation.
* **from agents import ...**: Imports the core components of the OpenAI Agents SDK:
  + **Agent**: Represents an AI agent.
  + **RunContextWrapper**: A wrapper for passing context to tools.
  + **Runner**: For running agents.
  + **function\_tool**: A decorator to turn functions into tools.
  + **ModelSettings**: For configuring model settings.
  + **InputGuardrail**: For defining guardrails.
  + **GuardrailFunctionOutput**: For defining the output of a guardrail function.
  + **InputGuardrailTripwireTriggered**: An exception raised when a guardrail is triggered.
* **from dotenv import load\_dotenv**: For loading environment variables from a .env file.
* **import logfire**: For logging and tracing.
* **import os**: For interacting with the operating system.

**2. Environment Variables and Logfire:**

unfold\_lesspython

content\_copyaddcompare\_arrowsopen\_in\_full

*# Load environment variables*

load\_dotenv()

*# Comment these lines out if you don't want Logfire tracing*

logfire.configure(send\_to\_logfire='if-token-present')

logfire.instrument\_openai\_agents()

model = os.getenv('MODEL\_CHOICE', 'gpt-4o-mini')

* **load\_dotenv()**: Loads environment variables from a .env file.
* **logfire.configure(...)**: Configures Logfire for logging.
* **logfire.instrument\_openai\_agents()**: Sets up Logfire to track agent activity.
* **model = os.getenv('MODEL\_CHOICE', 'gpt-4o-mini')**: Gets the MODEL\_CHOICE environment variable, defaulting to 'gpt-4o-mini'.

**3. Data Models (Pydantic):**

unfold\_lesspython

content\_copyaddcompare\_arrowsopen\_in\_full

class FlightRecommendation(BaseModel):

airline: str

departure\_time: str

arrival\_time: str

price: float

direct\_flight: bool

recommendation\_reason: str

class HotelRecommendation(BaseModel):

name: str

location: str

price\_per\_night: float

amenities: List[str]

recommendation\_reason: str

class TravelPlan(BaseModel):

destination: str

duration\_days: int

budget: float

activities: List[str] = Field(description="List of recommended activities")

notes: str = Field(description="Additional notes or recommendations")

class BudgetAnalysis(BaseModel):

is\_realistic: bool

reasoning: str

suggested\_budget: Optional[float] = None

* These are Pydantic models, defining the structure of data.
* **FlightRecommendation**: Represents a flight recommendation.
* **HotelRecommendation**: Represents a hotel recommendation.
* **TravelPlan**: Represents a complete travel plan.
* **BudgetAnalysis**: Represents the result of a budget analysis.
  + **is\_realistic: bool**: Whether the budget is realistic.
  + **reasoning: str**: The reason for the budget analysis.
  + **suggested\_budget: Optional[float] = None**: A suggested budget if the original is not realistic.
* **Field(description="...")**: This adds a description to the field, which can be helpful for the AI agent to understand what the field is for.

**4. User Context (Dataclass):**

unfold\_lesspython

content\_copyaddcompare\_arrowsopen\_in\_full

@dataclass

class UserContext:

user\_id: str

preferred\_airlines: List[str] = None

hotel\_amenities: List[str] = None

budget\_level: str = None

session\_start: datetime = None

def \_\_post\_init\_\_(self):

if self.preferred\_airlines is None:

self.preferred\_airlines = []

if self.hotel\_amenities is None:

self.hotel\_amenities = []

if self.session\_start is None:

self.session\_start = datetime.now()

* **@dataclass**: This decorator simplifies the creation of this data-centric class.
* **UserContext**: This class stores user-specific information.
  + **user\_id: str**: A unique ID for the user.
  + **preferred\_airlines: List[str] = None**: A list of preferred airlines.
  + **hotel\_amenities: List[str] = None**: A list of preferred hotel amenities.
  + **budget\_level: str = None**: The user's budget level (e.g., "budget", "mid-range", "luxury").
  + **session\_start: datetime = None**: The start time of the user's session.
* **\_\_post\_init\_\_(self)**: This method is called *after* the \_\_init\_\_ method.
  + It sets preferred\_airlines and hotel\_amenities to empty lists ([]) if they are None.
  + It sets session\_start to the current time (datetime.now()) if it's None.

**5. Tools (Function Tools):**

unfold\_lesspython

content\_copyaddcompare\_arrowsopen\_in\_full

@function\_tool

def get\_weather\_forecast(city: str, date: str) -> str:

"""Get the weather forecast for a city on a specific date."""

*# ... (weather data and logic) ...*

@function\_tool

async def search\_flights(wrapper: RunContextWrapper[UserContext], origin: str, destination: str, date: str) -> str:

"""Search for flights between two cities on a specific date, taking user preferences into account."""

*# ... (flight data and logic) ...*

@function\_tool

async def search\_hotels(wrapper: RunContextWrapper[UserContext], city: str, check\_in: str, check\_out: str, max\_price: Optional[float] = None) -> str:

"""Search for hotels in a city for specific dates within a price range, taking user preferences into account."""

*# ... (hotel data and logic) ...*

* **@function\_tool**: This decorator turns functions into tools that agents can use.
* **get\_weather\_forecast(city: str, date: str) -> str**: This tool gets the weather forecast.
  + It's the same as in v4\_handoffs.py.
* **search\_flights(wrapper: RunContextWrapper[UserContext], origin: str, destination: str, date: str) -> str**: This tool searches for flights.
  + **wrapper: RunContextWrapper[UserContext]**: This is how the tool accesses the UserContext.
  + **User Preferences:** It now considers preferred\_airlines from the UserContext.
    - It sorts the flight\_options to put preferred airlines first.
    - It adds a "preferred": True flag to flights that match the user's preferences.
  + It returns a JSON string of flight options.
* **search\_hotels(wrapper: RunContextWrapper[UserContext], city: str, check\_in: str, check\_out: str, max\_price: Optional[float] = None) -> str**: This tool searches for hotels.
  + **wrapper: RunContextWrapper[UserContext]**: Accesses the UserContext.
  + **User Preferences:** It now considers preferred\_amenities and budget\_level from the UserContext.
    - It calculates a "preference score" for each hotel based on how many preferred amenities it has.
    - It sorts hotels by preference score (higher scores first).
    - It sorts hotels by price if the budget\_level is "budget" or "luxury".
  + It returns a JSON string of hotel options.

**6. Guardrails:**

unfold\_lesspython

content\_copyaddcompare\_arrowsopen\_in\_full

budget\_analysis\_agent = Agent(

name="Budget Analyzer",

instructions="""

... (instructions for budget analysis) ...

""",

output\_type=BudgetAnalysis,

model=model

)

async def budget\_guardrail(ctx, agent, input\_data):

"""Check if the user's travel budget is realistic."""

*# ... (budget analysis logic) ...*

* **budget\_analysis\_agent**: This is a specialized agent for budget analysis.
  + It has instructions to guide its behavior.
  + It has an output\_type of BudgetAnalysis.
* **budget\_guardrail(ctx, agent, input\_data)**: This is the guardrail function.
  + **ctx**: The context of the agent run.
  + **agent**: The agent being run.
  + **input\_data**: The user's input.
  + **Logic:**
    1. It creates an analysis\_prompt based on the user's input.
    2. It runs the budget\_analysis\_agent with this prompt.
    3. It checks if the final\_output from the budget\_analysis\_agent indicates that the budget is not realistic.
    4. If the budget is not realistic, it prints a message.
    5. It returns a GuardrailFunctionOutput object.
       - **output\_info**: The BudgetAnalysis object.
       - **tripwire\_triggered**: True if the budget is not realistic, False otherwise.
  + **Error Handling:** It has a try...except block to handle errors gracefully.

**7. Specialized Agents:**

unfold\_lesspython

content\_copyaddcompare\_arrowsopen\_in\_full

flight\_agent = Agent[UserContext](

*# ... (flight agent definition) ...*

)

hotel\_agent = Agent[UserContext](

*# ... (hotel agent definition) ...*

)

conversational\_agent = Agent[UserContext](

*# ... (conversational agent definition) ...*

)

* **flight\_agent**: The flight specialist agent.
  + It's similar to v4\_handoffs.py, but it now uses UserContext and the improved search\_flights tool.
* **hotel\_agent**: The hotel specialist agent.
  + It's similar to v4\_handoffs.py, but it now uses UserContext and the improved search\_hotels tool.
* **conversational\_agent**: This agent is responsible for giving basic responses to the user to carry out a normal conversation.
  + It has instructions to guide its behavior.
  + It does not have any tools.

**8. Main Travel Agent:**

unfold\_lesspython

content\_copyaddcompare\_arrowsopen\_in\_full

travel\_agent = Agent[UserContext](

*# ... (travel agent definition) ...*

)

* **travel\_agent**: The main travel planning agent.
  + It's similar to v4\_handoffs.py, but it now:
    - Uses UserContext.
    - Has the budget\_guardrail in its input\_guardrails.
    - Has the conversational\_agent in its handoffs.

**9. Main Function:**

unfold\_lesspython

content\_copyaddcompare\_arrowsopen\_in\_full

async def main():

*# ... (user context and queries) ...*

for query in queries:

*# ... (run the agent and format the output) ...*

* **async def main()**: The main function.
* **user\_context**: Creates a UserContext object with example preferences.
* **queries**: A list of example queries.
* **Loop:** It loops through the queries:
  + It runs the travel\_agent with the query and user\_context.
  + It formats the output based on the type of response (flight, hotel, travel plan, or generic).
  + It prints the output.
  + It handles the InputGuardrailTripwireTriggered exception.

**Key Improvements in v5\_guardrails\_and\_context.py:**

1. **User Context:** The UserContext class and the RunContextWrapper allow the agents and tools to access user preferences.
2. **Guardrails:** The budget\_guardrail adds a safety check to ensure the user's budget is realistic.
3. **Enhanced Tools:** The search\_flights and search\_hotels tools are improved to consider user preferences.
4. **Conversational Agent:** The conversational\_agent allows the agent to have a basic conversation with the user.
5. **Streamlit Integration:** This file is used by v6\_streamlit\_agent.py to create a web app.

**Relationship to Other Files:**

* **v3\_tool\_calls.py**: This file is a much simpler version, with only one agent and no handoffs or user context.
* **v4\_handoffs.py**: This file introduces handoffs to specialized agents, but still lacks user context and guardrails.
* **v6\_streamlit\_agent.py**: This file builds a Streamlit web application that uses the travel\_agent and UserContext from v5\_guardrails\_and\_context.py. It provides a user interface for interacting with the travel planning assistant.

**In Summary:**

v5\_guardrails\_and\_context.py is a significant step forward in building a more sophisticated and personalized travel planning assistant. It introduces user context, guardrails, and enhanced tools, making the system more robust and user-friendly. It also lays the groundwork for the Streamlit application in v6\_streamlit\_agent.py.

I hope this detailed explanation is helpful! Let me know if you have any further questions.