AI agent to mail Weather report

Email Output

Please find the daily Atlanta weather report attached as a PDF.

Weather Description:
In Atlanta, today's weather is primarily sunny with a high temperature of 54 degrees Fahrenheit (12.2 degrees Celsius). Winds are light to gentle, blowing from the southwest at around 8 miles per hour.

One attachment • Scanned by Gmail

PDF Output

Daily Atlanta Weather Report

Weather Data:

Clear sky, 22.3°C, 12.7 km/h

Weather Description:

Currently in Atlanta, Georgia, the weather is clear with a comfortable temperature of 22.3 degrees Celsius (approximately 72 degrees Fahrenheit). Wind speeds are moderate at around 12.7 kilometers per hour. Enjoy your day!

The **Antopic Model Context Protocol (MCP)** is a design pattern for building modular, scalable, and maintainable AI agents or applications. It is inspired by the **Model-View-Controller (MVC)** pattern but is tailored for AI and automation workflows. The MCP framework divides the application into three main components:

Model (Modules):

- Represents the core functionality of the application.
- Each module is responsible for a specific task (e.g., fetching data, generating reports, sending emails).
- Modules are independent and reusable.

Context:

- Acts as a shared state or environment.
- Holds configuration, data, and other resources required by the modules.
- Ensures loose coupling between modules.

Protocol:

- Defines the workflow or interaction between modules.
- Orchestrates the execution of tasks in a specific sequence.
- Acts as the "brain" of the application.

1. Context

The Context class acts as the shared state or environment for the application. It holds all the configuration and data required by the modules.

Key Points:

- **Configuration**: SMTP server details, email credentials, and recipient email.
- **Data**: Weather data, AI-generated description, and PDF filename.
- **Purpose**: Centralizes all shared data and configuration, making it accessible to all modules

2. Modules

The functionality is divided into modular components, each responsible for a specific task.
These modules interact with the Context to access shared data and configuration.

a. WeatherFetcher

Fetches weather data from an external API (wttr.in).

```
class WeatherFetcher:
    def fetch_weather(self, context):
        url = "https://wttr.in/Atlanta?format=%C+%t+%w"  # wttr.in provides simple weather data
        try:
            response = requests.get(url)
            response.raise_for_status()
            context.weather_data = response.text.strip()
            print("Weather data fetched successfully.")
        except Exception as e:
            print(f"Failed to fetch weather data: {e}")
            context.weather_data = None
```

Fetches weather data and stores it in context.weather_data.

Handles errors gracefully and logs them.

b. WeatherDescriber

Uses **Ollama** (with the Mistral model) to generate a brief weather description.

```
class WeatherDescriber:
    def generate_description(self, context):
        if not context.weather_data:
            print("No weather data available to generate description.")
            return

        # Use Ollama to generate a brief weather description
        prompt = f"Write a brief and professional description of the weather
in Atlanta based on this data: {context.weather_data}."
        try:
            # Specify the model name as "mistral"
            response = ollama.generate(model="mistral", prompt=prompt)
            context.weather_description = response['response']
            print("Weather description generated successfully.")
        except Exception as e:
            print(f"Failed to generate weather description: {e}")
            context.weather_description = None
```

Key Points:

Generates a weather description using the Mistral model.

Stores the description in context.weather_description.

Handles errors gracefully and logs them.

c. PDFGenerator

Generates a PDF report from the fetched weather data and AI-generated description.

```
class PDFGenerator:
    def generate_pdf(self, context):
        if not context.weather_data:
            print("No weather data available to generate PDF.")
            return

        pdf = FPDF()
        pdf.add_page()
        pdf.set_font("Arial", size=12)

        # Add a title
        pdf.cell(200, 10, txt="Daily Atlanta Weather Report", ln=True, align="C")

        # Clean the weather data by removing unsupported Unicode characters
        weather_data_cleaned = "".join(char if ord(char) < 128 else " " for char in context.weather_data)

        # Add weather data
        pdf.ln(10)
        pdf.multi_cell(0, 10, txt=f"Weather Data:\n{weather_data_cleaned}")

        # Add AI-generated weather description
        if context.weather_description:
            pdf.ln(10)
            pdf.multi_cell(0, 10, txt=f"Weather Description:\n{context.weather_description}")

        # Save the PDF
        pdf.output(context.pdf_filename)
        print(f"PDF created: {context.pdf_filename}")
```

Key Points:

Creates a PDF with the weather data and AI-generate description.

Cleans the weather data to remove unsupported Unic characters.

Saves the PDF with the filename specified in context.pdf_filename.

d. EmailSender

Sends an email with the generated PDF as an attachment.

```
class EmailSender:
    def send_email(self, context, subject, body):
        if not context.weather_data:
            print("No weather data available to send email.")
            return

        # Create the email
        msg = MIMEMultipart()
        msg['From'] = context.EMAIL_ADDRESS
        msg['To'] = context.RECIPIENT_EMAIL
        msg['Subject'] = subject

        # Attach the body of the email
        msg.attach(MIMEText(body, 'plain'))

        # Attach the PDF file
        with open(context.pdf_filename, "rb") as attachment:
            part = MIMEBase("application", "octet-stream")
            part.set_payload(attachment.read())
            encoders.encode_base64(part)
            part.add_header(
                "Content-Disposition",
                f"attachment; filename= {context.pdf_filename}",
            )
            msg.attach(part)

        # Connect to the SMTP server and send the email
        try:
            server = smtplib.SMTP(context.SMTP_SERVER, context.SMTP_PORT)
            server.starttls()  # Upgrade the connection to secure
            server.login(context.EMAIL_ADDRESS, context.EMAIL_PASSWORD)
            server.sendmail(context.EMAIL_ADDRESS, context.RECIPIENT_EMAIL, msg.as_string())
            print("Email sent successfully!")
        except Exception as e:
            print(f"Failed to send email: {e}")
        finally:
            server.quit()
```

Key Points:

Composes and sends an email with the PDF attachment.

Uses the SMTP server and credentials from the Context.

Handles errors gracefully and logs them.

3. Protocol

The WeatherReportAgent class orchestrates the interaction between the modules and the Context. It defines the workflow for the application.

```
class WeatherReportAgent:
    def __init__(self):
        self.context = Context()
        self.weather_fetcher = WeatherFetcher()
        self.weather_describer = WeatherDescriber()  # Add AI module
        self.pdf_generator = PDFGenerator()
        self.email_sender = EmailSender()

    def run(self):
        # Step 1: Fetch weather data
        self.weather_fetcher.fetch_weather(self.context)

        # Step 2: Generate weather description using AI
        self.weather_describer.generate_description(self.context)

        # Step 3: Generate PDF
        self.pdf_generator.generate_pdf(self.context)

        # Step 4: Send email with PDF attachment
        email_subject = "Daily Atlanta Weather Report"
        email_body = "Please find the daily Atlanta weather report attached as a PDF.\n\n"
        if self.context.weather_description:
            email_body += f"Weather Description:\n{self.context.weather_description}"
        self.email_sender.send_email(self.context, email_subject, email_body)
```

Key Points:

Initializes the Context and all modules.

Defines the workflow:

Fetch weather data.

Generate weather description using AI.

Generate PDF.

Send email with PDF attachment.

Thank You