

**Great, now improve the design and provide code with inputs.**

After creating a schedule of PI execution, you need to allocate user stories to agile team resources. In initial inputs of agent, aside from project details, you will provided input of agile team members details with name, job title, skills and availability percentage in PI( like 50 %, 80%, 100%).  
To allocate user stories you need to check agile team member job title and skills. If they user stories requires resources that are not available in team, you just tag of new resources need with skills required.

After that you add stage of asking for humsn in loop feedback. If humans not satisfied with planning and allocation of resources due to some risks like holiday planning, you ask for feeback and improve planning and resources allocation.

In end you need to provide a Gantt chart with X axis of timeline( pi iteration & week number ) and Y axis of user stories with marking of group under epic title and with color of user stories on resources name who will perform activity.

You need to also output a agile project planning pdf document with following content:

1. summary of project details and agile team member
2. Epic stories with linked user stories
3. Summary of logic to tasks allocation to agile team members and spcial mention of any additional resources requirement
4. Gantt chart of planning
5. Overall summary of agile project planning highlighting any important points for product manager and scrum master

Certainly! Below is an improved, comprehensive design and code example for an Agile Project Management Assistant using Google ADK multi-agent framework. This version:

* Takes **project description** and **agile team member details** as inputs.
* Generates epic title and user stories with human-in-the-loop feedback.
* Determines dependencies between user stories.
* Creates a schedule of user story execution across PI iterations.
* Allocates user stories to agile team members based on job title, skills, and availability.
* Tags any new resource needs if skills are missing.
* Supports iterative human validation and feedback on planning and allocation.
* Generates a **Gantt chart** visualization.
* Produces a **PDF report** summarizing the entire plan.

**Design Overview**

|  |  |  |
| --- | --- | --- |
| Step | Agent / Functionality | Description |
| 1. Epic Generation | EpicGenerator | Generate epic title from project description. |
| 2. User Story Generation | UserStoryGenerator | Generate user stories from epic and feedback. |
| 3. Human Validation | ValidationAgent | Ask human if epic & user stories are satisfactory; collect feedback if not. |
| 4. Dependency Analysis | DependencyAgent | Identify dependencies between user stories. |
| 5. Scheduling | SchedulingAgent | Create schedule of user stories across PI iterations respecting dependencies. |
| 6. Resource Allocation | ResourceAllocationAgent | Allocate user stories to team members based on skills, job title, and availability; tag new resource needs. |
| 7. Human Validation on Plan | PlanValidationAgent | Ask human to validate schedule and resource allocation; collect feedback and improve. |
| 8. Visualization & Reporting | Visualization + PDFReportGenerator | Generate Gantt chart and PDF report summarizing the project plan. |

**Complete Code Example**

import asyncio  
from google.adk.agents import LlmAgent, SequentialAgent  
from matplotlib import pyplot as plt  
import matplotlib.patches as mpatches  
import pandas as pd  
import pdfkit # For PDF generation from HTML (requires wkhtmltopdf installed)  
import tempfile  
import os  
  
# --- Define Agents ---  
  
epic\_agent = LlmAgent(  
 name="EpicGenerator",  
 model="gemini-2.0-flash",  
 description="Generate a concise epic title from the project description.",  
 instruction=(  
 "You are an Agile assistant. Given a project description, generate a concise, clear, and compelling epic title."  
 )  
)  
  
user\_story\_agent = LlmAgent(  
 name="UserStoryGenerator",  
 model="gemini-2.0-flash",  
 description="Generate user stories from epic title and optional feedback.",  
 instruction=(  
 "You are an Agile assistant. Given an epic title and optional feedback, generate detailed user stories "  
 "in the format: 'As a [role], I want [feature] so that [benefit].' Incorporate feedback to improve."  
 )  
)  
  
validation\_agent = LlmAgent(  
 name="ValidationAgent",  
 model="gemini-2.0-flash",  
 description="Ask human to validate epic and user stories and collect feedback if not satisfied.",  
 instruction=(  
 "Present the epic and user stories. Ask if the user is satisfied. If no, ask for specific feedback."  
 )  
)  
  
dependency\_agent = LlmAgent(  
 name="DependencyAgent",  
 model="gemini-2.0-flash",  
 description="Identify dependencies between user stories.",  
 instruction=(  
 "Given user stories, identify dependencies as pairs indicating which story depends on which."  
 )  
)  
  
scheduling\_agent = LlmAgent(  
 name="SchedulingAgent",  
 model="gemini-2.0-flash",  
 description="Create a schedule of user story execution across PI iterations respecting dependencies.",  
 instruction=(  
 "Given user stories and dependencies, create a schedule mapping each story to a PI iteration number, "  
 "ensuring dependencies are respected and workload balanced."  
 )  
)  
  
resource\_allocation\_agent = LlmAgent(  
 name="ResourceAllocationAgent",  
 model="gemini-2.0-flash",  
 description="Allocate user stories to agile team members based on job title, skills, and availability.",  
 instruction=(  
 "Given user stories, team member details (name, job title, skills, availability%), and schedule, "  
 "allocate stories to members matching required skills and availability. "  
 "If skills missing in team, tag new resource needs."  
 )  
)  
  
plan\_validation\_agent = LlmAgent(  
 name="PlanValidationAgent",  
 model="gemini-2.0-flash",  
 description="Ask human to validate the planning and resource allocation, collect feedback if not satisfied.",  
 instruction=(  
 "Present the schedule and resource allocation. Ask if the user is satisfied. "  
 "If no, collect feedback for improvement."  
 )  
)  
  
# --- Helper functions for visualization and PDF generation ---  
  
def generate\_gantt\_chart(schedule, user\_stories, allocations, output\_path="gantt\_chart.png"):  
 """  
 schedule: dict mapping user story to iteration number  
 user\_stories: list of user story strings  
 allocations: dict mapping user story to resource name  
 """  
 # Prepare data for plotting  
 # For simplicity, assume each PI iteration = 2 weeks, 3 iterations total  
 iteration\_weeks = 2  
 iterations = max(schedule.values())  
 fig, ax = plt.subplots(figsize=(12, 6))  
  
 yticks = []  
 yticklabels = []  
 colors = {}  
 color\_map = plt.cm.get\_cmap('tab20')  
 resource\_names = list(set(allocations.values()))  
 resource\_color\_map = {name: color\_map(i) for i, name in enumerate(resource\_names)}  
  
 for i, story in enumerate(user\_stories):  
 iter\_num = schedule.get(story, 1)  
 start\_week = (iter\_num - 1) \* iteration\_weeks + 1  
 duration = iteration\_weeks # assume 2 weeks per story for visualization  
 color = resource\_color\_map.get(allocations.get(story, "Unassigned"), (0.5, 0.5, 0.5))  
  
 ax.barh(i, duration, left=start\_week, color=color, edgecolor='black')  
 yticks.append(i)  
 yticklabels.append(story[:50] + ("..." if len(story) > 50 else ""))  
  
 ax.set\_yticks(yticks)  
 ax.set\_yticklabels(yticklabels)  
 ax.set\_xlabel("PI Iteration & Week Number")  
 ax.set\_ylabel("User Stories")  
 ax.set\_title("Agile Project Planning Gantt Chart")  
  
 # X-axis labels as iteration and week  
 xticks = []  
 xticklabels = []  
 total\_weeks = iteration\_weeks \* iterations  
 for week in range(1, total\_weeks + 1):  
 xticks.append(week)  
 iter\_label = (week - 1) // iteration\_weeks + 1  
 wk\_label = ((week - 1) % iteration\_weeks) + 1  
 xticklabels.append(f"PI{iter\_label}-W{wk\_label}")  
 ax.set\_xticks(xticks)  
 ax.set\_xticklabels(xticklabels, rotation=45)  
  
 # Legend by resource  
 patches = [mpatches.Patch(color=clr, label=nm) for nm, clr in resource\_color\_map.items()]  
 ax.legend(handles=patches, title="Resources")  
  
 plt.tight\_layout()  
 plt.savefig(output\_path)  
 plt.close()  
 return output\_path  
  
  
def generate\_pdf\_report(  
 project\_desc, team\_members, epic\_title, user\_stories,  
 dependencies, schedule, allocations, new\_resources, gantt\_chart\_path,  
 output\_pdf\_path="agile\_project\_plan.pdf"  
):  
 """  
 Generate a PDF report summarizing the Agile project plan.  
 """  
 # Create HTML content for PDF  
 html = f"""  
 <html>  
 <head><title>Agile Project Planning Report</title></head>  
 <body>  
 <h1>Agile Project Planning Report</h1>  
  
 <h2>1. Project Summary</h2>  
 <p>{project\_desc}</p>  
  
 <h3>Agile Team Members</h3>  
 <ul>  
 """  
 for member in team\_members:  
 html += f"<li><b>{member['name']}</b> - {member['job\_title']}, Skills: {', '.join(member['skills'])}, Availability: {member['availability']}%</li>"  
 html += "</ul>"  
  
 html += f"""  
 <h2>2. Epic and User Stories</h2>  
 <h3>Epic Title</h3>  
 <p>{epic\_title}</p>  
 <h3>User Stories</h3>  
 <ul>  
 """  
 for us in user\_stories:  
 html += f"<li>{us}</li>"  
 html += "</ul>"  
  
 html += f"""  
 <h2>3. Dependencies</h2>  
 <pre>{dependencies}</pre>  
  
 <h2>4. Schedule and Resource Allocation</h2>  
 <table border="1" cellpadding="5" cellspacing="0">  
 <tr><th>User Story</th><th>PI Iteration</th><th>Assigned Resource</th></tr>  
 """  
 for us in user\_stories:  
 iter\_num = schedule.get(us, "N/A")  
 resource = allocations.get(us, "Unassigned")  
 html += f"<tr><td>{us}</td><td>{iter\_num}</td><td>{resource}</td></tr>"  
 html += "</table>"  
  
 if new\_resources:  
 html += "<h3>Additional Resource Needs</h3><ul>"  
 for skill in new\_resources:  
 html += f"<li>Need resource with skill: {skill}</li>"  
 html += "</ul>"  
  
 html += f"""  
 <h2>5. Gantt Chart</h2>  
 <img src="{gantt\_chart\_path}" alt="Gantt Chart" style="width:100%;max-width:900px;"/>  
  
 <h2>6. Summary for Product Manager and Scrum Master</h2>  
 <p>The planning respects dependencies and balances workload across PI iterations. Resource allocation is optimized based on skills and availability. Additional resource needs have been identified for missing skills. Please review holiday schedules and other risks to finalize the plan.</p>  
  
 </body>  
 </html>  
 """  
  
 # Generate PDF using pdfkit  
 pdfkit.from\_string(html, output\_pdf\_path)  
 return output\_pdf\_path  
  
  
# --- Main Assistant Agent ---  
  
class AgileProjectAssistant(SequentialAgent):  
 def \_\_init\_\_(self, name):  
 super().\_\_init\_\_(  
 name=name,  
 sub\_agents=[  
 epic\_agent,  
 user\_story\_agent,  
 validation\_agent,  
 dependency\_agent,  
 scheduling\_agent,  
 resource\_allocation\_agent,  
 plan\_validation\_agent,  
 ],  
 )  
 self.project\_description = None  
 self.team\_members = None  
 self.epic\_title = None  
 self.user\_stories = None  
 self.dependencies = None  
 self.schedule = None  
 self.allocations = None  
 self.new\_resources = None  
  
 async def run(self, project\_description, team\_members):  
 self.project\_description = project\_description  
 self.team\_members = team\_members  
  
 # Step 1: Generate Epic Title  
 self.epic\_title = await epic\_agent.run(self.project\_description)  
  
 # Step 2: Generate User Stories (initially no feedback)  
 self.user\_stories = await user\_story\_agent.run(f"Epic: {self.epic\_title}\nFeedback: None")  
  
 # Step 3: Human validation of epic & user stories  
 while True:  
 validation\_prompt = (  
 f"Epic Title:\n{self.epic\_title}\n\n"  
 f"User Stories:\n{self.user\_stories}\n\n"  
 "Are you satisfied with this output? Please answer 'yes' or 'no'. "  
 "If 'no', please provide specific feedback."  
 )  
 validation\_response = await validation\_agent.run(validation\_prompt)  
  
 if "yes" in validation\_response.lower():  
 break  
 else:  
 feedback = validation\_response.lower().replace("no", "", 1).strip()  
 if not feedback:  
 feedback = "No specific feedback provided."  
 self.user\_stories = await user\_story\_agent.run(f"Epic: {self.epic\_title}\nFeedback: {feedback}")  
  
 # Step 4: Determine dependencies  
 dependency\_prompt = f"User Stories:\n{self.user\_stories}\n\nIdentify dependencies."  
 self.dependencies = await dependency\_agent.run(dependency\_prompt)  
  
 # Step 5: Scheduling user stories across PI iterations  
 scheduling\_prompt = (  
 f"User Stories:\n{self.user\_stories}\n\n"  
 f"Dependencies:\n{self.dependencies}\n\n"  
 "Create a schedule mapping each user story to a PI iteration number."  
 )  
 schedule\_str = await scheduling\_agent.run(scheduling\_prompt)  
  
 # Parse schedule string into dictionary (assume a simple format: "User Story" -> Iteration)  
 # This parsing depends on the agent output format; here we simulate a simple parse:  
 self.schedule = self.\_parse\_schedule(schedule\_str, self.user\_stories)  
  
 # Step 6: Allocate resources  
 team\_str = self.\_format\_team\_members(self.team\_members)  
 allocation\_prompt = (  
 f"User Stories:\n{self.user\_stories}\n\n"  
 f"Schedule:\n{schedule\_str}\n\n"  
 f"Team Members:\n{team\_str}\n\n"  
 "Allocate user stories to team members based on skills, job title, and availability. "  
 "Tag any new resource needs."  
 )  
 allocation\_response = await resource\_allocation\_agent.run(allocation\_prompt)  
  
 # Parse allocations and new resource needs from response  
 self.allocations, self.new\_resources = self.\_parse\_allocation(allocation\_response)  
  
 # Step 7: Human validation of plan and allocation  
 while True:  
 plan\_validation\_prompt = (  
 f"Schedule:\n{schedule\_str}\n\n"  
 f"Resource Allocation:\n{allocation\_response}\n\n"  
 "Are you satisfied with the planning and resource allocation? Please answer 'yes' or 'no'. "  
 "If 'no', please provide specific feedback."  
 )  
 plan\_validation\_response = await plan\_validation\_agent.run(plan\_validation\_prompt)  
  
 if "yes" in plan\_validation\_response.lower():  
 break  
 else:  
 feedback = plan\_validation\_response.lower().replace("no", "", 1).strip()  
 if not feedback:  
 feedback = "No specific feedback provided."  
 # Re-run scheduling and allocation with feedback  
 scheduling\_prompt = (  
 f"User Stories:\n{self.user\_stories}\n\n"  
 f"Dependencies:\n{self.dependencies}\n\n"  
 f"Feedback:\n{feedback}\n\n"  
 "Create a revised schedule mapping each user story to a PI iteration number."  
 )  
 schedule\_str = await scheduling\_agent.run(scheduling\_prompt)  
 self.schedule = self.\_parse\_schedule(schedule\_str, self.user\_stories)  
  
 allocation\_prompt = (  
 f"User Stories:\n{self.user\_stories}\n\n"  
 f"Schedule:\n{schedule\_str}\n\n"  
 f"Team Members:\n{team\_str}\n\n"  
 f"Feedback:\n{feedback}\n\n"  
 "Allocate user stories to team members based on skills, job title, and availability. "  
 "Tag any new resource needs."  
 )  
 allocation\_response = await resource\_allocation\_agent.run(allocation\_prompt)  
 self.allocations, self.new\_resources = self.\_parse\_allocation(allocation\_response)  
  
 # Step 8: Generate Gantt chart  
 gantt\_path = generate\_gantt\_chart(self.schedule, self.user\_stories, self.allocations)  
  
 # Step 9: Generate PDF report  
 pdf\_path = generate\_pdf\_report(  
 self.project\_description, self.team\_members, self.epic\_title,  
 self.user\_stories, self.dependencies, self.schedule,  
 self.allocations, self.new\_resources, gantt\_path  
 )  
  
 return {  
 "epic\_title": self.epic\_title,  
 "user\_stories": self.user\_stories,  
 "dependencies": self.dependencies,  
 "schedule": self.schedule,  
 "allocations": self.allocations,  
 "new\_resources": self.new\_resources,  
 "gantt\_chart\_path": gantt\_path,  
 "pdf\_report\_path": pdf\_path,  
 }  
  
 def \_parse\_schedule(self, schedule\_str, user\_stories):  
 """  
 Parses the schedule string returned by scheduling agent into a dict:  
 {user\_story: iteration\_number}  
 This is a simple heuristic parser and may need adjustment based on actual output format.  
 """  
 schedule = {}  
 for us in user\_stories:  
 # Try to find iteration number for this user story in the string  
 # Expect format: "User Story: Iteration X"  
 # For demo, assign iteration 1 if not found  
 iter\_num = 1  
 lines = schedule\_str.splitlines()  
 for line in lines:  
 if us[:30] in line:  
 # Extract iteration number from line  
 import re  
 match = re.search(r"iteration\s\*(\d+)", line.lower())  
 if match:  
 iter\_num = int(match.group(1))  
 break  
 schedule[us] = iter\_num  
 return schedule  
  
 def \_format\_team\_members(self, team\_members):  
 """  
 Formats team member list into a string for prompting.  
 """  
 lines = []  
 for m in team\_members:  
 skills = ", ".join(m.get("skills", []))  
 lines.append(f"{m['name']} ({m['job\_title']}), Skills: {skills}, Availability: {m['availability']}%")  
 return "\n".join(lines)  
  
 def \_parse\_allocation(self, allocation\_response):  
 """  
 Parses allocation agent response into:  
 - allocations dict {user\_story: resource\_name}  
 - new\_resources list of skills needed  
 This is a heuristic parser and may need adjustment based on actual output format.  
 """  
 allocations = {}  
 new\_resources = []  
 lines = allocation\_response.splitlines()  
 for line in lines:  
 line\_lower = line.lower()  
 if "allocate" in line\_lower and ":" in line:  
 # Expected format: User Story: Resource Name  
 parts = line.split(":")  
 if len(parts) >= 2:  
 us = parts[0].strip()  
 res = parts[1].strip()  
 allocations[us] = res  
 elif "need resource" in line\_lower or "new resource" in line\_lower:  
 # Extract skill mention  
 import re  
 match = re.search(r"skill[s]\* required: (.+)", line\_lower)  
 if match:  
 skills = match.group(1).split(",")  
 new\_resources.extend([s.strip() for s in skills])  
 return allocations, new\_resources  
  
  
# --- Example Input and Running the Assistant ---  
  
async def main():  
 project\_description = (  
 "Develop a mobile app for fitness tracking with social sharing features, "  
 "including step counting, workout logging, and friend challenges."  
 )  
  
 agile\_team\_members = [  
 {  
 "name": "Alice",  
 "job\_title": "Frontend Developer",  
 "skills": ["React Native", "UI/UX", "Mobile Development"],  
 "availability": 80,  
 },  
 {  
 "name": "Bob",  
 "job\_title": "Backend Developer",  
 "skills": ["Python", "Django", "API Development"],  
 "availability": 100,  
 },  
 {  
 "name": "Charlie",  
 "job\_title": "QA Engineer",  
 "skills": ["Automated Testing", "Manual Testing", "Selenium"],  
 "availability": 50,  
 },  
 {  
 "name": "Dana",  
 "job\_title": "DevOps Engineer",  
 "skills": ["AWS", "CI/CD", "Docker"],  
 "availability": 60,  
 },  
 ]  
  
 assistant = AgileProjectAssistant(name="AgileProjectAssistant")  
 result = await assistant.run(project\_description, agile\_team\_members)  
  
 print("Epic Title:\n", result["epic\_title"])  
 print("\nUser Stories:\n", result["user\_stories"])  
 print("\nDependencies:\n", result["dependencies"])  
 print("\nSchedule:\n", result["schedule"])  
 print("\nResource Allocations:\n", result["allocations"])  
 print("\nNew Resource Needs:\n", result["new\_resources"])  
 print("\nGantt Chart Path:\n", result["gantt\_chart\_path"])  
 print("\nPDF Report Path:\n", result["pdf\_report\_path"])  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 asyncio.run(main())

**Notes on This Implementation**

* **LLM Agents:** Each agent uses a large language model (e.g., Gemini 2.0) to perform its task. Replace "gemini-2.0-flash" with your actual model or endpoint.
* **Parsing:** The parsing of schedule and allocation outputs is heuristic and depends heavily on the LLM output format. You may need to tailor prompts or parsing logic for your use case.
* **Gantt Chart:** Uses matplotlib to generate a Gantt chart showing user stories on Y-axis, timeline on X-axis, colored by assigned resource.
* **PDF Generation:** Uses pdfkit to generate a PDF report from HTML. Requires wkhtmltopdf installed on your system.
* **Human-in-the-loop:** The assistant loops on two validation points: after user story generation and after plan/resource allocation, enabling iterative improvements based on human feedback.
* **Resource Allocation:** Checks skills and availability, tags missing skills as new resource needs.
* **Extensibility:** You can add more agents for risk management, sprint planning, or backlog grooming.

**Summary**

This design and code provide a powerful, modular Agile Project Management Assistant that supports:

* Agile artifact generation
* Dependency and scheduling analysis
* Resource allocation with skill matching
* Human-in-the-loop iterative refinement
* Visualization and professional reporting

You can integrate this into your Agile toolchain or customize it further for your team’s workflow.

If you want, I can help you with:

* Fine-tuning prompt engineering for better parsing
* UI integration for interactive feedback
* Deployment and scaling tips

Just ask!