```
import time
from typing import List, Optional

from pydantic import BaseModel


class AgentSchema(BaseModel):
    name: str = None
    system_prompt: str = None
    task: str = None
    response: str = None


class JambaSwarmRequest(BaseModel):
    task: str = (None,)
    plan: str = None
    agents: List[AgentSchema] = None
    timestamp: int = int(time.time())


class JambaSwarmResponse(BaseModel):
    task: str = (None,)
    plan: str = None
    agents: List[AgentSchema] = None
    timestamp: int = int(time.time())
    response: str = None
```

```
class AgentSchema(BaseModel):
    name: Optional[str] = None
    system_prompt: Optional[str] = None
    task: Optional[str] = None
    response: Optional[str] = None


class DirectorSettings(BaseModel):
    name: str
    strategy: str
    objectives: List[str]


class BossSettings(BaseModel):
    name: str
    decision_making_strategy: str
    recruitment_strategy: str


class TaskDistribution(BaseModel):
    task: str
    assigned_agents: List[str]
```

```
class JambaSwarmRequest(BaseModel):
    task: Optional[str] = None
    plan: Optional[str] = None
    agents: Optional[List[AgentSchema]] = None
    director_settings: DirectorSettings
    boss_settings: BossSettings
    task_distribution: Optional[List[TaskDistribution]] = None
    timestamp: int = int(time.time())



class JambaSwarmResponse(BaseModel):
    task: Optional[str] = None
    plan: Optional[str] = None
    agents: Optional[List[AgentSchema]] = None
    response: Optional[str] = None
    timestamp: int = int(time.time())



# Sample usage:



# try:
#     request = JambaSwarmRequest(
#         task="Research on AI",
#         plan="Execute a comprehensive research plan",
#         agents=[
```

```
#                 AgentSchema(name="Agent1", system_prompt="Analyze recent AI papers", task="AI research task"),
#                 AgentSchema(name="Agent2", system_prompt="Summarize AI research findings", task="Summarization task"),
#         ],
#                         director_settings=DirectorSettings(name="Director1",  strategy="Hierarchical", objectives=["Efficiency", "Accuracy"]),
#             boss_settings=BossSettings(name="Boss1",  decision_making_strategy="Collaborative", recruitment_strategy="Pre-selected"),
#         task_distribution=[
#             TaskDistribution(task="Research on AI", assigned_agents=["Agent1", "Agent2"])
#         ]
#     )
#     print(request.json())
# except ValidationError as e:
#     print(e.json())
```