Bank support

Small but complete example of using PydanticAI to build a support agent for a bank.

Demonstrates:

- dynamic system prompt
- structured `result_type`
- tools

Running the Example

With dependencies installed and environment variables set, run:

pip

```
python -m pydantic_ai_examples.bank_support
```

uv

```
uv run -m pydantic_ai_examples.bank_support
```

(or `PYDANTIC_AI_MODEL=gemini-1.5-flash ...`)

Example Code

bank_support.py

```python
from dataclasses import dataclass

from pydantic import BaseModel, Field

from pydantic_ai import Agent, RunContext


class DatabaseConn:
    """This is a fake database for example purposes.

    In reality, you'd be connecting to an external database
    (e.g. PostgreSQL) to get information about customers.
    """

    @classmethod
    async def customer_name(cls, *, id: int) -> str | None:
        if id == 123:
            return 'John'

    @classmethod
    async def customer_balance(cls, *, id: int, include_pending: bool) -> float:
        if id == 123:
            return 123.45
        else:
            raise ValueError('Customer not found')


@dataclass
class SupportDependencies:
    customer_id: int
    db: DatabaseConn


class SupportResult(BaseModel):
    support_advice: str = Field(description='Advice returned to the customer')
    block_card: bool = Field(description='Whether to block their')
    risk: int = Field(description='Risk level of query', ge=0, le=10)


support_agent = Agent(
    'openai:gpt-4o',
    deps_type=SupportDependencies,
    result_type=SupportResult,
    system_prompt=(
        'You are a support agent in our bank, give the '
        'customer support and judge the risk level of their query. '
        "Reply using the customer's name."
    ),
)


@support_agent.system_prompt
async def add_customer_name(ctx: RunContext[SupportDependencies]) -> str:
    customer_name = await ctx.deps.db.customer_name(id=ctx.deps.customer_id)
    return f"The customer's name is {customer_name!r}"


@support_agent.tool
async def customer_balance(
    ctx: RunContext[SupportDependencies], include_pending: bool
) -> str:
    """Returns the customer's current account balance."""
    balance = await ctx.deps.db.customer_balance(
        id=ctx.deps.customer_id,
        include_pending=include_pending,
    )
    return f'${balance:.2f}'


deps = SupportDependencies(customer_id=123, db=DatabaseConn())
result = support_agent.run_sync('What is my balance?', deps=deps)
print(result.data)
"""
support_advice='Hello John, your current account balance, including pending transactions, is $123.45.' block_card=False risk=1
"""

result = support_agent.run_sync('I just lost my card!', deps=deps)
print(result.data)
```

```
"""
support_advice="I'm sorry to hear that, John. We are temporarily blocking your card to prevent unauthorized transactions." block_card=True risk=8
"""
```