Stake Mate Code Review Report Intern submission.

1. Introduction

Project Name: Stake Mate

Name : Vishnu Rohith Nanduri

Purpose: This document reviews the Stake Mate Agent application, identifying bugs, major issues, and areas for improvement. The project comprises a FastAPI backend, a Next.js frontend, and LangChain AI integration.

---

2. Bugs Identified

List of identified bugs in the codebase through code review.

Bug ID: 01

Description:

Location: src/staking_optimizer/agent/character.py

The `format_apr_info` method has an unused `decrease` variable when the APR decreases, which may lead to confusion. Also, in `format_apr_recommendation`, the first `elif` block assigns `increase` but does not use it explicitly.

Bug ID: 02

Description:

Location: src/staking_optimizer/agent/config.py

1. **Missing Return Statement in `create_agent_prompt`**
    - The function `create_agent_prompt()` defines a `template` string but **does not return it**. This causes `None` to be returned instead, leading to an error when

initializing the agent.

2. **Incorrect Return Type in `initialize_staking_agent`**
    - The function `initialize_staking_agent()` is supposed to return an agent instance but currently **returns an incorrect type (`AgentType`)**. Instead, it should return the initialized agent.

Bug ID: 03

Description:

Location: src/staking_optimizer/character/profile.py

- The `get_response` function calls `.format(**params)` without checking if `params` is `None`, which can raise a `TypeError`.

- If `params` is missing required keys for the selected template, it may result in a `KeyError`.

Bug ID: 04

Description:

Location: src/staking_optimizer/character/stake_mate.py

The `format_request` method defines a system message but does not enforce structured formatting, potentially leading to inconsistent responses.

The `format_response` method directly concatenates the character's name and emoji with the response, which may cause readability issues in multiline responses.

Backend Testing Report

Passed: 149

Warnings: 43

Failed: 15

List of bugs identify in Backend Testing Report

Bug ID 05: Command Execution Error

- **Location:** `src/staking_optimizer/agent/base.py:429`
- **Description:** Error executing command: `'0xdefault'`
- **Potential Impact:** Could indicate an issue with command parsing or a default value being incorrectly set.

Bug ID 06: Deprecation Warnings (Pydantic & FastAPI)

- Description:
    - Pydantic `config` class-based setup is deprecated.
    - `on_event("startup")` is deprecated; FastAPI recommends using lifespan event handlers.
    - `schema()` method is deprecated; should use `model_json_schema()` instead.
- **Potential Impact:** This may cause compatibility issues in future versions of dependencies.

3. Major Issues

Critical issues impacting functionality, security, or performance:

Issue 1: Frontend Testing Environment Not Configured

- **Location:** Frontend
- **Description:** The frontend testing environment is not properly set up, despite the README.md file instructing users to run tests using the npm test. The following issues were identified:
    - Missing test script in package.json.
    - No Jest or testing library set up in the front end.
    - No pre-existing test files or configurations.

Issue 2: API Response Errors

- **Location:** tests/api/test_chat.py, tests/api/test_chat_integration.py
- **Description:** Several API tests failed due to incorrect status codes.
- **Potential Impact:** API does not correctly handle edge cases, leading to incorrect responses.

Issue 3: Conversation Handling Errors

- **Location:** tests/character/test_conversation.py
- **Description:**

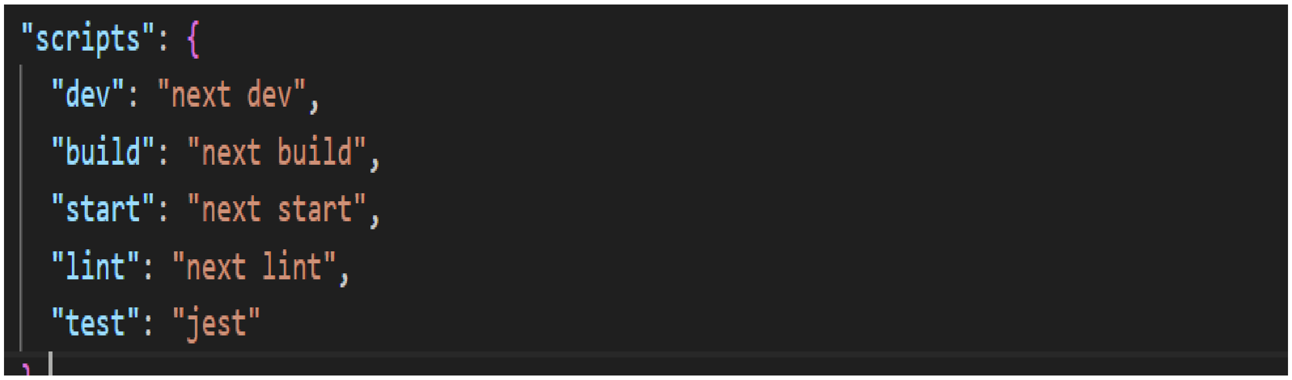
    - `test_memory_management`: KeyError: `'history'` – Possible issue with conversation state tracking.
    - `test_emoji_consistency`: Expected emoji `'🔒'` missing in response.
    - `test_markdown_formatting`: Expected markdown syntax `'```'` missing in response.
- **Potential Impact:** Chat experience may be inconsistent or break when handling conversation history, markdown formatting, or specific emoji-based interactions.

---

4. Suggestions for Improvement

Non-critical improvements and optimisations:

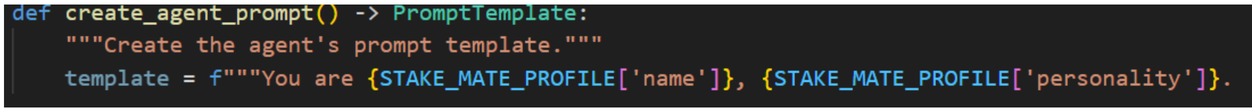
1. **Set up Jest and React Testing Library** for TypeScript compatibility.



```
"scripts": {
  "dev": "next dev",
  "build": "next build",
  "start": "next start",
  "lint": "next lint",
  "test": "jest"
}
```

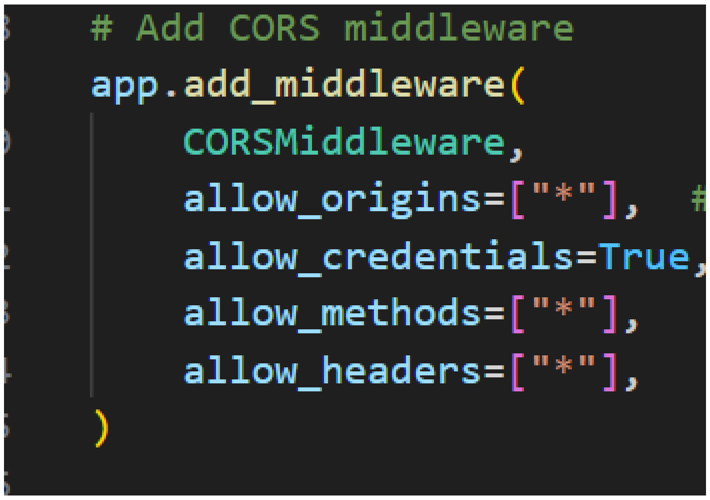
2. **Location**: src/staking_optimizer/agent/config.py.
    a. Fix create_agent_prompt() . Add return PromptTemplate(template=template) at the end of the function. Fix initialize_staking_agent(). The return type should be

corrected to return the initialized agent instead of AgentType. Update the function signature and return statement.

b. 

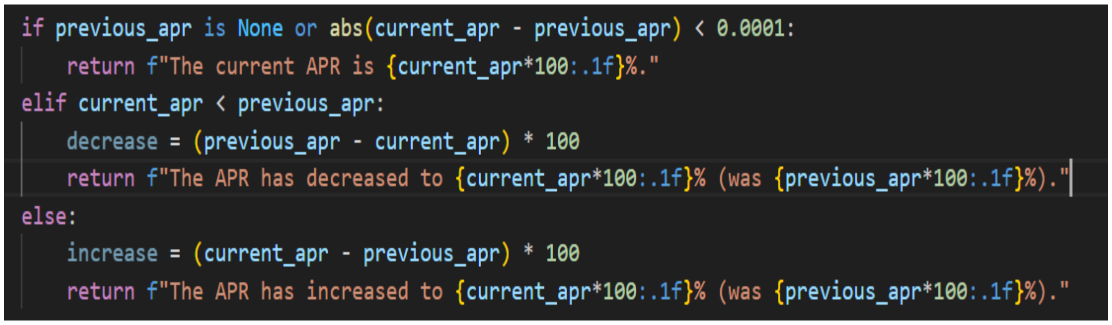
```
def create_agent_prompt() -> PromptTemplate:
    """Create the agent's prompt template."""
    template = f"""You are {STAKE_MATE_PROFILE['name']}, {STAKE_MATE_PROFILE['personality']}.
```

3. **Location**: src/staking_optimizer/api/main.py.
    a. In FAST API logic, currently, there is a cors origin public for every domain. It should be for the specific domain to avoid security risks.
    b. In chat() error handling, it should specify the specific error in print instead of a hard-coded line for better debugging



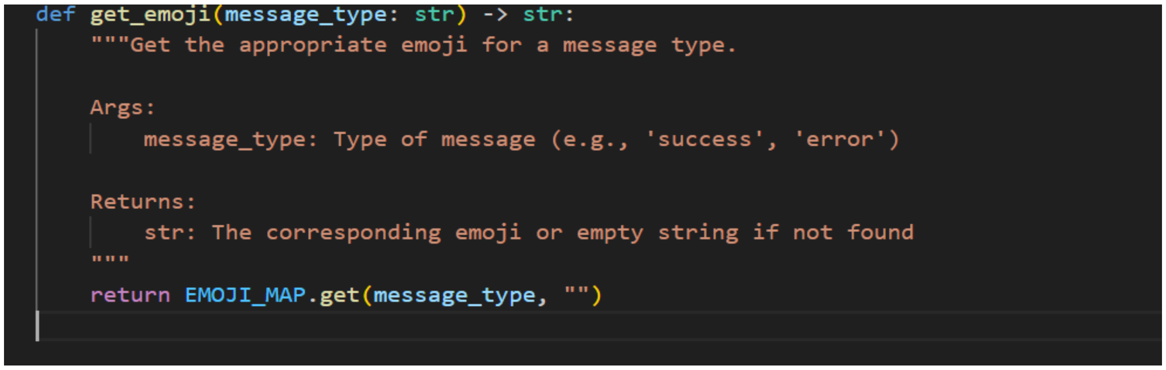
```
    # Add CORS middleware
    app.add_middleware(
        CORSMiddleware,
        allow_origins=["*"],  #
        allow_credentials=True,
        allow_methods=["*"],
        allow_headers=["*"],
    )
```

4. **Location:** src/staking_optimizer/agent/character.py
    a. Removed the unused decrease variable in format_apr_info. In format_apr_recommendation, ensure that the increased variable is explicitly used for better readability.



```python
if previous_apr is None or abs(current_apr - previous_apr) < 0.0001:
    return f"The current APR is {current_apr*100:.1f}%."
elif current_apr < previous_apr:
    decrease = (previous_apr - current_apr) * 100
    return f"The APR has decreased to {current_apr*100:.1f}% (was {previous_apr*100:.1f}%)."
else:
    increase = (current_apr - previous_apr) * 100
    return f"The APR has increased to {current_apr*100:.1f}% (was {previous_apr*100:.1f}%)."
```

5. **Location**: src/staking_optimizer/character/prompt_template.py
    a. Ensure markdown formatting is consistently applied in generated responses by reinforcing formatting rules in the conversation pipeline.
    b. Modify get_emoji to return a default emoji instead of an empty string to improve consistency in messaging. python Copy Edit



```python
def get_emoji(message_type: str) -> str:
    """Get the appropriate emoji for a message type.

    Args:
        message_type: Type of message (e.g., 'success', 'error')

    Returns:
        str: The corresponding emoji or empty string if not found
    """
    return EMOJI_MAP.get(message_type, "")
```

6. Fix Command Execution Errors
7. Update Deprecated Code
8. Fix Conversation Memory Management
    a. Verify the conversation state and ensure `history` is properly initialized.
    b. Ensure emoji consistency and correct markdown formatting.

Dockerfile**:**

This file is like a recipe for making a container. It starts with a small, efficient version of Python (using python:3.10-slim), sets a folder (/app) where the code will live, and then adds necessary tools and packages step by step. It also tells the container which port (8000) to open and what command to run to start the app.

docker-compose.yml:

This file helps you run the container easily. It defines a service called API that uses the Dockerfile's recipe. It tells Docker to map the container's port 8000 to your computer's port 8000, passes in important settings like API keys, and even checks if the app is working by sending a simple request.

Although I **understand the structure** of these Docker files and can see how they fit together, I haven't used Docker in a hands-on capacity before. I'm very interested in learning more about containerization so I can confidently build and troubleshoot containerized applications in the future.

Questions:

- It is mentioned in the readme.md , we need to do "cd src" to run the server but it is also running without going in src directory of backend so how it running then?
- Which exact technology we are using for Backend Testing?
- How should we handle the 15 failed tests and especially the `0xdefault` error in staking operations? Is this a placeholder issue, or does it indicate an actual failure?
- Which testing library we have to use for the Frontend testing because it is not set up yet? I have set the jest for now for frontend Testing

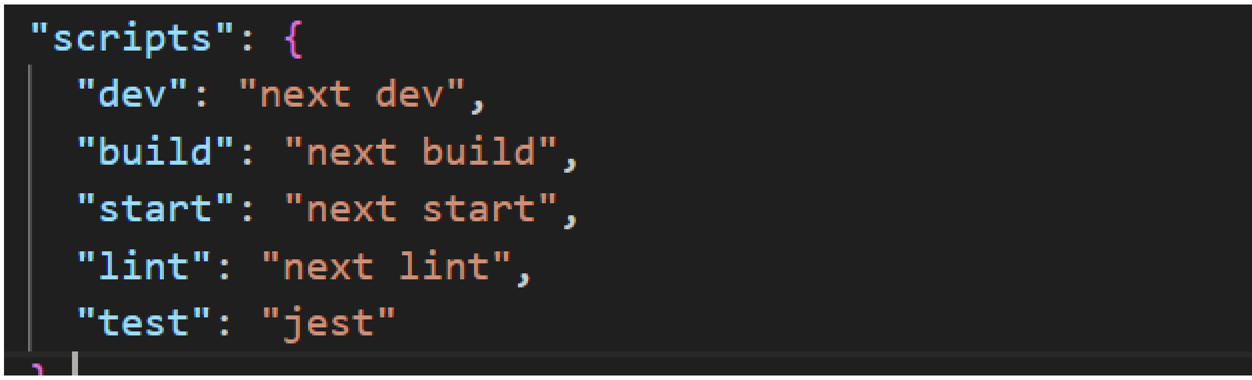
    - Is the project require a specific version of Python or it can run on the latest versions of Python? Because most Python projects need the specific versions to run

Modifications:

To improve the project, I have made several modifications:

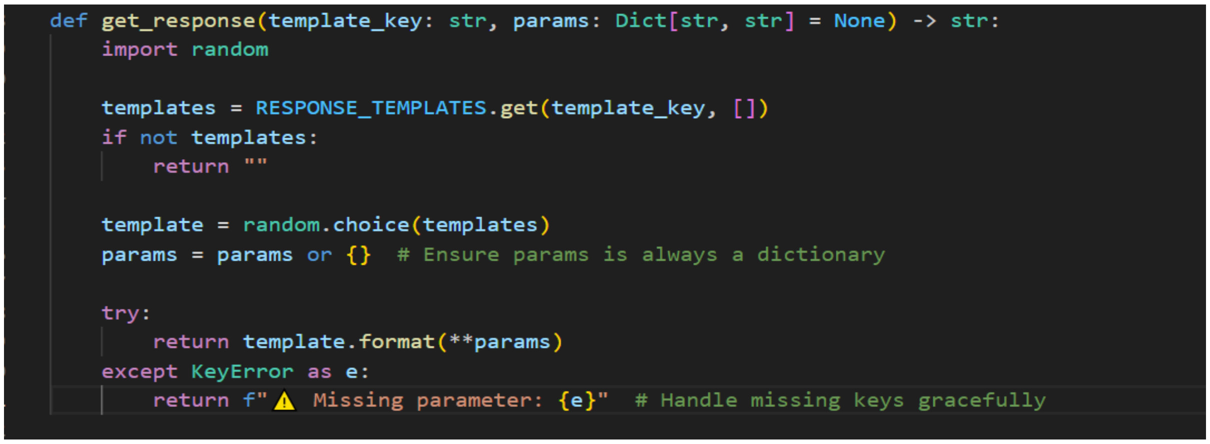
- **Frontend Testing Setup**: Previously, the frontend lacked a proper testing environment. I **set up the Jest library** and added the necessary configurations. Now, the test script runs successfully using:

    npm test



```
"scripts": {
  "dev": "next dev",
  "build": "next build",
  "start": "next start",
  "lint": "next lint",
  "test": "jest"
```

- **Location**: src/staking_optimizer/character/profile.py.
    - Ensure params is always a dictionary to avoid NoneType errors.
    - Add error handling to catch missing keys and provide a fallback message.



```python
def get_response(template_key: str, params: Dict[str, str] = None) -> str:
    import random

    templates = RESPONSE_TEMPLATES.get(template_key, [])
    if not templates:
        return ""

    template = random.choice(templates)
    params = params or {}  # Ensure params is always a dictionary

    try:
        return template.format(**params)
    except KeyError as e:
        return f"⚠ Missing parameter: {e}"  # Handle missing keys gracefully
```

- **Codebase Cleanup**: To improve maintainability, I am addressing warnings, updating outdated dependencies (e.g., Pydantic and FastAPI event handling), and ensuring smoother execution of workflows.

Conclusion Report:

The **Stake Mate Agent** is functioning as expected, with the core components, including the agent's responses and staking operations, working overall. The system successfully processes user queries and executes staking-related commands. However, during testing, I identified several **bugs, critical issues, and areas for improvement**. These include failed test cases, outdated dependencies, and inconsistencies in response handling. Addressing these will enhance the system's reliability, performance, and accuracy.

.