Functional Requirement

Functional requirements describe the specific functions, features, and capabilities that a system must possess. These requirements are concerned with what the system should do and the services it should provide. Functional requirements are typically task-oriented and can be defined in a way that they are testable. They answer questions like "What does the system do?" They can be broken down further into various subcategories:

1. \*\*User Authentication and Configuration:\*\*

- The application must allow users to configure their Steam account information, including username, password, and API key.

- Users should be able to specify additional settings, such as SSL certificate verification and local acceleration options.

- The application should provide an option for users to enable or disable specific plugins, such as BuffAutoAcceptOffer, BuffAutoOnSale, UUAutoAcceptOffer, and SteamAutoAcceptOffer.

2. \*\*Session Management:\*\*

- The application should support Steam session management, allowing users to cache their login session for future use.

- Users must have the ability to login to their Steam account and save the session information for subsequent launches.

- The session should be saved in a file specific to the Steam username and must be automatically loaded on application startup.

3. \*\*Plugin Integration:\*\*

- The application should support multiple plugins, including BuffAutoAcceptOffer, BuffAutoOnSale, UUAutoAcceptOffer, and SteamAutoAcceptOffer.

- Users should be able to enable and configure these plugins through the application's configuration file.

- Plugins must have the ability to interact with the Steam client and perform specific tasks, such as auto-accepting offers or managing items on sale.

4. \*\*Plugin Initialization and Execution:\*\*

- Plugins must be initialized properly during application startup, and they should be able to access the Steam client instance.

- Plugins should execute their tasks concurrently and independently, without causing conflicts or blocking the main application's functionality.

- The application should handle the exit code of each plugin and ensure a graceful exit when all plugins have completed their tasks.

5. \*\*Error Handling and Logging:\*\*

- The application must have a robust error-handling mechanism to catch and log exceptions.

- Detailed log files must be generated for debugging and issue reporting purposes.

- Users should receive informative error messages in the application's console for any issues that occur during execution.

6. \*\*Update Checking:\*\*

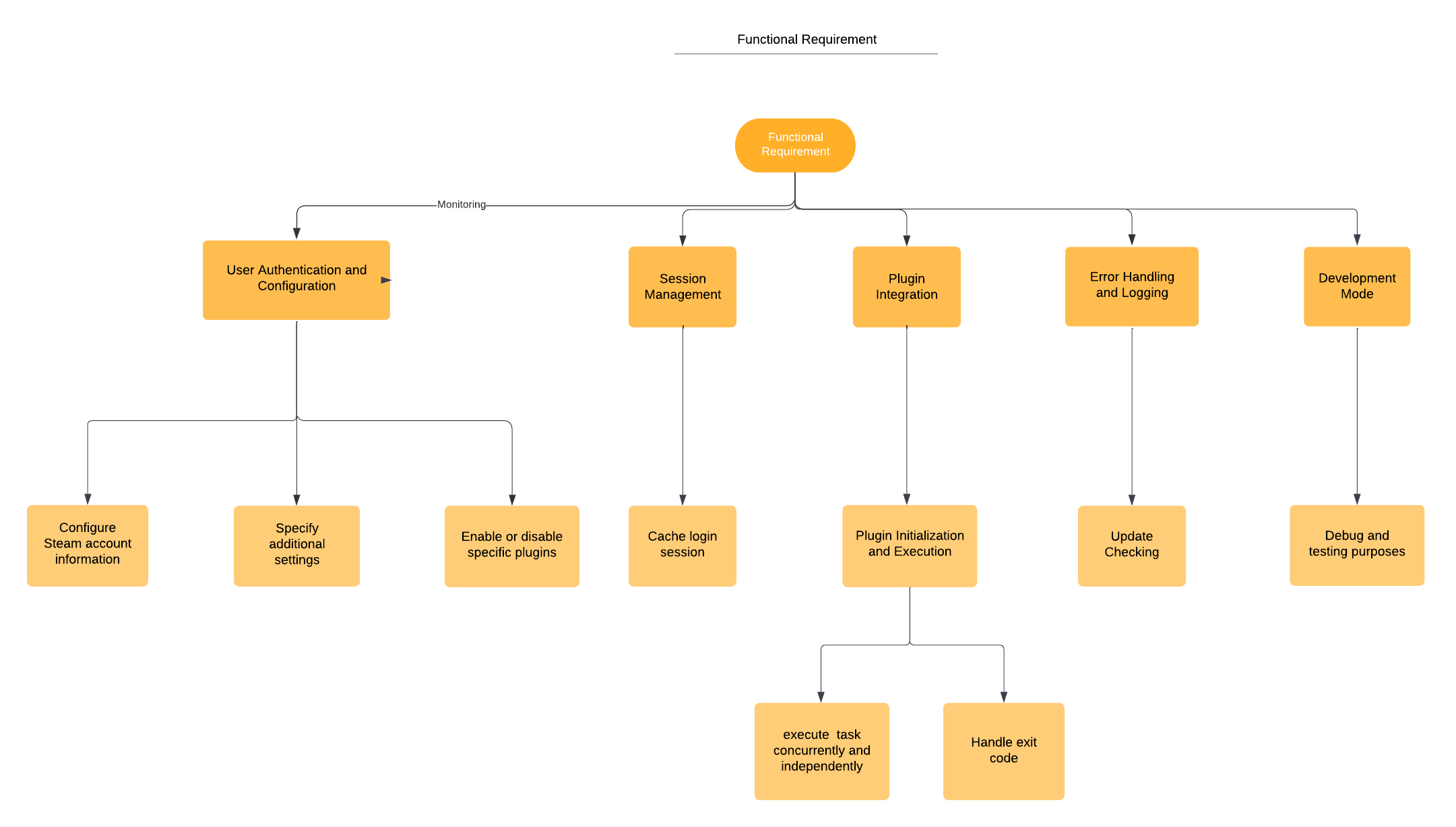
- The application should periodically check for updates by fetching version information from a remote server.

- Users should be notified if a new version is available, and they should have the option to update the application.

7. \*\*Development Mode:\*\*

- Users can enable a development mode for debugging and testing purposes.

- In development mode, additional debugging information should be provided in logs.



Non-Functional Requirement

Non-functional requirements define the quality attributes or constraints that a system must meet. These requirements are not directly related to the specific functions of the system but are essential for ensuring that the system performs adequately and satisfies various stakeholder needs. Non-functional requirements are often categorized as follows:

1. \*\*Usability:\*\*

- The application must have a user-friendly and well-documented configuration file, with clear explanations for each setting.

- It should provide user prompts and informative messages to guide users through the setup process.

- The application should be easy to set up and use for both technical and non-technical users.

2. \*\*Performance:\*\*

- The application should not significantly impact the system's performance while running in the background.

- Plugins should execute efficiently to prevent high CPU and memory usage.

- The use of multiple threads for concurrent plugin execution should be optimized to minimize resource consumption.

3. \*\*Security:\*\*

- The application must handle user credentials and session information securely, ensuring they are not exposed to unauthorized users.

- SSL certificate verification should be an option to enhance security but can be disabled if necessary.

- The application should not pose a security risk, even when running in development mode.

4. \*\*Reliability:\*\*

- The application should be reliable and able to recover gracefully from unexpected errors or exceptions.

- The use of session management should ensure that users do not need to frequently re-enter their credentials.

- Plugins should be well-tested and stable to avoid crashes or failures.

5. \*\*Scalability:\*\*

- The application should support the addition of new plugins in the future, making it extensible and adaptable to changing requirements.

- Plugins should be designed in a way that allows easy integration without major code changes.

6. \*\*Maintainability:\*\*

- The application's codebase should be well-structured and maintainable.

- Developers should be able to understand and extend the code without excessive effort.

- The application should adhere to coding best practices and follow a consistent coding style.

7. \*\*Update Mechanism:\*\*

- The application's update mechanism should be secure and protect users from downloading malicious updates.

- It should ensure that users can trust the source of updates and validate the integrity of the application.

8. \*\*Documentation:\*\*

- Comprehensive documentation should be provided to guide users on how to configure, use, and troubleshoot the application.

- Documentation should include detailed plugin integration instructions and explanations of each configuration option.

9. \*\*Availability:\*\*

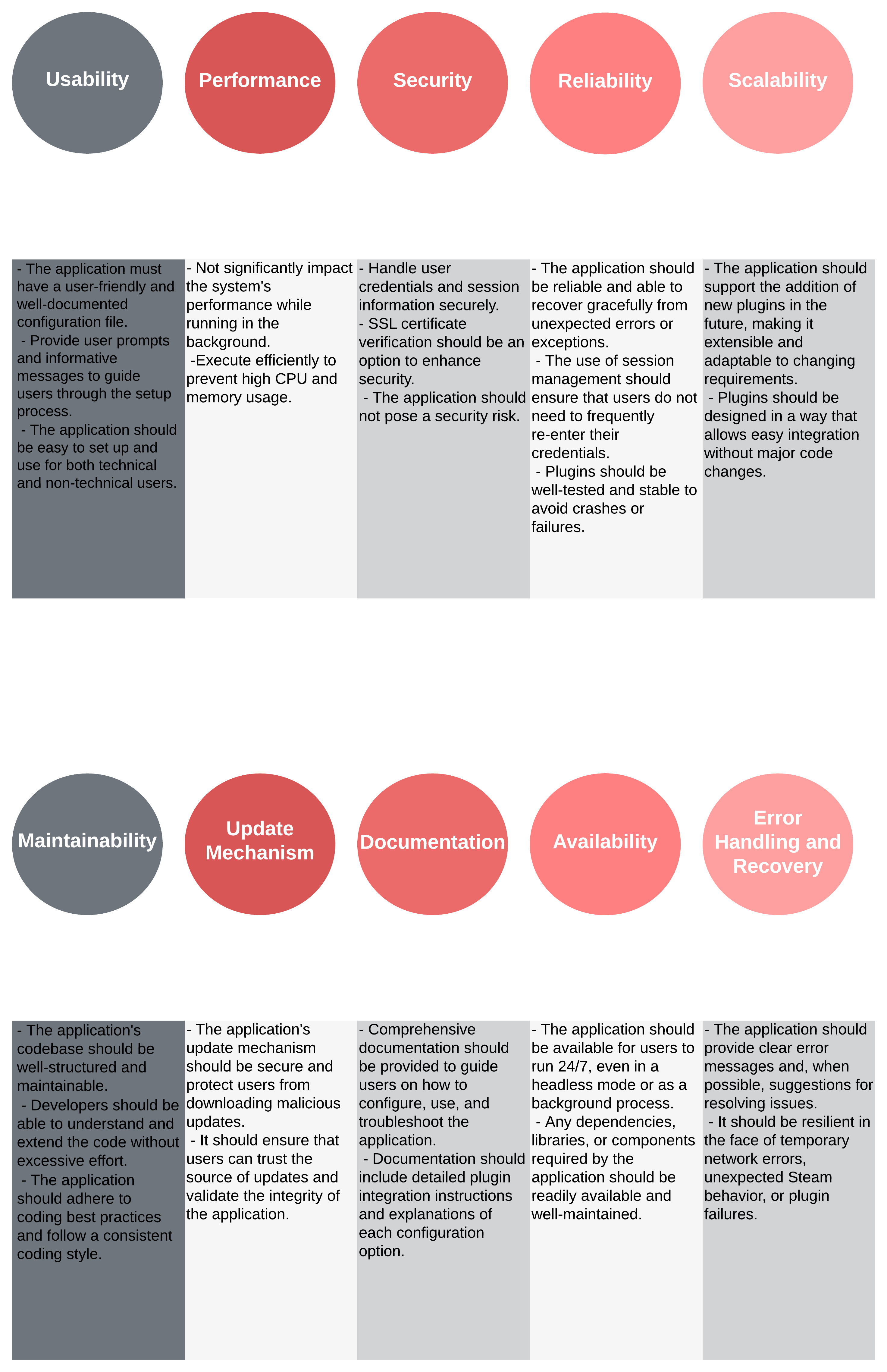
- The application should be available for users to run 24/7, even in a headless mode or as a background process.

- Any dependencies, libraries, or components required by the application should be readily available and well-maintained.

10. \*\*Error Handling and Recovery:\*\*

- The application should provide clear error messages and, when possible, suggestions for resolving issues.

- It should be resilient in the face of temporary network errors, unexpected Steam behavior, or plugin failures.



In summary, the functional and non-functional requirements of the given Python script, which appears to be a Steam trading automation tool, cover aspects related to user configuration, session management, plugin integration, error handling, update checking, development mode, usability, performance, security, reliability, scalability, maintainability, update mechanisms, documentation, availability, and error handling and recovery. These requirements ensure that the application is user-friendly, efficient, secure, reliable, and well-documented.