## Technology Stack and Framework

This section delves into the technology stack and frameworks that power the Discord bot, focusing on the tools and technologies that facilitate rapid development, seamless user interaction, and efficient data management.

### Programming Languages and Frameworks

**Python**:

* **Role**: Primary programming language for developing the bot.
* **Features**: Chosen for its readability, robust standard library, and extensive support through third-party libraries, Python underpins all major functionalities of the bot, from data scraping to process automation and interaction handling.

**Selenium**:

* **Role**: Automates web browsers to extract real-time product prices and availability.
* **Capabilities**: Simulates human interactions with web pages, allowing the bot to perform complex navigations and data extraction tasks, critical for accurate price monitoring.

**Discord.py**:

* **Role**: Handles communications with the Discord API.
* **Functionality**: Manages user interactions, receives commands, sends notifications, and embeds the bot seamlessly within Discord communities.

### Tools and Platforms

**Visual Studio Code**:

* **Role**: Preferred IDE for writing, testing, and debugging the bot’s code.
* **Advantages**: Offers extensive plugin support, built-in Git control, and integrated terminal, which streamline the coding and version control processes.

**Git**:

* **Role**: Manages source code versions and collaborates features.
* **Benefits**: Essential for tracking code changes, managing branches, and integrating changes from multiple contributors, ensuring consistency and continuity in the development process.

**GitHub**:

* **Role**: Hosts the source code repository and facilitates collaborative features like issue tracking and code reviews.
* **Integration**: Centralizes source control and acts as a platform for continuous integration and deployment strategies.

### Data Management and Storage

In lieu of a conventional database, our project utilizes a combination of configuration files, JSON, and direct file output mechanisms for managing both transient and persistent data:

**Configuration Files**:

* **Role**: Securely manage operational parameters and sensitive credentials, such as API keys and SMTP settings.
* **Implementation**: Stored in .env files, these parameters are loaded dynamically into the application environment, enhancing security by segregating configuration from the code.

**JSON Files**:

* **Role**: Handle transient data like session states and user preferences.
* **Advantages**: Offers flexibility and speed in accessing and updating data, ideal for non-sensitive, temporary information.

**Excel and HTML**:

* **Role**: Serve as formats for logging long-term data and generating reports.
* **Functionality**: Facilitates easy distribution and accessibility of data, allowing for comprehensive reporting and analysis through automated emails.

### Testing Strategy

Our project employs a robust testing framework using Python’s unittest library and unittest.mock for mocking external dependencies. This strategy ensures that each component of the bot functions as expected under various scenarios. A detailed exploration of our testing approach and methodologies will be presented in the subsequent chapter.

@startuml

skinparam rectangle {

BackgroundColor PaleGreen

BorderColor DarkSlateGray

}

skinparam database {

BackgroundColor Gold

BorderColor DarkSlateGray

}

skinparam node {

BackgroundColor LightBlue

BorderColor DarkSlateGray

}

rectangle "User Device" as UD {

[Discord] as Discord

}

node "Local Server" as LS {

package "IDE & Code" as IDEcode{

[Visual Studio Code] as VSCode

[Python] as Python

[Discord.py] as Dpy

[Selenium] as Selenium

[UnitTest]

[PyTest]

}

[Git] as Git

}

database "Data Storage" as Data {

[JSON] as JSON

[HTML] as HTML

[Excel] as Excel

}

database "Repository" as Repo {

[GitHub] as GitHub

}

UD -down-> LS : User Commands\nand Responses

IDEcode-left-> Data : Data Logging\nand Reporting

Git-down-> Repo : Code Storage\nand Version Control

@enduml