**FastAPI**

FastAPI is a web framework for building APIs based on standard Python type hints. It is used developing RESTful APIs and microservices.

1. **Fast Execution:** Built on Starlette and Pydantic, FastAPI offers performance close to that of Node.js and Go.
2. **Automatic Data Validation:** Uses Pydantic for data validation and serialization.
3. **Asynchronous Support:** Fully supports asynchronous programming with Python's async and await.
4. **Interactive API Documentation:** Provides built-in Swagger UI and ReDoc documentation.
5. **Dependency Injection:** Makes handling authentication, authorization, and database connections easier.
6. **Type Annotations:** Uses Python’s type hints to generate efficient and reliable APIs.
7. **Automatic OpenAPI and JSON Schema Generation:** Generates API specifications without additional effort.

**Libraries Used in FastAPI**

FastAPI is designed to work with several useful libraries that enhance its functionality:

1. **Starlette** – Provides the web framework functionalities such as routing, middleware, WebSockets, and background tasks.
2. **Pydantic** – Used for data validation, serialization, and type enforcement.
3. **Uvicorn** – A lightning-fast ASGI server that runs FastAPI applications.
4. **Tortoise ORM / SQLAlchemy / Gino** – ORM libraries for handling database operations.
5. **Databases** – An asynchronous database connector compatible with PostgreSQL, MySQL, and SQLite.
6. **JWT (PyJWT / authlib / jose)** – Libraries for authentication and security handling.
7. **Celery / APScheduler** – Used for task scheduling and background jobs.
8. **Alembic** – For handling database migrations when using SQLAlchemy.
9. **HTTPX / Requests** – For making HTTP requests in FastAPI applications.
10. **Motor** – An asynchronous MongoDB driver used with FastAPI.
11. **GraphQL (Ariadne / Strawberry / Graphene)** – Enables FastAPI to support GraphQL APIs.
12. **FastAPI-Security** – Helps in implementing OAuth2 authentication.

**Web Scraping**

Web scraping is the process of extracting data from websites automatically using scripts or programs. This technique is used to collect large amounts of data efficiently, which can then be analyzed, stored, or used for various applications such as market research, competitive analysis, price monitoring, and data-driven decision-making.

How Web Scraping Works

1. A request is sent to a website using an HTTP client.
2. The server responds with HTML content.
3. The HTML is parsed to extract useful information.
4. The desired data is extracted using specific selectors.
5. The extracted data is saved in a structured format (CSV, JSON, database, etc.).

**Libraries Used for Web Scraping**

Several libraries help in web scraping, each with its strengths:

**1. BeautifulSoup**

* A Python library that helps extract data from HTML and XML documents.
* It allows searching and navigating the parsed data using selectors.

**2. Requests**

* A simple HTTP library to send GET and POST requests.

**3. Selenium**

* Automates web browsers and is useful for scraping dynamic websites.

**4. Scrapy (For Large-Scale Web Scraping)**

* A powerful framework for building web crawlers.
* Best for handling multiple pages and large datasets.

**5. LXML (For High-Speed Parsing)**

* A fast parser for processing HTML and XML.

**6. Playwright (For Headless Browsing & JavaScript Rendering)**

* Similar to Selenium but more efficient for headless browsing.

**Automation**

Automation refers to the use of technology to perform tasks with minimal human intervention. It helps in improving efficiency, reducing errors, and increasing productivity. Automation can be applied in various fields such as software testing, web scraping, data processing, task scheduling, robotic process automation (RPA), and DevOps.

1. Automating browser interactions, such as form filling, data scraping, and testing.
2. Automating repetitive tasks like file management, email handling, and backups.
3. Automating software testing to ensure reliability and performance.
4. Automating data extraction, processing, and storage.
5. Automating deployment, monitoring, and infrastructure management.

**Libraries Used for Automation**

1. **Selenium**

Used to control web browsers programmatically.

* Automates tasks like web form submissions, UI testing, and web scraping.

**2. Playwright (For Web Automation & Testing)**

* Faster and more reliable than Selenium for browser automation.
* Works with Chromium, Firefox, and WebKit.

**3. PyAutoGUI (For GUI Automation)**

* Automates keyboard and mouse actions.
* Used for clicking buttons, typing text, taking screenshots, and more.

**4. AutoPy (For Advanced GUI Automation)**

* Similar to PyAutoGUI but supports multi-monitor setups and low-level keyboard/mouse control.

**5. Task Scheduler (APScheduler, Crontab, Task Scheduler)**

* Automates task execution based on a schedule.
* APScheduler is used for scheduling tasks in Python.

**6. Requests & BeautifulSoup (For Web Scraping Automation)**

* Used to fetch and extract data from websites.

**7. Pandas & OpenPyXL (For Data Automation)**

* Automates data processing and Excel file handling.

**8. Pytest & Unittest (For Software Testing Automation)**

* Automates software testing to ensure applications work as expected.

**9. Fabric & Ansible (For Server & DevOps Automation)**

* Used for deploying applications, configuring servers, and running remote commands.