**High-Level Design (HLD) for TDD Lab**

**1. Overview**

The High-Level Design (HLD) presents the modular architecture of the TDD Lab. The design will review significant components of TDD such as tests, code implementation, and refactoring. The approach for testing TDD is implementing test cases as a guide to instruct what the code will do and how it will perform. Tests are developed and designed for functionalities of an application.

**2. System Architecture   
2.1. Modular Architecture**

* **Inventory Module:** This module’s responsibility is to test the functionality of the fruit stand application’s inventory. It’s related to keeping track of the inventory for the fruits and updating the quantity.
* **Cart Module:** This module’s responsibility is to test the functionality of the user’s shopping cart. Tasks such as adding items, removing items, adjusting the updated quantity, and the correct calculation for total cost. The benefit of writing tests for the cart module before code implementation is that one can define the expected behavior of the cart module and verify that it meets the expected expectations.
* **User Interface Module:** This module is responsible for testing the user interface of the fruit stand application. During testing it’s ensured that the user interface is user-friendly, responsive, and visually appealing to users.

**3. Components  
3.1. Web Application**

* A test framework such as PyTest supports TDD and provides the structure and tools for writing and implementing tests.
* Unit tests can serve a purpose for providing documentation for expected behavior of the code. This ensures that one can maintain the application with ease.

**3.2. PyTest (Test Framework)**

* Tests can be written for individual functionalities that will be implemented. The assert statement should be used during tests to check whether the outcome meets the expected results.
* Code can be refactored to ensure that other tests pass. It gives one the option to improve the quality of code and eliminate things such as duplicated or complex code.

**3.3. Snipping Tool**

* The snipping tool can be useful for documenting and sharing information related to tests and code. This application can be used to capture screenshots of error messages, test failures, or specific code snippets. Screenshots of the following examples can provide context and assistance to others viewing the issues.

**4. Data Flow**

**Write Tests:** Test cases are written to define the expected behavior of a specific functionality or snippet of code. The test case usually includes input data and expected output from the functionality.

**Run Tests:** When one runs the test, it will fail because the code has not been implemented. The failure of the test means that the code does not work as expected.

**Code Implementation:** Write the amount of code necessary to make the test pass. The code should meet the requirements of the test case and produce the expected result.

**Rerun Test:** It’s time to rerun the test again. If the test passes, the code works as expected based on the test case.

**Refactoring Code:** Code can be refactored to improve its structure, maintainability, and conciseness. The purpose of refactoring code is to ensure that the code is understandable and easier to manage for efficiency.

**5.** **Integrations**

* **PyTest:** In conjunction with TDD, PyTest provides a structured format to write and organize tests.
* **Jenkins:** Jenkins is a continuous integration tool (CI) that can help with automating the testing process. This tool can be used to run tests automatically when new code is pushed to the codebase.

**6. Scalability and Maintenance**

* TDD provides scalability by ensuring that written tests are testable independently. This approach of TDD makes it simpler to add new features or changes without any effects to the functionalities.
* Bugs can be caught early on with the help of writing tests before code implementation. Changes can be made to code by relying on existing tests to verify that the expected behavior remains maintained.

**7. Conclusion**

TDD’s approach provides several benefits for testing purposes. Writing tests beforehand is beneficial because clear expectations are established for the behavior of functionalities. Problems regarding the code can be fixed in the early stages of software development. This HLD reviews the modular architecture conjoined with the TDD workflow. The process of TDD can simplify the testing process for those wanting to improve the accuracy and quality of code.