**Low-Level Design (LLD) for Selenium Lab**

**1. Introduction** The low-level design provides detailing information about specific modules that are implemented in the Selenium Lab.

**2. Food Module   
2.1. Initialization of Prices and Items**

* **Function:**  \_\_init\_\_(self)
* **Input:** Name and price for food item
* **Output:** The current list of items available on the application
* **Description:** Firstly, the Food class is created. Initializes items with the corresponding names and prices for the application. The initialization of the food module prevents redundant code and helps to promote interaction with other modules.

**2.2. Assigning values to attributes**

* **Function:**  \_\_init\_\_(self, input\_name, input\_price)
* **Input:** Name and price are assigned to input\_name and input\_price with the init constructor
* **Output:** None
* **Description:** Uses objects from the Food class to assign input values for item name and price.

**3. Cart Module   
 3.1. Initialization of cost information for items**

* **Function:**  \_\_init\_\_(self)
* **Input:** The class Cart is created with object as a parameter. The attribute self.items is initialized into a list. Other attributes such as subtotal, discount, discount\_applied, purchase\_message, and total are initialized in the constructor as well.
* **Output:** None
* **Description:** Attributes from the class Cart are initialized for the shopping cart. The current state of the cart is initialized. The purpose of initialization for the cart module is to ensure that the user can add, remove, or apply discounts to his or her order.

**3.2. Adding items**

* **Function:** add(self, item)
* **Input:** The item that will be added to the cart.
* **Output:** An updated state of the cart that includes the item that is added.
* **Description:** The functionality of the add function is to add items to the cart. The function updates the cart to show the newly added items and the total cost.

**3.2 Removing items**

* **Function:** remove(self, item)
* **Input:** The item that will be removed from the cart.
* **Output:** An updated state of the cart will show that the item has been removed.
* **Description:** The functionality of the remove function is to remove items from the cart. The function updates the cart to show that the item has been removed.

**3.3 Updating the state of the cart**

* **Function:** \_updatetotal(self)
* **Input:** The item’s price
* **Output:** The total cost for items in the cart.
* **Description:** The updatetotal function updates the current state of the cart when items have been added, removed, or a discount has been applied.

**3.4. Applying the discount**

* **Function:**  apply\_discount(self, discount\_percentage)
* **Input:** The discount percentage and the items in the cart.
* **Output:** The discounted price in the cart after the discount has been applied.
* **Description:** In this discount function, the user has the option to apply a discount to the order. However, the discount can only be applied once.

**3.5. Total cost for items**

* **Function:** calculate\_total
* **Input:** Items that are in the cart.
* **Output:** Shows the total cost for items with the discount applied.
* **Description:** The function produces a calculation of the discounted total. The discounted total is rounded to two decimal places.

**3.6. The view of the shopping cart**

* **Function:** display(self)
* **Input:** Items that are in the cart and the subtotal
* **Output:** Shows a visualization of the current state of the cart along with the subtotal.
* **Description:** The function displays the items that are currently in the cart and includes cost information such as the subtotal.

**3.7.** **Clearing the cart**

* **Function:** purchase(self)
* **Input:** Items in the cart and cost information
* **Output:** The list is empty. The subtotal, total, and discount code are reset to zero.
* **Description:** The function clears items in the cart and cost information after the user’s purchase. The following message is displayed: “Thank you for purchasing!”

**4. App Module**

**4.1. Main function of the module: Index**

* **Function:** index()
* **Input:** The user accesses the root URL of the app.
* **Output:** The app runs showing the shopping application.
* **Description:** The function is used to handle the route of the web application. It returns to the main page of the application when the user accesses the root URL of the app.

**5. Constant\_values Module**

* **Function:** None
* **Input:** None
* **Output:** None
* **Description:** The module assigns prices for the items, including the set discount. These constant values are accessible to other modules. The upside of this module is that it prevents redundant code.

**6. Tester Module**

**6.1. The browser**

* **Function:** browser
* **Input:** Initializes the driver
* **Output:** The user can perform web automation tasks with the driver already initialized.
* **Description:** The function initializes the driver, then the driver is yielded to perform specific tasks. After the user is done with the driver, the driver automatically quits the session.

**6.2. Testing the Flask Framework**

* **Function:** test\_flask\_app(browser)
* **Input:** The URL of the Flask app
* **Output:** The output determines whether the assertion passes or fails the testing. If the assertion fails, an AssertionError will be raised.
* **Description:** The function performs testing on a Flask application. The browser.get method is used to navigate to the specified URL in the parenthesis. An assertion is used to check whether the application runs as expected.

**7. Conclusion**

The Low-Level Design highlights the overall architecture of the application. The document provides information about the specific functions and their specifications. This also would include descriptions for how the functions operate.