1)Develop the test plan for testing an e-commerce web/mobile application (www.amazon.in)

Creating a test plan for an e-commerce web/mobile application like Amazon.in involves several key components to ensure comprehensive coverage. Here's a concise outline of a test plan:

1. Test Plan Introduction:

- Objectives and goals of testing.
- Scope of testing within the application.
- Test strategy and approach.

2. Test Environment:

- Hardware and software requirements.
- Network configurations.
- Tools and resources needed.

3. Test Items:

- Features and functionalities to be tested.
- Out of scope items.

4. Risk Analysis:

- Potential risks and mitigation strategies.
- Prioritization of test scenarios based on risk.

5. Test Data:

- Data requirements for testing.
- Data privacy and security considerations.

6. Test Cases:

- Detailed test cases for:
    - Homepage
    - User registration and login
    - Product search and listing
    - Product details
    - Shopping cart functionality
    - Checkout process including payment gateway integration
    - Order confirmation and tracking
    - User account and order history
- Test cases for mobile-specific functionalities.

7. Types of Testing:

- Functional Testing: To verify each function of the application.
- Integration Testing: To check the data flow between integrated components.
- Performance Testing: To ensure the application performs well under expected loads.
- Security Testing: To validate data protection and vulnerability checks.
- Usability Testing: To confirm the application is user-friendly and intuitive.
- Compatibility Testing: To check the application's performance across different devices and browsers.

8. Test Execution:

- Test cycles and phases.
- Entry and exit criteria.
- Test case execution order.

9. Defect Management:

- Defect tracking and reporting process.
- Severity and priority classification.

10. Reporting and Metrics:

- Test progress reporting.
- Metrics for test coverage, defect density, etc.

11. Test Closure:

- Criteria for test completion.
- Lessons learned and documentation.

12. Tools and Resources:

- List of tools for test management, automation, performance, etc.

13. Schedule and Estimation:

- Time estimates for each testing phase.
- Resource allocation.

14. Approvals:

- Sign-off from stakeholders.

This is a high-level overview, and each section would need to be elaborated with specific details pertinent to www.amazon.in.

2)Develop the test plan for testing an e-commerce web/mobile application (www.flipkart.com).

Test Plan Outline for E-commerce Application

1. **Introduction**

   - Objectives and Scope

   - Test Strategy

   - Resources (Personnel, Tools, etc.)

2. **Test Environment**

   - Hardware and Software Requirements

   - Network Configurations

   - Test Data Management

3. **Test Types**

   - Functional Testing: To verify all functions of the application work as expected.

   - Integration Testing: To ensure different modules work together seamlessly.

   - Performance Testing: To assess the application's stability and responsiveness under various conditions.

   - Security Testing: To check for vulnerabilities and data protection mechanisms.

   - Usability Testing: To evaluate the user experience and ease of use.

   - Compatibility Testing: To confirm the application works across different devices and browsers.

4. **Test Cases**

   - **Functional Test Cases**: Covering user registration, product search, cart management, checkout process, payment gateway integration, etc.

   - **Non-Functional Test Cases**: Including load times, stress testing scenarios, security checks, etc.

5. **Test Execution**

   - Test Cycle Planning

   - Test Case Prioritization

   - Bug Tracking and Reporting

6. **Risk Analysis**

   - Identification of potential risks

   - Mitigation Strategies

   - Contingency Plans

7. **Reporting and Documentation**

   - Test Reports Format

   - Summary of Findings

   - Recommendations

8. **Exit Criteria**

   - Definition of 'Done'

   - Acceptance Criteria

9. **Review and Approval**

   - Stakeholder Review

   - Sign-off

3)Develop the test plan for testing an e-commerce web/mobile application (www.ebay.com).

Creating a test plan for an e-commerce web/mobile application like eBay involves several key components to ensure comprehensive coverage. Here's a concise outline of a test plan:

1. Test Plan Introduction:

- Objectives and scope of testing
- Testing strategy and focus areas
- Resources and timelines

2. Test Environment:

- Hardware and software requirements
- Network configurations
- Test data setup

3. Test Types:

- **Functional Testing:** Verify individual functions like product search, shopping cart, and payment processing[1].
- **Integration Testing:** Ensure components and third-party services work together seamlessly[1].
- **Performance Testing:** Assess the application's speed, efficiency, and reliability[1].
- **Security Testing:** Check for vulnerabilities and data protection measures.
- **Usability Testing:** Evaluate user-friendliness and design consistency.
- **Compatibility Testing:** Confirm the application works across different devices and browsers[1].

4. Test Cases:

- Define specific scenarios for each feature, including edge cases and error conditions.
- Prioritize test cases based on risk and impact.

5. Test Execution:

- Schedule and sequence of test case execution
- Tracking and reporting of defects
- Retesting and regression testing procedures

6. Test Reporting:

- Summary of test activities
- Detailed defect reports
- Test metrics and final assessment

7. Tools and Resources:

- List of testing tools for various test types, like functional, integration, and performance testing[1].

8. Risk Management:

- Identification of potential risks

- Mitigation strategies and contingency plans

9. Approval and Sign-off:

- Criteria for test completion

- Sign-off from stakeholders

4)Develop the test plan for testing an e-commerce web/mobile application (www.shopify.com).

1. Test Strategy:

- Define the scope and objectives of testing.

- Identify the test items and features to be tested.

- Determine the risk factors and prioritize the test cases accordingly.

2. Test Environment:

- Set up the testing environment that simulates real-world user conditions.

- Ensure cross-platform compatibility testing on various devices and browsers.

3. Test Cases:

- **Functional Testing:** Verify all user functions such as account registration, login, product search, cart operations, and payment processing.

- **Integration Testing:** Check the interaction between different modules of the application and third-party services like payment gateways.

- **Performance Testing:** Assess the application's performance under various loads to ensure stability and speed.

- **Security Testing:** Conduct vulnerability assessments and security checks to protect user data and transactions.

- **Usability Testing:** Evaluate the application's ease of use and overall user experience.

- **API Testing:** Ensure that all APIs used by the application are reliable, perform well, and secure data transfer.

4. Test Data:

- Prepare test data for various test scenarios, including positive and negative cases.

5. Test Execution:

- Execute test cases as per the plan and document the results.

- Use automated testing tools where applicable to increase efficiency.

6. Defect Tracking:

- Log and track any defects found during testing.

- Work with the development team to resolve issues and retest as needed.

7. Reporting:

- Provide regular updates on the testing progress, coverage, and outcomes.

- Prepare a final test report summarizing the testing activities and results.

8. Tools and Resources:

- List the tools and resources required for testing, including software, hardware, and human resources.

9. Schedule:

- Develop a timeline for all testing activities, from test planning to final reporting.

10. Review and Approval:

- Have the test plan reviewed by stakeholders.

- Obtain approval to proceed with the testing as planned.

5)Design the test cases for testing the e-commerce application

1. User Interface (UI) Test Cases:

- Verify that the company logo and name are clearly visible.

- Ensure all links and banners redirect correctly and are not broken.

- Check that product information (name, category, price, description) is clearly visible.

- Confirm that product and banner images are clear and properly displayed.

2. Product Buy-Flow Test Cases:

- Validate that users can select product attributes (size, color, etc.).

- Check the functionality of adding one or more products to the cart.

- Verify that users can add products to the wishlist.

- Ensure that previously added products are visible in the cart after signing in.

3. User Registration and Account Management:

- Test the user registration process for new users.

- Verify the login functionality for returning users.

- Check the account recovery process for forgotten passwords.

- Validate the update and deletion of user account information.

4. Payment Gateway Integration:

- Verify that all listed payment methods are working correctly.

- Test the transaction process for each payment method.

- Check the security of the payment process and data encryption.

- Validate the refund process and transaction reversal.

5. Product Search and Filtering:

- Test the search functionality returns relevant results.

- Verify that filtering options correctly narrow down the search results.

- Check that sorting options correctly order the products.

6. Checkout Process:

- Test the checkout process from cart to order confirmation.

- Verify that all necessary information is required and validated.

- Check that the final order summary is accurate before submission.

7. Coupon and Discount Application:

- Verify that coupon codes apply the correct discount.

- Test the functionality of different types of discounts.

- Check that expired coupons are not accepted.

8. Shipping and Delivery:

- Test the selection of different shipping options.

- Verify that shipping costs are correctly calculated.

- Check the tracking functionality for shipped orders.

9. Customer Service and Support:

- Verify the functionality of the help and support sections.

- Test the responsiveness of customer service channels.

- Check the accuracy and helpfulness of FAQs and automated chatbots.

10. Performance and Security:

- Conduct load testing to ensure the application handles high traffic.

- Perform security testing to identify any vulnerabilities.

- Test the application’s performance on different devices and browsers.

6)Test the e-commerce application and report the defects in it.

Functional Testing:

- Verify the functionality of all features like product search, filters, add to cart, checkout, payment gateways, etc.

- Check for proper integration with databases, third-party services, and APIs.

Usability Testing:

- Assess the ease of navigation, clarity of information, and overall user experience.

- Ensure the application is intuitive and the workflow is smooth from landing to checkout.

Security Testing:

- Test for vulnerabilities to SQL injection, XSS, CSRF, and other common web security threats.

- Confirm that customer data is handled securely, especially during transactions.

Performance Testing:

- Evaluate the application's performance under various loads to ensure it can handle peak traffic.

- Check for speed and responsiveness of the site on different devices and browsers.

Database Testing:

- Ensure data integrity and consistency across the application.

- Validate queries for accuracy and performance.

Mobile Application Testing:

- Confirm that the mobile version of the application is optimized for various screen sizes and resolutions.

- Test the app on different mobile operating systems for compatibility.

A/B Testing:

    - Compare different versions of the application to determine which features or designs yield better user engagement and conversion rates.

7)Develop the test plan and design the test cases for an inventory control system.

Test Plan for Inventory Control System:

1. **Introduction:**
   - Objectives and goals of testing.
   - System overview and scope.
2. **Test Items:**
   - List of components and features to be tested (e.g., product catalog, stock levels, order processing).
3. **Features to be Tested:**
   - Detailed features and functionalities that require testing.
4. **Features not to be Tested:**
   - Out-of-scope features or those that will not be tested.
5. **Approach:**

- Testing strategy and types of testing to be performed (e.g., unit, integration, system, user acceptance).

6. **Pass/Fail Criteria:**
   - Define the criteria for passing and failing test cases.
7. **Test Deliverables:**
   - Documents and reports to be produced (e.g., test cases, bug reports).
8. **Testing Tasks:**
   - Specific tasks required to prepare and execute testing.
9. **Environmental Needs:**
   - Hardware, software, network configurations, and other requirements.
10. **Responsibilities:**
    - Roles and responsibilities of the testing team members.
11. **Schedule:**
    - Timeline for test preparation, execution, and evaluation.
12. **Risks and Contingencies:**
    - Potential risks and their mitigation plans.
13. **Approval:**
    - Sign-off from stakeholders.

Test Cases for Inventory Control System:

1. **Product Catalog Management:**
   - TC01: Add a new product to the catalog.
   - TC02: Edit product details.
   - TC03: Delete a product from the catalog.
2. **Stock Level Management:**
   - TC04: Update stock levels after receiving items.
   - TC05: Verify stock levels decrease after a sale.
   - TC06: Check for stock level alerts when below threshold.
3. **Order Processing:**
   - TC07: Create a new order.
   - TC08: Process an order for existing inventory.
   - TC09: Cancel an order.
4. **Reporting:**

- TC10: Generate inventory level report.

- TC11: Generate sales report.

- TC12: Generate order history report.

8)Execute the test cases against a client server or desktop application and identify the defects.

1. **Test Case Preparation:**
   - Create detailed test cases based on requirements, user stories, or use cases.
   - Include both positive and negative scenarios.
   - Specify input data, expected outcomes, and any preconditions.
2. **Test Execution:**
   - Execute the test cases against the application.
   - Use manual testing or automated testing tools (e.g., Selenium, Appium, JUnit, TestNG).
   - Verify that the application behaves as expected.
3. **Defect Identification:**
   - While executing test cases, pay attention to unexpected behaviors, errors, or inconsistencies.
   - Common defects include:
     - **Functional Defects:** Features not working as intended.
     - **Usability Defects:** Poor user experience, confusing UI, or broken links.
     - **Performance Defects:** Slow response times, memory leaks, or resource issues.
     - **Security Defects:** Vulnerabilities, authentication issues, or data leaks.
     - **Compatibility Defects:** Issues on specific browsers, devices, or platforms.
4. **Logging Defects:**
   - When you encounter a defect, log it in a defect tracking system (e.g., JIRA, Bugzilla).
   - Include details like steps to reproduce, actual vs. expected results, and screenshots if possible.
   - Assign severity and priority levels to defects.
5. **Regression Testing:**
   - After fixing defects, perform regression testing to ensure that existing functionality remains intact.
   - Re-execute previously passed test cases to catch any unintended side effects.
6. **Collaboration:**
   - Communicate defects to developers, providing clear information.
   - Collaborate with the development team to understand the root cause and verify fixes.

9)Test the performance of the e-commerce application.

Functionality Testing: Check all the features of your application, such as product search, shopping cart, payment gateway, etc., to ensure they are working as expected[1].

1. **Usability Testing**: Evaluate the user interface and overall user experience of your application. This includes checking the intuitiveness of the navigation, the clarity of the instructions, and the responsiveness of the buttons[2].
2. **Performance Testing**: Use tools like Lighthouse, PageSpeed Insights, and WebPageTest to assess your application's load times, performance metrics, and areas for improvement[3][4]. This includes:
   - **Page Speed Testing**: Measure how quickly the pages of your application load[5].
   - **Load Testing**: Determine how well your application performs under heavy traffic[6].
   - **Stress Testing**: Test the limits of your application by increasing the load beyond its normal capacity[6].
3. **Security Testing**: Ensure that your application is secure from threats and vulnerabilities. This includes testing the security of user data and payment information[1].
4. **Compatibility Testing**: Check if your application works well across different devices, operating systems, and browsers[2]..

10)Automate the testing of e-commerce applications using Selenium

1. **Understand Selenium:**
   - Selenium is a powerful tool for cross-browser automated testing. It supports various browsers and allows you to interact with web elements programmatically.
   - Familiarize yourself with Selenium's concepts, such as WebDriver, locators, and page objects.
2. **Choose a Sample E-Commerce Website:**
   - For practice, you can choose an e-commerce website similar to Amazon. This will allow you to automate common tasks like adding addresses, searching for products, and checking out.
   - In this example, let's automate the address-filling process on Amazon.
3. **Set Up Your Environment:**
   - Install Java and set up your development environment.
   - Download the necessary Selenium JAR files or use Maven to manage dependencies.
4. **Write Your Selenium Script:**
   - Create a Java class (e.g., `AmazonWebsiteAddressAutoSubmit`) to automate the address submission process.
   - Use the Chrome WebDriver (chromedriver) to launch the Amazon address page.
   - Fill in the required fields (e.g., full name, phone number, postal code, address lines, city).
   - Locate the submit button and click it to add the address.
   - Optionally, verify that the address was added successfully.
5. **Sample Java Code:**

```
6. import java.util.List;
7. import java.util.concurrent.TimeUnit;
8. import org.openqa.selenium.By;
9. import org.openqa.selenium.WebDriver;
10.  import org.openqa.selenium.WebElement;
11.  import org.openqa.selenium.chrome.ChromeDriver;
12.  import org.openqa.selenium.chrome.ChromeOptions;
13.  import io.github.bonigarcia.wdm.WebDriverManager;
14.
15.  public class AmazonWebsiteAddressAutoSubmit {
16.      public static void main(String[] args) {
17.          ChromeOptions chromeOptions = new ChromeOptions();
18.          WebDriverManager.chromedriver().setup();
19.          WebDriver driver = new ChromeDriver(chromeOptions);
20.          driver.manage().timeouts().pageLoadTimeout(15, TimeUnit.MINUTES);
21.
22.          // Launch the Amazon address page
23.
   driver.get("https://www.amazon.in/a/addresses/add?ref=ya_address_book_add_post");
24.
25.          // Replace with your email and password
26.          LoginPage login = new LoginPage(driver);
27.          login.set_username("tltxxx@gmail.com");
28.          login.continueButtonClick();
29.          login.set_password("xxxx");
30.          login.click_button();
31.
32.          // Replace the fields as per your requirement
33.          NewAddressDetails addAddress = new NewAddressDetails(driver);
34.          addAddress.set_fullname("xxxxx");
35.          addAddress.set_phonenumber("1234567890");
36.          addAddress.set_postalCode("625 016");
37.          addAddress.set_addressLine1("AddressLine1");
38.          addAddress.set_addressLine2("AddressLine2");
39.          addAddress.set_city("City");
40.
41.          // Click the submit button
42.          List<WebElement> allElements =
   driver.findElements(By.xpath("//input[@class='a-button-input']"));
43.          WebElement clickableElement = null;
44.          for (WebElement element : allElements) {
```

```
45.              if (element.getAttribute("aria-labelledby").equals("address-ui-
   widgets-form-submit-button-announce")) {
46.                  clickableElement = element;
47.                  clickableElement.click();
48.              }
49.          }
50.
51.          // Edit address part (optional)
52.          // You can check whether the address got added successfully
53.          driver.get("https://www.amazon.in/a/addresses");
54.          System.out.println("Successfully performed the operation of adding
   address");
55.      }
56.  }
```

57. **Additional Resources:**
    - GeeksforGeeks has an article on automating Amazon-like e-commerce websites with Selenium.
    - You can explore other e-commerce websites and automate different functionalities using similar principles.

11)Build a data-driven framework using Selenium and TestNG

1. **Introduction to Data-Driven Testing in Selenium WebDriver:**
    - Data-driven testing allows us to test different scenarios using various data sets.
    - Instead of hardcoding test data within test cases, we keep it external (e.g., in Excel sheets, CSV files).
    - This separation of test code and test data makes maintenance easier.
2. **Advantages of Data-Driven Testing:**
    - Reusability: Same test code can be used with different data sets.
    - Scalability: Easily add new test scenarios without modifying existing code.
    - Flexibility: Adapt to changes in business rules without rewriting tests[1].
3. **What is Apache POI?**
    - Apache POI (Poor Obfuscation Implementation) is a Java library for reading and writing Microsoft Office formats (e.g., Excel).
    - We'll use Apache POI to handle Excel files in our data-driven framework.
4. **Handling Excel Files using Apache POI:**
    - Read and write data from/to Excel sheets (.xls or .xlsx).
    - Use `HSSFWorkbook` for .xls files and `XSSFWorkbook` for .xlsx files.
5. **Creating Data-Driven Framework in Selenium WebDriver using Apache POI and TestNG:**
    - Set up your Selenium WebDriver project with TestNG.
    - Add Apache POI dependencies to your project.
6. **Read and Write Data from/to Excel Sheets:**
    - Create an Excel file with test data (e.g., login credentials, search queries).
    - Use Apache POI to read data from the Excel sheet and pass it to your test methods.
    - Update the Excel sheet with test results (if needed).
7. **Data-Driven Framework using TestNG DataProvider with Excel:**
    - Define a data provider method in your TestNG class.
    - Inside the data provider, read data from the Excel sheet and return it as a 2D array.
    - Annotate your test methods with `@Test(dataProvider = "dataProviderMethodName")`.
8. **Perform Cross-Browser Testing using Apache POI:**
    - Extend your data-driven framework to handle different browsers.

- Read browser names (e.g., Chrome, Firefox) from the Excel sheet.
- Initialize the corresponding WebDriver based on the browser name.

12)Build Page object Model using Selenium and TestNG

1. **What is the Page Object Model (POM)?**
   - POM treats each web page of an application as a separate class.
   - Each class represents a page and contains methods to interact with the elements on that page.
   - The advantage is that it reduces code duplication and improves test case maintenance.
2. **Creating Page Objects:**
   - Start by creating a separate class for each page of your application.
   - For example, if you have a login page, create a `LoginPage` class.
   - Initialize the WebDriver in the constructor of each page class.
3. **Page Object Example (Login Page):**

```
package com.example.pages;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;

public class LoginPage {
    private WebDriver driver;

    public LoginPage(WebDriver driver) {
        this.driver = driver;
        // Optionally, navigate to the login page here
    }

    public void enterUsername(String username) {
        driver.findElement(By.id("username")).sendKeys(username);
    }

    public void enterPassword(String password) {
        driver.findElement(By.id("password")).sendKeys(password);
    }

    public void clickLoginButton() {
        driver.findElement(By.id("loginButton")).click();
    }
}
```

29. **Creating Page Objects for Other Pages:**
   - Similarly, create classes for other pages (e.g., `HomePage`, `SearchPage`, etc.).
   - Each class should contain methods to interact with the elements specific to that page.
30. **TestNG Tests Using Page Objects:**
   - In your test classes, initialize the WebDriver and create instances of page objects.
   - Use the methods from page objects to perform actions (e.g., login, search, etc.).
   - Here's a simple example:

```
package com.example.tests;

import org.openqa.selenium.WebDriver;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.Test;
import com.example.pages.LoginPage;

public class LoginTest {
    private WebDriver driver;
    private LoginPage loginPage;

    @BeforeClass
    public void setUp() {
        // Initialize WebDriver (e.g., ChromeDriver)
```

```
        // Navigate to the login page
        // Create LoginPage instance
    }

    @Test
    public void testValidLogin() {
        loginPage.enterUsername("myusername");
        loginPage.enterPassword("mypassword");
        loginPage.clickLoginButton();
        // Assert home page elements
    }
}
```

31. **Benefits of POM:**
    - **Modularity:** Each page class encapsulates its elements and actions.
    - **Reusability:** Easily reuse page objects across multiple tests.
    - **Maintenance:** If UI changes, update only the relevant page class.

13)Build BDD framework with Selenium, TestNG and Cucumber

What is Cucumber in Selenium?

Cucumber is an open-source testing framework that supports BDD for automating web applications. It bridges the gap between business stakeholders and technical teams by allowing test scenarios to be written in plain English. These scenarios describe the expected behavior of the system from the user's perspective. Cucumber tests are written in a human-readable language called **Gherkin**[1].

Cucumber and Selenium Testing: A Collaborative Approach

When combined with Selenium, Cucumber enhances the testing process by making it more collaborative and accessible to non-technical stakeholders. Here's how it works:

1. **Feature File:**
    - The **Feature File** contains high-level descriptions of features or functionalities.
    - Written in Gherkin syntax, it outlines scenarios and their expected outcomes.
    - Example feature file snippet:

```
Feature: Login functionality
  Scenario: Successful login
    Given the user is on the login page
    When they enter valid credentials
    Then they should be logged in
```

2. **Step Definitions:**
    - The **Step Definitions** map Gherkin steps to actual code.
    - Each step corresponds to a method in Java (or other languages).
    - Example step definition:

```
@Given("the user is on the login page")
public void navigateToLoginPage() {
    // Navigate to the login page
}
```

3. **Test Runner File:**
    - The **Test Runner File** executes the Cucumber scenarios.
    - It specifies which feature files to run and where to find step definitions.
    - Example test runner class:

```
import io.cucumber.testng.AbstractTestNGCucumberTests;
import io.cucumber.testng.CucumberOptions;

@CucumberOptions(features = "src/test/resources/features", glue =
"stepdefinitions")
public class TestRunner extends AbstractTestNGCucumberTests {
```

```
        // Configuration options
    }
```

Setting up Cucumber BDD Framework for Selenium

Here are the steps to set up your Cucumber BDD framework with Selenium and TestNG:

1. **Prerequisites:**
   - Ensure you have Java and Maven (or Gradle) installed.
   - Set up your Selenium WebDriver (e.g., ChromeDriver, FirefoxDriver).
   - Create a new Maven or Gradle project.
2. **Add Dependencies:**
   - In your `pom.xml` (for Maven) or `build.gradle` (for Gradle), add the following dependencies:

```
<!-- For Cucumber -->
<dependency>
    <groupId>io.cucumber</groupId>
    <artifactId>cucumber-java</artifactId>
    <version>7.2.3</version>
</dependency>
<dependency>
    <groupId>io.cucumber</groupId>
    <artifactId>cucumber-testng</artifactId>
    <version>7.2.3</version>
</dependency>

<!-- For Selenium -->
<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>4.1.3</version>
</dependency>
```

   - Configure your build tool to execute Cucumber features and generate reports.
3. **Create Feature Files:**
   - Under `src/test/resources/features`, create your feature files.
   - Write scenarios in Gherkin syntax.
4. **Write Step Definitions:**
   - Create Java classes (e.g., `LoginSteps`) to define step implementations.
   - Map Gherkin steps to Java methods.
5. **Run Tests:**
   - Execute your tests using the Test Runner class.
   - View Cucumber reports (HTML, JSON, etc.) to analyze results.

Best Practices in Cucumber Testing

- Keep feature files concise and focused on one functionality.
- Use descriptive step definitions for better readability.
- Leverage tags to organize and filter scenarios.
- Regularly update your test data in feature files.

14)Develop the test plan for testing an e-commerce web/mobile application (www.amazon.in). (1)

15)Design the test cases for testing the e-commerce application (15)

16)Test the e-commerce application and report the defects in it. (16)

17)Develop the test plan and design the test cases for an inventory control system. (17)

18)Execute the test cases against a client server or desktop application and identify the defects. (18)

19)Test the performance of the e-commerce application. (19)

20)Automate the testing of e-commerce applications using Selenium (10)