Testing Review

Manual and Automated Testing

- **Manual Testing**: In manual testing, a tester will manually go through each part of the application and test out different features to make sure it works correctly. If any bugs are uncovered in the testing process, the tester will report those bugs back to the software team so they can be fixed.
- **Automated Testing**: In automated testing, you can automate your tests by writing a separate program that checks whether your application behaves as expected.

Unit Testing

- **Unit Testing**: In unit testing, you test each function to ensure that everything is working as expected. Unit tests can also serve as a form of documentation for your application because they are meant to represent the expected behavior for your code.
- **Single Responsibility Principle**: The single responsibility principle recommends keeping each function small and responsible for one thing.
- **Common JavaScript Testing Frameworks**: Some common testing frameworks include Jest, Mocha, and Vitest. Jest is a popular testing framework for unit tests.

Here is an example of unit tests using Jest.

First, you can create a function that is responsible for returning a newly formatted string:

```
export function getFormattedWord(str) {
  if (!str) return '';
  return str.charAt(0).toUpperCase() + str.slice(1);
}
```

In a separate `getFormattedWord.test.js` file, you can write some tests to verify that the function is working as expected. The `getFormattedWord.test.js` file will look like this:

```
import { getFormattedWord } from "./getFormattedWord.js";
import { test, expect } from "jest";

test('capitalizes the first letter of a word', () => {
  expect(getFormattedWord('hello')).toBe('Hello');
});
```

- **`expect` Function**: The `expect` function is used to test a value.
- **Matcher**: Matcher is a function that checks whether the value behaves as expected. In the above example, the matcher is `toBe()`. Jest has a variety of matchers.

To use Jest, you first need to install the `jest` package by using `npm i jest`. You will also need to add a script to your `package.json` file like this:

```
"scripts": {
  "test": "jest"
},
```

Then, you can run the command `npm run test` to run your tests.

Software Development Lifecycle

- **Different Stages of Software Development Lifecycle**:
    - **Planning Stage**: The development team collects requirements for the proposed work from the stakeholders.
    - **Design Stage**: The software team breaks down the requirements and decides on the best approaches for solutions.
    - **Implementation Stage**: The software team breaks down the requirements into manageable tasks and implements them.
    - **Testing Stage**: This involves manual and automated testing for the new work. Sometimes, the team tests out the application throughout the entire development stage to catch and fix any issues that come up.
    - **Deployment Stage**: The team deploys the new changes to a build or testing environment.
    - **Maintenance Stage**: This involves fixing any issues that arise from customers in the production application.

- **Different Models of the Software Development Lifecycle**:
    - **Waterfall Model**: The Waterfall model is where each phase of the lifecycle needs to be completed before the next phase can begin.
    - **Agile Model**: The Agile model focuses on iterative development by breaking down work into sprints.

BDD and TDD

- **TDD**: Test-driven development is a methodology that emphasizes writing tests first. Writing tests before building out features provides real-time feedback to developers during the development process.
- **BDD**: Behavior-driven development is the approach of aligning a series of tests with business goals. The test scenarios in BDD should be written in a language that can be understood by both technical and non-technical individuals. An example of such syntax is Gherkin.
- **BDD Testing Frameworks**: Examples of BDD testing frameworks include Cucumber, JBehave, and SpecFlow.

Assertions in Unit Testing

- **Assertion**: Assertions are used to test that the code is behaving as expected.
- **Assertion Libraries**: Chai is a commonly used assertion library. Other common JavaScript assertion libraries are `should.js` and `expect.js`.

Here is an example of an assertion using Chai that checks that the return value from the `addThreeAndFour` function is equal to the number 7:

```
assert.equal(addThreeAndFour(), 7);
```

> **Best Practices**: Regardless of which assertion library you use, you should write clear assert and failure messages that will help you understand which tests are failing and why.

Mocking, Faking, and Stubbing

- **Mocking**: Mocking is the process of replacing real data with false data that simulates the behavior of real components. For example, you could mock the API response in testing instead of making continuous API calls to fetch the data.
- **Stubbing**: Stubs are objects that return pre-defined responses or dummy data for an expected behavior in an application. For example, you can stub the behavior for a database connection in your tests without having to rely on an actual database connection.
- **Faking**: Fakes are simplified versions of real components without the complexity or side effects. For example, you can fake a database by storing the data in memory instead of interacting with the real database. This will allow you to mimic database operations in memory, which will be much faster than dealing with the real database.

Functional Testing

- **Functional Testing**: Functional testing checks if the features and functions of the application work as expected. The goal of functional testing is to test the system as a whole against multiple scenarios.
- **Non-Functional Testing**: Non-functional testing focuses on things like performance and reliability.
- **Smoke testing**: Smoke testing is a preliminary check on the system for basic or critical issues before beginning more extensive testing.

End-to-End Testing

- **End-to-End Testing**: End-to-end testing, or E2E, tests real-world scenarios from the user's perspective. End-to-end tests help ensure that your application behaves correctly and is predictable for users. However, it is time-consuming to set up, design, and maintain.
- **End-to-End Testing Frameworks**: Playwright is a popular end-to-end testing framework developed by Microsoft. Other examples of end-to-end testing tools include Cypress, Selenium, and Puppeteer.

Here is an example of E2E tests from the freeCodeCamp codebase using Playwright. The `beforeEach` hook will run before each of the tests. The tests check that the donor has a supporter link in the menu bar, as well as a special stylized border around their avatar:

```
test.beforeEach(async ({ page }) => {
  execSync("node ./tools/scripts/seed/seed-demo-user --set-true isDonating");
  await page.goto("/donate");
});

...

test("The menu should have a supporters link", async ({ page }) => {
  const menuButton = page.getByTestId("header-menu-button");
  const menu = page.getByTestId("header-menu");

  await expect(menuButton).toBeVisible();
  await menuButton.click();

  await expect(menu).toBeVisible();

  await expect(page.getByRole("link", { name: "Supporters" })).toBeVisible();
});

test("The Avatar should have a special border for donors", async ({ page }) => {
  const container = page.locator(".avatar-container");
  await expect(container).toHaveClass("avatar-container gold-border");
});
```

Usability Testing

- **Usability Testing**: Usability testing is when you have real users interacting with the application to discover if there are any design, user experience, or functionality issues in the app. Usability testing focuses on the intuitiveness of the application by users.
- **Four Common Types of Usability Testing**:
  - **Explorative**: Explorative usability testing involves users interacting with the different features of the application to better understand how they work.
  - **Comparative**: Comparative testing is where you compare your application's user experience with similar applications in the marketplace.
  - **Assessment**: Assessment testing is where you study how intuitive the application is to use.
  - **Validation**: Validation testing is where you identify any major issues that will prevent the user from using the application effectively.
- **Usability Testing Tools**: Examples of tools for usability testing include Loop11, Maze, Userbrain, UserTesting, and UXTweak.

Compatibility Testing

- **Compatibility Testing**: The goal of compatibility testing is to ensure that your application works in different computing environments.
- **Different Types of Compatibility Testing**:
  - **Backwards Compatibility**: Backwards compatibility refers to when the software is compatible with earlier versions.
  - **Forwards Compatibility**: Forwards compatibility refers to when the software and systems will work with future versions.
  - **Hardware Compatibility**: Hardware compatibility is the software's ability to work properly in different hardware configurations.
  - **Operating Systems Compatibility**: Operating systems compatibility is the software's ability to work on different operating systems, such as macOS, Windows, and Linux distributions like Ubuntu and Fedora.
  - **Network Compatibility**: Network compatibility means the software can work in different network conditions, such as different network speeds, protocols, security settings, etc.
  - **Browser Compatibility**: Browser compatibility means the web application can work consistently across different browsers, such as Google Chrome, Safari, Firefox, etc.
  - **Mobile Compatibility**: It is important to ensure that your software applications work on a variety of Android and iOS devices, including phones and tablets.

Performance Testing

- **Performance Testing**: In performance testing, you test an application's speed, responsiveness, scalability, and stability under different workloads. The goal is to resolve any type of performance bottleneck.

- **Different Types of Performance Testing**:
  - **Load Testing**: Load testing determines how a system behaves during normal and peak load times.
  - **Stress Testing**: Stress testing is where you test your application in extreme loads and see how well your system responds to the higher load.
  - **Soak Testing (Endurance Testing)**: Soak testing or endurance testing is a type of load testing where you test the system with a higher load for an extended period of time.
  - **Spike Testing**: Spike testing is where you dramatically increase and decrease the loads and analyze the system's reactions to the changes.
  - **Breakpoint Testing (Capacity Testing)**: Breakpoint testing or capacity testing is where you slowly increment the load over time to the point where the system starts to fail or degrade.

Security Testing

- **Security Testing**: Security testing helps identify vulnerabilities and weaknesses.
- **Security Principles**:
  - **Confidentiality**: This protects against the release of sensitive information to other recipients that aren't the intended recipient.
  - **Integrity**: This involves preventing malicious users from modifying user information.
  - **Authentication**: This involves verifying the user's identity to ensure that they are allowed to use that system.
  - **Authorization**: This is the process of determining what actions authenticated users are allowed to perform or which parts of the system they are permitted to access.
  - **Availability**: This ensures that information and services are available to authorized users when they need it.
  - **Non-Repudiation**: This ensures that both the sender and recipient have proof of delivery and verification of the sender's identity. It protects against the sender denying having sent the information.
- **Common Security Threats**:
  - **Cross-Site Scripting (XSS)**: XSS attacks happen when an attacker injects malicious scripts into a web page and then executes them in the context of the victim's browser.
  - **SQL Injection**: SQL injection allows malicious users to inject malicious code into a database.
  - **Denial-of-Service (DoS) Attack**: DoS attack is when malicious users flood a website with a high number of requests or traffic, causing the server to slow down and possibly crash, making the site unavailable to users.
- **Categories of Security Testing Tools**:
  - **Static Application Security Testing**: These tools evaluate the source code for an application to identify security vulnerabilities.
  - **Dynamic Application Security Testing**: These tools interface with the application's frontend to uncover potential security weaknesses. DAST tools do not have access to the source code.

- **Penetration Testing (pentest)**: Penetration testing is a type of security testing that involves creating simulated cyberattacks on the application to identify any vulnerabilities in the system.

A/B Testing

- **A/B Testing**: A/B testing involves comparing two versions of a page or application and studying which version performs better. It is also known as bucket or split testing. A/B testing allows you to make more data-driven decisions and continually improve the user experience.
- **Tools for A/B Testing**: Examples of tools to use for A/B testing include GrowthBook and LaunchDarkly.

Alpha and Beta Testing

Once the initial development and software testing are complete, it is important to have the application tested by testers and real users. This is where alpha and beta testing come in.

- **Alpha Testing**: Alpha testing is done by a select group of testers who go through the application to ensure there are no bugs before it is released into the marketplace. Alpha testing is part of acceptance testing and utilizes both white and black box testing techniques.
- **Beta Testing**: Beta testing is where the application is made available to real users. Users can interact with the application and provide feedback. Beta testing is also a form of user acceptance testing.
- **Acceptance Testing**: Acceptance testing ensures that the software application meets the business requirements and the needs of users before its release.
- **Black Box Testing**: Black box testing only focuses on the expected behavior of the application.
- **White Box Testing**: White box testing involves the tester knowing the internal components and performing tests on them.

Regression Testing

- **Regression**: Regression refers to situations where new changes unintentionally break existing functionality.
- **Regression Testing**: Regression testing helps catch regression issues. In regression testing, you re-run functional tests against parts of your application to ensure that everything still works as expected.
- **Tools for Regression Testing**: Tools that you can use to perform regression testing include Puppeteer, Playwright, Selenium, and Cypress.

- **Techniques for Regression Testing**:
  - **Unit Regression Testing**: This is where you have a list of items that need to be tested each time major changes or fixes are implemented into the app.
  - **Partial Regression Testing**: This involves targeted approaches to ensure that new changes have not broken specific aspects of the application.
  - **Complete Regression Testing**: This runs tests against all the functionalities in the codebase. This is the most time-consuming and detailed option.
- **Retesting**: Retesting is used to check for known issues and ensure that they have been resolved. In contrast, regression testing searches for unknown issues that might have occurred through recent changes in the codebase.

Assignment

Review the Testing topics and concepts.

Please complete the assignment