TEMA – Testimi i automatizuar i aplikacioneve web

ABSTRAKT

**Tabela e permbajtjes**

ABSTRAKTI  
LISTA E TABELAVE  
Lista e figurave

Lista e shkurtimeve

KAPITULLI I – disa shpjegime per qellimin e ketij studimi

Kapitulli 2 – Cfare eshte testimi – (ISTQB infos)

Kapitulli 3 – Testimi i web apps (selenium, cypress) – FREE

Kapitulli 4 Testimi I web apps (paid tools) – Neoload, Testcomplete ? To be decided

Kapitulli 5 Krahasimi i free charge dhe paid tools (PROS CONS)

Kapitulli 6 – Load Testing (Neoload ?) – to be decided

QELLIMI I STUDIMIT

Framework-et e automatizimit te testeve po perdoren gjithmone dhe me teper ne procesin e zhvillimit te softuerve, pasi ato permiresojne procesin e testimit te produktit. Kto frameworks lejojne zhvilluesit te automatiozojne testet qe perseriten shpesh dhe shpesh jane TIME-CONSUMING. Ne vitet e fundit ne treg mund te gjejme shume frameworks te ndryshem falas ose me pagese qe ndihmojne ne automatizimin e procesit te testimit.

Pavarsisht avantazheve qe cdo framework testmi ka, secila nga ato ka specifikat e veta dhe ne varesi te natyres se projektit nje framework mund te jete me i mire se tjetri. Ne kete studim ne do te perpiqemi te krahasojme dy framework shume te njohura qe jane falas si Selenium dhe Cypress dhe te vleresojme gjithashtu si qendrojne ato ne raport me njera tjetren dhe ne raport me framework ose tools te tjera per automatizin e testeve qe jane me pagese.

OBJEKTIVAT E KETIJ STUDIMI

1. Rishikoni literaturën mbi kornizat e automatizimit dhe krahasoni kornizat e ndryshme bazuar në veçoritë dhe aftësitë e tyre
2. Vlerësoni pikat e forta dhe të dobëta të Selenium dhe Cypress bazuar në kritere të tilla si lehtësia e përdorimit, pajtueshmëria, besueshmëria, shpejtësia dhe mbështetja për gjuhë të ndryshme programimi
3. Krahasoni kornizat e automatizimit falas dhe me pagesë bazuar në veçoritë, modelet e çmimeve, mbështetjen e klientit dhe miratimin e përdoruesit
4. Jepni rekomandime për organizatat që po shqyrtojnë përdorimin e kornizave të automatizimit bazuar në analizën dhe vlerësimin e kornizave të ndryshme

Qellimi I ketij studimi eshte qe te bej nje krahasim midis te githave frameworks dhe tools te siperpermendura dhe tju ndihmoj ju te beni zgjedhje cilen framework/tool te perdorni per projektin tuaj.

KAPITULLI 2 Cfare eshte testimi? (ISTQB)

Testimi i software eshte procesi I vleresimit te nje software ose sistemi per te percaktuar nese permbush kerkesat e specifikuar dhe funksionon sic duhet. Procesi perfshin verfikimin dhe validimin e aspekteve te ndryshme te nje softueri si, funksionaliteti, perdorshmeria, besueshmeria, performanca dhe siguria.

Testimi i softureit eshte nje proces shume i rendesishem ne ciklin e zhvillimit te softuerit sepse ndihmon ne identifikimin e erroreve ne softuer dhe sigurohet qe pritshmerite e perdoruesve ne lidhje me cilesine e produktit permbushen. Testimi si proces mund te jete manual, qe do te thote testerat ndjekin nje set me hapa te parapercaktuar ose automatic duke bere ekzukitimin e testeve ne nje kohe me te shpejt dhe rezultate te sakta.

Qellimi i testimit te softuerit eshte te sigurohet qe produkti funksioni sic parashikohet, eshte i lehte per tu perdorur dhe permbush nevojat e perdoruesit. Testimi mund te ndihmoje gjithashtu per te identifikuar permiresime te mundshme ne produkt dhe te sugjeroje modifikime qe mund te permiresojne cilesine e produktit ne pergjithesi.

Shkurtimisht, testimi i softuerit eshte nje proces shume kritik qe ndihmon ne sigurimin e cilesise, funksionalitetit, performances te softuerit dhe sistemeve te ndryshme. Ai ndihmon ne identifikin e defekteve dhe sigurohet qe software permbush nevojat dhe pritshmerite e perdoruesve te tij, duke e bere ate nje aspekt esencial/shume te rendesishem ne ciklin e zhvillimit te nje softueri.

TESTIMI Sipas ISTQB permbledhje 5-10 faqe

Cfare eshte testimi (permbledhur me SIPER)

Gjate 5 dekadeve te fundit nje numer i madh sugjerimesh per principet e testimit jane bere per te ofruar disa udhezimie te pergjitheshme per te gjitha llojet e testimit.

**7 Principet e testimit**

1. Testimi tregon pranine e defekteve, por jo mungesen e tyre. -> Testimi redukton mundesine e defekteve ne aplikacion, por edhe nese asnje defekt nuk eshte gjetur, testimi nuk mund të provojë saktësinë e objektit të testit.
2. Te testosh gjithcka eshte e pamundur -> Te testosh gjithcka eshte e pamundur pervec ne disa raste specifike. Fokusi kryesor duhet te jene tekninat e testimit, prioritizimi i test cases, dhe testimi I bazuar ne rrezik sesa testimi i gjithckaje.
3. Testimi i hershem kursen kohe dhe para -> Testimi i hershem qe ne fillim te procesit te zhvillimit te aplikacioneve nuk do te shkaktoje defekte te mevonshme ne produktet e punes qe pasojne. Kostoja do te reduktohet pasi do te kete me pak defekte.
4. Grumbullimi i defekteve -> Shumica e defekteve gjenden në një numër të vogël komponentesh. Sipas parimit Paereto, i njohur gjithashtu si rregulli 80/20, 20% e komponenteve përmbajnë 80% të defekteve të gjetura.
5. Kujdes nga paradoksi i pesticideve -> Nese te njejtet teste perseriten shume here, ato nuk do te jene me efektive per te zbuluar defekte te reja. Prandaj eshte e rendesishme qe tested te rishikohen dhe modifikohen vazhdimisht.
6. Testimi varet nga konteksti -> Lloje të ndryshme të aplikacioneve testohen në mënyra të ndryshme
7. Mungesa e erroreve eshte ide e gabuar -> Nëse aplikacioni ose sistemi nuk është funksional dhe nuk i përmbush kërkesat e klientit dhe nevojat e përdoruesve, atëherë gjetja dhe rregullimi e defekteve është e padobishme.

**Aktivitet dhe detyrat e testimit 1.4.2**

Nje proces testimi konsiston ne nje sere aktivitetesh:

1. Plani i testimit – Konsiston ne definimin e objektivave te testimit dhe me pas ne zgjedhjen e nje **approach**  qe na ndihmon me se miri ne arritjen e ketyre objektivave.
2. Monitorimi dhe kontrolli I testimit -> Monitorimi I testimit perfshin kontrollin e vazhdueshem te te gjitha aktiviteve te testimit dhe krahasimin e progresin aktual perkundrejt planit. Kontrolli i testimit konsisiton ne marrjen e veprimeve te nevojshme per te arritur objektivat e testimit.
3. Analiza e testimit -> Identifikimi i **features** qe jane te testueshme dhe prioritizimi i tyre. Gjithashtu dhe vleresimi I rreziqeve.
4. Dizanji I testimit -> Perpunimi I test conditions ne test cases. Identifikimi I mbulimit te testimit, teknikat e testimit, definimi I datave qe do te perdoren per testim, mjediset e testimit, infrastruktura dhe mjetet qe do te perdoren. Pyetja qe ngrihet ne kete aktivitet: Si te testojme?
5. Implementimi I testimit -> konsiston ne krijimin e nje software per ekzekutimin e testeve.
6. Ekzekutimi I testit -> Konsiston ne ekzekutimin manual ose te automatizuar te testeve. Rezultate aktuale te testimit krahason me rezultatet e pritura. Identifikohen anomalite dhe analizohen per te gjetur shkakun e tyre.
7. Perfundimi I testimit -> Ky aktivitet ndodh zakonisht pas cdo release ose iteracioni. Konsiston ne Krijimin e nje raporti I cili u dorezohet paleve te interesuara.

**Nivelet e testimit 2.2.2**

1. Component testing
2. Integration testing
3. System testing
4. Acceptance testing

**Llojet e testimit 2.3**

1. Functional Testing – Testimi Funksional
2. Non-functional Testing – Testimi jo funksional
3. White-box Testing – Testimi Kutia e Bardhe
4. Change-related Testing – Testimi I varur nga ndryshimet
5. Maintenance Testing – Testimi mirembajtes
6. Static Testing – Testimi statik

**Differences between Static and Dynamic Testing – Diferencat midis testimit statik dhe atij dinamik**

**Teknikat e dizenjemit te testeve 4.1**

1. Black-box Test Techniques – Teknina e kutise se zeze
2. White-box Test Techniques – Teknika e kuties se bardhe
3. Experience-based Test Techniques – Teknika ne baze te eksperiences

Test management – Menaxhimi i procesit te testimit

1. Test Organization – Organizimi i testimit
2. Test Planning and Estimation – Planifikimi dhe estimimi I testimit
3. 5.3 Test Monitoring and Control – Monitorimi dhe kontrolli I testimit
4. 5.4 Configuration Management – menaxhimi i konfigurimeve
5. 5.5 Risks and Testing – Rreziqet dh testimi
6. 5.6 Defect Management – Menaxhimi i defekteve

Psikologjia e testimit

1. 1.5 The Psychology of Testing

**Testimi I automatizuar INTRO dhe disa nga frameworks dhe tools me te perdorur**

Testimi i automatizuar i softuerit është procesi i përdorimit të mjeteve të specializuara softuerike për të ekzekutuar teste të parashkruara në një aplikacion ose sistem softueri, në vend të testimit manual. Ai përfshin shkrimin e skripteve ose rastet e testimit që mund të ekzekutohen automatikisht për të testuar softuerin, në vend që testuesit të kryejnë çdo rast testimi manualisht. Qëllimi i testimit të automatizuar është të përmirësojë efikasitetin, shpejtësinë dhe saktësinë e procesit të testimit, duke reduktuar përpjekjen e kërkuar për testimin manual.

Mjetet e automatizuara të testimit mund të simulojnë ndërveprimet e përdoruesve me softuerin, si futja e të dhënave, klikimi i butonave dhe lundrimi nëpër ekrane. Mjetet mund të verifikojnë më pas rezultatet kundrejt rezultateve të pritshme, të tilla si kontrolli nëse shfaqen të dhënat e sakta ose nëse mesazhet e gabimit shfaqen kur është e përshtatshme. Testimi i automatizuar mund të përdoret për një sërë llojesh testimi, duke përfshirë testimin funksional, testimin e regresionit, testimin e performancës dhe testimin e ngarkesës.

Përfitimet e testimit të automatizuar përfshijnë përmirësimin e efikasitetit të testimit, rritjen e mbulimit të testeve, reagime më të shpejta dhe reduktimin e rrezikut të gabimit njerëzor. Megjithatë, testimi i automatizuar kërkon ekspertizë në mjetet dhe kornizat e automatizimit të testimit, si dhe aftësi programimi dhe skriptimi. Është gjithashtu e rëndësishme që të dizajnohen dhe mirëmbahen me kujdes skriptet e testimit për t'u siguruar që ato të mbeten efektive dhe të sakta ndërsa softueri evoluon.

* + 1. Benefits and Risks of Test Automation - ISTQB

Automated software testing is important for several reasons:

1. Time-saving: Kursimi i kohës: Testimi i automatizuar mund të ekzekutojë testet shumë më shpejt se testimi manual, duke i lejuar testuesit të ekzekutojnë më shumë teste në më pak kohë. Kjo mundëson reagime më të shpejta dhe mund të zvogëlojë kohën në treg për produktet softuerike.
2. Consistency: Konsistenca: Testet e automatizuara kryejnë të njëjtat hapa dhe kontrolle sa herë që ekzekutohen, duke siguruar qëndrueshmëri në procesin e testimit. Kjo zvogëlon gjasat e gabimit njerëzor dhe përmirëson besueshmërinë e rezultateve të testit.
3. Coverage: Mbulimi: Testimi i automatizuar mund të mbulojë një numër të madh rastesh testimi, duke përfshirë ato komplekse që mund të jenë të vështira për t'u ekzekutuar me dorë. Kjo siguron që të gjitha aspektet e softuerit të testohen tërësisht dhe redukton rrezikun e defekteve të pazbuluara.
4. Reusability: Ripërdorshmëria: Testet e automatizuara mund të ripërdoren nëpër versione dhe ndërtime të ndryshme të softuerit, duke kursyer kohë dhe përpjekje në krijimin e testeve të reja për çdo version.
5. Cost-effective: Me kosto efektive: Ndërsa investimi fillestar në testimin e automatizuar mund të jetë më i lartë se testimi manual, testimi i automatizuar përfundimisht mund të kursejë kostot në afat të gjatë duke reduktuar nevojën për testues manualë dhe duke zvogëluar kohën dhe përpjekjen e kërkuar për testim.
6. Quality assurance: Sigurimi i cilësisë: Testimi i automatizuar mund të ndihmojë në përmirësimin e cilësisë së softuerit duke identifikuar defektet në fillim të ciklit të zhvillimit, duke reduktuar gjasat që klientët të zbulojnë defekte pas lëshimit.

Në përgjithësi, testimi i automatizuar i softuerit mund të përmirësojë efikasitetin, efektivitetin dhe cilësinë e procesit të testimit, duke e bërë atë një mjet thelbësor për ekipet moderne të zhvillimit të softuerit.

There are many popular test automation frameworks and tools available, both free and paid. Here are some examples:

**Free Test Automation Frameworks:**

1. Selenium WebDriver: Selenium WebDriver is an open-source framework used for automating web browsers. It supports multiple programming languages and has a large community of users and contributors.
2. Appium: Appium is an open-source framework used for automating mobile apps. It supports iOS, Android, and Windows platforms.
3. Robot Framework: Robot Framework is a generic test automation framework that can be used for a variety of testing types. It has a user-friendly syntax and supports multiple programming languages.
4. Cypress: Cypress is an open-source framework used for testing web applications. It offers fast, reliable, and easy-to-use testing capabilities, with a focus on end-to-end testing.

**Paid Test Automation Frameworks:**

1. TestComplete: TestComplete is a commercial tool used for automated functional testing of web, desktop, and mobile applications. It supports multiple scripting languages and has built-in test management capabilities.
2. Ranorex: Ranorex is a commercial tool used for automated testing of desktop, web, and mobile applications. It has a user-friendly interface and supports multiple programming languages.
3. Telerik Test Studio: Telerik Test Studio is a commercial tool used for automated functional testing of web and desktop applications. It has a user-friendly interface and supports multiple programming languages.
4. Katalon Studio: Katalon Studio is a commercial tool used for automated testing of web, mobile, and API applications. It has a built-in test management system and supports multiple scripting languages.

**SELENIUM dhe CYPRESS (disa info mbi APPIUM)**

**-Selenium**

Selenium is an open-source test automation framework that is widely used for automating web browsers. It was originally created by Jason Huggins in 2004 as an internal tool at ThoughtWorks, a software consultancy firm. The tool quickly gained popularity and was released as an open-source project in 2008. Since then, it has become one of the most popular test automation frameworks in the industry, with a large community of users and contributors.

Selenium provides a suite of tools for automated testing of web applications, including Selenium IDE, Selenium WebDriver, and Selenium Grid. Selenium IDE is a record-and-playback tool that allows users to create automated tests in a user-friendly interface. Selenium WebDriver is a programmatic interface for interacting with web browsers, allowing users to create more sophisticated tests using programming languages such as Java, Python, and C#. Selenium Grid is a tool for distributed testing, allowing tests to be run across multiple machines and browsers simultaneously.

One of the key benefits of Selenium is its flexibility. It supports a wide range of programming languages and can be integrated with a variety of other testing tools and frameworks. It also supports multiple browsers and operating systems, allowing tests to be run across different platforms. Additionally, Selenium is open-source and free to use, making it accessible to a wide range of organizations and individuals.

Overall, Selenium is a powerful and versatile tool for automated testing of web applications. Its popularity and community support make it a valuable asset for any testing team.

**Selenium IDE** (Integrated Development Environment) is a record-and-playback tool for creating automated tests in Selenium. It is a simple and easy-to-use interface that allows testers to record their interactions with a web application and then play back those interactions as automated tests. Selenium IDE also allows testers to manually edit and customize the recorded tests, making it a flexible and powerful tool for automated testing.

One of the key benefits of Selenium IDE is its simplicity. It is a lightweight tool that is easy to install and requires minimal configuration. It also includes a number of useful features such as element locators, assertions, and variables, making it easy to create and customize tests.

However, it is worth noting that Selenium IDE is primarily intended for creating simple and straightforward tests. For more complex tests, testers may need to use other tools in the Selenium suite, such as Selenium WebDriver.

**Selenium Grid** is a tool in the Selenium suite that allows users to run tests across multiple machines and browsers in parallel. It is designed to provide distributed testing capabilities, allowing tests to be executed across multiple machines and browsers simultaneously.

With Selenium Grid, testers can create a hub that acts as a central point for distributing tests to different nodes, which are machines or virtual machines that have browsers installed. The hub can be configured to distribute tests based on a variety of criteria, including browser type, operating system, and available resources.

One of the key benefits of Selenium Grid is its scalability. It allows testers to run tests in parallel across multiple machines and browsers, which can significantly reduce test execution times. It also provides a cost-effective solution for testing across different platforms and browsers, as it enables testers to run tests on machines with different configurations without the need for physical devices.

Selenium Grid is compatible with Selenium WebDriver, which means that tests created using WebDriver can be executed on Grid. This makes it easy for testers to scale their test automation efforts and run tests across a variety of environments.

Overall, Selenium Grid is a powerful tool for distributed testing and can significantly improve test automation efficiency and scalability. Its compatibility with Selenium WebDriver and ability to run tests in parallel across multiple machines and browsers make it a valuable asset for any testing team.

**Selenium WebDriver** is a popular open-source test automation tool that allows testers to automate web applications. It provides a simple and powerful interface for interacting with web elements such as buttons, forms, and dropdowns.

One of the key benefits of WebDriver is its ability to interact with web elements using real browsers, allowing testers to perform more realistic tests. It supports multiple programming languages such as Java, Python, and JavaScript, making it flexible and accessible to a wide range of users.

WebDriver also includes features such as element locators, waits, and assertions, which make it easy to create robust and reliable tests. It also supports headless browsers, making it possible to run tests without a user interface.

Overall, Selenium WebDriver is a powerful and versatile tool for web application testing that is widely used by testing teams around the world. Its flexibility, compatibility, and ease of use make it an ideal choice for both beginners and experienced testers alike.

**PROS/CONS SELENIUM**

Pros:

Open Source: Selenium is an open-source tool, which means it is free to use, and users have access to the source code. This makes it a cost-effective option for test automation.

Cross-Browser Compatibility: Selenium supports multiple browsers such as Chrome, Firefox, and Internet Explorer, making it easy to run tests across different platforms.

Flexibility: Selenium supports multiple programming languages such as Java, Python, and JavaScript, making it accessible to a wide range of users.

Robust Framework: Selenium provides a robust framework for web application testing, with features such as element locators, waits, and assertions.

Community Support: Selenium has a large and active community of users who provide support and share knowledge and resources.

Cons:

Limited Support for Mobile Testing: Selenium is primarily designed for testing web applications, and has limited support for mobile testing.

Requires Technical Knowledge: Using Selenium requires a certain level of technical knowledge and programming skills, which can be a barrier for non-technical testers.

Setup and Maintenance: Setting up and maintaining Selenium can be time-consuming and requires expertise in configuring the test environment.

Flakiness: Selenium tests can be prone to flakiness or inconsistency, which can be challenging to troubleshoot and fix.

Overall, while there are some challenges associated with using Selenium, its flexibility, robust framework, and community support make it a popular choice for web application testing.

Selenium komandat kryesore (Provide examples per secilin)

1. Komandat e browser
2. Locating Web Elements - Gjetja e elementeve të uebit
3. Interacting with Web Elements - Ndërveprimi me Elementet e Uebi
4. Navigation – Lundrimi/Navigimi
5. Assertions - Pohimet

get(url): Kjo komandë hap një faqe interneti në URL-në e dhënë.

back(): Kjo komandë e kthen shfletuesin një faqe prapa.

forward(): Kjo komandë e drejton shfletuesin përpara një faqe.

refresh(): Kjo komandë rifreskon faqen aktuale të internetit.

maximize\_window(): Kjo komandë maksimizon dritaren e shfletuesit.

set\_window\_size(width, height): Kjo komandë vendos madhësinë e dritares së shfletuesit në gjerësinë dhe lartësinë e specifikuar.

get\_current\_url(): Kjo komandë merr URL-në aktuale të faqes së internetit.

get\_title(): Kjo komandë merr titullin e faqes aktuale të internetit.

FirefoxDriver() -

Close() -

Quit() -

find\_element\_by\_id(): Kjo metodë gjen një element nga atributi i saj ID.

find\_element\_by\_name(): Kjo metodë gjen një element sipas atributit të emrit të tij.

find\_element\_by\_xpath(): Kjo metodë gjen një element duke përdorur një shprehje XPath.

find\_element\_by\_link\_text(): Kjo metodë gjen një element nga teksti i saktë i një lidhjeje.

find\_element\_by\_partial\_link\_text(): Kjo metodë gjen një element nga një përputhje e pjesshme e tekstit të lidhjes së saj.

find\_element\_by\_tag\_name(): Kjo metodë gjen një element me emrin e etiketës HTML.

find\_element\_by\_class\_name():Kjo metodë gjen një element me emrin e klasës së saj CSS.

find\_element\_by\_css\_selector(): Kjo metodë gjen një element duke përdorur një përzgjedhës CSS.

Këto metoda përdoren për të lokalizuar elementët e uebit në faqe bazuar në atribute të ndryshme, të tilla si ID, emri, teksti, emri i etiketës, emri i klasës ose përzgjedhësi CSS. Pasi të gjendet një element, mund të ndërveprohet me të duke përdorur komanda të tjera Selenium, të tilla si click(), send\_keys(), ose get\_attribute().

click(): Kjo metodë klikon mbi elementin.

send\_keys(): Kjo metodë dërgon tekst te elementi.

clear(): Kjo metodë pastron tekstin nga një element hyrës.

get\_attribute(): Kjo metodë kthen vlerën e atributit të specifikuar të elementit.

is\_displayed(): Kjo metodë kthen një vlerë boolean që tregon nëse elementi shfaqet aktualisht në faqe.

is\_enabled(): Kjo metodë kthen një vlerë boolean që tregon nëse elementi është aktualisht i aktivizuar.

is\_selected(): Kjo metodë kthen një vlerë boolean që tregon nëse elementi është zgjedhur aktualisht.

Këto metoda mund të përdoren për të bashkëvepruar me elementët e uebit në mënyra të ndryshme, si klikimi mbi butona, plotësimi i fushave të formularit ose marrja e informacionit nga faqja. Për të përdorur këto metoda, fillimisht duhet të lokalizoni elementin duke përdorur një nga metodat për gjetjen e elementeve të ueb-it në Selenium, si find\_element\_by\_id(), find\_element\_by\_xpath(), ose find\_element\_by\_css\_selector(). Pasi të keni gjetur elementin, mund të përdorni këto metoda për të bashkëvepruar me të.

Në testimin e automatizuar, pohimet përdoren për të kontrolluar nëse rezultatet e pritura të një testi përputhen me rezultatet aktuale. Pohimet ju lejojnë të verifikoni nëse disa kushte janë përmbushur, dhe nëse jo, testi do të dështojë.

Këtu janë disa pohime të përdorura zakonisht në Selenium:

* assertEqual(): Ky pohim kontrollon që dy vlera janë të barabarta.
* assertNotEqual(): Ky pohim kontrollon që dy vlera të mos jenë të barabarta.
* assertTrue(): Ky pohim kontrollon që një vlerë boolean është e vërtetë.
* assertFalse(): Ky pohim kontrollon që një vlerë boolean është false.
* assertIn(): Ky pohim kontrollon që një vlerë është e pranishme në një listë, tuple ose varg.
* assertNotIn(): Ky pohim kontrollon që një vlerë nuk është e pranishme në një listë, tuple ose varg.
* assertIs(): Ky pohim kontrollon që dy objekte janë i njëjti objekt.
* assertIsNot(): Ky pohim kontrollon që dy objekte nuk janë i njëjti objekt.

Pohimet janë një komponent kritik i automatizimit të testit, pasi ato ju lejojnë të siguroheni që testet tuaja po prodhojnë rezultatet e pritura. Duke përdorur pohimet, ju mund të kapni gabimet dhe defektet në fillim të procesit të zhvillimit, gjë që mund t'ju kursejë kohë dhe burime në afat të gjatë.

**-Cypress**

Cypress is a modern and popular open-source test automation framework that is designed for testing modern web applications. It was first released in 2014 by Brian Mann, Drew Lanham, and Alan Chang, and has gained significant popularity in recent years. Unlike other testing frameworks, Cypress is not based on Selenium, but rather uses a unique architecture that allows it to run tests in the same context as the application being tested.

One of the key benefits of Cypress is its ease of use. It has a simple and intuitive API that allows developers and testers to write tests using JavaScript. Cypress also includes a user-friendly dashboard that provides real-time feedback on test results, making it easy to debug and troubleshoot issues.

Another benefit of Cypress is its fast execution times. Because it runs tests in the same context as the application being tested, it is able to provide quick and reliable feedback on test results. It also includes built-in support for modern front-end technologies such as React, Angular, and Vue.js, making it ideal for testing modern web applications.

Cypress also includes a number of features that make it ideal for end-to-end testing, including support for network requests and responses, visual testing, and custom timeouts and intervals. It can also be integrated with popular testing frameworks such as Mocha and Chai.

Overall, Cypress is a powerful and user-friendly test automation framework that is ideal for testing modern web applications. Its unique architecture and fast execution times make it a valuable tool for any testing team.

Here are some Cypress commands compared to their equivalents in Selenium:

cy.get(): This is the primary command for selecting DOM elements in Cypress, similar to Selenium's findElement(). However, Cypress automatically retries the selection until the element becomes available or times out.

cy.click(): This command is similar to Selenium's click(), but it automatically waits for the element to become clickable.

cy.type(): This command is similar to Selenium's sendKeys(), but it allows you to type text into an element with more natural keyboard behavior and automatically waits for the element to become available.

cy.visit(): This command is similar to Selenium's get(), but it allows you to visit a URL and automatically waits for the page to finish loading.

cy.contains(): This command allows you to select an element by its text content, similar to Selenium's findElement(By.xpath("//\*[contains(text(),'text')]")).

cy.url(): This command allows you to retrieve the current URL of the page, similar to Selenium's driver.getCurrentUrl().

cy.wait(): This command allows you to pause the test for a specified amount of time, similar to Selenium's time.sleep().

Overall, Cypress provides a more modern and streamlined API for interacting with web elements compared to Selenium. It also has built-in support for waiting for elements to become available and handling asynchronous behavior, which can be more cumbersome in Selenium.

Here are some commands that are present in Cypress but not in Selenium:

cy.intercept(): This command allows you to intercept and modify HTTP requests and responses made by your application, which can be useful for mocking APIs and testing network-related functionality.

cy.clock(): This command allows you to control the built-in JavaScript clock in the browser, which can be useful for testing time-dependent functionality.

cy.wrap(): This command allows you to wrap a non-Cypress object or value and treat it as a Cypress object, which can be useful for integrating external libraries or utilities with your Cypress tests.

cy.fixture(): This command allows you to load and use test data from external JSON or YAML files, which can be useful for parameterizing your tests and separating data from test logic.

cy.task(): This command allows you to execute custom Node.js code as a task within your Cypress test, which can be useful for integrating with external systems or performing complex setup/teardown logic.

Overall, Cypress provides a more extensive and flexible API for testing web applications compared to Selenium, allowing developers to test a wider range of functionality with less code.

PROJEKTET – expected 15-30 pages

TEST CASES

PROJEKTI SELENIUM  
PROJEKTI CYPRESS

KRAHASIM (ex. Exe time, reports, challenges,)

**PAID AUTOMATION TOOL**

**KRAHASIM FREE dhe PAID Framework/tools – tabele krahasuese selenium/cypress vs paid tool tbd**

**Disa insights per lloje te tjera te testimi load testing, security penetration (still to be decided) qe jane pjese e testimit te app web/mobile**

REFERENCAT