Joni Lassila

iOS-mobiilisovelluksen testaus Robot Frameworkin avulla

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

28.3.2024

Tiivistelmä

Tekijä:	Joni Lassila
Otsikko:	iOS-mobiilisovelluksen testaus Robot Frameworkin avulla
Sivumäärä:	35 sivua + 6 liitettä
Aika:	28.3.2024
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaajat:	Lehtori Ilpo Kuivanen

Tämän opinnäytetyön tarkoituksena on suunnitella sekä toteuttaa testiautomaatio iOS-sovellukselle käyttäen Robot Framework -kehystä. Aiemmin sovelluksen testit on suoritettu manuaalisesti, ja niiden automatisointi voisi parantaa sovelluksen laadunvarmistusta.

Opinnäytetyön teoriaosuudessa tarkastellaan ohjelmistotestauksen perusteita, eri testaustasoja ja Robot Frameworkin roolia testauksessa. Lisäksi esitellään käytetyt kirjastot ja sovellukset, jotka auttavat testien kehittämisessä. Tämän jälkeen käydään läpi testattavaa sovellusta ja sen ominaisuuksia.

Opinnäytetyön käytännön osuudessa esitellään testausympäristön käyttöönotto, testien suunnittelu ja toteutus sovellukselle. Käyttöönotossa esitellään tarvittavat työkalut sekä niiden asennus. Testien suunnittelussa sekä toteutuksessa käydään läpi parhaimmat käytännöt sekä kuinka testeistä saadaan luotettavia ja tehokkaita. Lopuksi testien toteutuksessa analysoidaan testien tehokkuutta raportin perusteella.

Opinnäytetyön tuloksena saatiin tehokas sekä luotettava testausautomaatio iOS-mobiilisovellukselle. Testejä suorittaessa löydettiin myös sovelluksesta ohjelmointivirhe. Lisäksi tuloksena saatiin runko, johon on helppo lisätä uusia testejä uusien ominaisuuksien lisääntyessä mobiilisovellukseen.

Avainsanat: iOS-testiautomaatio, Robot Framework

Abstract

Author:	Joni Lassila
Title:	iOS Mobile Application Testing with Robot Framework
Number of Pages:	35 pages + 6 appendices
Date:	28 March 2024
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Software Engineering
Supervisor:	Ilpo Kuivanen, Senior Lecturer

The purpose of the study was to design and implement test automation for an iOS application using Robot Framework. Previously, the tests for the application were done manually, and by automating them the quality of the application could be enhanced.

The theory part of the thesis introduces the basics of software testing, different levels of testing, and the role of Robot Framework. Additionally, Robot Framework and its associated libraries and applications which assist in test development are introduced. After that, the application to be tested and its features are reviewed.

Next, the implementation of the testing environment, as well as the design and execution of tests for the application, are introduced together with the necessary tools and their installation. The paper also discusses, the best practices, as well as how to make tests reliable and efficient. The thesis provides analysis of the effectiveness of the tests based on the report.

As a result, an efficient and reliable test automation was developed for the iOS mobile application. During the testing, a programming error was also found in the application. Additionally, a test environment frame was developed, which makes it easy to add new tests as new features to the mobile application.

Keywords: iOS automated testing, Robot Framework

Sisällys

Liite 3: Sovelluksen kalasaaliin tarkastelu kartalla

Liite 4: Karttatestitiedostot

Liite 5: Lisätyn kalasaaliin testitiedostot

Liite 6: Kalasaaliin lisäys testitiedostot

Lyhenteet

iOS: *iPhone OS*. Applen iPhone-laitteissa oleva käyttöjärjestelmä.

TDD: *Test Driven Development*. Testivetoinen testausmenetelmä, jossa testit kirjoitetaan ennen varsinaista koodia.

API: *Application Programming Interface*. Ohjelmointirajapinta, jonka avulla sovellukset voivat kommunikoida toistensa kanssa.

NPM: *Node Package Manager*. Paketinhallintatyökalu JavaScript-ohjelmointikielelle.

PIP: *Package Installer for Python*. Paketinhallintatyökalu Python-ohjelmointikielelle.

1 Johdanto

Nykypäivänä mobiilisovellukset ovat monimutkaisia sekä niitä käytetään useilla eri laitteilla. Laitteiden eri kokoonpanot tuovat omat haasteensa ohjelmistotuotannon testausprosessiin. Automaatiotestaus pyrkii ratkaisemaan näitä haasteita. Tämän opinnäytetyön tarkoituksena on tarkastella ohjelmistotestausta sekä kuinka automaattista testausta voidaan soveltaa iOS-sovellukseen. Testausprosessin parantamiseksi testauksessa käytetään Robot Framework -kirjastoa, joka on valittu sen tehokkuuden sekä monipuolisuuden ansiosta.

Tutkimusongelmana on, miten voidaan kehittää luotettava ja nopea testausautomaatio iOS-sovellukselle. Testausautomaatiolla voidaan vähentää tarvetta manuaaliseen testaukseen, joka on alttiimpi inhimilliselle virheelle. Opinnäytetyön tavoitteena on suunnitella sekä luoda testiautomaatio, jonka avulla pystytään varmistamaan sovelluksen toimivuus sekä laatu ennen uusien versioiden jakamista asiakkaiden saataville.

Opinnäytetyö jakautuu teoriaosuuteen sekä käytännön osuuteen. Teoriaosuudessa käsitellään ohjelmistotestauksen periaatteita sekä Robot Frameworkin ominaisuuksia. Lisäksi teoriaosuudessa käsitellään Robot Frameworkin kanssa käytettäviä kirjastoja sekä sovelluksia, jotka auttavat testien kehittämisessä. Käytännön osuudessa käsitellään iOS-sovellusta sekä laitteita, joita käytetään testauksessa ja kuinka luomme testausympäristön. Tämän jälkeen käsitellään testien suunnittelua sovellukselle sekä niiden toteutusta. Lopuksi saatuja tuloksia tarkastellaan ja pohditaan jatkokehitysideoita sekä sitä, kuinka luotettava testausautomaatio opinnäytetyön tuloksena on saatu.

2 Ohjelmistotestaus

Ohjelmistotestaus on laaja prosessi, jonka tarkoituksena on varmistaa ohjelmiston laatu sekä sen toiminnallisuus. Testauksen tavoite on havaita ohjelmistossa mahdollisia virheitä sekä häiriöitä ennen ohjelmiston tai uuden version jakamista käyttäjien saataville. (Ohjelmistotestaus 2023.)

Ohjelmistotestauksen merkitys korostuu, kun otetaan huomioon sen vaikutus brändin maineeseen sekä ohjelmiston toimitusaikaan. Esimerkiksi Nissan-automerkki joutui takaisinkutsumaan yli miljoona autoa turvatyynyanturin vian takia. Ohjelmistovika on aiheuttanut myös 1,2 miljardin dollarin arvoisen satelliitin laukaisun epäonnistumisen. Ohjelmistoviat ovat maksaneet Yhdysvaltain taloudelle 1,1 biljoonaa dollaria, mikä on vaikuttanut 4,4 miljardiin asiakkaaseen. (IBM software testing.) Tämän perustella voimme havainnollistaa ohjelmistotestauksen tärkeyden.

Ohjelmistotestaus ei ole pelkästään koodin erilaisten metodien paluuarvojen tarkistamista, sillä voidaan myös tarkistaa ohjelmiston suorituskykyä, turvallisuutta sekä käytettävyyttä. (Ohjelmistotestaus 2023.) Testaus voi olla kallis prosessi, mutta oikeanlaisilla testaustekniikoilla sekä prosesseilla voidaan säästää miljoonia vuodessa. Kun kehityksestä jättää riittävästi tilaa testaukselle, se parantaa ohjelmiston laatua sekä asiakasodotuksia, joka johtaa mahdollisesti parempaan sovelluksen myyntiin. (IBM software testing.)

2.1 Automaatiotestaus

Automaatiotestaus on ohjelmistotyökalujen ja skriptien käyttöä ihmisen ohjaaman manuaalisen prosessin automatisoimiseksi. Automaatiotestausta harjoitetaan yleisesti ketterissä ohjelmistoprojekteissa alkuvaiheesta lähtien. (Rehkopf 2023.) Ketterä menetelmä on projektinhallinnan lähestymistapa, joka keskittyy asiakasyhteistyöhön, jatkuvaan parantamiseen sekä joustavuuteen. Ketterä menetelmä luotiin vastauksena vesiputousmallin rajoituksiin. Rajoitukset sisälsivät pitkiä kehityssyklejä ja jäykkiä prosesseja. Ketterässä menetelmässä

painotetaan iteratiivista kehitystä eli jatkuvasti parantaen sekä kykyä vastata muuttuviin ohjelmistovaatimuksiin. Suosituin ketterä menetelmä on nimeltään Scrum, jonka tarkoituksena on jakaa projektit sprintteihin eli lyhyisiin iteraatioihin. Sprintit kestävät yleensä kaksi tai neljä viikkoa. Sprinttien aikana on päivittäin tilannekatsaus, jonka tarkoituksena on käydä läpi päivän tilanne kaikkien tiimin jäsenien osalta. Tilannekatsaukset parantavat projektin etenemistä sekä varmistavat tehokkaan viestinnän ja jatkuvan parantamisen. (The Agile Coach 2024.)

Manuaalisen testauksen ollessa normi, laadunvarmistustiimien palkkaaminen oli yleistä ohjelmistoyrityksissä. Tämän tiimin tehtävänä oli testisuunnitelmien luominen askel askeleelta eteneviksi tarkastuslistoiksi, joiden avulla ohjelmistoprojektin ominaisuuksien toimivuus varmistetaan. Projektiin tulleiden päivitysten yhteydessä tarkastuslista käytäisiin manuaalisesti läpi ja tulokset palautettaisiin ohjelmointitiimille esille tulleiden ongelmien korjaamiseksi. (Rehkopf 2023.) Tämä prosessi on virhealtis, hidas sekä kallis, jonka takia prosessi halutaan automatisoida. Automaattisessa testauksessa vastuu siirtyy ohjelmointitiimille, jolloin testisuunnitelmat kehitetään rinnakkain tuotekehityssuunnitelman kanssa ja suoritetaan automaattisesti projektin edetessä. (Rehkopf 2023.)

Kaikki testit, jotka voidaan automatisoida, tulisi automatisoida. Mutta on tilanteita, joissa automatisoitu testi ei ole sen arvoista tai edes mahdollista suorittaa. Kyseisiä tilanteita, joissa testaus on tehtävä manuaalisesti, ovat esimerkiksi tutkivatestaus (engl. Exploratory testing), visuaalinen regressiotestaus (engl. Visual regression testing) sekä käyttäjäkokemustestaus (engl. User experience testing). Yleisimmät testausmenetelmät, joita halutaan automatisoida ovat yksikkötestaus, regressiotestaus sekä hyväksymistestaus. (Rehkopf 2023.) Näistä testitapauksesta kerrotaan lisää seuraavissa kappaleissa.

Tutkivassa testauksessa ohjelmaa tutkitaan matkimalla loppukäyttäjän toimintaa ja etsitään bugeja sekä virheitä ilman skriptien tai testitapauksien käyttöä. Tutkivassa tapauksessa käydään läpi neljä erilaista vaihetta. Ensimmäisessä vaiheessa opitaan tuote, sillä sovellukset saattavat vaatia opetusta, ennen kuin

tutkiva testaus on tehokasta. Toisessa vaiheessa käydään tulosten oppiminen, eli halutaan tietää mitä sovelluksen tulisi palauttaa käyttäjälle. Kolmannessa vaiheessa sovellusta suoritetaan, tarkkaillaan, tutkitaan ja kootaan raportoitu dokumentti tuloksista. Neljännessä vaiheessa kaikki edellä mainitut vaiheet toistetaan uudestaan, jolloin jokaisella kierroksella tulisi uutta tietoa mahdollisista ongelmista. (Sachedina 2017.)

Visuaalisessa regressiotestauksessa testaajat varmistavat, että sovellukseen tehdyt muutokset eivät vaikuta sovelluksen käyttöliittymän visuaaliseen ulkonäköön. Kyseistä testausta voidaan automatisoida, mutta jos varmistus tehdään koneellisesti pikseli pikseliltä, tulokset voivat olla virheellisiä. (Shain 2022.) Sovelluksen tarkistus pikseli pikseliltä voi olla mahdoton, jos sivun kuvat tai tekstit vaihtuvat useasti kuten uutisointi sivuilla. Kyseiseen ongelmaan on kehitetty visuaalinen tekoäly, joka tunnistaa visuaaliset elementit sivulla värin, sisällön, koon ja sijainnin perusteella. Jokaisen testiympäristön muutoksessa tallentuu kuvankaappaus sovelluksesta, jota malli käyttää vertailun lähtökohtana. (Ericsson 2022.)

Käyttäjäkokemuksen testausta käytetään sovelluksen käyttäjäystävällisyyden arviointiin. Käyttäjäkokemus testausta ei voida automatisoida, sillä kyseinen testausmenetelmä vaatii useita erilaisia ihmisiä arvioimaan sekä testaamaan sovellusta, jotta löydetään mahdolliset ongelmakohdat sovelluksen käytössä. (Mohit 2023.)

2.2 Yksikkötestaus

Yksikkötestauksessa keskitytään sovelluksen pienten osien, kuten yksittäisten funktioiden tai luokkien toiminnallisuuden testaamiseen. Tämä menetelmä auttaa havaitsemaan ohjelmointivirheet mahdollisimman varhaisessa vaiheessa. (Ohjelmistotestaus 2022.) Yksikkötestit tulisi ajaa aina, kun ohjelmistokoodiin tehdään muutoksia. Mikäli testi epäonnistuu, on koodin ongelma-alueen löytäminen nopeaa. (Amazon Unit Testing 2023.)

Yksikkötestauksessa perustekniikoita, joiden avulla voidaan varmistaa kattavat testitapaukset, ovat loogiset tarkastukset, rajatarkastukset, virheenkäsittely sekä olio-ohjelmointitarkastukset. Loogisessa tarkastuksessa tarkistetaan, suorittaako järjestelmä oikeat laskelmat ja meneekö ohjelma seuraavissa vaiheissa oikeisiin sijainteihin odotetuilla syötteillä. Rajatarkistuksissa tarkistetaan, miten ohjelmisto reagoi tyypillisiin, virheellisiin sekä rajatapaussyötteisiin. Rajatapaussyötteellä varmistetaan, että ohjelmisto onnistuu käsittelemään äärimmäisiä tai epätavallisia syötteitä. Kyseisellä syötteellä tarkoitetaan sellaista, joka sijoittuu ohjelman syötteiden hyväksyttyjen arvojen rajoille. Esimerkiksi, jos ohjelmisto odottaa syötteenä kokonaislukua väliltä 1-10, rajatapaussyötteitä olisivat 1 ja 10, sekä arvot näiden ulkopuolelta kuten 0 ja 11. Virheenkäsittely tarkoittaa sitä, kuinka ohjelmisto reagoi virheellisiin syötteisiin. Olio-ohjelmointitarkastuksessa tarkistetaan, että olioiden tila päivittyy oikein, jos niitä muutetaan ohjelman ajon aikana. (Amazon Unit Testing 2023.)

Yksikkötestauksessa käytetään useasti TDD-lähestymistapaa eli testivetoista kehitystä (engl. test-driven development), jonka tarkoituksena on kirjoittaa testit ennen varsinaisen koodin kirjoittamista. Tämän lähestymistavan tarkoitus on luoda ensimmäisenä testit, jotka aluksi epäonnistuvat. Tämän jälkeen kirjoitetaan koodi, jotta testi menee läpi ja toimii halutulla tavalla. Tämän avulla ohjelmoijan pitää havainnollistaa koodin paluuarvot sekä syötteet, jotka parantavat koodin laatua sekä vähentävät virheiden esiintymistä. (Sonmez.)

Yksikkötestauksen isoimmat hyödyt ovat tehokas virheiden löytäminen ja dokumentointi. Yksikkötestit ajetaan aina kun koodiin tehdään muutoksia, jolloin virheet paikannetaan nopeasti. Yksikkötestit toimivat myös osana koodin dokumentointia, sillä testien avulla muut kehittäjät näkevät, mitä koodilta odotetaan sen suorittaessaan ja mitä arvoja se palauttaa. (Amazon Unit Testing 2023.)

2.3 Regressiotestaus

Regressiotestauksen tavoitteena on varmistaa, että uudet ominaisuudet ohjelmistossa eivät riko vanhoja jo entuudestaan testattuja ja toimivia ominaisuuksia.

Tämä on erityisen tärkeää nykypäivänä, sillä ketterien menetelmien ansiosta ohjelmistot kokevat jatkuvia muutoksia tiheään tahtiin. Muutoksia tehdessä on tärkeää varmistaa, että ne eivät aiheuta sivuvaikutuksia muihin ominaisuuksiin. Regressiotestaus on suunniteltu kyseistä ongelmaa varten. (Srinivasan & Gopalaswamy: 194.)

Regressiotestaus jaetaan yleisesti kahteen eri osaan, säännölliseen ja lopulliseen regressiotestaukseen. Säännöllistä regressiotestausta suoritetaan projektin aikana useasti varmistamaan, että vikakorjaukset ja uudet toiminnallisuudet ovat kunnossa. Lopullinen regressiotestaus tehdään ennen ohjelmiston julkaisua asiakkaille. Kyseiseen lopputestaukseen määritellään sopiva regressiotestien sarja. Tätä sarjaa toistetaan jatkuvasti tietyn määritellyn ajan, sillä jotkin ongelmat kuten muistivuodot voivat tulla esille vasta, kun ohjelmistoa on käytetty tietyn ajan. Tämän ansiosta voidaan varmistaa, että asiakas saa toimivan ohjelmiston. (Srinivasan & Gopalaswamy: 195.)

Regressiotestausta voidaan suorittaa manuaalisesti, mutta ohjelmiston testitapausten kasvaessa regressiotestitapausten manuaalisesti hallitseminen ei ole käytännöllistä. Automaattisen regressiotestauksen suurin hyöty on se, että se vapauttaa ohjelmoijien resursseja. Tämän ansiosta he voivat keskittyä enemmän monimutkaisiin ja vaativiin tehtäviin, jotka parantavat ohjelmistojen laatua. (Homann.)

2.4 Hyväksymistestaus

Hyväksymistestauksen tarkoituksena on testata ohjelmisto sen hyväksyttävyyden varmistamiseksi. Kyseisen testin tarkoituksena on arvioida, vastaako ohjelmisto liiketoimintavaatimuksia ja onko se hyväksyttävässä kunnossa. Testauksessa käytetään yleensä musta laatikko -testausmenetelmää, jonka tarkoituksena on testata ohjelmisto ilman tietoa sen rakenteesta tai toteutuksesta. Tällöin testit luodaan asiakkaan näkökulmasta. (Acceptance Testing 2022.)

Hyväksymistestaus voidaan suorittaa joko sisäisesti, ulkoisesti tai asiakkaan toimesta. Sisäisen hyväksymistestauksen suorittavat ohjelmiston kehittäneen yrityksen jäsenet, jotka eivät ole suoraan osallisia projektissa. Ulkoisen testauksen suorittavat ohjelmiston kehittäneen yrityksen ulkopuolinen toimija. Asiakkaan hyväksymistestauksen suorittavat ohjelmiston kehittäneen yrityksen asiakas. (Acceptance Testing 2022.)

Hyväksymistestaus suoritetaan yleensä aina ohjelmiston elinkaarenlopussa, kun kehitys- ja testausvaiheet on suoritettu. Tässä vaiheessa asiakas päättää, hyväksytäänkö ohjelmisto käyttöön vai hylätäänkö se. (Acceptance Testing 2022.)

3 Testausteknologia

Testausteknologian valinta on tärkeä päätös ohjelmistokehityksessä, varsinkin mobiilisovelluksissa, joissa laitteiden monimuotoisuus asettaa korkeita vaatimuksia. Tämä luku keskittyy siihen, kuinka oikein valituilla teknologioilla voidaan luoda nopeasti testiautomaatio mobiilisovellukseen.

3.1 Robot Framework

Robot Framework on Python-pohjainen avoimen lähdekoodin automaatiokehys, jota käytetään testiautomaatiossa. Sen kehitys alkoi vuonna 2005 Nokia Networksille. Ensimmäinen versio Robot Frameworkista julkaistiin samana vuonna, ja sen käyttö alkoi levitä nopeasti. Suosion ansiosta Robot Framework julkaistiin avoimena lähdekoodina vuonna 2008 kaikkien käyttöön. (Robot Framework: past, present and future 2016.)

3.1.1 Ominaisuudet ja käyttö

Robot Framework käyttää avainsanapohjaista lähestymistapaa, joka tarjoaa selkokielisen ja helposti ymmärrettävän syntaksin. Tämän lähestymistavan avulla

testitapaukset määritellään käyttäen selkeitä ja kuvailevia avainsanoja, mikä tekee testeistä helposti ymmärrettäviä ja ylläpidettäviä. (Robot Framework.)

Robot Framework -testejä kirjoitetaan ".robot" päättyviin tiedostoihin. Testitiedoston rakenne koostuu "Settings"-osiosta, "Variables"-osiosta ja lopuksi "Test Cases"-osiosta. Robot Framework -testejä voidaan käynnistää käyttämällä “robot exampleTest.robot" -nimistä komentoa. Komennossa "exampleTest.robot" on testitiedosto, joka halutaan käynnistää. (Robot Framework.)

Settings eli asetukset -kohdassa määritellään testeissä käytettävät kirjastot sekä mahdolliset resurssitiedostot. Resurssitiedosto on erillinen aputiedosto, johon voidaan listata esimerkiksi kaikki testeissä käytettävät kirjastot, jolloin jokaiseen testitiedostoon ei tarvitse erikseen listata tarvittavia kirjastoja. Lisäksi resurssitiedostossa voidaan määritellä avainsanoja, joita voidaan testeissä hyödyntää. Näiden tarkoituksena on luoda organisoitu rakenne projektiin sekä mahdollisuus avainsanojen uudelleenkäyttöön eri Robot Framework -testitiedostoissa. (Robot Framework.)

Variables eli muuttujat -kohtaan voidaan määritellä testitapauksissa käytettävät muuttujat. Muuttujien avulla voidaan helposti vaihtaa eri ympäristöihin, esimerkiksi käyttämällä nettisivu- tai käyttäjätunnusmuuttujaa. Tämän avulla testeissä käytettävät muuttujat voidaan nopeasti ja helposti vaihtaa ilman koodin läpikäymistä. (Robot Framework.)

Test Cases eli testitapaukset ovat varsinaisten testitapausten osio, jotka suoritetaan järjestyksessä ylhäältä alaspäin. Jokaiseen testitapaukseen on mahdollista lisätä "Setup", joka suoritetaan testin alussa, sekä "Teardown", joka suoritetaan testin lopuksi. Näiden tarkoituksena on alustaa sekä palauttaa sovellus haluttuun kohtaan seuraavia testejä varten. (Robot Framework.)

Robot Framework testitapauksia voidaan myös eritellä käyttäen "[Tags]"-merkintää. Näiden merkintöjen tarkoitus on eritellä testit eri kategorioihin, jotta tiettyjä toiminnallisuuksia voidaan testata ilman, että kaikkia testejä tarvitsisi käydä läpi. Esimerkiksi käyttämällä "Profile"-merkintää voimme kategorisoida, että

kyseinen testi varmistaa sovelluksen profiilisivun toiminnallisuuksi. Tällainen testisetti voitaisiin käynnistää käyttämällä "robot -include Profile example.robot"-komentoa. Haluttu "Tag"-merkintä lisätään "-include"-kohdan jälkeen. (Robot Framework.)

3.1.2 Robot Framework hyödyt verrattuna XCUITest-työkaluun

XCUITest on Applen Kehittämä kehys iOS-sovellusten käyttöliittymätestaukseen. XCUITest-kehys toimii samanlailla kuin Robot Framework eli mahdollistaa testien kirjoittamisen. Se automatisoi käyttäjän vuorovaikutuksen sovelluksen kanssa. (Akshay 2023.)

Robot Frameworkin käyttää avainsanapohjaista lähestymistapaa; XCUITest käyttää testien kirjoittamiseen Swift- ja Objective-C-ohjelmointikieltä.

Esimerkkikoodin 1 avulla havainnollistamme avainsanapohjaisen syntaksin sekä kuinka Robot Frameworkin avulla profiilinimen vaihtamisen toimivuus voidaan todentaa.

```
*** Settings ***
Library   AppiumLibrary
Resource  resources.robot
Suite Setup         Open App
Suite Teardown      Logout And Close App

*** Variables ***
${name}   Joni Lassila

*** Test Cases ***
Updating Profile Name Should Succeed
     [Documentation] Navigates to profile page and changes profile name
     to random string, after that changes back to default $name.
     [Tags] Regression

     Open Profile Page
     Change Profile Name     ${name}
     Open Profile Page
     Verify Profile Name Change   ${name}
```

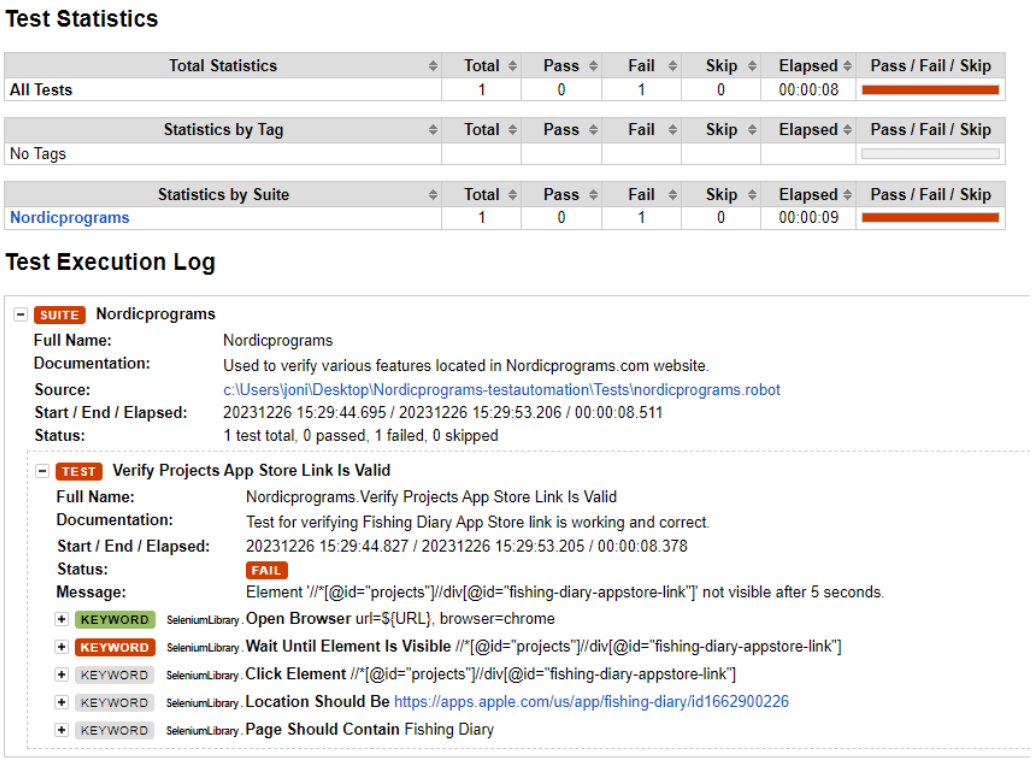
Esimerkkikoodi 1. Robot Framework testi profiilinimen päivittämiseen. Koodi käyttää hyödykseen AppiumLibrary-kirjastoa ja sisältää vaiheet profiilisivulle siirtymiseen, nimen vaihtamiseen satunnaiseen merkkijonoon ja takaisin alkuperäiseen nimeen sekä varmistaa muutokset.

Esimerkkikoodin 2 avulla havainnollistetaan, kuinka Applen kehittämällä XCUI-Test-kirjastolla voidaan todentaa sama profiilinimen vaihto-ominaisuus. Näiden esimerkkien perusteella voidaan todeta, että Robot Frameworkilla kirjoitetut testit ovat helpommin ymmärrettäviä sekä luettavia. Lisäksi Robot Frameworkin avainpohjainen syntaksi mahdollistaa avainsanojen käytön muissa testeissä.

```
import XCTest

class XCUITest: XCTestCase {
override func setup() {
super.setUp()
XCUIApplication.launch()
}

func updateProfileName() {
let app = XCUIApplication()
app.buttons["profile"].tap()
app.buttons["editProfile"].tap()
app.textFields["profileTextField"].tap()
app.textFields["profileTextField"].typeText("Joni Lassila")
app.buttons["saveButton"].tap()
XCTAssert(app.statisticTexts["Joni Lassila"].exists)
}
```

Esimerkkikoodi 2. XCUITest-kirjastolla tehty testi profiilinimen vaihtamisen todentamiseen.

Robot Framework testien ajamisen jälkeen muodostuu automaattisesti HTML-raportti, joka voidaan nähdä kuvassa 1. Tulokset sisältävät kaikki ajetut testit sekä kyseisten testien tilanne, eli menikö testi läpi vai ei. Raportista voidaan myös nähdä mihin avainsanaan kyseinen testi epäonnistui. Kuvassa 1 oleva testi epäonnistui, sillä annettua elementtiä ei löytynyt nettisivulta. Epäonnistunut avainsana on merkitty punaisella, ja tässä tapauksessa se on "Wait Until Element Is Visible". Testiraportti tarjoaa lisäksi jokaiseen avainsanaan ja testiin käytetyn ajan, jonka avulla testejä voidaan optimoida nopeammaksi.



Test Statistics

Total Statistics	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
All Tests	1	0	1	0	00:00:08	

Statistics by Tag	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
No Tags						

Statistics by Suite	Total	Pass	Fail	Skip	Elapsed	Pass / Fail / Skip
Nordicprograms	1	0	1	0	00:00:09	

Test Execution Log

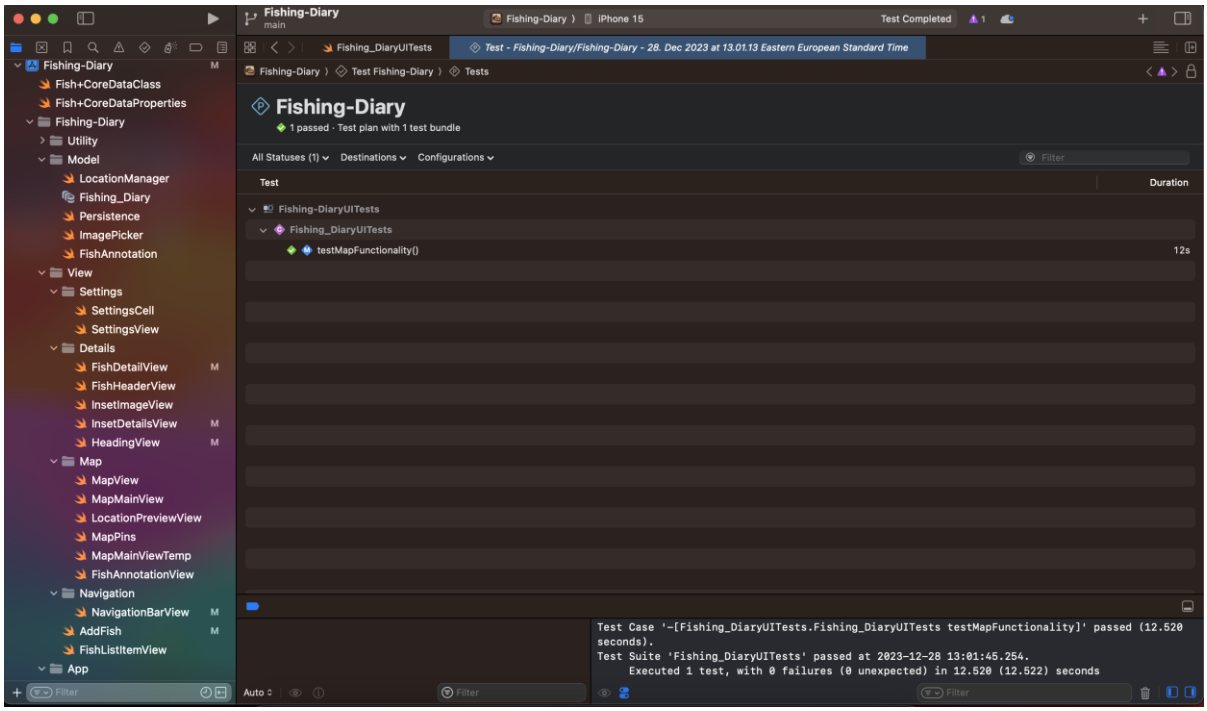
SUITE Nordicprograms
Full Name: Nordicprograms
Documentation: Used to verify various features located in Nordicprograms.com website.
Source: c:\Users\joni\Desktop\Nordicprograms-testautomation\Tests\nordicprograms.robot
Start / End / Elapsed: 20231226 15:29:44.695 / 20231226 15:29:53.206 / 00:00:08.511
Status: 1 test total, 0 passed, 1 failed, 0 skipped

TEST Verify Projects App Store Link Is Valid
Full Name: Nordicprograms.Verify Projects App Store Link Is Valid
Documentation: Test for verifying Fishing Diary App Store link is working and correct.
Start / End / Elapsed: 20231226 15:29:44.827 / 20231226 15:29:53.205 / 00:00:08.378
Status: FAIL
Message: Element '//*[@id="projects"]//div[@id="fishing-diary-appstore-link"]' not visible after 5 seconds.

KEYWORD SeleniumLibrary . **Open Browser** url=${URL}, browser=chrome
KEYWORD SeleniumLibrary . **Wait Until Element Is Visible** //*[@id="projects"]//div[@id="fishing-diary-appstore-link"]
KEYWORD SeleniumLibrary . **Click Element** //*[@id="projects"]//div[@id="fishing-diary-appstore-link"]
KEYWORD SeleniumLibrary . **Location Should Be** https://apps.apple.com/us/app/fishing-diary/id1662900226
KEYWORD SeleniumLibrary . **Page Should Contain** Fishing Diary

Kuva 1. Robot Framework -raportti testistä.

XCUITest-testien ajamisen jälkeen muodostuu raportti, joka voidaan nähdä kuvassa 2. Tulokset sisältävät kaikki ajettujen testien tulokset sekä kuinka kauan kyseisessä testissä kului aikaa. Tämän avulla voidaan havainnollistaa, että Robot Framework tarjoaa kattavamman sekä selkeämmän testiraportin kuin XCUITest. XCUITest-kehys ei generoi HTML-raporttia, mutta se on mahdollista, kun käytetään kolmannen osapuolen työkaluja.



Kuva 2. XCUITest-raportti testistä.

Robot Framework valikoitui XCUITest-kirjaston sijasta opinnäytetyön automaatiotyökaluksi, sillä se tarjoaa useita merkittäviä etuja laadun ja tehokkuuden näkökulmasta. Isoin etu on Robot Frameworkin ollessa avainsanapohjainen. Testien kirjoittaminen on helppoa ja tehokasta.

Robot Frameworkin laajat kirjastot mahdollistavat monipuolisten testien luomisen. Lisäksi Robot Framework tarjoaa mahdollisuuden laajentaa ja räätälöidä testitapauksia tarpeen mukaan käyttäen Python-ohjelmointikieltä, mikäli kirjastoista puuttuu tarvittava avainsana. Yksi tärkeimmistä ominaisuuksista on Robot Frameworkin laaja raportti, jonka ansiosta testien optimoinnista sekä virheiden havaitsemisesta on tehty helppoa.

3.1.3 Laajennettavuus

Robot Framework tarjoaa rikkaan ekosysteemin automaatiokehittäjille, jotka ratkaisevat yleisimmät automaatio-ongelmat. Robot Frameworkin avainsanoja voidaan kuitenkin laajentaa käyttäen Python-ohjelmointikieltä, joita ei ole saatavilla

Robot Frameworkin sisäänrakennetuissa kirjastoissa. Esimerkkikoodi 3 on esimerkki, kuinka Python-ohjelmointikieltä voidaan hyödyntää luomaan erillinen avainsana käyttäjän iän laskemiseksi. (Robot Framework.)

```
def calculate_user_age(date):
    birthdate = datetime.strptime(birthdate, "%d-%m-%Y")
    today = datetime.today()
    age = today.year - birthdate.year - ((today.month, today.day) <
(birthdate.month, birthdate.day))
    return age
```

Esimerkkikoodi 3. Esimerkki Robot Frameworkin avainsanojen laajentamisesta käyttäen Python-ohjelmointikieltä.

Tämän jälkeen Python-ohjelmointikielellä luotua avainsanaa voidaan käyttää Robot Framework -testissä laskemaan käyttäjän ikä sekä kirjoittamaan sen ohjelmiston lomakkeeseen. Kyseinen avainsana voidaan nähdä Esimerkkikoodissa 4.

```
*** Test Cases ***
Example Test
     ${age}=   Calculate User Age 01-01-2001
     Input Text    age-field     ${age}
```

Esimerkkikoodi 4. Robot Framework testi, joka käyttää Esimerkkikoodissa 3 olevaa Python-funktiota.

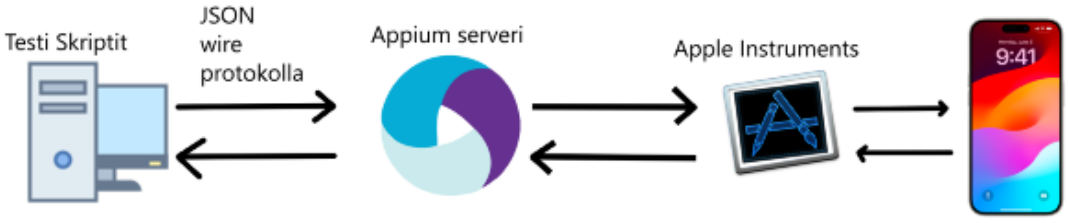
3.2 AppiumLibrary

AppiumLibrary on mobiilisovellusten testaamiseen tarkoitettu kirjasto, joka tarjoaa Robot Frameworkille korkean tason avainsanoja. Kirjaston tarjoamat avainsanat mahdollistavat sovelluksen hallinnan sekä toimintojen testaamisen. Tällaisia avainsanoja ovat esimerkiksi "Activate Application", jonka tarkoituksena on avata määritetty sovellus sekä "Click Element", jonka tarkoituksena on esimerkiksi painaa tiettyä määriteltyä nappia. (AppiumLibrary.) Tämän kirjaston avulla nopeutamme sekä helpotamme testien luomista mobiilisovellukselle valmiiden avainsanojen ansiosta.

3.3 Appium

Appium on avoimen lähdekoodin automaatiotyökalu, joka on suunniteltu mobiilisovellusten testaamiseen sekä Android- ja iOS-alustoilla. Appium on alustariippumaton. Se käyttää yhtä API:a, minkä ansiosta sama testi voidaan suorittaa useilla eri alustoilla. (Appium.) Appium on node.js-pohjainen HTTP-palvelin, joka tarjoaa REST API -rajapinnan. Se hallinnoi pyyntöjä alustan mukaan kommunikoimalla JSON wire -protokollan mukaisesti. (Manoj 2015: 16.)

Kuvassa 3 esitetään Appiumin testausarkkitehtuuri iOS-laitteille. Testiskriptit suoritetaan lähettämällä JSON-muotoisia komentoja Appium-palvelimelle käyttäen JSON wire -protokollaa. Appium-palvelin lähettää komentoja instrumentille mahdollistaen käyttöliittymän hallinnoimisen. Appium toimii viestintämenetelmänä Robot Frameworkin ja testilaitteiden välillä.



Kuva 3. Appium-testausarkkitehtuuri iOS-laitteille.

4 Testattava sovellus

Opinnäytetyössä testattava sovellus on nimeltään Fishing Diary, joka on iOS:lle kehitetty kalastuspäiväkirja. Sovelluksessa käyttäjä voi kirjata talteen kalasaaliin tiedot, kuten painon, kuvan, mitan ja kalastuspaikan (liite 1). Näitä tietoja käyttäjä pääsee tarkastelemaan myöhemmin (liite 2). Kalastuspaikka tallentuu sovelluksen kartalle, jotta käyttäjä voi löytää saman kalastuspaikan uudelleen (liite 3). Opinnäytetyössä toteutetaan automaatiotestaus edellä mainituille ominaisuuksille.

Fishing Diary on kehitetty Swift-ohjelmointikielellä, joka on Applen kehittämä ohjelmointikieli. Sovellus hyödyntää Core Dataa, jonka avulla tiedot tallennetaan käyttäjän kännykkään ilman tarvetta ulkoiselle tietokannalle. (Core Data.)

5 Testausympäristön kehittäminen

Opinnäytetyön luvussa 3 käsiteltiin testausteknologiaa teoreettisesti. Tämän luvun tarkoituksena on siirtyä konkreettiseen toimintaan, eli kuinka opinnäytetyössä käytettävä testausympäristö otetaan käyttöön iOS-mobiilisovelluksen testausta varten. Testausympäristö luodaan macOS-käyttöjärjestelmälle, joka on välttämätön iOS-sovellusten testien ajamiseen sekä luomiseen. Testien suorittaminen tapahtuu fyysisellä iOS-laitteella.

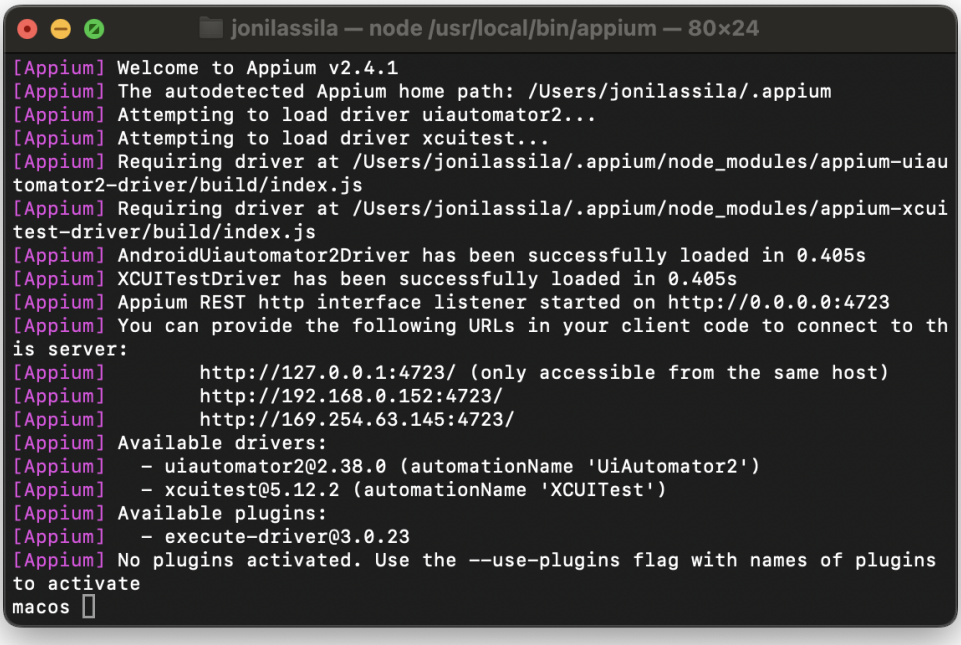
Luvun aluksi käsitellään miten Robot Framework sekä siihen vaadittavat ohjelmistot asennetaan. Lisäksi käsittelemme Visual Studio Code -tekstieditoria sekä siihen vaadittavia laajennuksia, jotka tarjoavat hyödyllisiä lisäominaisuuksia testien kirjoittamiseen. Lopuksi tarkastelemme, kuinka Appium Inspector otetaan käyttöön ja kuinka se toimii.

5.1 Robot Frameworkin ja kirjastojen asennus

Opinnäytetyön testausympäristön pystyttämisessä on ensimmäisenä Robot Frameworkin sekä vaadittavien kirjastojen asennus. Robot Frameworkin käyttö edellyttää Python-ohjelmointikielen asentamista, jonka asennus onnistuu virallisilta nettisivuilta. Python-asennuspaketti sisältää automaattisesti PIP-paketinhallintatyökalun, joka on välttämätön tarvittavien ohjelmistojen ja kirjastojen asennuksessa. Robot Frameworkin asennus onnistuu komennolla "pip install robotframework".

Puhelinsovelluksen testiautomaatioon hyödynnetään AppiumLibrary-kirjastoa, jonka asentaminen onnistuu komennolla "pip install robotframework-appiumlibrary". Kirjasto vaatii Appium-työkalun laitteiden kommunikointiin. Appium on Node.js-pohjainen työkalu, joka tulee asentaa ensimmäisenä. Tämän jälkeen

Appium asennetaan komennolla "npm install -g appium". Appium-palvelimen käynnistäminen onnistuu komennolla "appium". Käynnissä oleva palvelin voidaan nähdä kuvassa 4.



Kuva 4. Käynnissä oleva Appium-palvelin.

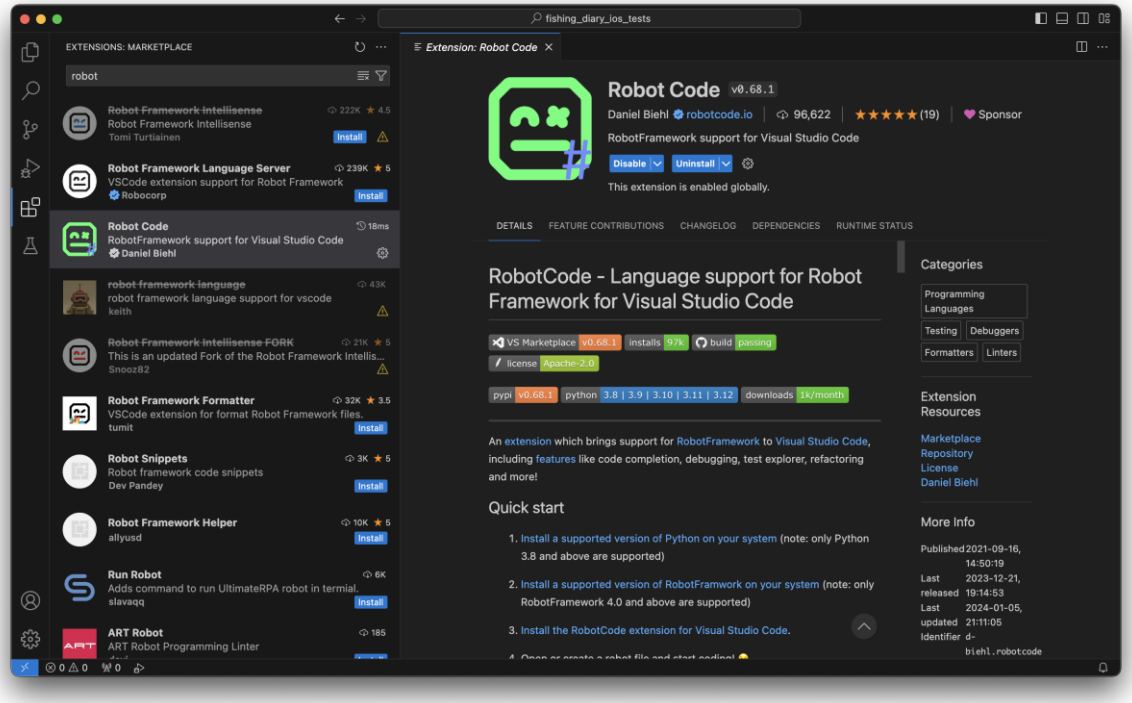
Appiumin kommunikointi iOS-laitteiden kanssa vaatii XCUITest-ajurien asennuksen. Ajurien asennus onnistuu komennolla "appium install xcuitest".

5.2 Koodieditorin valmistelu

Testien kirjoittamiseen on tarjolla useita erilaisia koodieditoreja, mutta opinnäytetyöhön on valikoitunut Microsoft Visual Studio Code -tekstieditori. Editori on valikoitunut helppokäyttöisyyden ja laajennettavuuden ansiosta. Laajennuksilla tarkoitetaan editoriin lisättäviä lisäominaisuuksia, joita ei ole entuudestaan

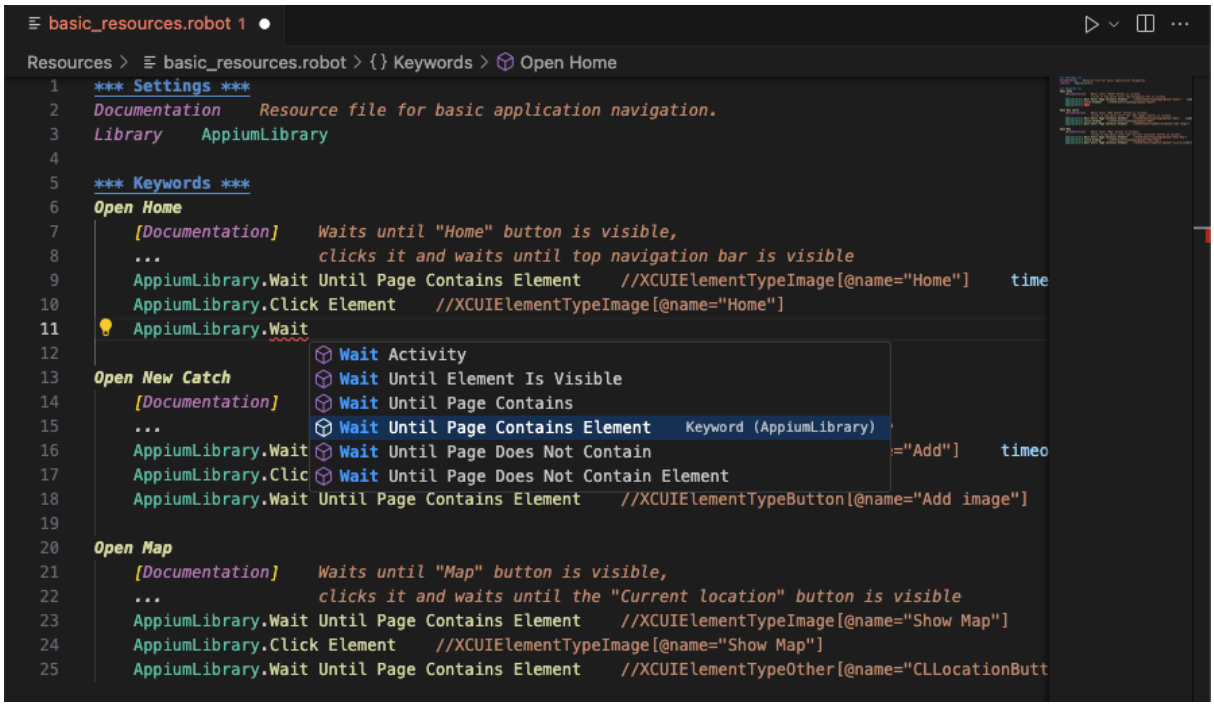
asennettu kyseiseen editoriin. Laajennuksien avulla editori voidaan räätälöidä omiin tarpeisiin, kuten Robot Framework testien kirjoittamiseen sekä ajamiseen.

Microsoft Visual Studio asennuksen jälkeen laajennuksiin lisättiin "Robot Code" (kuva 5), joka on tarkoitettu Robot Framework testien kirjoittamiseen. Laajennus tarjoaa lisäominaisuuksia, joita ovat koodin automaattinen täydennys, vianetsintätoiminto ja koodin navigointi.



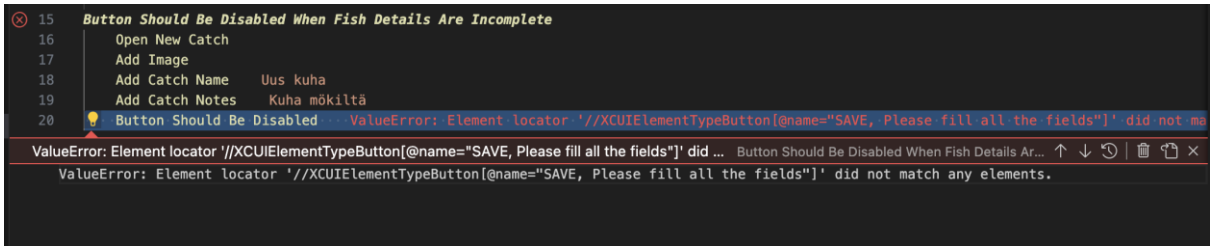
Kuva 5. Robot Code -laajennus Visual Studio Codessa.

Automaattisella täydennyksellä koodieditori ehdottaa avainsanoja, kun käyttäjä aloittaa kirjoittamisen. Tämä ominaisuus nopeuttaa testien kirjoittamista ja vähentää avainsanojen kirjoitusvirheitä. Kyseinen toiminto voidaan nähdä kuvassa 6, jossa toiminto ehdottaa kaikkia "Wait"-sanalla alkavia avainsanoja. Koodin navigointiominaisuus mahdollistaa siirtymisen avainsanan määrittelyyn klikkaamalla avainsanaa. Kyseinen ominaisuus helpottaa avainsanojen löytämistä mahdollisten virheiden korjaamista varten ja uusien lisäominaisuuksien lisäämisen.



Kuva 6. Visual Studio Code -koodin automaattinen täydennys.

Laajennuksen tarjoama vianetsintätoiminto on tärkeä osa testien virheiden korjaamisessa. Testin epäonnistuessa editori näyttää rikkinäisen avainsanan sekä virheilmoituksen. Tämä voidaan nähdä kuvassa 7, jossa tekstieditori osoittaa rikkinäisen avainsanan ja syyn. Tässä tapauksessa "Button Should Be Disabled" -avainsana on rikki, sillä määriteltyä elementtiä ei ole löytynyt. Tämä ominaisuus nopeuttaa sekä helpottaa testien vikojen havaitsemista, sillä ongelman näkee suoraan editorista eikä tarvitse avata testistä erikseen luotua HTML-raporttia.



Kuva 7. Robot Code -vianetsintätoiminto Visual Studio Codessa.

Laajennuksen asennuksen jälkeen Visual Studio Code on valmis testien kirjoittamista varten. Edellä mainitun laajennuksen ominaisuuksien avulla testien kirjoittaminen sekä mahdollisten virheiden löytäminen on helpompaa.

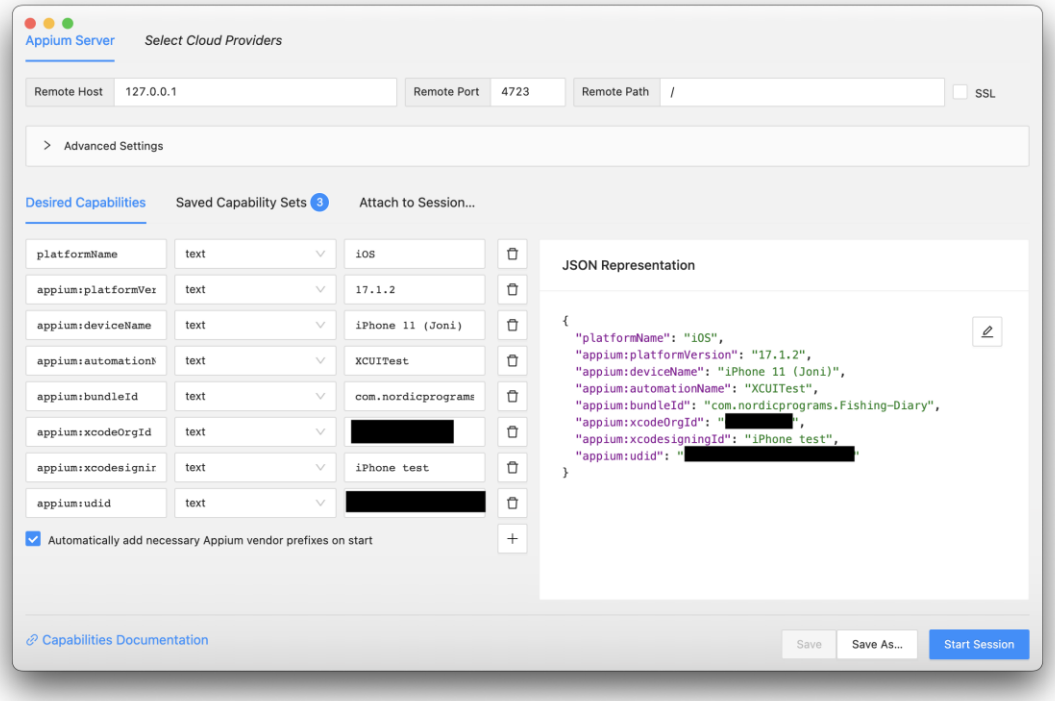
5.3 Appium Inspector

Appium Inspector on olennainen työkalu mobiilisovellusten testauksessa. Työkalu tarjoaa visuaalisen tavan elementtien löytämisen ja niiden vuorovaikutukseen graafisen käyttöliittymän kautta. Työkalu on valikoitunut opinnäytetyöhön, sillä se mahdollistaa paikantimien nopean löytämisen vähentäen niiden manuaalisen kirjoittamisen.

Appium Inspectorin yhdistäminen puhelimeen sekä testattavaan sovellukseen vaatii laitteen sekä sovelluksen tiedot. Kuvassa 8 esitetään Appium Inspectorin asetukset, jossa määritetään testiympäristön tiedot. Asetusnäkymä sisältää kentät seuraaville tiedoille:

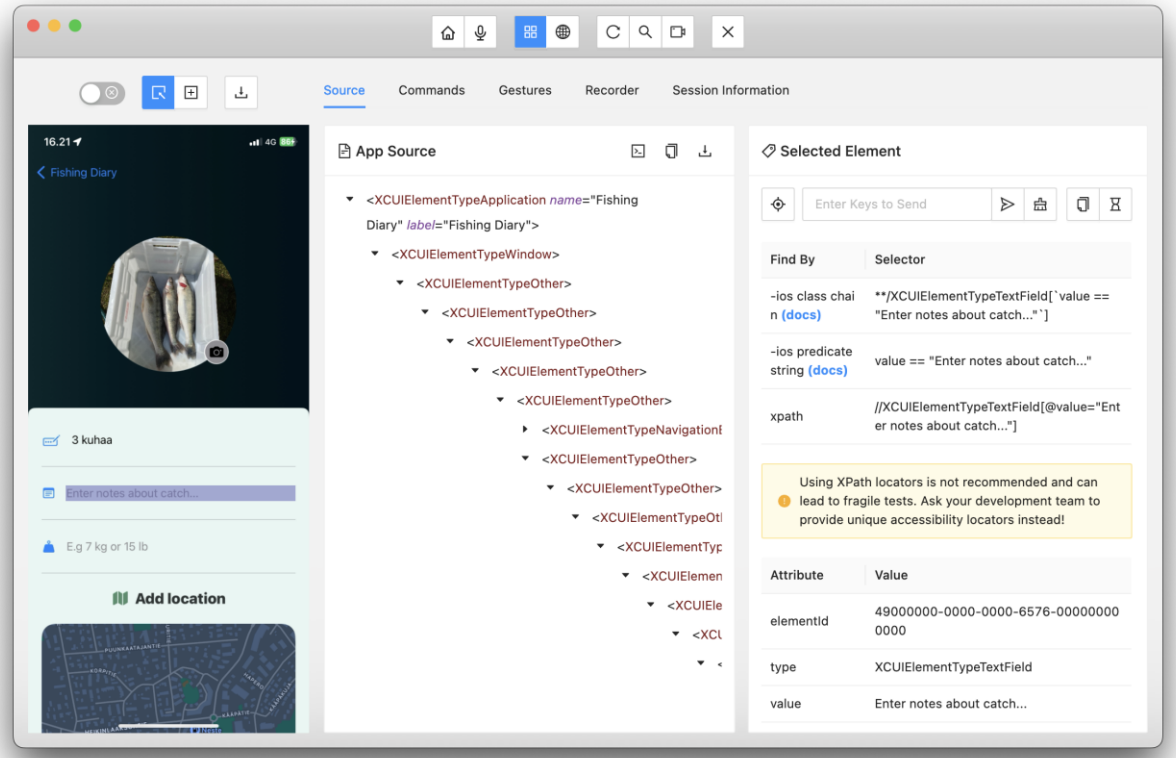
- alustan nimi (platformname)
- alustan versio (platformVersion)
- laitteen nimi (deviceName)
- testauskirjaston nimi (automationName)
- sovelluksen tunniste (bundleId)
- kehittäjän tilin tunnistenumero (xcodeOrgId)
- xcode allekirjoitusavain (xcodesigningId)
- laitteen tunnistenumero (udid).

Edellä mainitut tiedot ovat välttämättömiä, jotta Appium Inspector onnistuu yhdistämään puhelimeen sekä testattavaan sovellukseen.



Kuva 8. Appium Inspector -asetukset.

Yhteys puhelimeen on mahdollista luoda asetusten määrittelyn jälkeen sekä Appium-palvelimen ollessa käynnissä (kuva 4). Kuvassa 9 esitetään Appium Inspector yhdistettynä puhelimeen. Vasemmalta puolelta valitaan puhelimen näytöltä elementti, jonka jälkeen oikealla puolella näkyy valitun elementin tiedot. Elementin tietoihin kuuluvat paikannin sekä useita eri attribuutteja. Paikantimen avulla testit onnistuvat käsittelemään kyseistä elementtiä. Attribuutteihin sisältyy esimerkiksi elementin tyyppi eli onko kyseessä tekstikenttä tai vaikka kuva.



Kuva 9. Appium Inspector -käyttöliittymä.

6 Testien suunnittelu ja toteuttaminen

Edellisessä luvussa käsiteltiin, kuinka testausympäristö luotiin iOS-mobiilisovelluksen testausta varten. Tämän luvun tarkoituksena on käydä läpi testien suunnittelemista ja toteuttamista. Testien suunnittelussa käsitellään ensimmäisenä testien priorisointia, jonka jälkeen testit suunnitellaan näiden perusteella. Lopuksi testit toteutetaan suunnitelmien perusteella ja käsitellään parhaimpia käytäntöjä testien kirjoittamisessa.

6.1 Testien suunnittelu

Testien suunnittelemisessa tulee aina tunnistaa ohjelmiston kriittiset toiminnot eli priorisoida testitapaukset. Kriittiset toiminnot ovat sovelluksen toiminnot, jotka ovat tärkeimpiä käyttäjille. Esimerkiksi verkkokaupassa ostosprosessi on

kriittinen toiminto, sillä se vaikuttaa asiakkaan ostoprosessin onnistumiseen. Testitapausten priorisointi ja suorittaminen mahdollisimman tehokkaassa järjestyksessä on olennainen osa ohjelmiston virheiden havaitsemista. Priorisoimalla testaukseen käytettävät resurssit voidaan kohdistaa tärkeimpiin osa-alueisiin sovelluksessa. Testitapausten priorisoinnissa voidaan hyödyntää riskipohjaista, historiaan perustuvaa ja kattavuuteen perustuvaa priorisointia. Edellä mainittujen tekniikoiden avulla voidaan varmistaa, että testit kattavat tietyt vaatimukset sekä kriittiset toiminnot. (Son 2023.)

Riskipohjaisessa priorisoinnissa analysoidaan sovelluksen alueita, jotka voivat aiheuttaa ongelmia epäonnistuessaan. Kyseinen menetelmä perustuu riskianalyysiin, jossa otetaan huomioon virheen todennäköisyys sekä sen vaikutus ohjelmistoon. Tavoitteena on luoda testit kyseisille alueille, joissa todennäköisyys virheille on suurin. (Son 2023.)

Historiaan perustuvassa priorisoinnissa otetaan huomioon vikaherkät ominaisuudet. Tämä lähestymistapa käyttää aikaisempien testien tuloksia, jotta helposti vikaantuvat ominaisuudet voidaan priorisoida. (Son 2023.)

Kattavuuteen perustuvassa priorisoinnissa varmistetaan, että testit kattavat mahdollisimman suuren osan ohjelmiston toiminnoista. Menetelmä keskittyy ohjelmiston eri alueiden kattavaan testaukseen, joka auttaa varmistamaan ohjelmiston toimivuuden sekä laadun kokonaisvaltaisesti. (Son 2023.)

Opinnäytetyössä testattavaan sovelluksen ei ole luotu aikaisempia testejä. Testien suunnittelussa käytetään hyödyksi edellä mainittuja tekniikoita priorisoimaan testit, jotta ohjelmiston riskialueet ovat testattu. Testien suunnittelemisessa haluamme ensimmäisenä priorisoida riskipohjaiset testit. Suunnitteleminen alkaa käymällä sovellus läpi sekä kirjata mahdolliset ongelma osa-alueet taulukkoon 1. Kyseisessä taulukossa riskipohjaisessa priorisoinnissa ovat saaliin lisäys ja lisätyn saaliin katselu, koska kyseiset toiminnot ovat ohjelmiston kriittisimmät toiminnot. Historiaan perustuvaan priorisointiin on lisätty saaliin lisääminen puutteellisilla tiedoilla, sillä aikaisemmissa sovellusversioissa

kyseinen toiminto on ollut ongelmallinen. Kattavuuteen perustuvaan priorisointiin on valikoitunut lisätyn saaliin katselu kartalla ja sen avaaminen sieltä käsin. Tällä varmistamme, että toiminto on testattu.

Taulukko 1. Sovelluksen suunnitellut testitapaukset.

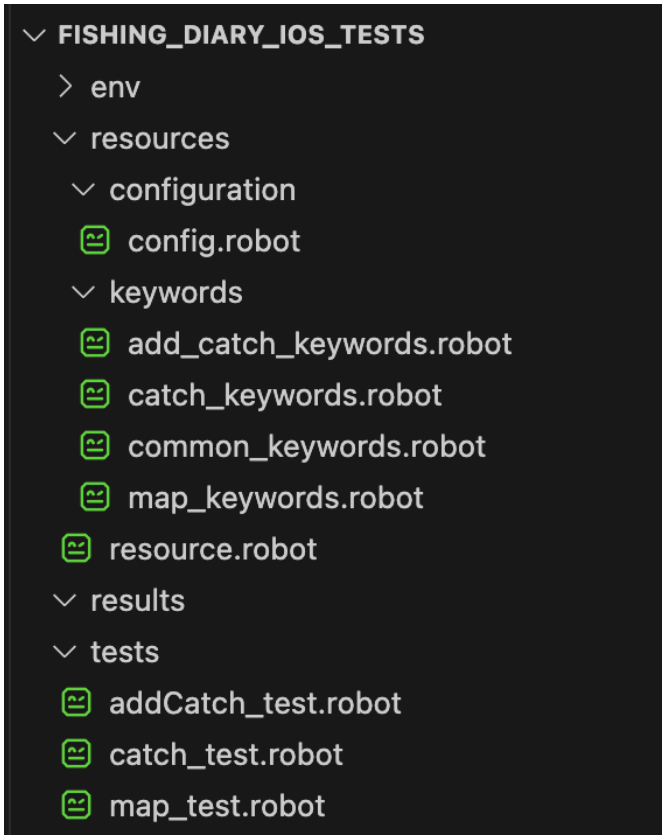
Testitapaus	**Kuvaus**	**Priorisointi**
Saaliin lisäys	Kalasaaliin lisääminen onnistuu	Riskipohjainen
Lisätyn saaliin katselu	Lisätyn saaliin katseleminen onnistuu ja tiedot ovat oikein	Riskipohjainen
Lisätyn saaliin katselu kartalla	Lisätyn saaliin katselu onnistuu kartalla. Kartalla näkyy saaliin sijainti sekä tiedot sitä painaessa	Kattavuuteen perustuva
Saaliin avaaminen kartalta	Kartalla olevan saaliin avaaminen onnistuu sekä kaikki tiedot ovat oikein	Kattavuuteen perustuva
Saaliin lisääminen puutteellisilla tiedoilla	Kalan lisäys tulisi onnistua, vaikka esimerkiksi painoa ei ole annettu	Historiaan perustuva

Tämän opinnäytetyön testit ovat suunniteltu kattamaan luvussa 4 mainittujen sovelluksien ominaisuuksien kriittiset toiminnot. Lisäksi testeillä halutaan varmistaa hyvä käyttäjäkokemus sovelluksen asiakkaille.

6.2 Testien toteuttaminen

Testitapausten kehittämisessä tulee ottaa huomioon kehittäjät sekä testaajat. Ohjelmiston testien kehittyessä testausympäristön monimutkaisuus voi kasvaa, ellei ympäristö ole organisoitu. Organisoituneeseen testausympäristöön sisältyy esimerkiksi oikeanlainen kansiorakenne.

Opinnäytetyössä käytettävä testausympäristön kansiorakenne jakautuu kolmeen pääosaan, resurssit, tulokset ja testiskriptit (kuva 10). Resurssikansio sisältää yleisimmät resurssit, joita käytetään eri testitapauksissa. Se on jaettu "configurations"-alikansioon, joka sisältää asetustiedostot, jotka mahdollistavat esimerkiksi yhteyden puhelimen ja testien välillä. Lisäksi se jakautuu "keywords"-alikansioon, joka kokoaa yleisimmät toiminnot ohjelmistossa, kuten kotinäkymään siirtyminen. Tulosten "results"-kansio sisältää testiajojen jälkeiset suoritusraportit. Testit "tests"-kansio sisältää varsinaiset testitiedostot, jotka suoritetaan ohjelmiston laadunvarmistusta varten. Organisoitu rakenne mahdollistaa tiedostojen nopean löytämisen sekä uudelleen käytettävien tiedostojen luomisen.



Kuva 10. Organisoitu kansiorakenne.

Organisoituneen kansiorakenteen luomisen jälkeen ensimmäisenä on asetustiedoston kirjoittaminen. Tällä mahdollistetaan yhteyden luominen laitteeseen ja sovellukseen. Asetustiedosto (esimerkkikoodi 5) sisältää samat asetukset kuin

edellisessä luvussa määritellyssä Appium Inspector -työkalussa. Asetukset määritellään muuttujiin, joiden ansiosta testitiedosto pysyy organisoituneena, helposti luettavana sekä muutettavana. Muuttujien lisäksi asetustiedostoon lisätään avainsana "Open Fishing Diary Application" jonka tarkoituksena on avata testattava sovellus määritetyillä muuttujilla.

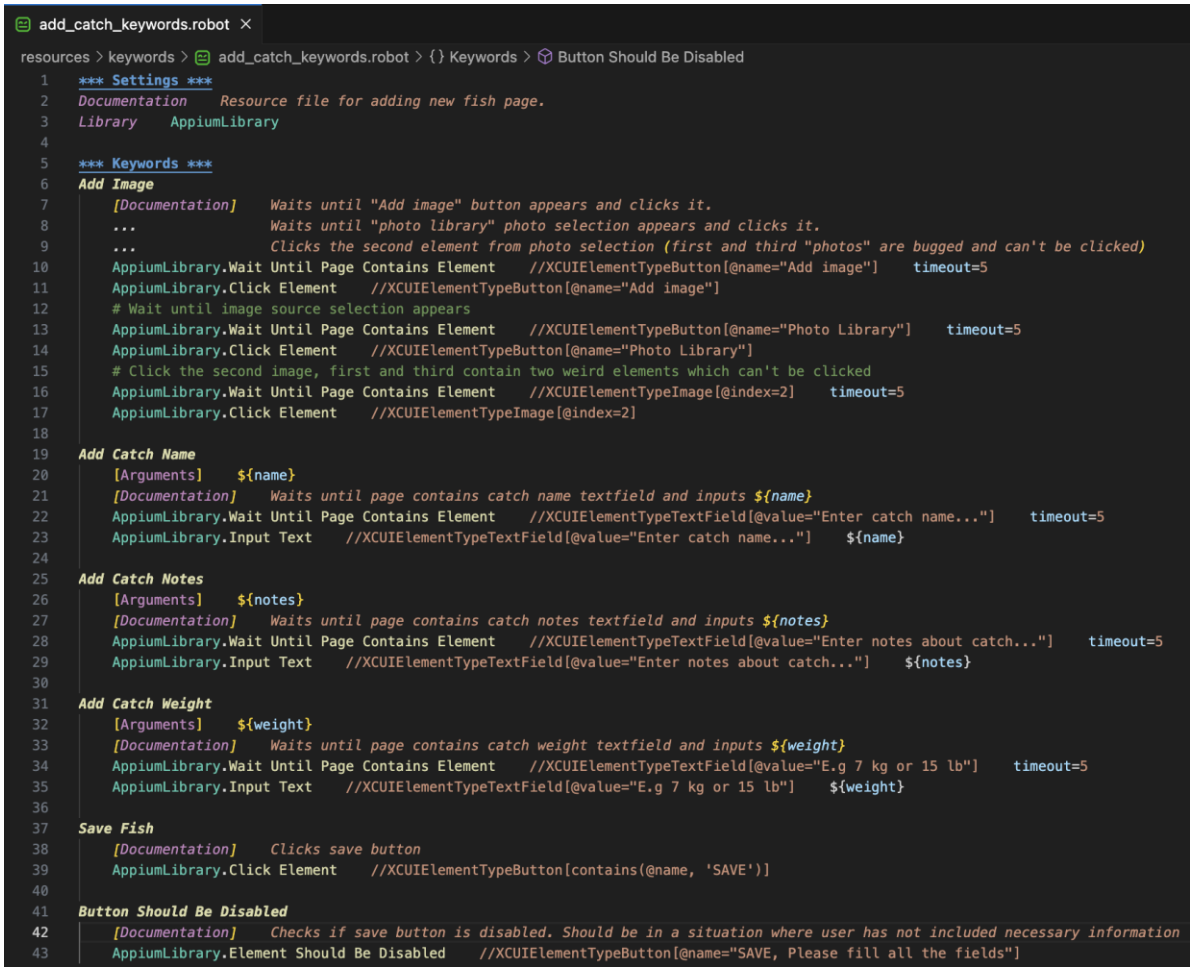
```
*** Settings ***
Documentation  File for connecting to Fishing Diary application
Library    AppiumLibrary

*** Variables **

${REMOTE_URL}          http://127.0.0.1:4723
${PLATFORM_NAME}    iOS
${PLATFORM_VERSION}    17.2.1
${DEVICE_NAME}    Joni's iPhone
${BUNDLE_ID}      com.nordicprograms.Fishing-Diary
${AutomationName}    XCUITest
${UDID}    Test phone UDID
${XCODEORGID}    Xcode Organisation ID
${XCODESIGNINGID}    iPhone test


*** Keywords ***
Open Fishing Diary Application
  Open Application    ${REMOTE_URL}    platformName=${PLATFORM_NAME}
platformVersion=${PLATFORM_VERSION}
  ...         deviceName=${DEVICE_NAME}      bundleId=${BUNDLE_ID}
automationName=${AutomationName}
  ...         udid=${UDID}    xcodeOrgId=${XCODEORGID}    xcodesign-
ingId=${XCODESIGNINGID}
```

Esimerkkikoodi 5. Robot Framework -asetustiedosto. Tiedosto sisältää tarvittavat muuttujat puhelimen yhteyden luomiseen sekä sovelluksen avaamiseen.

Testejä lähdetään kirjoittamaan testisuunnitelman pohjalta. Ensimmäisenä testinä on, että kalasaaliin lisääminen onnistuu. Ennen varsinaisen testin kirjoittamista ominaisuus käydään läpi puhelimella, minkä avulla havainnollistetaan ominaisuuden toiminta ja mahdolliset ongelmakohdat. Tämän jälkeen luodaan tarvittavat avainsanat kalasaaliin lisäämiseksi resurssitiedostoon. Kyseiset avainsanat ovat kuvan, nimen, lisätiedon ja painon lisääminen. Lopuksi lisätään tallennuspainikkeen painaminen. Kyseiset avainsanat voidaan nähdä kuvassa 11.

Resurssitiedostoihin halutaan kirjoittaa tehokkaita avainsanoja. Avainsanat, jotka odottava esimerkiksi tietyn elementin ilmestymistä, voidaan hyödyntää

"timeout"-asetusta. Kyseisellä asetuksella voidaan asettaa, kuinka kauan määriteltyä elementtiä odotetaan enimmillään. Tämän avulla testi ei odota turhaan elementtiä, joka ei välttämättä tulisi koskaan näkyville. Esimerkiksi "Wait Until Page Contains Element" -avainsana, jossa on viiden sekunnin yläraja, voidaan nähdä kuvassa 11.



Kuva 11. Saaliin lisäyksen avainsanat.

Kun avainsanat on määritelty, kalasaaliin lisäystesti voidaan kirjoittaa (Esimerkkikoodi 6). Testi alkaa "Suite Setup" -avainsanalla, joka on määritelty avaamaan sovellus ennen testien suorittamista. Testeissä käytettävät muuttujat kirjoitetaan "Variables"-osioon, joka helpottaa testin luettavuutta sekä muokkaamista. Tämän jälkeen "Test Cases" -osioon kirjoitetaan varsinainen testi. Ensimmäisenä on testin nimi, jonka avulla tulisi ymmärtää, mitä testillä halutaan varmistaa.

Tässä tapauksessa se nimetään "New Fish Should Be Added" -testiksi. Testissä käytettävät avainsanat listataan testin nimen alle. Testi etenee ensimmäisenä kalasaaliin lisäyssivun avaamisella ja tarvittavien tietojen syöttämisellä. Lopuksi kalasaalis tallennetaan sekä varmistetaan, että se on lisätty listaan. Testin loputtua suoritetaan "Teardown"-osio ja palautetaan sovellus aloitustilaan. Tässä tapauksessa kutsutaan "Delete Fish" -avainsanaa ja poistetaan juuri lisäämä kalasaalis. Tällä halutaan varmistaa, että testit eivät ole riippuvaisia toisistaan.

```
*** Settings ***
Resource    resource.robot
Suite Setup    Open Fishing Diary Application

*** Variables ***
${name}    Hauki
${notes}    Hauki mökiltä
${weight}    noin 4 kiloa

*** Test Cases ***

New Fish Should Be Added
    [Documentation]    Test for adding fish and verifying it's added
to the list
    Open New Catch
    Add Image
    Add Catch Name    ${name}
    Add Catch Notes    ${notes}
    Add Catch Weight    ${weight}
    Save Fish
    List Should Contain Fish    ${name}
    [Teardown]    Delete Fish    ${name}
```

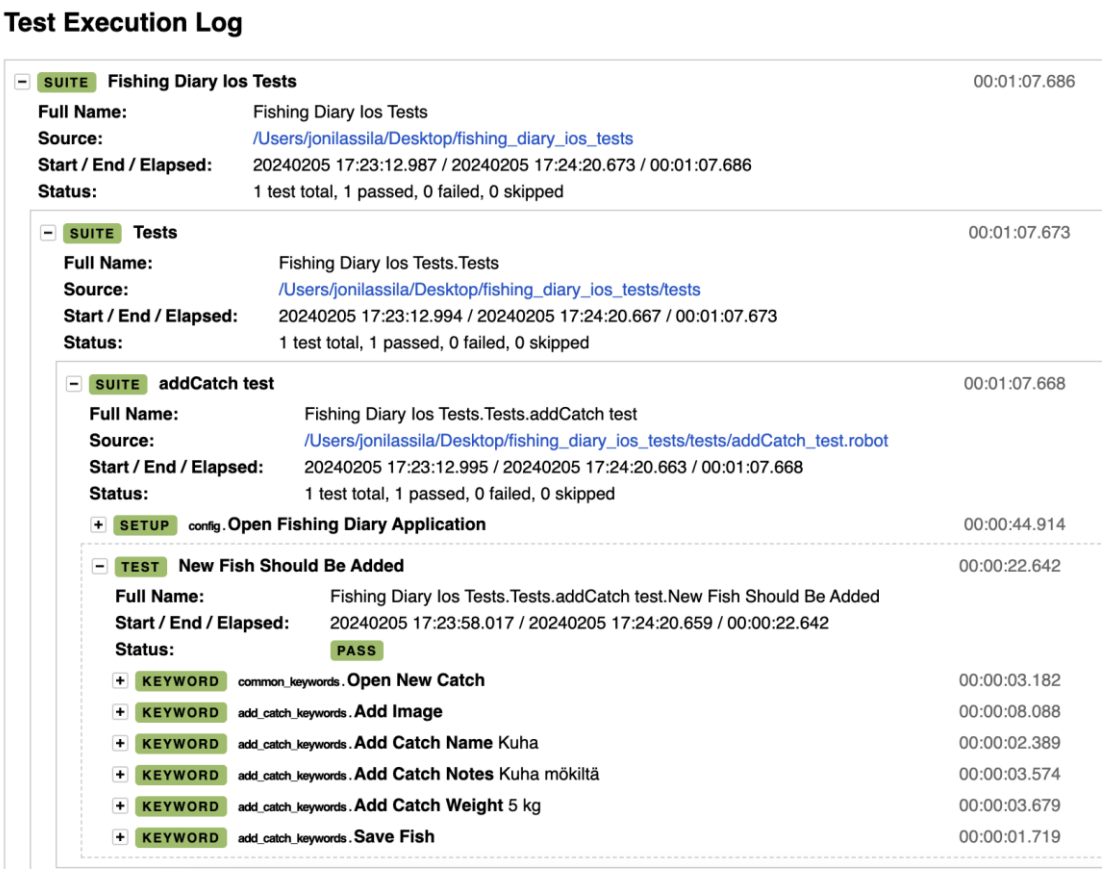
Esimerkkikoodi 6. Robot Framework -testi uuden kalasaaliin lisäämiseksi.

Testin ajamisen jälkeen muodostuu automaattisesti raportti, joka voidaan nähdä kuvassa 12. Raportista nähdään tietoja testin suoritusajasta, mukaan lukien kunkin avainsanan suorittamisen käytetty aika. Tämä antaa mahdollisuuden tarkastella suoritusaikoja sekä tunnistaa mahdolliset hidasteet.

Raportin analyysiin sisältyy avainsanojen suoritusaikojen tarkastelu. Jos havaitaan, että avainsana vie huomattavan kauan suorittaa, kyseinen avainsanan tai sovelluksen toimivuus voi vaatia optimointia. Tässä testitapauksessa raportti (kuva 12) osoittaa, että testi suoriutui tehokkaasti eikä mikään avainsanoista vaadi optimointia. Tämä voidaan nähdä siitä, että mikään avainsanoista ei

vienyt huomattavasti enempää aikaa kuin toiset. Tämä on hyvä havainto, sillä se osoittaa, että testattava sovellus sekä testausympäristö toimivat tehokkaasti.

Raporttien analyysia tulisit suorittaa säännöllisesti, jotta mahdolliset suorituskykyongelmat havaitaan ajoissa. Tämä ylläpitää sovelluksen sekä testausympäristön tehokkuutta ja laatua.

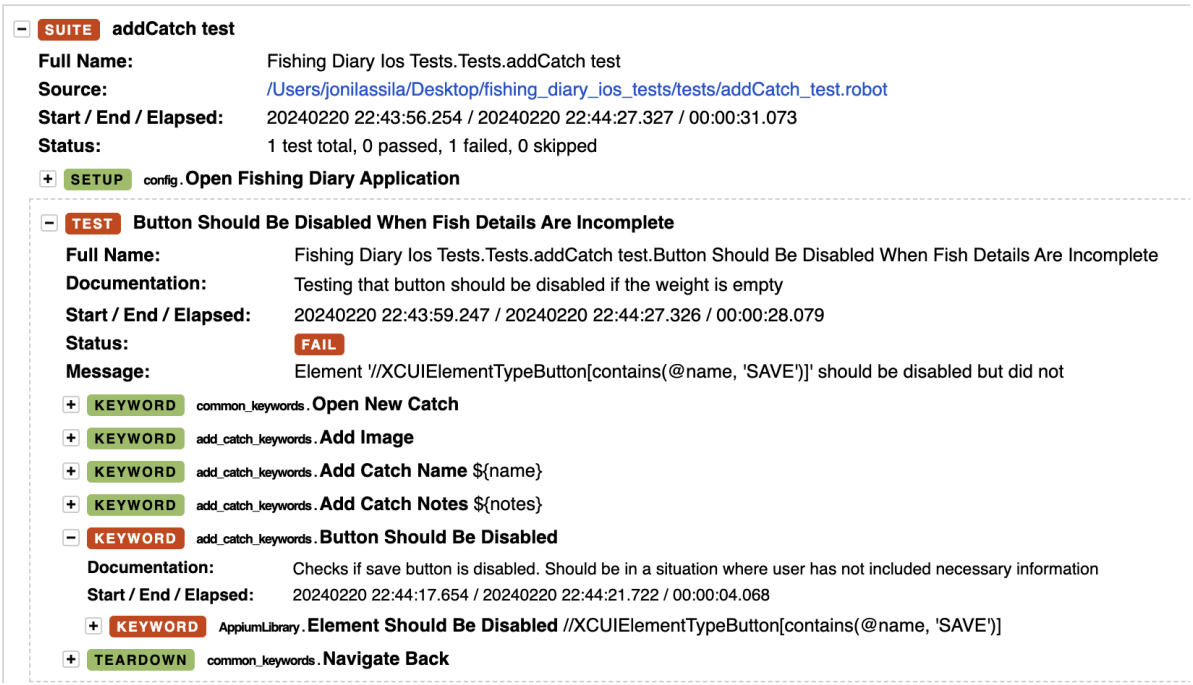


Kuva 12. Raportti testistä.

Edellä mainitun testin onnistunut läpivienti osoittaa, että testausympäristö toimii halutulla tavalla. Tämän jälkeen siirrytään eteenpäin ja toteutetaan loput testitapauksista. Nämä testitapaukset seuraavat samanlaista prosessia, jota käytettiin edellisessä testin luonnissa, eli tarvittavien avainsanojen luominen, testin toteuttaminen ja raportin analysointi.

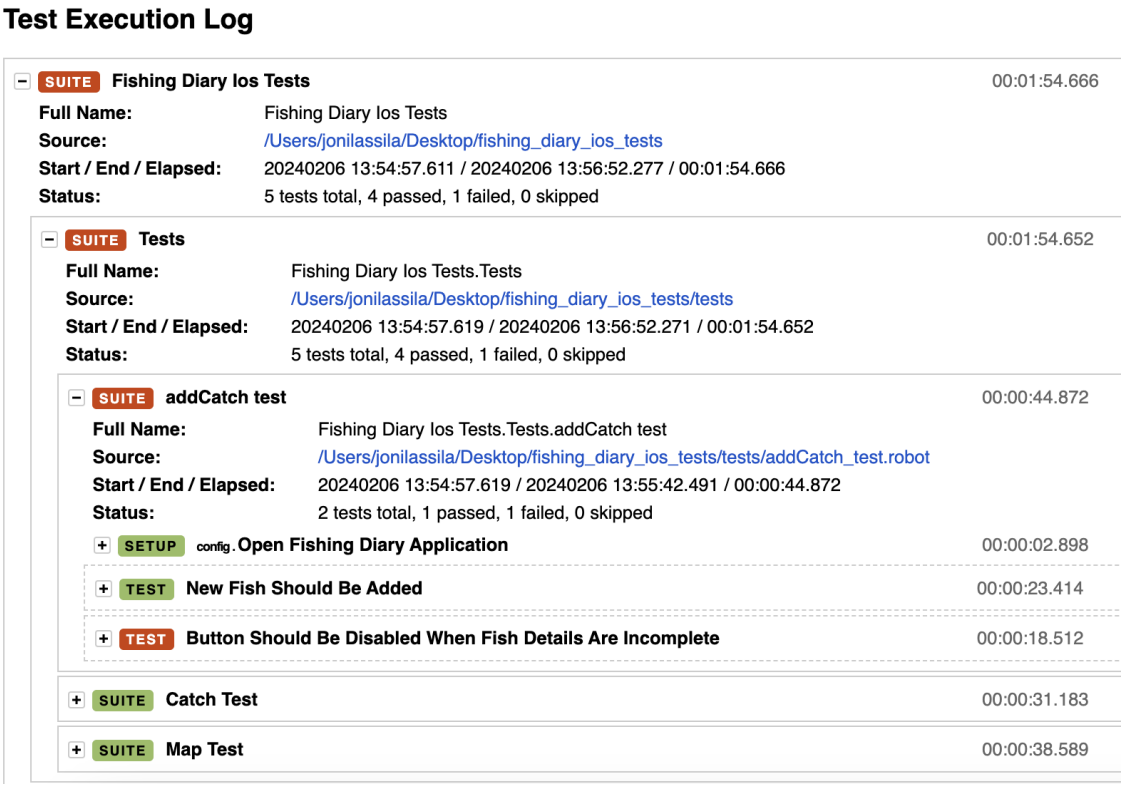
Seuraavaksi kirjoitetaan testitapaukset karttaominaisuudelle. Testitapauksen tavoitteena on varmistaa, että kalasaaliin lisäämisen jälkeen karttaan ilmestyy sen sijainti karttaneulana ja saaliin tiedot ovat nähtävissä, kun neulaa painetaan. Kyseinen toiminto sovelluksessa voidaan nähdä liitteestä 3 sekä siihen luotu testi liitteestä 4. Seuraava testitapaus liittyy kalasaaliin tietojen oikeellisuuden varmistamiseen. Testissä ensimmäisenä luodaan uusi kalasaalis, jonka jälkeen varmistetaan, että tiedot tallentuvat oikein ja näkyvät sovelluksen aloitusnäytöllä sijaitsevassa listassa. Lopuksi tarkistetaan, että saliin tietosivu näyttää kaikki tiedot oikein. Kyseinen toiminto sovelluksessa voidaan nähdä liitteestä 2 sekä siihen luotu testi liitteestä 5. Lopuksi kirjoitetaan viimeinen testi saaliin lisääminen puutteellisilla tiedoilla, jonka avulla varmistetaan, että tallennuspainiketta ei voi painaa, mikäli osa saaliin tiedoista on tyhjiä. Testissä ensimmäiseksi avataan kalasaaliin lisäämisen mahdollistava sivu, jonka jälkeen lisätään kaikki muut vaadittavat tiedot lukuun ottamatta painoa. Lopuksi varmistetaan, että painikkeen painaminen on estetty. Tämä testi voidaan nähdä liitteestä 6.

Edellä mainittu viimeinen testitapaus paljasti sovelluksessa ohjelmointivirheen. Testin tarkoituksena oli testata, että tallennuspainike ei ole painettavissa, jos saaliille ei ole annettu painoa. Tämä voidaan nähdä kuvassa 13, jossa testi epäonnistui punaisella merkittyyn avainsanaan "Button Should Be Disabled". Tämän testin avulla havaittiin, että napin logiikasta puuttui toiminto, joka tarkistaisi onko saaliin painokenttään kirjoitettu tekstiä. Tämä havainto osoittaa kuinka tärkeää on suorittaa perusteellista testausta ja priorisoida testejä, jopa testata niitä ominaisuuksia, jotka on jo kertaalleen korjattu.



Kuva 13. Epäonnistunut testi.

Kaikkien testien kirjoittamisen jälkeen seuraavana oli suorittaa kaikkien testien ajo. Tämä vaihe on tärkeä, sillä se tarjoaa kokonaisen näkökulman testien sekä sovelluksen tehokkuudesta. Ajetun testisarjan raportista (kuva 14) voidaan nähdä, että testisarjat suoriutuivat tehokkaasti. Lisäksi testisarjassa mukana oleva rikkinäinen testi ei hidastanut testien suoritusaikaa, mikä osoittaa "timeout"-asetuksen tehokkuuden.



Kuva 14. Lopullinen testiajo.

Opinnäytetyön testausympäristön suunnittelu ja toteutus olivat onnistuneita. Organisoitu kansiorakenne mahdollisti tehokkaan testauksen hallinnan. Tämä tuki testausprosessia ja skaalautuvuutta. Lisäksi työssä käytetyt työkalut osoittautuivat tehokkaiksi sovelluksen laadunvarmistuksessa.

7 Yhteenveto

Tämän opinnäytetyön tarkoituksena oli suunnitella sekä toteuttaa testiautomaatio iOS-sovellukselle käyttäen Robot Frameworkia. Lisäksi haluttiin selvittää, kuinka voidaan kehittää luotettava ja nopea testausautomaatio iOS-sovellukselle. Tavoitteiden saavuttamiseksi käytettiin Robot Framework -kehystä yhdistettynä AppiumLibrary-kirjaston kanssa. Näiden kahden työkalun avulla onnistuttiin luomaan tehokas sekä luotettava testiautomaatio iOS-sovellukselle. Johdannossa asetettiin tavoite suunnitella sekä luoda testiautomaatio, jonka avulla pystytään varmistamaan sovelluksen toimivuus sekä laatu ennen uusien versioiden jakamista asiakkaiden saataville. Tämä tavoite saavutettiin.

Testien suorittamisen aikana havaittiin ohjelmointivirhe sovelluksesta. Ohjelmointivirhe havaittiin testissä, jonka tarkoituksena oli testata, että tallennuspainike ei ole painettavissa, jos saaliille ei ole annettu painoa.

Opinnäytetyössä kirjoitettu testausautomaatio helpottaa uusien testien kirjoittamisesta uusien ominaisuuksien lisääntyessä, koska avainsanat on tehty uudelleenkäytettäväksi sekä kansiorakenne helpoksi ja organisoituneeksi.

Opinnäytetyöprosessi oli kiinnostavaa sekä opettavainen, jonka aikana opin paljon testausautomaatiosta sekä sen vaikutuksesta ohjelmistokehitykseen. Lisäksi testauksessa käytettävät työkalut tulivat tutuiksi.

Jatkokehityksenä testiautomaation testikattavuutta voitaisiin laajentaa suuremmaksi, esimerkiksi lisäämällä testitapauksia, jotka kattavat harvinaisemmat käyttötapaukset. Tämä parantaisi sovelluksen laadunvarmistusta.

Lähteet

Acceptance Testing. 2022. Verkkoaineisto. Software Testing Help. <https://softwaretestingfundamentals.com/acceptance-testing/>. 29.8.2022. Luettu 29.11.2023.

Akshay, Pai. 2023. Getting started with XCUITesti: UI Automation Framework on iOS. Verkkoaineisto. BrowserStack. <https://www.browserstack.com/guide/getting-started-xcuitest-framework>. 8.2.2023. Luettu 28.12.2023.

Amazon Unit Testing. 2023. Verkkoaineisto. Amazon. <https://aws.amazon.com/what-is/unit-testing/>. Luettu 14.1.2024.

Appium. Verkkoaineisto. JavaTpoint. <https://www.javatpoint.com/appium>. Luettu 20.12.2023.

Appium Inspector. Verkkoaineisto. GitHub. <https://github.com/appium/appium-inspector>. Luettu 10.1.2024,

Core Data. Verkkoaineisto. Apple <https://developer.apple.com/documentation/coredata/>. Luettu 20.12.2023.

Homann, Maria. Why Automate Regression Testing? 5 Reasons. Verkkoaineisto. Leapwork. <https://www.leapwork.com/blog/5-reasons-why-you-should-automate-regression-testing>. Luettu 2.12.2023.

Manoj, H. 2015. Appium Essentials. Kirja. <https://www.google.fi/books/edition/Appium_Essentials/bQoDCAAAQBAJ?hl=en&gbpv=1&dq=appium&printsec=frontcover>. Luettu 20.12.2023.

Mohit, Joshki. 2023. What is UX testing with example. Verkkoaineisto. BrowserStack. <https://www.browserstack.com/guide/what-is-ux-testing>. 3.2.2023. Luettu 26.12.2023.

Ohjelmistotestaus. 2022. Verkkoaineisto. Valagroup. <https://www.valagroup.com/fi/blogi/mita-on-ohjelmistotestaus-ja-mita-hyotya-siita-on/>. 10.11.2022. Luettu 25.11.2023.

Ohjelmistotestaus. 2023. Verkkoaineisto. Haltu. <https://www.haltu.fi/blogi/ohjelmistotestaus#ohjelmistotestaus>. 4.7.2023. Luettu 27.11.2023.

Patil, A. 2020. Regression testing in era of internet of things and machine learning. Kirja. <https://www.google.fi/books/edition/Regression_Testing_in_Era_of_Internet_of/gRXBDwAAQBAJ?hl=en&gbpv=1&dq=regression+testing&printsec=frontcover>. Luettu 16.1.2024.

Rehkopf, Max. 2023. Automated software testing. Verkkoaineisto. Atlassian. <https://www.atlassian.com/continuous-delivery/software-testing/automated-testing>. Luettu 23.12.2023.

Robot Framework. Verkkoaineisto. Robot Framework <https://robotframework.org/>. Luettu 20.12.2023.

Robot Framework AppiumLibrary. Verkkoaineisto. AppiumLibrary. <https://serhatbolsu.github.io/robotframework-appiumlibrary/AppiumLibrary.html>. Luettu 20.12.2023.

Robot Framework: past, present and future. 2016. Verkkoaineisto. Eficode. <https://www.eficode.com/blog/en/blog/robot-framework>. Päivitetty 1.6.2021. Luettu 23.12.2023.

Sachedina, Fahim. 2017. What is Exploratory Testing?. Verkkoaineisto. Global App Testing. <https://www.globalapptesting.com/blog/what-is-exploratory-testing>. 1.8.2017. Luettu 26.12.2023.

Shain, Danny. 2022. What is visual regression testing? Verkkoaineisto. Applitools. <https://applitools.com/blog/visual-regression-testing/>. 17.6.2022. Luettu 26.12.2023.

Son, Hannah. 2023. Test Case Prioritization Techniques and Metrics. Verkkoaineisto. Testtrail. <https://www.testrail.com/blog/test-case-prioritization/>. 4.8.2023. Luettu 26.1.2024.

Sonmez, John. A Survival Guide to Test Driven Development and Unit Testing. Verkkoaineisto. Usersnap. <https://usersnap.com/blog/tdd-unit-testing/>. Luettu 26.12.2023.

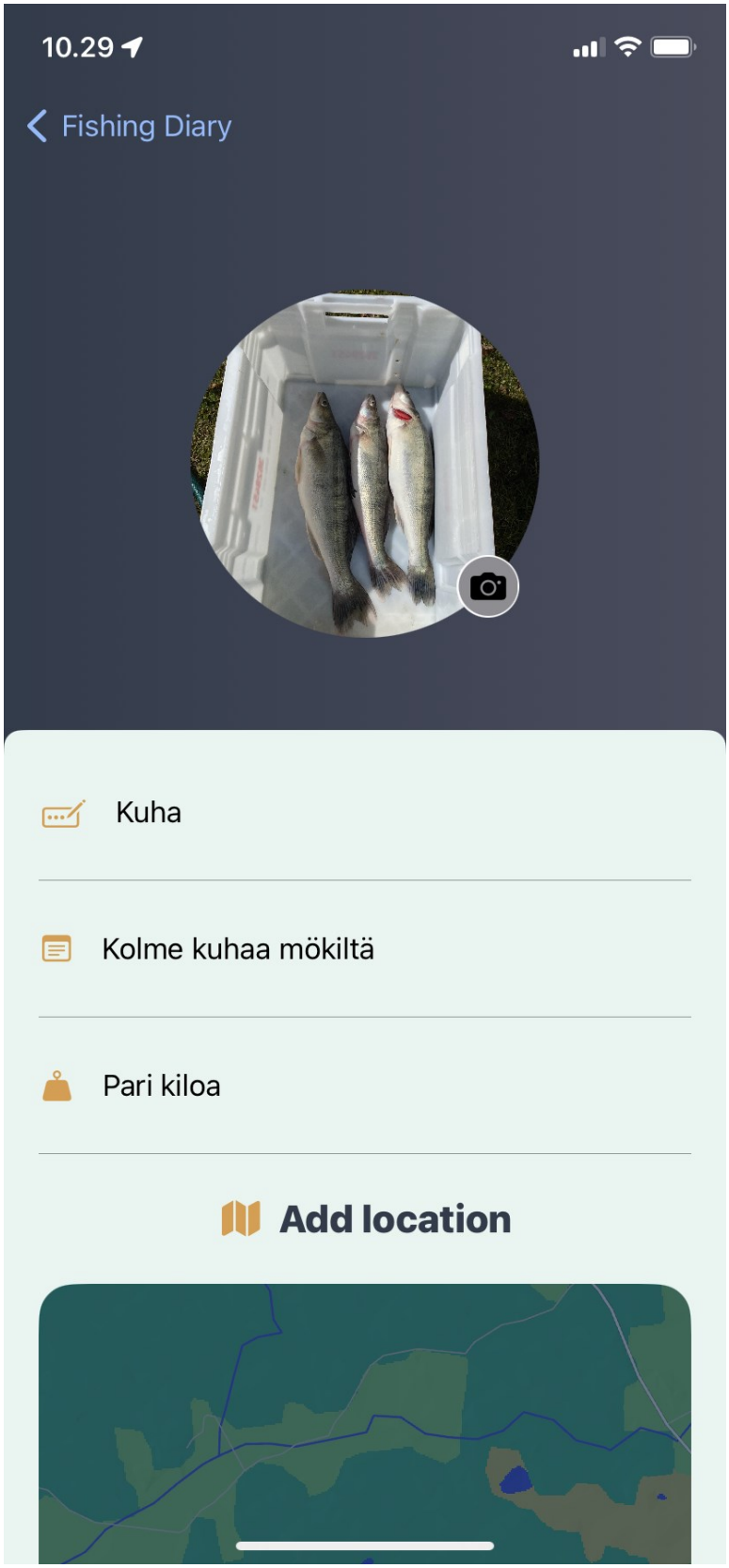
Srinivasan, Desikan & Gopalaswamy, Ramesh. Software Testing: Principles and Practices. Kirja. <https://www.google.fi/books/edition/Appium_Essentials/bQoDCAAAQBAJ?hl=en&gbpv=1&dq=appium&printsec=frontcover>. Luettu 16.1.2024.

The Agile Coach. Verkkoaineisto. Atlassian. <https://www.atlassian.com/agile>. Luettu 25.12.2023.

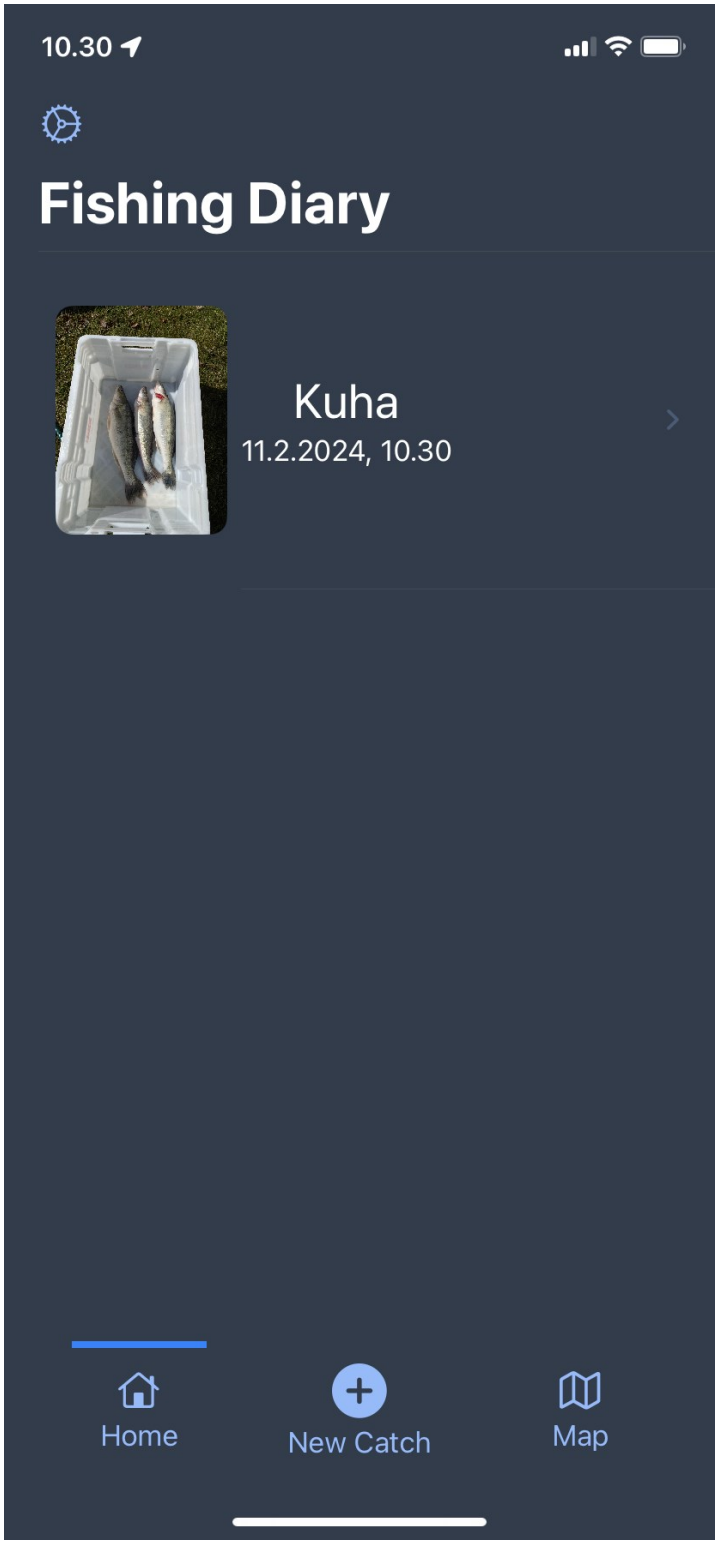
Visual regression testing: Why visual AI can improve your user's digital experiences. 2022. Verkkoaineisto. Ericsson. <https://www.ericsson.com/en/blog/2022/12/visual-regression-testing-ai>. 5.12.2022. Luettu 26.12.2023.

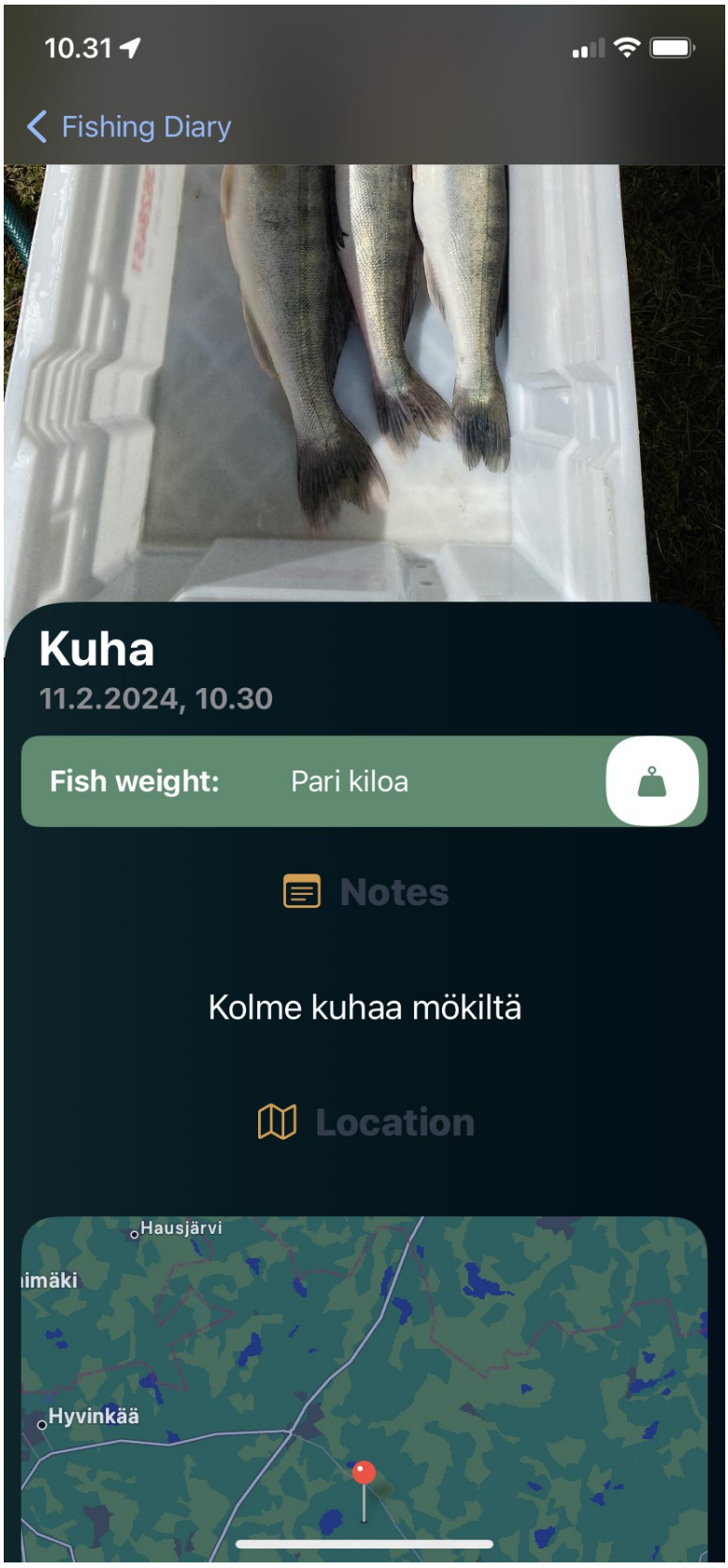
IBM software testing. Verkkoaineisto. IBM. <https://www.ibm.com/topics/software-testing>. Luettu 29.11.2023.

Sovelluksen kalasaaliin lisäys

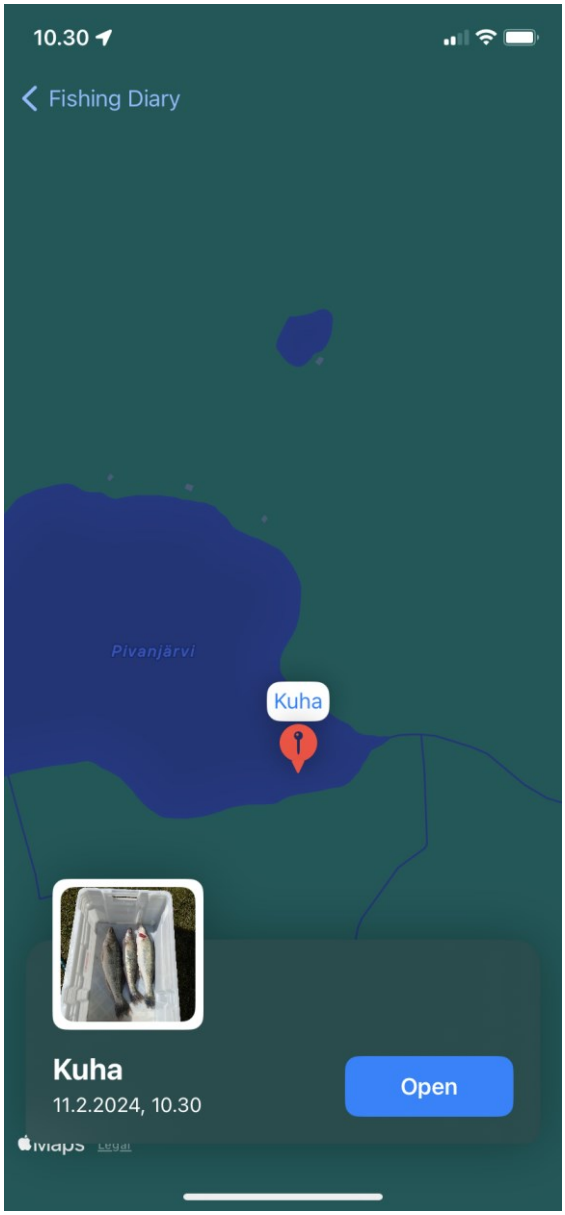


Sovelluksen kalasaaliin tarkastelu




10.31
Fishing Diary
Kuha
11.2.2024, 10.30
Fish weight:
Pari kiloa
Notes
Kolme kuhaa mökiltä
Location
Hausjärvi
imäki
Hyvinkää

Sovelluksen kalasaaliin tarkastelu kartalla



Karttatestitiedostot

```
*** Settings ***
Documentation    Resource file for adding new fish page.
Library    AppiumLibrary

*** Keywords ***
Verify Map Contains Pin
    [Arguments]    ${name}
    [Documentation]    Verifies the pin with ${name} is on the map
    AppiumLibrary.Wait Until Element Is Visible    //XCUIElementTypeStaticText[@name="${name}"]/following-sibling::XCUIElementTypeImage[@name="Map Pin"]    timeout=5
Click Pin
    [Arguments]    ${name}
    [Documentation]    Clicks the pin that contains ${name}
    AppiumLibrary.Wait Until Element Is Visible    //XCUIElementTypeStaticText[@name="${name}"]/following-sibling::XCUIElementTypeImage[@name="Map Pin"]    timeout=5
    AppiumLibrary.Click Element    xpath=(//XCUIElementTypeStaticText[@name="${name}"]/following-sibling::XCUIElementTypeImage[@name="Map Pin"])

Open Selected Catch
    [Documentation]    Clicks the "Open" button on selected catch Pin
    AppiumLibrary.Wait Until Element Is Visible    //XCUIElementTypeButton[@name="Open"]    timeout=5
    AppiumLibrary.Click Element    //XCUIElementTypeButton[@name="Open"]

Verify Selected Catch
    [Arguments]    ${name}
    [Documentation]    Verifies the selected catch is correct based on ${name}
    AppiumLibrary.Wait Until Element Is Visible    //XCUIElementTypeStaticText[@name="${name}"]    timeout=5
    AppiumLibrary.Click Element    //XCUIElementTypeStaticText[@name="${name}"]
```

```
*** Settings ***
Resource    resource.robot
Suite Setup    Open Fishing Diary Application
Suite Teardown    Delete Fish    ${name}

*** Variables ***
${name}    Hauki Pin
${notes}    Hauki mökiltä saatu uistellessa
${weight}    Olisko ollut noin 6 kiloa

*** Test Cases ***

Verify Pin Is Generated
    [Documentation]    Test for verifying new pin is generated
    ...                on the map, and it can be clicked.
    Open New Catch
    Add Image
    Add Catch Name    ${name}
    Add Catch Notes    ${notes}
    Add Catch Weight    ${weight}
    Save Fish
    Open Map
    Click Pin    ${name}
    [Teardown]    Run Keywords    Navigate Back
    ...            AND            Delete Fish    ${name}

Verify Selected Pin Is Correct
    [Documentation]    Test for verifying new pin is generated
    ...                on the map, and it can be clicked.
    Open New Catch
    Add Image
    Add Catch Name    ${name}
    Add Catch Notes    ${notes}
    Add Catch Weight    ${weight}
    Open Map
    Click Pin    ${name}
    Open Selected Catch
    Verify Selected Catch    ${name}
    [Teardown]    Run Keywords    Navigate Back
    ...            AND            Delete Fish    ${name}
```

Lisätyn kalasaaliin testitiedostot

```
*** Settings ***
Documentation    Resource file for added fishes.
Library    AppiumLibrary

*** Keywords ***
Open Catch From List
    [Arguments]    ${name}
    [Documentation]    Opens catch from list based on ${name}
    AppiumLibrary.Wait Until Page Contains Element    //XCUIElementTypeButton[contains(@name, "${name}")]
    AppiumLibrary.Click Element    //XCUIElementTypeButton[contains(@name, "${name}")]

Verify Name
    [Arguments]    ${name}
    [Documentation]    Verifies the ${name}
    AppiumLibrary.Page Should Contain Element    //XCUIElementTypeStaticText[@name="${name}"]

Verify Weight
    [Arguments]    ${weight}
    [Documentation]    Verifies the ${weight}
    AppiumLibrary.Page Should Contain Element    //XCUIElementTypeStaticText[@name="${weight}"]

Verify Notes
    [Arguments]    ${notes}
    [Documentation]    Verifies the ${notes}
    AppiumLibrary.Page Should Contain Element    //XCUIElementTypeStaticText[@name="${notes}"]

Delete Fish
    [Arguments]    ${name}
    Open Catch From List    ${name}
    AppiumLibrary.Wait Until Page Contains Element    //XCUIElementTypeButton[@name="Delete"]    timeout=5
    AppiumLibrary.Click Element    //XCUIElementTypeButton[@name="Delete"]
    # Confirm deletion
    AppiumLibrary.Wait Until Page Contains Element    //XCUIElementTypeButton[@name="Confirm"]    timeout=5
    AppiumLibrary.Click Element    //XCUIElementTypeButton[@name="Confirm"]
```

```
*** Settings ***
Resource    resource.robot
Suite Setup    Open Fishing Diary Application

*** Variables ***
${name}    Kuha
${weight}    Noin 4 kiloa
${notes}    Hauki mökiltä. Tuli uistellessa.

*** Test Cases ***
Open Catch And Verify Details
    [Documentation]    Creates a new catch and then verifies that
    Open New Catch
    Add Image
    Add Catch Name    ${name}
    Add Catch Notes    ${notes}
    Add Catch Weight    ${weight}
    Save Fish
    List Should Contain Fish    ${name}
    #Open and verify
    Open Catch From List    ${name}
    Verify Name    ${name}
    Verify Weight    ${weight}
    Verify Notes    ${notes}
    [Teardown]    Run Keywords    Navigate Back
    ...             AND             Delete Fish    ${name}
```

Kalasaaliin lisäys testitiedostot

```
*** Settings ***
Resource    resource.robot
Suite Setup    Open Fishing Diary Application

*** Variables ***
${name}    Hauki
${notes}    Hauki mökiltä
${weight}    noin 4 kiloa

*** Test Cases ***

New Fish Should Be Added
    [Documentation]    Test for adding fish and verifying it's added to the list
    Open New Catch
    Add Image
    Add Catch Name    ${name}
    Add Catch Notes    ${notes}
    Add Catch Weight    ${weight}
    Save Fish
    List Should Contain Fish    ${name}
    [Teardown]    Delete Fish    ${name}

Button Should Be Disabled When Fish Details Are Incomplete
    [Documentation]    Testing that button should be disabled if the weight is empty
    Open New Catch
    Add Image
    Add Catch Name    ${name}
    Add Catch Notes    ${notes}
    Button Should Be Disabled
    [Teardown]    Navigate Back
```