Alexandra Dedikova

Mobiilisovellusten testauksen automatisointi tekoälyllä

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto-ja viestintätekniikan tutkinto-ohjelma

Insinöörityö

8.4.2024

Tiivistelmä

Tekijä:	Alexandra Dedikova
Otsikko:	Mobiilisovellusten testauksen automatisointi tekoälyllä
Sivumäärä:	38 sivua
Aika:	8.4.2024
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikan tutkinto-ohjelma
Ammatillinen pääaine:	Mobile Solutions
Ohjaajat:	Vanhempi Lehtori Peter Hjort

Opinnäytetyön tarkoituksena oli tutkia mobiilisovellusten testausprosessien perusteita, tutkia olemassa olevia ratkaisuja mobiilisovellusten testauksen automatisointiin ja selvittää, miten näitä ratkaisuja voidaan parantaa ChatGPT:n integraation kautta. Työssä luotiin toimiva testiprototyyppi käyttäen tekoälyteknologiaa. Insinöörityö on jaettu teoreettiseen osaan ja projektiosaan. Ensimmäinen osa selittää, mitä mobiilisovellusten testaus yleisesti ottaen käsittää, mitkä ovat testauksen nyanssit ja miten automatisoitu testaus eroaa manuaalisesta testauksesta. Lisäksi osiossa avataan, mitä ChatGPT-teknologia oikeastaan on. Lisäksi tässä osassa käsitellään ChatGPT:n mahdollisuudet ja heikkoudet sekä projektissa käytettyjen työkalujen valintaa.

Projektin toinen osio keskittyy käytännön soveltamiseen ja kehyksen luomiseen ChatGPT:n avulla sekä Android- että iOS-sovellusten testaamiseen. Tutkimusprojekti tavoitteli myös ChatGPT:n ominaisuuksien, kuten suorituskyvyn, tarkkuuden ja tehokkuuden, tunnistamista ja arviointia testaustilanteissa. Lisäksi insinöörityön tavoitteena oli selvittää, miten ChatGPT voisi vähentää manuaalista työtä, parantaa testikattavuutta ja nopeuttaa testausprosessia. Testiprototyypin toteutuksessa hyödynnettiin useita kehittyneitä teknologioita, mukaan lukien Appium, WebDriver.IO, Mocha, WebDriver.IO Allure Reporter ja OpenAI:n tarjoama API-avain tai rajapinta-avain. Tutkimuksessa suoritettiin testejä käyttäen erilaisia tällä hetkellä saatavilla olevia kielimalleja. Näiden mallien vaikutusta testituloksiin analysoitiin huolellisesti.

Opinnäytetyön lopputuloksena tehtiin yksityiskohtainen analyysi suoritetusta työstä ja saaduista tuloksista sekä arvioitiin ChatGPT:n integraation kokonaisvaikutusta mobiilisovellusten testauksen automatisointiin ja tämän lähestymistavan relevanssia. Lisäksi ehdotettiin mahdollista tutkimusaiheen jatkokehityssuuntaa. Tutkimusprojektista saadut tiedot ovat sekä tärkeitä että arvokkaita ja mahdollistavat tulosten ja teknisen toteutuksen jakamisen ja hyödyntämisen. Tulokset tarjoavat arvokasta tietoa organisaatioille, jotka harkitsevat tai suunnittelevat OpenAI:n tai muiden kielimallien käyttöönottoa automatisointitestausprojekteissa ja muissa sovelluksissa.

Avainsanat: ChatGPT, GPT, OpenAI, tekoäly, testaus, E2E, Android, iOS, JavaScript, Appium, Webdriverlo, Mocha, Node.js

---

Abstract

Author:	Alexandra Dedikova
Title:	Automating mobile app testing with artificial intelligence
Number of Pages:	38 pages
Date:	8 April 2024
Degree:	Bachelor of Engineering
Degree Programme:	Information and communication technology
Professional Major:	Mobile Solutions
Supervisors:	Senior Lecturer Peter Hjort

The purpose of the thesis was to examine the fundamental aspects of testing processes for mobile applications, delve into current methods for automating such tests, and ascertain avenues for enhancement through the incorporation of ChatGPT technology. A practical test model utilizing this innovative approach was constructed. The study comprises a theoretical discussion, which outlines the essentials of mobile app testing, differentiates between automated and manual testing strategies, and introduces ChatGPT technology along with its capabilities and limitations. It further details the selection of tools deployed in the project.

The second part of the project focuses on the practical development of an actual testing framework leveraging ChatGPT to assess both Android and iOS applications. This investigation evaluated ChatGPT's effectiveness, precision, and efficiency in testing, aiming to discern its role in reducing manual testing efforts, augmenting test coverage, and expediting the testing cycle. The prototype was built employing cutting-edge technologies such as Appium, WebDriver.IO, Mocha, and WebDriver.IO Allure Reporter, in conjunction with the OpenAI API, examining the influence of different language models on testing outcomes.

The conclusion of this study provided a comprehensive review and assessment of the conducted work and findings, highlighting the significance of integrating ChatGPT in automating mobile app testing processes and suggesting future research directions. This work offers insightful information for entities contemplating the adoption of OpenAI or similar language models in their automation testing ventures or other applications, presenting a valuable resource derived from the practical experiences and technical executions documented throughout this research.

Keywords: ChatGPT, GPT, OpenAI, AI, testing, E2E, Android, iOS, JavaScript, Appium, WebdriverIo, Mocha, Node.js

Sisällys

Lyhenteet

AI: *Artificial Intelligence*. Tekoäly. Tässä insinöörityössä käytetty ChatGPT on tekoälypohjainen työkalu.

API: *Application Programming Interface*. Rajapinta, jonka avulla kaksi erilaista ohjelmaa voi vaihtaa tietoja keskenään määriteltyjä sääntöjä noudattaen.

E2E: *End-to-End*-testit joissa testaaja validoi koko ohjelmistotuotteen alusta loppuun. Tämä testausmenetelmä varmistaa, että sovellus käyttäytyy odotetusti kaikissa integroiduissa komponenteissa ja järjestelmissä.

GPT: *Generative Pre-trained Transformer*. Generoiva esikoulutettu transformer -malli tai laaja kielimalli (LLM).

JSON: *JavaScript Object Notation* tekstipohjainen tietoformaatti, jota käytetään tietojen tallentamiseen ja siirtämiseen.

LLM: *Large language model*. Laaja kielimalli on tekoälyyn ja koneoppimiseen perustuva malli, joka on suunniteltu ymmärtämään, tuottamaan ja käsittelemään luonnollista kieltä monimutkaisella ja syvällisellä tavalla.

MMLU: *Massive Multitask Language Understanding*. Massiivinen monitehtäväinen kielten ymmärtäminen.

NLP: *Natural language processing*. Luonnollisen kielen käsittely on tietojenkäsittelytieteen ja tekoälyn haara, joka keskittyy ihmiskielen ymmärtämiseen ja tuottamiseen koneiden avulla.

1 Johdanto

Teknologian nopea kehittyminen on tuonut mobiilisovellusten kehittämiseen uusia mahdollisuuksia ja haasteita. Mobiilisovellukset ovat nyt monimutkaisempia ja niiden odotetaan toimivan saumattomasti eri laitteissa ja käyttöympäristöissä. Tämä asettaa lisääntyneitä vaatimuksia sovellusten testaamiselle. Samanaikaisesti tekoälyn ja koneoppimisen kehittyminen on avannut ovia testausmenetelmien automatisoimiseksi, jolloin manuaalista työtä voidaan vähentää ja testaus voidaan suorittaa tehokkaammin.

Tässä opinnäytetyössä selvitetään, miten ChatGPT:llä voidaan hyödyntää mobiilisovellusten testauksen automatisoinnissa. ChatGPT on yksi viimeaikaisista tekoälymallien innovaatioista, ja se on osoittanut potentiaalinsa monilla alueilla, mukaan lukien luonnollisen kielen käsittely ja vuorovaikutteiset chatbotit. Työssä käydään läpi mobiilisovellustestauksen nykyisiä menetelmiä, ChatGPT:n potentiaalia ja haasteita testauksessa sekä sitä, miten tekoäly vaikuttaa ohjelmistotestaukseen ja muuttaako se testausprosesseja. Tämän tutkimuksen tavoitteena on tarjota syvällinen ymmärrys ChatGPT:n mahdollisuuksista mobiilisovellustestauksessa ja esittää suuntaviivoja sen tehokkaaseen hyödyntämiseen.

2 Manuaalinen ja automatisoitu mobiilisovellusten testaus

Mobiilisovellusten testaaminen tarkoittaa ohjelmiston testaamista älypuhelimissa ja tableteissa. Se on välttämätön osa sovelluksen kehitysprosessia monista syistä, erityisesti siksi, että se varmistaa sovellusten laadun, toimivuuden ja turvallisuuden. Koska mobiililaitteiden käyttö on nykyään yleistä ja monipuolista, sovellusten on toimittava moitteettomasti erilaisissa laitteissa, käyttöjärjestelmissä ja verkko-olosuhteissa.

Sovelluskehittäjillä on käytössään useita erilaisia testityyppejä ja menetelmiä näiden testien toteuttamiseen. On suositeltavaa, että laadunvarmistuksen tiimi testaa mobiilisovellukset. Riippumattoman testauksen etuna on, että se

vähentää kehittäjien nopeaan tuotteiden julkaisuun liittyvän puolueellisuuden riskiä. Matkapuhelinsovellusten testauksen lopullisena tavoitteena on havaita ohjelmistossa olevat ongelmat, määrittää ratkaisut näihin ongelmiin ja lopulta toimittaa korkealaatuinen tuote. (1.)

Matkapuhelinsovellusten testaaminen eroaa huomattavasti muiden ohjelmistojen, kuten verkkosivustojen tai tietokoneohjelmien, testaamisesta. Mobiilisovellusten kohdalla itse puhelimet ja tabletit, joilla sovellukset toimivat, vaikuttavat merkittävästi sovellusten toimintaan. Verkkosivustojen tai tietokoneohjelmien kohdalla käytettävän laitteen merkitys ei ole yhtä suuri, toisin kuin mobiilisovellusten testauksessa, jossa laitteisto vaikuttaa olennaisesti sovellusten toimintaan. Koska mobiilisovellusten käyttäjät ovat usein liikkeellä ja käyttävät erilaisia internet-yhteyksiä, testaajien täytyy huomioida myös näiden vaihtelevien olosuhteiden vaikutus. Kaiken tämän lisäksi on tärkeää ymmärtää, mitä käyttäjät odottavat mobiilisovelluksilta, ja tämä on olennainen osa testaajan tehtävää, jota ei voi sivuuttaa. (2, s. 8).

Manuaalisessa testauksessa testaaja ajaa testitapaukset käsin, kun taas automaattisessa testauksessa testitapaukset ajetaan automaattisesti ohjelmiston avulla. Mobiilisovelluksia voidaan testata sekä manuaalisesti että käyttäen automatisoitua testausta tai yhdistämällä näitä kahta menetelmää. Automaattinen testaus on joissakin tilanteissa nopeampaa kuin manuaalinen testaus. Automatisoidut testit mahdollistavat testien toistamisen tarpeen mukaan useita kertoja. Tämän ansiosta testaajat voivat keskittyä muihin tehtäviinsä.

Tuotekehitystä voidaan luotettavasti seurata reaaliajassa suorittamalla testejä jokaisen muutoksen jälkeen. Automaattinen testaus on erinomainen työkalu ohjelmistojen toimintahäiriöiden ja yleisten virheiden tunnistamiseen. Automaattista mobiilisovellustestausta hyödynnetään diagnostisena työkaluna, joka kattaa sovelluksen keskeiset tiedot ja tarjoaa arvokkaita tietoja, joihin voidaan nojata myöhemmissä testausvaiheissa. Testiautomaatio on avainasemassa, kun tavoitteena on saavuttaa ja ylläpitää laadunvarmistuksen

kokonaisvaltaista ja johdonmukaista lähestymistapaa. Tässä lähestymistavassa kaikki prosessit, menetelmät ja toimet on suunniteltu ja toteutettu tavalla, joka tähtää tuotteen tai palvelun laadun jatkuvaan parantamiseen ja ylläpitämiseen. Testiautomaation keskeinen hyöty on sen kyky mahdollistaa ohjelmistovirheiden tunnistaminen ja korjaaminen jo varhaisessa vaiheessa. Se auttaa hillitsemään myöhään havaittujen virheiden vahingollisia vaikutuksia ja ylläpitää myös palautteen kiertoa kehittäjä- ja tuotetiimien välillä. (3, § 1). Testin automatisointi on järkevää, jos sen manuaalinen suorittaminen vie paljon aikaa tai on muuten vaikeaa. Testiautomaation sovelluksia ovat yksikkö- ja integraatiotestaus, savutestaus, regressiotestaus ja osittain hyväksymistestaus.

Yksikkötestit ovat perustason testausta, jossa tarkistetaan, että ohjelmakoodin pienet, itsenäiset osat toimivat niin kuin pitää. Tämä tarkoittaa, että koodi jaetaan pieniin osiin, ja jokaiselle osalle tehdään omat testinsä. Yksikkötestit keskittyvät yksittäisen osan toimintaan, eivätkä ota huomioon kyseisen osan yhteyksiä muihin koodin osiin. Ohjelmistokehittäjät kirjoittavat yleensä yksikkötestit samanaikaisesti ohjelmiston kehittämisen kanssa. Heidän syvällinen ymmärryksensä koodin rakenteesta antaa heille mahdollisuuden kehittää tehokkaita testejä.

Integraatiotestauksessa keskitytään tarkistamaan, miten eri osat järjestelmässä toimivat yhdessä. Keskeistä on varmistaa, että niiden välillä oleva kommunikaatio on asianmukaista, jotta ne muodostaisivat saumattomasti toimivan kokonaisuuden.

Savutestaus on kuin sovelluksen ensimmäinen tarkastus. Siinä varmistetaan, että sovelluksen päätoiminnot toimivat tarpeeksi hyvin, jotta voidaan edetä tarkempiin testauksiin. Sovelluksen täytyy läpäistä tämä alustava savutestaus, ennen kuin sitä voidaan testata enemmän.

Regressiotestauksessa tarkistetaan kaksi asiaa. Ensiksi tarkistetaan, että ohjelmistoon tehtyjen muutosten jälkeen kaikki aiemmin toimineet ominaisuudet toimivat edelleen. Toiseksi varmistetaan, että jo korjatut virheet eivät ole

palanneet. Regressiotestauksessa tarkistetaan kaksi asiaa. Ensiksi, että ohjelmistoon tehtyjen muutosten jälkeen kaikki aiemmin toimineet ominaisuudet toimivat edelleen. Toiseksi varmistetaan, että jo korjatut virheet eivät ole palanneet. Tavoitteena on varmistaa, että uudet päivitykset eivät aiheuta ongelmia vanhoille toiminnoille.

Hyväksymistestauksessa loppuasiakas tai loppuasiakkaan valtuuttama edustaja varmistaa, että valmis tai lähes valmis tuote toimii suunnitellusti käytännön tilanteissa. Tämä testaus edellyttää ymmärrystä liiketoiminnasta, asiakkaiden vaatimuksista ja toimintatavoista sekä alaan liittyvistä säännöistä ja laeista. Tyypillisesti hyväksymistestaus ei keskity syvälliseen tekniseen vianetsintään tai koodin sisäiseen laadunvarmistukseen, jotka kuuluvat enemmän yksikkö- ja integraatiotestauksen sekä regressiotestauksen piiriin. Sen sijaan hyväksymistestauksen pääpaino on varmistaa, että tuote toimii suunnitellusti loppukäyttäjän näkökulmasta.

Toisin kuin ihminen, automaatio ei koskaan väsy pitkien ja uuvuttavien testitapausten suorittamiseen. Kun automatisointi on toteutettu oikein, testitehtäviä voidaan suorittaa samanaikaisesti useissa ympäristöissä. Vaikka automaatio tehostaa testausprosessia, se ei yksinään pysty korvaamaan manuaalista testausta kokonaan, sillä inhimillinen harkinta ja käyttäjäkokemuksen arviointi ovat korvaamattomia testauksen osa-alueita. (4.)

3 ChatGPT

ChatGPT on OpenAI:n kehittämä tekoälyyn perustuva luonnollisen kielen käsittelytyökalu, joka hyödyntää GPT-arkkitehtuuria. Generative Pre-trained Transformers (GPT), eli suomeksi generoivat esikoulutetu transformer -mallit, jotka kuuluvat Large Language Models (LLM, käsitteet määritellään kattavasti lyhenteet-osiossa) -kategoriaan eli suomeksi laajoihin kielimalleihin, on suunniteltu tuottamaan ihmisen tuottaman kaltaista tekstiä analysoimalla ja ymmärtämällä annettuja syötteitä. Näiden mallien ainutlaatuisuus perustuu esikoulutusprosessiin, jossa ne oppivat ennustamaan seuraavan tokenin, joka

voi edustaa sanaa tai sen osaa, ja ymmärtämään tokenien välisiä suhteita analysoimalla tekstikokoelmia, jotka kattavat miljardeja sanoja. Tämä oppimisprosessi mahdollistaa monipuolisten ja kontekstuaalisten vastausten tuottamisen. (5.)

Termi *token* GPT:ssä ja kielimalleissa viittaa tekstiyksikköön, jota malli käsittelee. Tämä voi olla sana, osa sanaa (kuten taivutuspääte tai juurimorfeemi), merkki tai jopa laajempi tekstinpätkä riippuen siitä, miten malli on koulutettu ja mitä tokenisointimenetelmää käytetään. Tokenisointi on siis prosessi, jossa syöteteksti jaetaan pienempiin osiin eli tokeneihin. Tokenisointimenetelmä voi vaikuttaa merkittävästi mallin suorituskykyyn ja kykyyn ymmärtää kieltä. Esimerkiksi jotkin menetelmät voivat paremmin käsitellä harvinaisia sanoja tai monimutkaisia kielioppirakenteita jakamalla ne ymmärrettävämpiin osiin. Kielimallit, kuten GPT, käyttävät näitä tokeneita oppimisprosessin aikana ja ennustettaessa uutta tekstiä, mahdollistaen monimutkaisten ja luonnolliselta vaikuttavien tekstien generoinnin.

Transformerit, jotka muodostavat GPT:n ytimen, ovat edistyneitä koneoppimisen malleja, jotka tutkijat esittelivät ensimmäisen kerran vuonna 2017. Ne on alun perin suunniteltu käännöstehtäviin, mutta niiden käyttö on laajentunut kattamaan erilaisia NLP-sovelluksia (NLP, käsitteet määritellään kattavasti lyhenteet-osiossa). Transformerit käyttävät itseohjautuvaa oppimista ja huomiokerroksia keskittyäkseen syötteen kannalta olennaisiin osiin, mikä tekee niistä erityisen tehokkaita kielen ymmärtämisessä ja tuotannossa. Niiden arkkitehtuuri, joka jakautuu syötettä ymmärtäviin enkoodereihin ja tuloksia tuottaviin dekoodereihin, mahdollistaa monimutkaisten tehtävien, kuten kääntämisen, suorittamisen sekvenssistä toiseen.

Siirto-oppiminen on keskeinen osa transformereiden tehokkuutta, sillä se mahdollistaa aiemmin hankitun tiedon soveltamisen uusiin, samankaltaisiin tehtäviin. Tämä oppimismenetelmä nopeuttaa uusien aihealueiden omaksumista ja tehostaa hienosäätöä erityisissä tehtävissä pienemmillä

resurssivaatimuksilla. Sen ansiosta GPT-mallit ovat erittäin joustavia ja soveltuvat monenlaisiin käyttötarkoituksiin. (6.)

Erilaiset organisaatiot hyödyntävät GPT-malleja ja generatiivista tekoälyä lukuisissa sovelluksissa, kuten kysymys- ja vastausbottien kehittämisessä, tekstin tiivistämisessä, sisällöntuotannossa ja tiedonhaussa. ChatGPT:n kyky tuottaa ihmismäistä tekstiä ja sisältöä, mukaan lukien kuvat, musiikki, videot ja muut, mahdollistaa sen käydä keskusteluja, jotka muistuttavat ihmisten välistä vuoropuhelua.

ChatGPT on koulutettu laajalla internet-tekstien valikoimalla, minkä ansiosta se pystyy suorittamaan laajan valikoiman tehtäviä, vastaamaan kysymyksiin, tarjoamaan selityksiä, osallistumaan keskusteluun ja tuottamaan luovaa sisältöä. Mallin kyky ymmärtää annettuja kehotteita ja tuottaa niihin sopivia vastauksia juontuu sen kyvystä oppia syvällisesti koulutusaineistosta. On kuitenkin tärkeää muistaa, että ChatGPT:n tietämys on rajattu siihen, mitä se on oppinut koulutuksensa aikana, eikä se kykene hankkimaan tai päivittämään tietojaan reaaliajassa. Viimeisimmän päivityksen, huhtikuun 2023, tietojen mukaan ChatGPT ei sisällä tietoa tämän päivämäärän jälkeen tapahtuneista tapahtumista tai kehityksistä. (7.)

3.1 GPT-mallit

GPT-1, joka julkaistiin vuonna 2018, oli suora sovellus transformer-mallista kielellisiin tehtäviin. GPT-1:ssä oli 117 miljoonaa parametria ja se koulutettiin BooksCorpus-tietoaineistolla. Sen suorituskyky erilaisissa luonnollisen kielen käsittelyn tehtävissä osoitti suurten kielimallien potentiaalin. (8.)

GPT-2 julkaistiin vuonna 2019 ja oli merkittävä parannus edeltäjäänsä verrattuna. GPT-2 oli 1,5 miljardin parametrin malli, joka oli koulutettu 8 miljoonan verkkosivun aineistolla nimeltä WebText. Se esitteli huomattavia kielellisen ymmärryksen ja tuottamisen kykyjä tuottaen yhtenäisiä ja kontekstiin sopivia tekstikappaleita. (8.)

Alun perin OpenAI epäröi julkaista GPT-3-mallia huolien vuoksi, jotka liittyivät potentiaaliseen väärinkäyttöön, kuten valeuutisten tuottamiseen. Vuonna 2020 esitelty GPT-3 oli suuri harppaus koossa ja kyvyissä. Malli, jossa oli 175 miljardia parametria, koulutettiin vielä suuremmalla ja monipuolisemmalla aineistolla, johon kuului kirjoja, verkkosivustoja ja muita tekstejä. Käytetyt viisi tietoaineistoa olivat Common Crawl, WebText2, Books1, Books2 ja Wikipedia. GPT-3 osoitti edistyneitä kykyjä tuottaa ihmisen kaltaista tekstiä, vastata kysymyksiin, kääntää, tiivistää, ja jopa luoda sisältöä, kuten runoja tai koodia. Sen API:n saatavuus mahdollisti erilaiset kolmannen osapuolen sovellukset. (8.)

GPT-3.5 on alaluokka GPT-3-malleista, jonka OpenAI loi vuonna 2022. GPT-3 kehitettiin edelleen GPT-3.5:ksi, jota sitten käytettiin Chatbot-palvelun kehittämiseen, joka tunnetaan nimellä ChatGPT. GPT-3.5-Turbo julkaistiin vuonna 2023. GPT-3.5-Turbon pääominaisuus on sen tarkka räätälöintimahdollisuus. GPT-3.5-Turbon räätälöintimahdollisuus tarjoaa kehittäjille ja yrityksille enemmän valinnanvaraa ja kontrollia siitä, miten he haluavat tekoälymallin toimivan heidän tarpeidensa mukaan. Se tarjoaa parannellun työkalun luoda entistä tarkempia, kontekstuaalisempia ja käyttäjäystävällisempiä tekoälysovelluksia. (9.)

GPT-4 julkaistiin maaliskuussa ja tehtiin yleisesti saataville kaikille kehittäjille heinäkuussa. GPT-4 Turbo julkaistiin marraskuussa 2023. GPT-4 edustaa merkittävää edistystä tekoälyn kielimallinnuksessa. Se on suuren mittakaavan multimodaalinen malli, mikä tarkoittaa, että se pystyy käsittelemään sekä teksti- että kuvatietoja tuottaakseen tekstiä. Tämä edustaa merkittävää edistysaskelta verrattuna sen edeltäjiin, jotka keskittyivät lähinnä tekstiin. GPT-4 koulutettiin kahdessa vaiheessa. Ensin mallille syötettiin suuria tekstiaineistoja internetistä, josta se koulutettiin ennustamaan seuraavaa sanaa. Toisessa vaiheessa ihmisarviointeja käytetään järjestelmän hienosäätämiseen prosessissa, jota kutsutaan vahvistusoppimiseksi ihmispalautteesta. Tämä menetelmä kouluttaa mallia kieltäytymään kehotteista, jotka ovat vastoin OpenAI:n määritelmiä haitallisesta käytöksestä. Tällaisia ovat esimerkiksi kysymykset laittomien toimintojen suorittamisesta, neuvot itselle tai muille vahingon aiheuttamiseksi,

tai pyynnöt järkyttävien, väkivaltaisten tai seksuaalisten sisältöjen kuvailemiseksi. GPT-4-Turbo on viimeisin OpenAI:n Generative Pretrained Transformer -mallien sarjassa. GPT-4-Turbo pystyy säilyttämään enemmän tietoa kuin GPT-4 suuremman konteksti-ikkunansa ansiosta. Tämä laajennettu kapasiteetti mahdollistaa mallin käsitellä laajempia tietomääriä yhdellä kehotteella, mikä tekee siitä tehokkaamman monimutkaisten tehtävien suorittamisessa. Molemmat ovat käytössä ChatGPT Plus -palvelussa. (9; 10). Alla olevassa taulukossa 1 on lueteltu viimeisimmät GPT-mallit tähän päivään mennessä, mukaan lukien päivämäärät, joihin asti ne on esikoulutettu, ja tokenien määrät.

Taulukko 1. Ajankohtaisimmat ChatGPT-mallit vuoden 2023 lopussa ja vuoden 2024 alussa (12).

Malli	**Opetus data**	**Kuvaus**	**Tokenit**
gpt-4-1106-preview	Huhti 2023 asti	GPT-4-Turbo Uusin GPT-4-malli, jossa on parannettu ohjeiden seuraamista. JSON-tila, toistettavat tulokset, rinnakkaiset funktionkutsut ja muuta. Tämä esikatselumalli ei ole vielä sopiva tuotantoliikenteeseen.	128 000
gpt-4-vision-preview	Huhti 2023 asti	GPT-4-vision-preview: Kyky ymmärtää kuvia, kaikkien muiden GPT-4-Turbon kykyjen lisäksi. Tämä on esikatselumallin versio, eikä se sovellu vielä tuotantoliikenteeseen.	128 000
gpt-4	Syys 2021 asti	GPT-4 ylittää suorituskyvyllään aiemmat suuret kielimallit. MMLU-testissä, joka on englanninkielinen monivalintakysymysten kokoelma 57 eri aihealueesta, GPT-4 ei ainoastaan merkittävästi ylitä olemassa olevia malleja englannin kielessä, vaan osoittaa myös vahvaa suorituskykyä muilla kielillä.	8 192
gpt-3.5-turbo-1106	Syys 2021 asti	GPT-3.5-Turbo – sarjan uusin malli, jossa on parannettu ohjeiden	16 385

		seuraaminen, JSON-tila, toistettavat tulokset, rinnakkaiset funktiokutsut ja muuta.	

Tulevat mallit, kuten GPT-5 ja niiden jälkeiset, jatkanevat trendiä kohti suurempia ja monipuolisempia kielimalleja. Tämä kehitys saattaa mahdollistaa vielä tarkemman kielen ymmärryksen ja tuotannon tekoälyjärjestelmissä sekä uusien sovellusalueiden kehittymisen.

On kuitenkin tärkeää muistaa, että suuremmat ja monimutkaisemmat mallit tuovat mukanaan omat haasteensa. Esimerkiksi laajempien mallien kouluttaminen ja ylläpito vaativat merkittäviä laskennallisia resursseja, ja suurten mallien käyttö voi olla energiatehokkuuden kannalta haastavaa. Myös eettisten kysymysten, kuten tekoälyn käytön seurausten ja vastuiden, pohtiminen on tärkeää.

Tällä hetkellä on vaikea ennustaa tarkasti, mitä tulevat mallit tuovat tullessaan. Kuitenkin alalla tehdään jatkuvasti tutkimusta tekoälyn ja koneoppimisen parissa, mikä auttaa paremman ymmärryksen saavuttamisessa näistä teknologioista ja niiden potentiaalista. (11.)

3.2 ChatGPT:n rajoitukset

Vaikka ChatGPT on tehokas, se ei ole virheetön. Se voi tuottaa virheellisiä tai epäloogisia, eikä se pysty hakemaan tai vastaanottamaan tietoa internetistä reaaliajassa. ChatGPT voi hyödyntää vain niitä tietoja, joihin se on koulutettu viimeisimmän päivityksensä ajankohtaan asti.

Joskus käyttäjien on muotoiltava kysymyksensä uudelleen useita kertoja saadakseen tyydyttävän vastauksen ChatGPT:ltä. Suurempi rajoitus on vastausten vaihteleva laatu: vastaukset saattavat toisinaan näyttää uskottavilta, vaikka eivät olekaan, tai ne voivat olla tarpeettoman monimutkaisia ja vaikeaselkoisia. Sen sijaan, että pyytäisi selvennystä epäselviin kysymyksiin,

malli vain arvaa, mitä kysymys tarkoittaa, mikä voi johtaa odottamattomiin vastauksiin. (7.)

GPT-malli ei todellisuudessa ymmärrä kieltä samalla tavalla kuin ihmiset, vaikka se pystyykin muodostamaan erittäin yksityiskohtaisen käsityksen siitä, miten eri käsitteet liittyvät toisiinsa. Sen kyky tuottaa tekstiä ei perustu todelliseen tunneymmärrykseen tai loogiseen ymmärrykseen, vaan ennemminkin siihen, miten usein tietyt sanat ja lauseet esiintyvät koulutusaineistossa. Vaikka GPT voi jäljitellä tunteiden ilmaisua tekstissä, se ei itse tunne eikä prosessoi tunteita. Samoin, kuin ihmiset käyttävät kieltä tietyn tarkoituksen tai intention välittämiseen, GPT:n tuottama teksti syntyy ilman tällaista tietoista päämäärää tai itsetietoisuutta. Malli pystyy ymmärtämään välittömän kontekstin, kuten keskustelun aiemmat viestit, mutta sen kyky muistaa pitkän aikavälin tietoja tai ymmärtää laajoja keskusteluhistorioita ja syvällisiä kontekseja on rajallinen.

ChatGPT:n kehittäjät ovat tunnustaneet ensimmäisinä, että se voi tuottaa virheellistä (ja mahdollisesti haitallista) tietoa, mutta he työskentelevät kovasti sen korjaamiseksi. (11; 13).

3.3 ChatGPT:n hyödyt

ChatGPT, edistynyt tekoälyyn perustuva chatbot-järjestelmä, tarjoaa lukuisia etuja monenlaisiin käyttöyhteyksiin. Sen korkean tarkkuuden kielimalli ymmärtää ja reagoi ihmisten tekstiin tehokkaasti, mikä tekee siitä arvokkaan työkalun erityisesti asiakaspalveluviestinnässä. ChatGPT on erityisen hyödyllinen yrityksille, sillä se voi automatisoida rutiininomaisia tehtäviä ja vapauttaa ihmistyövoiman keskittymään vaativampiin tehtäviin. Tämä ei ainoastaan tehosta toimintaa ja takaa yhtenäistä palvelukokemusta, vaan myös varmistaa vastausten korkean laadun. Malli kehittyy jatkuvasti käyttäjävuorovaikutuksen myötä parantaen kykyään ymmärtää kieliä ja tuottaa tarkempia vastauksia ilman, että se säilyttäisi henkilökohtaisia tietoja. Monikielinen tuki laajentaa sen soveltuvuutta globaalisti.

ChatGPT:n tehokkuus monimutkaisten tehtävien käsittelyssä on erityisen arvokasta digitaalisessa markkinoinnissa, mahdollistaen yritysten sisällön tuottamisen ja optimoinnin hakukoneoptimointia varten. Lisäksi ChatGPT:n luonnollinen vuorovaikutuskyky asiakkaiden kanssa edistää positiivisempia käyttäjäkokemuksia ja vahvistaa asiakassuhteita. Kaiken kaikkiaan ChatGPT tarjoaa merkittävää lisäarvoa asiakaspalveluun ja on tehokas väline yritysten kehityksen tukemiseen parantaen asiakastyytyväisyyttä ja sitoutumista. (15.)

Yllä annetut esimerkit ovat vain osa ChatGPT:n mahdollisista eduista. Lisäksi ChatGPT:n oppimis- ja parantamisprosessi on jatkuvaa. Ajan myötä malli muuttuu tarkemmaksi, luotettavammaksi ja kontekstuaalisesti keskittyneemmäksi jatkuvan analyysin ja miljoonien käyttäjien palautteen sisällyttämisen ansiosta. Tämä tarkoittaa, että vastaukset tulevat yhä enemmän kohdennetuiksi ja oleellisiksi käyttäjien esittämiin kysymyksiin. Jatkuvan oppimisprosessin ansiosta ChatGPT kykenee pitämään itsensä ajan tasalla uusimmista tiedoista ja trendeistä taaten käyttäjille erittäin täsmälliset ja ajanmukaiset vastaukset. Lisäksi ChatGPT:n kyky oppia virheistään ja parantaa suorituskykyään takaa, että ajan myötä se tarjoaa luotettavamman ja miellyttävämmän käyttäjäkokemuksen.

3.4 ChatGPT:n rooli ja merkitys testauksessa

Tekoälyn kehittyessä on nähty useita innovatiivisia sovelluksia ohjelmistokehityksessä ja -testauksessa. ChatGPT on noussut merkittäväksi työkaluksi monilla alueilla, mukaan lukien ohjelmistotestaus.

Yksi ChatGPT:n vahvuuksista on sen kyky prosessoida luonnollista kieltä, mikä nostaa sen erinomaiseksi työkaluksi automatisoiduissa testiskenaarioissa, joissa vaaditaan monimutkaisia komentoja tai ohjeita. ChatGPT voi mukautua uusiin testausympäristöihin nopeasti, mikä vähentää tarvetta jatkuvasti päivittää testiskenaarioita.

Testausprosessia voi nopeuttaa ja yksinkertaistaa testitapausten suunnittelulla, jossa hyödynnetään ChatGPT:n kaltaisia tekoälypohjaisia työkaluja. Esimerkiksi tilanteissa, joissa korkean tason suunnitellut testitapaukset jättävät huomiotta kriittiset yksityiskohdat, voi testaajan työ helposti suuntautua harhaan. Tällöin tekoälyä hyödyntävä ChatGPT voi olla arvokas apuväline. Se kykenee luomaan tarkempia ja kattavampia testitapauksia, jotka paranevat ja tarkentuvat joka kerta, kun samaa kysymystä toistetaan. Tämän ansiosta testauksen laatua voidaan merkittävästi parantaa. Lisäksi ChatGPT voi luoda testitapauksia lukuisiin pyyntöihin, tarjoamalla esimerkiksi

- kirjautumislomakkeen esimerkkitiedot
- virheellisten tietojen testaus
- sähköisen kaupankäynnin transaktioiden testausideat
- varauksien ja päivämäärän tai ajan valinnan testit.

Toinen tapa nopeuttaa testausta on suunnitella intuitiivisia ja käyttäjäkeskeisiä testiskenaarioita, joissa ChatGPT:n kyky käsitellä ja tuottaa luonnollista kieltä tulee esiin. Tämä ominaisuus on erityisen arvokas käyttäjäkokemuksen testaamisessa, jossa keskeisenä päämääränä on selvittää, kuinka loppukäyttäjät kokevat sovelluksen ja miten he sitä käyttävät.

Kolmas keino on automaattisen testauksen optimoinnin hyödyntäminen, jossa ChatGPT:n rooli korostuu. Antamalla sille tarkat ohjeet, kuten koodin generoinnin testitapausten automatisointiin käyttäen Javaa, JUnitia ja muita teknologioita, ChatGPT voi automatisoida testitapauksia. Lisäksi se tarjoaa kattavat selitykset luomalleen automaatiokoodille, mikä tehostaa testausprosessia merkittävästi.

ChatGPT:n neljäs käyttötapa on auttaa tunnistamaan virheitä tai puutteita testikoodissa ja ehdottaa korjauksia reaaliaikaisesti. Tämä tehostaa testausprosessia ja parantaa ohjelmistojen laatua. Lisäksi ChatGPT mahdollistaa virhetietojen raportoinnin luonnollisella kielellä, tarjoten näin käyttäjäystävällisen välineen virheiden käsittelyyn.

Vielä yksi tapa nopeuttaa testausprosessia on kirjoittaa kyselyitä tietokantaan. Esimerkiksi ChatGPT voi tarjota esimerkkikoodia SQL- tai NoSQL-kyselyihin, jotka liittyvät haun toteuttamiseen ja tarvittavien tietojen etsimiseen tietyistä taulukoista. (16.)

4 ChatGPT:n integrointi Appium-kehykseen

Testausprosessien automatisointi ChatGPT:n avulla vaati järjestelmäriippumattoman lähestymistavan, koska suunnitelmissa on testata mobiilisovelluksia sekä Androidilla että iOS:lla. Tähän tavoitteeseen JavaScript osoittautui sopivaksi kieleksi. Itse sovellustestaukseen valitsin työkaluiksi Appiumin, WebDriverIO-sovelluskehyksen sekä OpenAI:n API-avaimen, joiden avulla voin automatisoida testauksen ja mahdollistaa kommunikoinnin GPT:n kanssa.

4.1 Appium

Appium on avoimen lähdekoodin työkalu, joka on suunniteltu erityisesti ohjelmistotestauksen automatisointiin. Sen tarkoituksena on varmistaa, että sovelluksen toiminnot toimivat odotetusti. Erilaisten testaustyyppien joukossa automatisointi erottuu kyvyllään sallia testaajien luoda koodiskriptejä. Nämä skriptit jäljittelevät käyttäjien toimintaa sovelluksen käyttöliittymässä pyrkien jäljittelemään reaalimaailman käyttäytymistä niin uskollisesti kuin mahdollista.

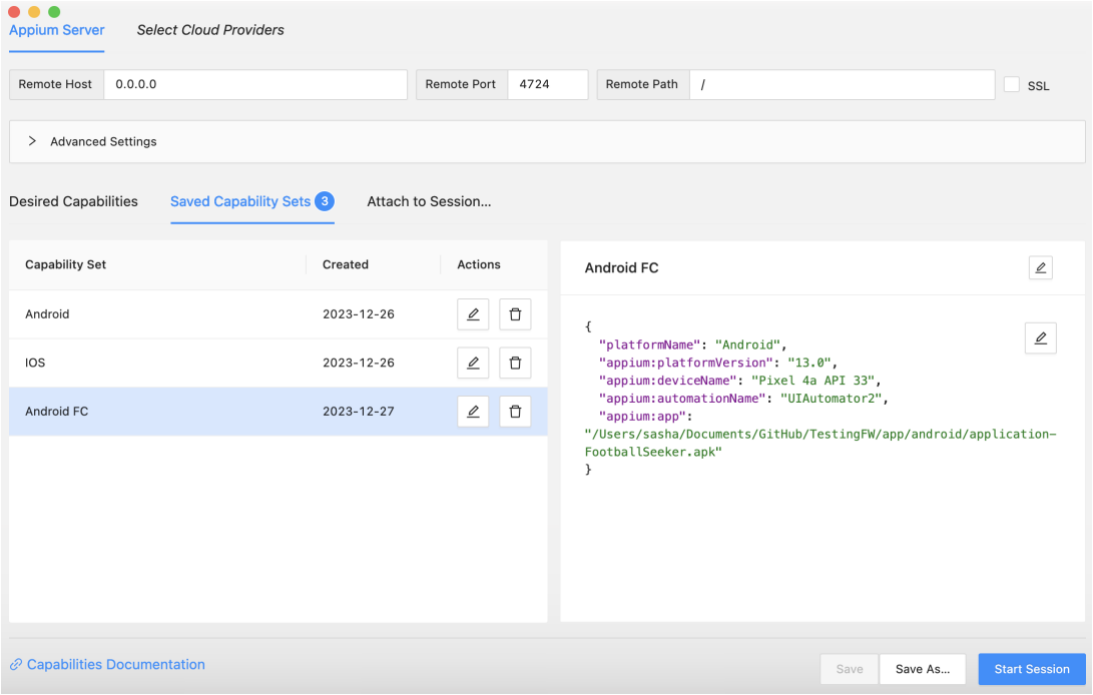
Lisäksi tämä lähestymistapa hyödyntää automaation tarjoamia etuja, kuten tehostettua nopeutta, laajennettavuutta ja toistettavuutta. (17.) Appium ei ole itsenäisesti toimiva työkalu. Se suorittaa testitapaukset käyttäen WebDriverIO-rajapintaa. Se mahdollistaa kehittäjille natiivien tai hybridien iOS- ja Android-sovellusten testauksen automatisoinnin.

Appiumin ominaisuudet ovat

- Appium mahdollistaa laadunvarmistustestaajille testiskriptien luomisen useilla ohjelmointikielillä: Java, JavaScript, PHP, Ruby, Python ja C#.
- Appiumin monialustainen kehys mahdollistaa yhteensopivien testiskriptien luomisen Windowsille, iOS:lle ja Androidille käyttäen samaa APIa.
- Käyttäjät voivat uudelleenkäyttää lähdekoodiaan Androidille ja iOS:lle; Appium ei vaadi sovelluksen lähdekoodin muokkausta.
- Appium tukee natiivien, hybridien ja mobiiliverkkosovellusten automatisointia kattaen laajan valikoiman sovellustyyppejä.
- Appium integroituu saumattomasti monialustaisten mobiilisovellusten kehitysalustoihin, kuten React Nativeen, Xamarinin ja Flutteriin.
- Appium tarjoaa sisäänrakennettuja työkaluja, kuten Appium Desktopin ja Appium Inspectorin, sovelluselementtien tarkasteluun ja testiskriptien generointiin. (18.)

On olemassa kahta eri versiota kuten: Appium 1.x ja Appium 2.x. Appium 2.x versio. Appium 2.x voi käyttää Appium Inspector kautta. Appium Inspector on mobiilisovellusten ja muiden sovellusten GUI-tarkastaja, joka toimii erikseen asennettavalla Appium-palvelimella (kuva 1).

Käytännössä Appium Inspector toimii Appium-ohjelmana, samankaltaisesti kuin WebDriverIO, Appiumin Java-asiakas ja Appiumin Python-asiakas, mutta se on varustettu käyttöliittymällä. Appium Inspector tarjoaa käyttöliittymän, jonka avulla voi valita Appium-palvelimen, määritellä asetuksia ja vuorovaikuttaa elementtien sekä muiden Appium-komentojen kanssa.

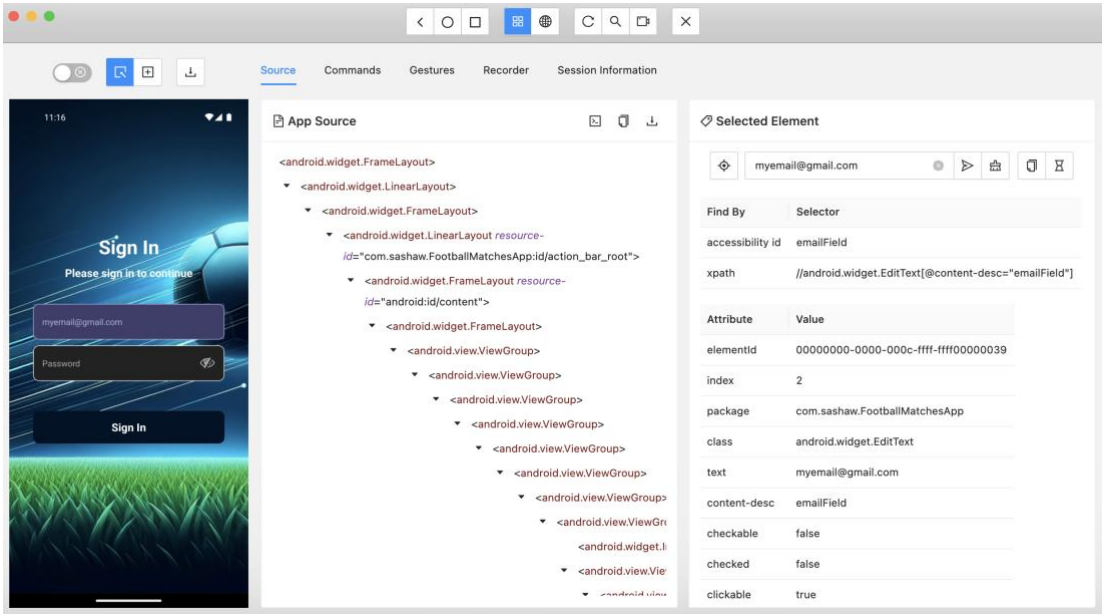


Kuva 1. Appium Inspector -käyttöliitymä.

Tämä versio sisältää tiettyjä ominaisuuksia. Se tarjoaa visuaalisen esityksen sovelluksen käyttöliittymästä, jolloin testaajat voivat olla vuorovaikutuksessa sovelluksen elementtien kanssa ja tarkastella niiden ominaisuuksia. Appium Inspector on hyödyllinen elementtien tunnistamiseen. Se on erityisen arvokas, kun on tarve tunnistaa ja vuorovaikuttaa elementtien kanssa mobiilisovelluksen käyttöliittymässä. Appium Inspector tarjoaa visuaalisen esityksen sovelluksen elementeistä, mikä helpottaa tiettyjen käyttöliittymäkomponenttien, kuten painikkeiden, tekstikenttien, kuvien ja muiden, paikantamista ja valitsemista. Jos tarvitsee kirjoittaa automaatioskriptejä mobiilisovellusten testaamiseen ja vuorovaikuttaa sovelluksen kanssa Inspectorissa, on mahdollista generoida koodikatkelmia sellaisiin toimiin kuten painikkeiden painallukset, tekstin syöttäminen ja elementtien tarkistukset.

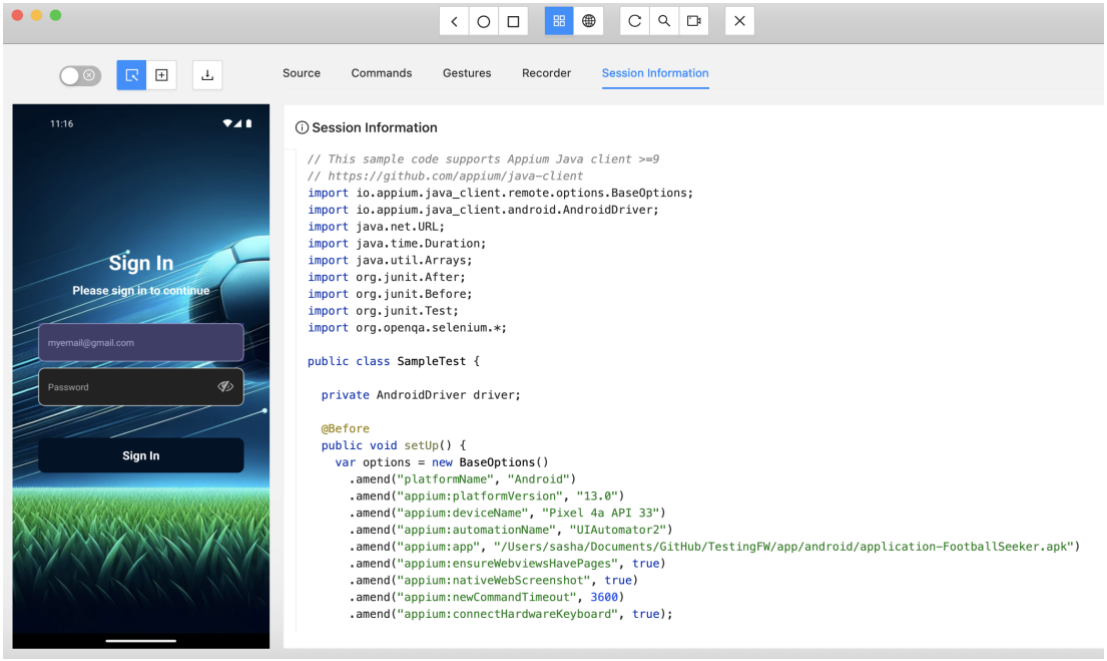
Appium Inspector näyttää erilaisia elementtien attribuutteja ja ominaisuuksia, jotka se on tunnistanut. Tämä tieto on erittäin tärkeää luotettavien automaatioskriptien luomisessa. Appium Inspectorin avulla voidaan luoda XPath-lausekkeita elementeille. XPath on tehokas menetelmä elementtien

etsimiseen XML-kaltaisissa rakenteissa, ja sen käyttö voi auttaa varmistamaan, että automaatioon kohdistetut skriptit ovat oikein kohdistettuja, vaikka niiden sijainti tai attribuutit olisivat muuttuneet (kuva 2).



Kuva 2. Istunto Appium Inspectorissa Android-sovellukselle. Näyttö, joka esittää sovelluksen komponentteja.

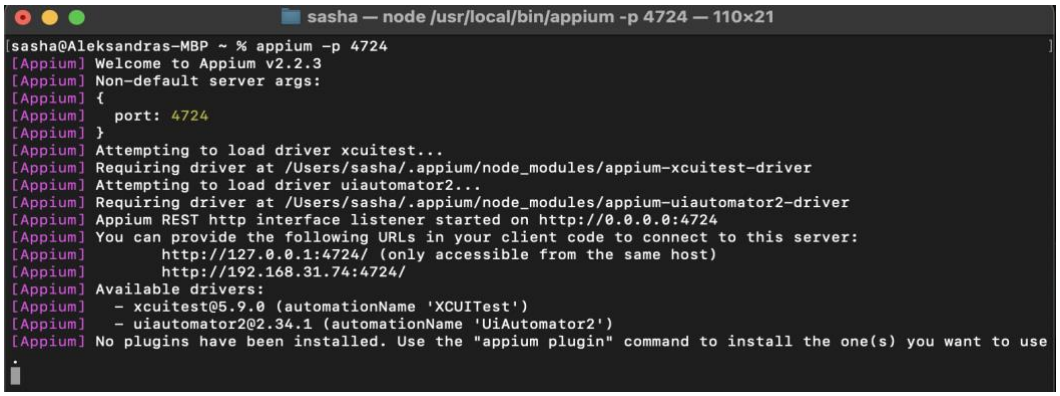
Jos on tarvetta testata mobiilisovellusta, joka on suunniteltu toimimaan useilla alustoilla (kuten iOS ja Android), Appium Inspector tarjoaa yhtenäisen käyttöliittymän elementtien tarkasteluun ja vuorovaikutukseen eri alustoilla. Jotkut Appium Inspectorin versiot tukevat käyttäjän vuorovaikutuksen tallentamista sovelluksen kanssa ja vastaavien automaatioskriptien generointia. Tämä voi olla erityisen hyödyllistä, jos tarvitsee nopeasti generoida testausskriptejä manuaalisten vuorovaikutusten perusteella (kuva 3).



Kuva 3. Istunto Appium Inspectorissa Android-sovellukselle. Näkymä, joka esittelee valmiiksi asennetun skriptin.

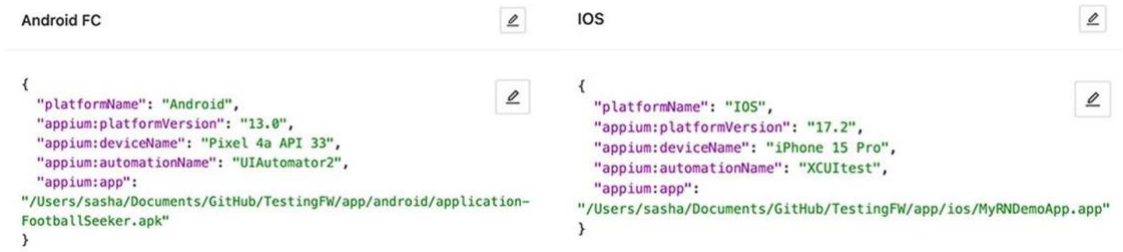
Kun automaatioskriptit eivät toimi odotetulla tavalla, Appium Inspectorin käyttö voi auttaa diagnosoimaan ongelmia. Testaajat voivat tarkistaa elementtien tunnistuksen tarkkuuden, tutkia elementtien ominaisuuksia ja tehdä tarvittavia säätöjä skripteihin. (19.)

Appium Inspectorin käyttöönottoon kuuluu tarvittavien työkalujen ja komponenttien asentaminen, Inspectorin käynnistäminen ja sen käyttäminen elementtien tunnistamiseen ja vuorovaikutukseen mobiilisovelluksesi käyttöliittymässä. Appium Inspectorin käynnistämiseen ja oikeaan toimintaan tarvitaan Appium-palvelimen käynnistäminen terminaalin kautta. Tämän voi suorittaa käyttämällä komentoa *appium* tai *appium -p* ####, jossa #### viittaa käytettävään porttinumeroon (kuva 4).



Kuva 4. Terminaalissa käynnistetty Appium-palvelin 4724 portilla.

Seuraavaksi tulee määritellä käytettävät ominaisuudet eli *capabilities*, jotka vaihtelevat iOS:lla ja Androidilla (kuva 5). Esimerkiksi Android-sovellusten testauksessa keskeinen työkalu on UIAutomator2, joka on erityisesti suunniteltu automatisoimaan natiiveja, hybridejä ja web-sovelluksia Android-laitteissa, mukaan lukien emulaattoreissa ja fyysisissä laitteissa. Appium UiAutomator2 Driver on osa Appiumin mobiilites-tausautomaation työkalua. iOS-sovellusten testauksessa käytetään vastaavia, mutta erilaisia työkaluja, jotka on räätälöity Applen ekosysteemin tarpeisiin. iOS:lle on oma ajurinsa XCUItest. XCUItest on Appium-ajuri iOS-sovellusten automatisointiin iOS:ssä, iPadOS:ssa ja tvOS:ssa.



Kuva 5. Android ja iOS ominaisuudet.

Capabilities-osiossa tulee myös määritellä testattava alusta, alustan versio, laite, ajuri ja polku testattavan sovelluksen tiedostoon. Viimeisenä tarvittavana elementtinä Appium Inspectorin käynnistämiseen ovat Android-emulaattori tai iOS-simulaattori. Kun kaikki on asennettu, voidaan aloittaa istunto.

Appium Inspectorin avulla voidaan nopeuttaa ja osittain automatisoida testausta käyttämällä ChatGPT:tä. Helpoin tapa on syöttää kyselyt suoraan verkkoversion chattiin ja saada niihin vastauksia. Kuitenkaan tämä lähestymistapa ei täysin vastaa tämän insinöörityön tavoitetta. Siitä huolimatta Appium Inspector voi olla hyödyllinen käyttöliittymäelementtien etsinnässä ja niiden reittien löytämisessä, erityisesti jos testaaja ei ole etukäteen päässyt käsiksi testattavaan sovellukseen tai ei ole sen kehittäjä.

On olemassa toinen vaihtoehto, miten Appiumia voidaan käyttää. Jotta Appium voidaan käynnistää erillisessä JavaScript-projektissa, johon myöhemmin liitetään OpenAI API, on käytettävä WebDriverIO:ta, joka sisältää oman testiajonsa.

4.2 WebDriverIO

WebDriverIO on Node.js:lle tarkoitettu avoimen lähdekoodin testaustyökalu, jonka avulla voit automatisoida testauksen. WebDriverIO on monipuolinen kehys, joka soveltuu sekä verkkosivustojen että mobiilisovellusten testaamiseen. Sen avulla on mahdollista suorittaa pieniä ja kevyitä komponenttitestejä sekä ajaa End-to-End-testiskenaarioita (End-to-End, käsitteet määritellään kattavasti lyhenteet-osiossa) selaimessa tai mobiililaitteella. Tämä takaa, että testaus on mahdollista suorittaa ympäristössä, jota käyttäjät käyttävät. Sisältäen älykkäitä valintastrategioita, jotka helpottavat vuorovaikutusta esimerkiksi React-komponenttien kanssa. Koska vuorovaikutukset tapahtuvat standardoidun automaatioprotokollan kautta, on taattua, että ne toimivat natiivisti eivätkä ole vain JavaScriptin emuloimia.

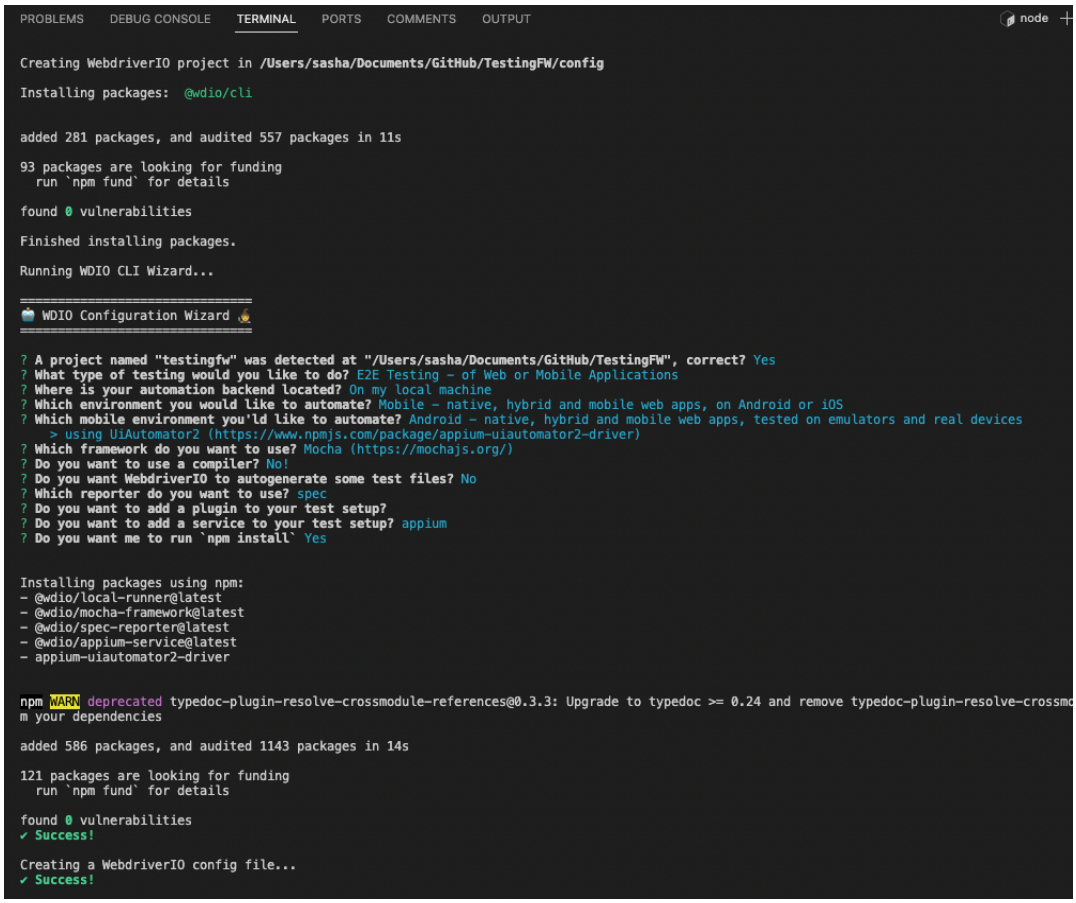
WebDriverIO tarjoaa oman testiajonsa, joka nopeuttaa testauksen aloittamista huomattavasti. Testiajon tavoitteena on hoitaa suurin osa testausprosessista, mahdollistaa integraation kolmansien osapuolten palveluihin ja tukea testien suorittamista mahdollisimman tehokkaasti (esimerkkikoodi 1).

```
exports.config = {
    // Runner Configuration
    runner: 'local',
    port: 4723,
    // Specify Test Files
    specs: [
        '../test/specs/android/*.js'
    ],
    // Patterns to exclude.
    exclude: [
        // 'path/to/excluded/files'
    ],
    // Capabilities
    maxInstances: 10,

    capabilities: [{
        // capabilities for local Appium web tests on an Android
Emulator
        "platformName": 'Android',
        "appium:deviceName": 'Pixel 4a API 33',
        "appium:platformVersion": "13.0",
        "appium:automationName": "UIAutomator2",
        "appium:app": androidAppPath,
    }],
    // Test Configurations
    // bail (default is 0 - don't bail, run all tests).
    bail: 0,
    // Set a base URL in order to shorten url command calls. If your
`url` parameter starts
    baseUrl: '',
    // Default timeout for all waitFor* commands.
    waitforTimeout: 10000,
    // Default timeout in milliseconds for request
    connectionRetryTimeout: 120000,
    connectionRetryCount: 3,
    services: ['appium'],
    framework: 'mocha',

    reporters: [['allure', {
        outputDir: 'allure-results',
        disableWebdriverStepsReporting: true,
        disableWebdriverScreenshotsReporting: true,
    }]],
    // Options to be passed to Mocha.
    mochaOpts: {
        ui: 'bdd',
        timeout: 60000
    },
}
```

Esimerkkikoodi 1. WebDriverIO:n konfiguraatiotiedosto.

WebDriverIO:n käyttöönotto alkaa komennolla *npm install @wdio/cli*. Sen jälkeen on määriteltävä konfiguraatiotiedosto, johon määritellään tiedot testeistä, mahdollisuuksista ja asetuksista (kuva 6). WebDriverIO:n konfiguraatioavustajan avulla konfiguraatiotiedoston luominen on tehty hyvin

yksinkertaiseksi. Tämä tapahtuu suorittamalla komento *npx wdio config*. Avustajaohjelma käynnistyy, esittää käyttäjälle kysymyksiä ja luo tarvittavan konfiguraatiotiedoston.



Kuva 6. WebDriverIO:n konfiguraatio apuri näyttö.

Konfiguraatiotiedoston asentamisen jälkeen on tarpeen syöttää tiedot ominaisuudesta ja polku testiskriptiin tai kansioon, joka sisältää kaikki tarvittavat skriptit, koska wdio.config voi testata skriptejä jonotusjärjestyksessä. Konfiguraatiotiedoston asetusten määrittelyn jälkeen testauksen voi aloittaa suorittamalla komennon *npx wdio run wdio.conf.js*. (20.)

4.3 Allure Reporter

Allure Reporter on WebDriverIO:n laajennus, joka tuottaa Allure-testiraportteja. Allure WebDriverIO -adapteri parantaa WebDriverIO:n

vakioraportointiominaisuuksia lisäämällä informatiivisia ja rakenteellisesti parempia testiraportteja varten tarkoitettuja lisäominaisuuksia. Allure raporteissa voidaan käyttää monenlaista metatietoa, joka antaa kontekstin ja yksityiskohdat kullekin testitapaukselle, tekee raporteista entistä hyödyllisempiä. Tämän laajennuksen avulla on mahdollista hyödyntää seuraavia ominaisuuksia

- Kuvausten, linkkien ja muiden metadatatietojen lisääminen testiraportteihin.
- Testien järjestäminen selkeisiin hierarkioihin, mikä parantaa niiden luetettavuutta ja organisointia.
- Testien jakaminen pienempiin askeleisiin, mikä helpottaa niiden ymmärtämistä ja ylläpitoa.
- Parametrisoitujen testien parametrien selkeä kuvaus erilaisten skenaarioiden määrittämiseksi.
- Kuvakaappausten ja muiden tiedostojen automaattinen tallentaminen testin suorituksen aikana.
- Täydellisten ympäristötietojen sisällyttäminen testiraporttiin. (21.)

Allure Reporter helpottaa testaustulosten tarkastelua ja analysointia. Sen avulla kehittäjät ja laadunvarmistusammattilaiset voivat nopeasti tunnistaa ja paikantaa ohjelmiston kehitysprosessin ongelmia, mikä nopeuttaa virheiden korjaamista ja parantaa tuotteen laatua. Esimerkiksi Alluren integrointi projekteihiin mahdollistaa automatisoidun testiraportoinnin tarjoten jatkuvaa palautetta ohjelmiston tilasta ja edistyksestä. Tämä integrointi tekee testiraporteista olennaisen osan ohjelmistokehityksen ja laadunvarmistuksen prosessia.

4.4 Mocha

Testien generointiin käytetään Mochaa. Mocha on suosittu JavaScript-testikehys, joka on kehitetty Node.js- ja selainympäristöissä toimivien sovellusten testaamiseen. Se tarjoaa kehittäjille mahdollisuuden kirjoittaa testitapauksia eri tavoin, esimerkiksi asynkronisesti käyttäen kutsuntafunktioita, promiseja tai async/await-syntaksia. Mocha mahdollistaa joustavan ja helppokäyttöisen lähestymistavan testien suorittamiseen ja raportointiin, mikä

tekee siitä erinomaisen valinnan monimutkaisten sovellusten testaukseen. Mocha-testit suoritetaan peräkkäin, mikä takaa tarkan ja joustavan raportoinnin samalla kun käsittelemättömät poikkeukset yhdistetään oikeisiin testitapauksiin. Koska Mocha on suosittu, siihen on saatavilla runsaasti lisäosia ja integraatioita, kuten Chai-kirjasto, joka tarjoaa voimakkaampia väitteitä testaukseen. (22.)

Mocha mahdollistaa *before*, *after*, *beforeEach* ja *afterEach* -koukkujen käytön, joiden avulla voidaan suorittaa koodia ennen ja jälkeen testien tai testisarjojen. Tämä on hyödyllistä esimerkiksi resurssien alustamisessa ja siivoamisessa. Mochan avulla kehittäjät pystyvät varmistamaan, että heidän koodinsa toimii odotetusti ja tunnistamaan mahdolliset ongelmat jo kehitysprosessin varhaisessa vaiheessa. Tämä prosessi auttaa parantamaan ohjelmiston laatua ja vähentämään virheitä tuotannossa.

4.5 OpenAi API

Yhteyden muodostamiseksi ChatGPT:hen ja kommunikointiin sen kanssa käytetään sen API:a ja siihen liittyvää avainta. Tässä projektissa hyödynnetään myös gpt-3.5-Turbo-mallia. Kaikki API-pyynnöt ovat maksullisia, ja niiden hinta määräytyy käytössä olevan mallin sekä tokenien määrän perusteella. Mikäli tarve vaatii laajempien pyyntöjen käsittelyä, tarvitaan todennäköisesti enemmän tokeneita, mikä puolestaan saattaa edellyttää kalliimman mallin käyttöä. Tällä hetkellä gpt-3.5-turbo on OpenAI:n tarjoamista malleista edullisin. Kalliimpia vaihtoehtoja, kuten gpt-4 ja gpt-4-Turbo, testataan myös erojen tutkimiseksi vastausten laadussa.

OpenAI API:n käyttöönottoa varten on ensin asennettava OpenAI:n Node.js-kirjasto. Sen jälkeen pyynnön lähetys tulee konfiguroida virallisen dokumentaation ohjeiden mukaisesti. Alla on konfiguraatio (esimerkkikoodi 2), joka on käytössä tässä projektissa GPT-3.5-kielimallin kanssa. (23.)

```
const OpenAI = require("openai").default;
require('dotenv').config();

// Initialize the OpenAIApi client using the API key
const openai = new OpenAI({
    apiKey: process.env.OPENAI_API_KEY
});
/**
 * Generates test code or text based on a prompt using the OpenAI API.
 * @param {string} prompt The prompt to send to the OpenAI API for
generating text.
 * @returns {Promise<string|null>} The generated text or null if an
error occurs.
 */
async function generateTestCode(prompt) {
    try {
        // Call the OpenAI API's Completions.create method
        const completion = await openai.chat.completions.create({
            model: "gpt-4", // Ensure to use the correct model name
"gpt-4-1106-preview" "gpt-3.5-turbo-1106" "gpt-4"
            messages: [
            { role: "system", content: "You are a helpful assistant."
},
            { role: "user", content: prompt }
            ],
            max_tokens: 1550 // maximum tokens
        });
        console.log("GPT Response:",
completion.choices[0].message.content);
        // Return the text content of the first choice in the response
        return completion.choices[0].message.content;
    } catch (error) {
        console.error("Error calling OpenAI API:", error);
        return null;
    }
}

module.exports = {
    generateTestCode,
};
```

Esimerkkikoodi 2. Kehyksen kytkiminen GPT:hen OpenAI:n rajapinta-avaimella.

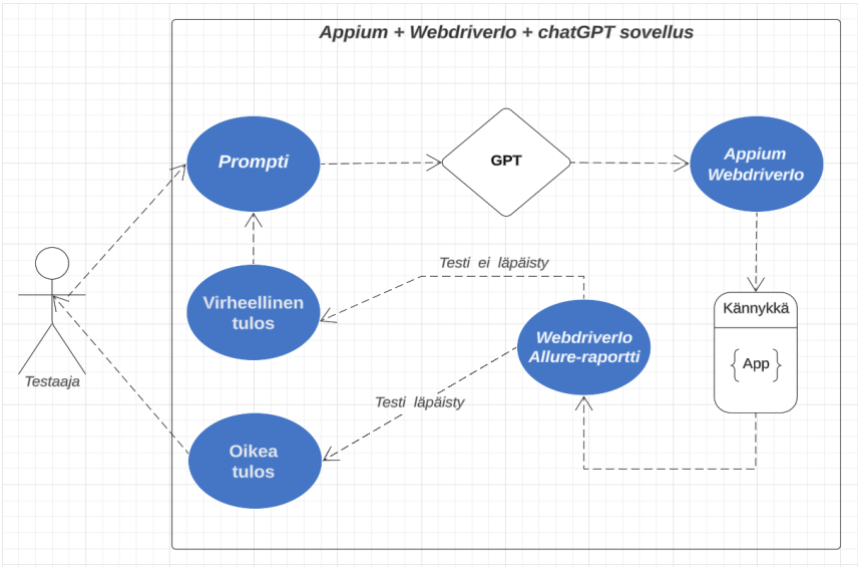
API-avaimen hankkiminen edellyttää aluksi rekisteröitymistä OpenAI:n verkkopalveluun. Kun rekisteröintiprosessi on suoritettu, käyttäjän profiilin API-avaimet-nimisestä osiosta on mahdollista luoda uusi avain. Samaisessa osiossa tarjotaan mahdollisuus tarkastella yhteyksien käyttötilastoja.

5 Testaus

Testauksessa käytettiin pääasiassa React Native -sovelluksen Android-versiota, sillä APK-tiedoston luominen on helpompaa. iOS-sovelluksen testaamiseen

valittiin sovellus avoimista lähteistä. (24.) Periaatteessa itse testausprosessi eri alustoilla ei käytännössä eroa toisistaan, erot ovat ainoastaan WebDriverIO:n konfiguraatiossa.

Testausprosessin käynnistämiseen ja sen mahdollisimman laajaan automatisointiin käytetään JavaScriptillä kirjoitettua skriptiä, joka sisältää useita toimintoja. Ensimmäinen toiminto lähettää ChatGPT:lle kehotteen (kuva 7).



Kuva 7. Kehyksen työkierto.

Kehote on tekstikysely chatille, joka on muotoiltu millä tahansa luonnollisella kielellä. Mitä tarkemmin ja kattavammin kehote on muotoiltu, sitä suuremmat ovat mahdollisuudet saada laadukas vastaus. Kehotteen laatimisella on merkittävä osuus tuloksessa, ja tämä osa ei ole automatisoitu tässä versiossa, vaan se perustuu käyttäjän tai testaajan pyyntöön. Lähettämiseen käytetään API-yhteyttä.

Chatin generoima koodi palautetaan vastauksena ja tallennetaan erilliseen tiedostoon testiskriptin muodossa. Tämän generoidun skriptin suorittamiseksi on käynnistettävä WebDriverIO:n konfiguraatiotiedosto. Myös tämä prosessi käynnistyy automaattisesti, kun generoitu testiskripti on vastaanotettu. Seuraava toiminto analysoi suoritetun testauksen tulokset. Jos tuloksista löytyy

virheitä, kehys lähettää pyynnön chattiin uudelleen. Uudessa pyynnössä ohjelma pyytää kirjoittamaan skriptin uudelleen perustuen analyysin löytämiin ongelmiin.

Teoriassa koko prosessi toimii automaattisesti vapauttaen käyttäjän tarpeesta hallita testitapausten kirjoittamiseen vaadittavia tietoja tai ohjelmointitaitoja. Ensiksi kirjoitetaan kehote, joka tulee muotoilla mahdollisimman selkeästi. Tämän perusteella määräytyy, kuinka oikein vastaus muodostuu. Alla on yksi esimerkki kehotteesta, jota käytettiin tässä projektissa (esimerkkikoodi 3).

```
const prompt = `Please provide a Mocha/Chai test script tailored for
an Android app login feature.
        The script should use async functions and the concise
$('~selector').setValue('value') syntax for element interactions.
        Two test cases are needed: one for an incorrect login attempt
that checks for an error message using selectors
        like '~emailField' and '~passwordField'; and another for a
correct login attempt that confirms the user is logged in.
        Initialize the app at the login screen in the beforeEach hook.
The script should begin with the 'describe'
        block and exclude any setup of capabilities.`;
```

Esimerkkikoodi 3. Esimerkki kehotteesta, jota käytettiin tässä projektissa.

Vaikka ChatGPT on koulutettu ymmärtämään yli 100 kieltä, on parempi kirjoittaa kehotteet englanniksi. Tämä johtuu siitä, että kielimalli on ensisijaisesti koulutettu englanninkielisestä materiaalista, jota on saatavilla runsaasti. GPT-4 ymmärtää vieraita kieliä paljon paremmin kuin GPT-3.5:n kieliversio. Internetissä on riittävästi resursseja, joissa kerrotaan yksityiskohtaisesti, miten GPT:lle tarkoitetut kyselyt tulisi kirjoittaa oikein. Siksi tähän aiheeseen ei syvennytä enempää.

Testausprosessin aloittamiseksi on tarpeen käynnistää emulaattori Android-sovelluksia varten tai simulaattori iOS-sovelluksia varten. Sovelluksia voidaan ajaa rinnakkain. Vaihtoehtoisesti sovelluksia voidaan testata käyttämällä todellisia laitteita: Android-sovelluksille Android-älypuhelimia ja iOS-sovelluksille iPhoneja. Kehyksen käynnistämiseen käytetään komentoa *node filePath/testScript.js*. Omassa tapauksessani komento on *node utils/generateTestCode.js*, joka käynnistää koko testausprosessin. Voi myös

suorittaa pelkästään testit käyttämällä komentoa *npm run wdio*, joka käynnistää WebDriverIO:n konfiguraatiotiedoston.

Suurimmaksi ongelmaksi muodostui käyttöliittymäelementtien tunnistaminen generoidusta skriptistä, mikä on välttämätöntä skriptin välittömäksi käynnistämiseksi. Appium tarjoaa useita menetelmiä käyttöliittymäelementtien viittaamiseen, mukaan lukien *accessibilityLabel*. Haittapuolena kuitenkin on, että tämä tieto on oltava kehittäjän määrittämä sovellusta luotaessa, ja testaajan on tunnettava se.

On myös vaihtoehto käyttää XPathia tai etsiä tekstiä. XPathin käytön haittapuolena on, että XPathin on oltava tiedossa, tai sen voi nähdä vain käyttämällä Appium Inspector -työkalua. Toisaalta hakeminen nimen perusteella ei välttämättä tuota tuloksia tai jotkut elementit voivat olla samannimisiä. Projektissa, Android-sovelluksen testaamiseksi, päätettiin turvautua accessa-bilityLabeliin, koska testattava sovellus oli kirjoitettu itse (esimerkkikoodi 4).

```
<TextInput
    style={styles.input}
    onChangeText={(text) => {
      setEmail(text);
      setMissingEmailError(false);
      setEmailError("");
    }}
    value={email}
    placeholder={t('email')}
    accessibilityLabel="emailField"
    placeholderTextColor="#aaa"
    keyboardType="email-address"
    autoComplete="email"
    autoCapitalize="none"
/>
```

Esimerkkikoodi 4. Esimerkki accessibilityLabelin käytöstä koodissa.

Seuraavaksi GPT tulee luomaan testiskriptin pyynnön mukaan. GPT-3.5:n luomat testit ovat kuitenkin usein täysin käyttökelvottomia, joko puuttuvien koodinosien tai odotusten vastaamattomuuden vuoksi. GPT-4 ja GPT-4-Turbo suoriutuvat huomattavasti paremmin, mutta ne ovat samalla kalliimpia. Alla on esimerkki GPT-4-mallilla generoidusta skriptistä (esimerkkikoodi 5).

```
sasha@Aleksandras-MBP TestingFW % node utils/generateTestCode.js
GPT Response: Certainly! Below is a Mocha/Chai script assuming that
you are using WebDriverIO with Mocha and Chai for an Android
application. You also mentioned to exclude any setup of capabilities,
so we'll jump straight into the `describe` block.

Please remember to replace the actual values for selectors
('~emailField', '~passwordField', etc.), error messages, and the login
confirmation step with those relevant to your Android app.

const { expect } = require('chai');
describe('Android App Login Feature', function () {
  beforeEach(async function() {
    // Assuming this command initializes the app in the login screen
    await driver.reset();
  });

  it('should display an error message for incorrect login attempt',
async function () {
    await $('~emailField').setValue('wrong@example.com');
    await $('~passwordField').setValue('wrongPassword');
    await $('~loginButton').click();

    // Add a pause or wait here if necessary, e.g., await
driver.pause(1000);

    const errorMessage = await $('~errorMessage').getText();
    expect(errorMessage).to.equal('Invalid email or password'); //
Replace with actual error message
  });

  it('should confirm the user is logged in after a correct login
attempt', async function () {
    await $('~emailField').setValue('correct@example.com'); // Replace
with correct email
    await $('~passwordField').setValue('correctPassword');  // Replace
with correct password
    await $('~loginButton').click();

    // After a successful login, an element unique to the home screen
might appear
    const uniqueElement = await
$('~uniqueElementAfterLogin').isDisplayed();

     // This confirms the user is logged in
    expect(uniqueElement).to.be.true;
    // Optionally, could verify redirection to the home or dashboard
page if there’s a URL change; or other post-login confirmation method
  });
});

Ensure you have the correct selectors for all the elements used in the
script. Also, error handling and additional test cases may be
```

```
necessary depending on the specifics of your app's login functionality
and how it responds to user input.

Remember to adjust synchronization (pauses or explicit waits) as per
the dynamics of your application to ensure that elements are
interacted with at the appropriate times.

Lastly, WebDriverIO's `setValue` will clear the existing value and set
the new value in inputs. If the app's behavior expects anything
different, you might need to handle it accordingly.
Generated test script saved to:
/Users/sasha/Documents/GitHub/TestingFW/chatGptHelper/generatedTest.js
updateTestScriptWithGetElement called
```

Esimerkkikoodi 5. ChatGPT:n vastaus ja suositukset generoidusta koodista terminaalissa.

Chatin suositusten mukaan testin läpäisemiseksi on tarpeen muuttaa käyttöliittymäelementtien nimiä joko käsin tai skriptin avulla. Jotkut niistä on jo merkitty oikein, mutta toiset vaativat korjausta. Lisäksi on välttämätöntä täyttää kentät oikeilla tiedoilla sähköpostiosoitteen ja salasanan syöttämiseksi, koska GPT ei tunnista niitä eikä niitä ole mainittu kehotteessa (esimerkkikoodi 6).

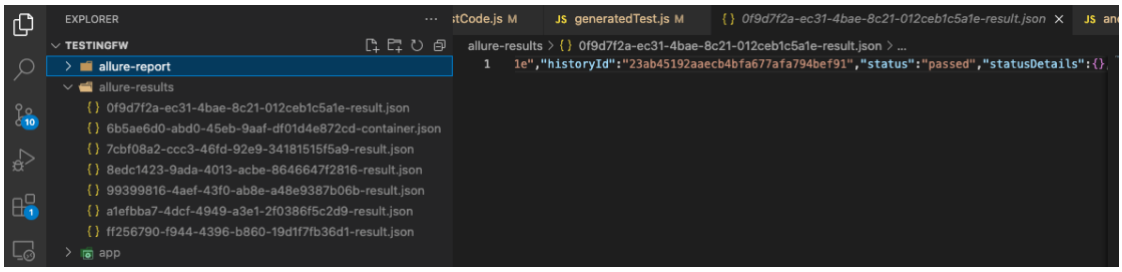
```
const { expect } = require('chai');
describe('Android application login tests', function() {

  // Precondition: Runs before each test case
  beforeEach(async function() {
    // Add your setup code here to make sure the app is on the login
screen
    const signInButton = await $('//*[@text="Sign In"]');
        await signInButton.waitForDisplayed();
        await signInButton.click();
});

it('should display an error for incorrect login credentials', async
function() {
    // Input incorrect username
    await $('~emailField').setValue('wrongUser');
    // Input incorrect password
    await $('~passwordField').setValue('wrongPassword');
    // Click on Sign In button
    await $('~loginButton').click();
    // Check for error message visibility
    const errorMessage = '//android.widget.TextView[@text="Something
went wrong. Please try again."]';
    $(errorMessage).waitForDisplayed();
    const errorMessageText = await $(errorMessage).getText();
    expect(errorMessageText).to.equal('Something went wrong. Please
try again.');
```

```
  it('should allow user to successfully log in with correct
credentials', async function() {
    // Input valid email
    await $('~emailField').setValue('sanya@gmail.com');
    // Input valid password
    await $('~passwordField').setValue('Sasha123');
    // Click on the Sign In button
    await $('~loginButton').click();
    // Check if 'Upcoming Matches' text is displayed post-login
    const upcomingMatchesElement = await
$('~upcomingMatces').getText();
    expect(upcomingMatchesElement).to.include('Upcoming Matches');
  });;

  // Additional tests and their respective assertions could go here

});
```

Esimerkkikoodi 6. Muokattu testiskripti, jossa käyttöliittymäelementtien nimet on muutettu.

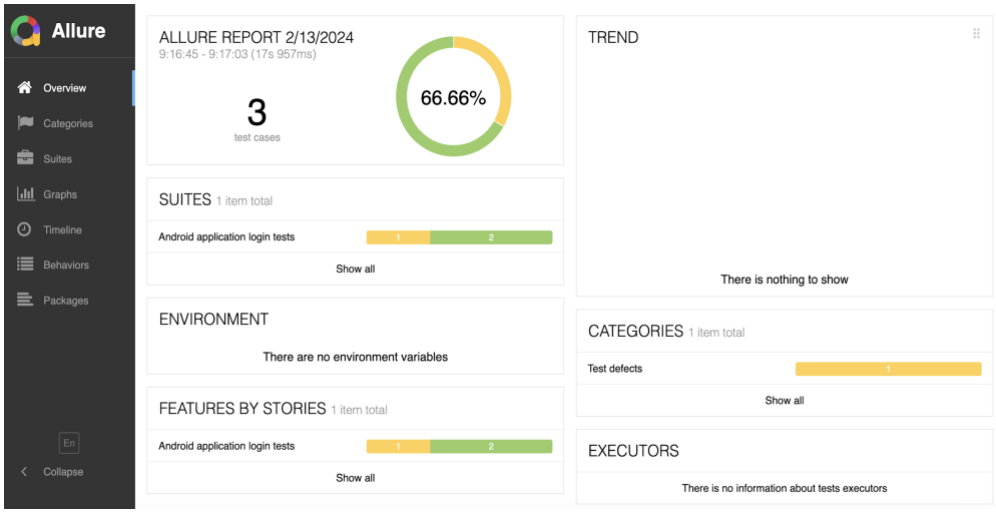
Tämän skriptin avulla testataan kirjautumismahdollisuutta sekä olemassa oleville että olemattomille tileille käyttäen oikeita ja vääriä tunnistetietoja. Kuten luvussa 4.2 on mainittu, testin generoinnin jälkeen käynnistetään WebDriverIO, joka suorittaa testiskriptin emulaattorissa tai simulaattorissa ja laatii sen jälkeen raportin tuloksista.

Tehokkaamman työskentelyn mahdollistamiseksi WebDriverIO:n testitulosten kanssa Allure Reporter -laajennus otettiin käyttöön. Se tarjoaa sekä web-käyttöliittymän että mahdollisuuden tallentaa testitulokset JSON-muodossa (kuva 8). Tämä mahdollistaa testitulosten hyödyntämisen myöhemmin. Lisäksi testituloksia voi seurata kätevästi web-käyttöliittymässä, joka visualisoi tilastoja ja kaavioita. Jotta tämän käyttöliittymän saisi käynnistettyä, on käytettävä seuraavaa komentoa *allure generate allure-results --clean && allure open allure-report*.



Kuva 8. Allure-raportin tulokset JSON-muodossa.

Testauksen tuloksista on saatavilla melko yksityiskohtainen yhteenveto (kuva 9), jossa on mukana läpäistyjen testien prosenttiosuus ja testitapausten määrä. Testitulokset voivat saada erilaisia tiloja, jotka raportissa esitetään väreittäin. Lisäksi raportti sisältää erilaisia kaavioita ajankäytöstä, käyttäytymisestä ja muista tekijöistä.



Kuva 9. Allure-raportin web-versio.

6 Automatisoinnin tulokset ja tulevaisuuden parannukset

Tämän työn tavoitteena oli Appium-kehyksen automaatiotason lisääminen ChatGPT API:n toteuttamisen kautta. Vaikka ajatus järjestelmästä, joka toimisi täysin autonomisesti ilman testaajien panosta, on houkutteleva, se ei valitettavasti ole realistinen tällä hetkellä. Vaikka kehys selviytyy datan generoinnista erinomaisesti ja soveltuu siihen erittäin hyvin, työn päätavoitteena

ei ollut datan generointi. Viimeisin GPT-malli suoriutuu koodin generoinnista kohtuullisen hyvin, mutta voi silti helposti tehdä virheitä vastauksissaan, mikä estää prosessin täydellisen automatisoinnin. Testaajan tai kehittäjän apu on edelleen tarpeen testausskenaarioiden virheiden korjaamisessa, vaikkakin ongelmat ovat usein vähäisiä. Paljon riippuu myös kehotteen muotoilusta. Joitakin yksityiskohtia on parempi ilmaista suoraan, esimerkiksi tietty tapa viitata UI-elementteihin. Joka tapauksessa, jopa tällaisessa muodossa, kehys pystyy merkittävästi nopeuttamaan testausprosessia. Se toimii, vaikkei täysin itsenäisesti. Välillä WebDriverIO ei toimi vakaasti. Joskus sama testiskripti läpäisee testauksen, kun taas toisinaan ei, mikä edellyttää korjausta. On mahdollista, että muiden kehysten käyttö voisi johtaa parempiin tuloksiin.

Idea on viety käytäntöön, mutta työn tavoitetta ei ole saavutettu täysin. Sitä ei kuitenkaan pidetä epäonnistumisena, sillä on saatu tuloksia, jotka voidaan parantaa ja kehittää edelleen. Suuri osa tuloksista riippuu edelleen kielimallista; mitä paremmin se on koulutettu testausmateriaaleissa, sitä tarkempia vastauksia se pystyy antamaan. Projekti voidaan helposti skaalata, ja siihen voidaan sisällyttää monia lisäominaisuuksia.

Kehykseen voidaan toteuttaa käyttöliittymä, jossa käyttäjä syöttää tarvittavat tiedot, esimerkiksi valitsee käyttämänsä kielimallin, syöttää kehotteen ja määrittää WebDriverIO:n asetukset. Kehyksen toiminnallisuutta voidaan kehittää edelleen sopivien toimintojen avulla, esimerkiksi kirjoittamalla funktio, joka syöttää tarvittavat tiedot tai estää tiettyjen toistuvien virheiden esiintymisen. Useita yleisiä kehotteita voidaan valmistella ja tallentaa uudelleenkäyttöä varten. Kehotteiden valinta voidaan myös lisätä käyttöliittymään pudotusvalikon muodossa.

GPT:hen voidaan lisätä muitakin toimintoja testiskriptien luomisen lisäksi, esimerkiksi datan generointi. Muita tekoälytyökaluja voidaan kytkeä työskentelemään visuaalisen sisällön kanssa. Esimerkiksi sovelluksen näyttökuvia voidaan ottaa ja kirjoittaa niiden perusteella skriptejä GPT:lla. Tilanteessa, jossa kaikki toimii ihanteellisesti ja saumattomasti, aikaa voidaan

säästää merkittävästi sovellusten ja käyttöliittymien testauksessa ja virheenkorjauksessa.

Tämä on vain osa mahdollisista parannuksista, ja paljon riippuu myös GPT:n, OpenAI:n ja koko tekoälyteollisuuden jatkokehityksestä.

7 Yhteenveto

Tämän työn päätavoitteena oli tutkia mobiilisovellusten testauksen automatisoinnin mahdollisuuksia ChatGPT:n integroinnin avulla ja tämän kehyksen käytännön toteutusta. Kehitetyn ratkaisun testauksen perusteella saatiin vastaus työn pääkysymykseen: onko tällainen toteutus mahdollinen ja käytännössä sovellettavissa.

Opinnäytetyössä käsiteltiin yksityiskohtaisesti mobiilisovellusten testauskehityksen teknistä toteutusta Appiumin ja WebDriverIon avulla yhdistäen ChatGPT:n. Ottaen huomioon GPT:n lukuisat käyttömahdollisuudet, tässä työssä keskityttiin erityisesti testien automaattiseen generointiin, testaustulosten arviointiin ja virheiden korjaukseen. Tässä toteutuksessa jotkut toiminnallisuuden konfiguraatiot ovat käytettävissä. Yksinkertaisin näistä on tarvittavien tietojen generointi, kuten uuden käyttäjän luominen tai profiilitietojen täyttäminen, mikä nopeuttaa testausprosessia. Seuraavana askeleena on testiskriptien generointi. Ohjelma luo testiskriptin perustuen annettuun ChatGPT-pyyntöön ja tallentaa skriptin erilliseen tiedostoon, jota voi käyttää testauksen aikana. Tässä vaiheessa virheitä voi ilmetä, ja skripti saattaa toisinaan vaatia uudelleenkirjoittamista, mikä on kuitenkin edelleen nopeampaa kuin manuaalinen kirjoittaminen. Tehokkaamman ja nopeamman virheenkorjauksen takaamiseksi kehykseen on integroitu virheenkorjauslaajennus, joka mahdollistaa virheiden analysoinnin ChatGPT:n avulla. GPT analysoi raportin ja pyrkii korjaamaan testien puutteita. Lisäksi kehyksen testauksen aikana kävi selvästi ilmi, että GPT-4:n kielimallit suoriutuvat tästä tehtävästä huomattavasti paremmin kuin niiden edeltäjä, GPT-3.5:n versio.

Työssä on kuvattu yksityiskohtaisesti kehyksen luomisen lähestymistapa ja prosessi. Noudattamalla annettua ohjeistusta voi varmistua, että prosessi ei vie paljoa aikaa, ja integroitu GPT nopeuttaa testausprosessia huomattavasti, koska sen avulla ei tarvitse kirjoittaa testejä käsin eikä käsitellä virheitä. Työssä on myös esitetty esimerkkejä iOS- ja Android-sovellusten testaamisesta ja niiden tuloksista.

Projektin tuloksena kehitettiin kehys, joka ei automatisoi mobiilisovellusten testausprosessia Appiumin pohjalta täysin, mutta lyhentää työaikaa ja nopeuttaa prosessia merkittävästi. Asiantuntijan valvonta on edelleen tarpeellista, mutta ChatGPT yksinkertaistaa testausta käyttämällä tätä kehystä. Tarkasti muotoiltu GPT-kysely ja oikein valittu kielimalli ovat tärkeitä. Yhtenä kehyksen etuna on, että uuden toiminnallisuuden lisääminen ei edellytä koko sovelluksen uudelleenkirjoittamista, vaan pelkkä uusien toimintojen määrittely riittää. Lisäksi kehys soveltuu web-sovellusten testaamiseen, ja GPT:n kielimallin vaihtaminen on suunniteltu yksinkertaiseksi.

Kehyksen asennus on tarpeen vain kerran, ja sen jälkeen jatkokäytössä voidaan muuttaa ainoastaan kehotetta ja testattavaa sovellusta, mikä tekee sen käytöstä vaivatonta. Lisäksi tarvittavat testiskriptit voidaan generoida etukäteen ja käyttää niitä tarpeen mukaan.

Opinnäytetyön aikana saadut tulokset ovat ensisijaisesti arvokkaita kokemuksen näkökulmasta. Suurin osa internetissä jo ehdotetuista aiheeseen liittyvistä ratkaisuista keskittyy chatin käyttöön avustajana, erillään itse testauskehityksestä. Tällaista ratkaisua ei voida pitää täysin automatisoituna. Lisäksi opinnäytetyön johtopäätökset voivat olla hyödyllisiä ja sovellettavissa laajemmin missä tahansa tilanteessa, jossa kielimallien integrointia ohjelmistoon harkitaan prosessien nopeuttamiseksi, parantamiseksi tai automatisoimiseksi.

Opinnäytetyössä luottu kehyksellä on suuri potentiaali jatkokehitykseen mukaan lukien uusien toimintojen ja parannusten lisääminen. Käytännön kokemus voi

myös tukea GPT:n integroimista muihin testausohjelmiin. Lisääntyvien kielimallien myötä kehys todennäköisesti muuttuu älykkäämmäksi ja sen mahdollisuudet laajenevat.

Lähteet

1 Chernyak, Alex Zap. 2023. Mobiilisovellusten testaus – Mitä se on, tyypit, prosessit, lähestymistavat, työkalut ja paljon muuta! Verkkoaineisto. Zaptest Unlimited Software Automation. <https://www.zaptest.com/fi/mobiilisovellusten-testaus-mita-se-on-tyypit-prosessit-lahestymistavat-tyokalut-ja-paljon-muuta>. Päivitetty 9.5.2023. Luettu 23.11.2023.

2 Knott, Daniel. 2015. Hands-On Mobile App Testing. Addison-Wesley Professional. Addison-Wesley Professional.

3 Sambamurthy, Manikandan. 2023. Test Automation Engineering Handbook. Packt Publishing.

4 Mitä on ohjelmistotestaus ja mitä hyötyä siitä on? 2022. Verkkoaineisto. Vala. <https://www.valagroup.com/fi/blogi/mita-on-ohjelmistotestaus-ja-mita-hyotya-siita-on/>. Päivitetty 10.11.2022. Luettu 25.11.2023.

5 What is GPT? Verkkoaineisto. AWS. <https://aws.amazon.com/what-is/gpt/>. Luettu 5.1.2024.

6 How do Transformers work? Verkkoaineisto. Hugging Face. <https://huggingface.co/learn/nlp-course/chapter1/4>. Luettu 16.2.2024.

7 Ortiz, Sabrina. 2023. What is ChatGPT and why does it matter? Here's what you need to know. Verkkoaineisto. ZDNET. <https://www.zdnet.com/article/what-is-chatgpt-and-why-does-it-matter-heres-everything-you-need-to-know/>. Päivitetty 15.9.2023. Luettu 1.12.2023.

8 Shree, Priya. 2020. The Journey of Open AI GPT models. Verkkoaineisto. Walmart Global Tech Blog. <https://medium.com/walmartglobaltech/the-journey-of-open-ai-gpt-models-32d95b7b7fb2>. Päivitetty 10.11.2020. Luettu 5.12.2023.

9 OpenAI. 2023. New models and developer products announced at DevDay. Verkkoaineisto. OpenAi Blog. <https://openai.com/blog/new-models-and-developer-products-announced-at-devday>. Päivitetty 6.11.2023. Luettu 20.12.2023.

10 GPT-4 System Card. 2023. Verkkoaineisto. OpenAi. <https://cdn.openai.com/papers/gpt-4-system-card.pdf>. Päivitetty 23.3.2023. Luettu 20.12.2023.

11 Generative Language Models and the GPT Algorithm. Verkkoaineisto. Turun yliopiston kirjasto. <https://utguides.fi/c.php?g=712454&p=5147021#s-lg-box-16135176>. Luettu 1.12.2023.

12 Models. 2023. Verkkoaineisto. OpenAi. <https://platform.openai.com/docs/models/continuous-model-upgrades>. Päivitetty 20.12.2023. Luettu 23.11.2023.

13 Guinness, Harry. 2023. How does ChatGPT work? Verkkoaineisto. Zapier. <https://zapier.com/blog/how-does-chatgpt-work/#>. Päivitetty 6.9.2023. Luettu 1.12.2023.

14 John. 2023. Verkkoaineisto. AI For Folks. <https://aiforfolks.com/does-chatgpt-learn-from-users/?utm_content=cmp-true>. Päivitetty 22.5.2023. Luettu 1.12.2023.

15 ChatGPT Advantages. 2023. Verkkoaineisto. Twelverays blog. <https://twelverays.agency/blog/chatgpt-advantages>. Päivitetty 17.10.2023. Luettu 10.11.2023.

16 Andrades, Geosley. 2023. ChatGPT and Its Role in Test Automation. Verkkoaineisto. Acceloq blog. <https://www.accelq.com/blog/chatgpt-and-its-role-in-test-automation/>. Päivitetty 27.1.2023. Luettu 15.11.2023.

17 Appium Documentation. Verkkoaineisto. Appium. <https://appium.io/docs/en/2.1/>. Luettu 26.12.2023.

18 Chowdhury, Arnab Roy. 2023. Best Automation Mobile Testing Tools and Frameworks. Verkkoaineisto. BrowserStack. <https://www.browserstack.com/guide/mobile-application-testing-frameworks>. Päivitetty 24.9.2023. Luettu 26.12.2023.

19 Sourojit, Das. 2023. What is Appium Inspector? (Benefits & How to Use it?). Verkkoaineisto. BrowserStack. <https://www.browserstack.com/guide/appium-inspector>. Päivitetty 18.8.2023. Luettu 5.1.2024.

20 Testrunner. Verkkoaineisto. WebdriverIO Documenation. <https://webdriver.io/docs/testrunner/>. Luettu 5.1.2024.

21 Allure Reporter. Verkkoaineisto. WebdriverIO Documenation. <https://webdriver.io/docs/allure-reporter/>. Luettu 8.1.2024.

22 Mocha simple, flexible, fun. Verkkoaineisto. Mocha. <https://mochajs.org/>. Luettu 26.12.2023.

23 Developer quickstart Get up and running with the OpenAI API. Verkkoaineisto. OpenAi. <https://platform.openai.com/docs/quickstart?lang=ChatCompletions&context=node>. Luettu 5.1.2024.

24 GitHub. 2021. WebdriverIO Native-demo-app. <https://github.com/webdriverio/native-demo-app/releases>. Päivitetty 17.05.2021. Luettu 6.11.2023.