LAB University of Applied Sciences

Selainlaajennuksen suunnittelu ja kehittämi-nen

ChatGPT-keskustelubotti

LAB-ammattikorkeakoulu
Tradenomi (AMK), Tietojenkäsittely
2023
Topi Laakso, David Legart

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Topi Laakso, David Legart	Opinnäytetyö, AMK	2023
	Sivumäärä 37	

Työn nimi

Selainlaajennuksen suunnittelu ja kehittäminen

ChatGPT-keskustelubotti

Tutkinto ja koulutusala

Tradenomi (AMK), Tietojenkäsittely

Toimeksiantajaorganisaatio (jos opinnäytetyöllä on toimeksiantaja)

-

Tiivistelmä

Projektin tavoitteena oli suunnitella ja kehittää selainlaajennus, joka toimii keskustelubottina hyödyntäen OpenAI:n ChatGPT-rajapintaa, sekä dokumentoida työn tulokset opinnäyteraporttiin. Laajennuksen tarkoitus on madaltaa kynnystä ChatGPT-kielimallin käyttöönottoon helposti ilman kirjautumisvaatimusta. Samalla se tarjoaa hauskan käyttökokemuksen ja parantaa tehokkuutta sekä tuottavuutta.

Raportissa käydään lyhyesti läpi verkkoselaimien, selainlaajennusten sekä kielimallien ominaisuuksia, mutta pääpaino on itse kehitystyössä. Laajennus kehitettiin kolmelle Suomen tilastollisesti käytetyimmälle työpöytäselaimelle HTML-, CSS- ja JavaScript-selainteknologioita hyödyntäen. Työn tuloksena syntyi kevyt ja yksinkertainen selainlaajennus, joka sisältää viihdyttäviä lisäominaisuuksia.

Asiasanat

selain, laajennus, kielimalli, chatbot, tekoäly, JavaScript, ChatGPT, ohjelmointi

Abstract

Author(s)	Type of Publication	Published
Topi Laakso, David Legart	Thesis, UAS	2023
	Number of Pages 37	

Title of Publication

Designing and developing a browser extension

Chatbot powered by ChatGPT

Degree, Field of Study

Bachelor's Degree Programme in Business Information Technology

Organisation of the client (if the thesis work is commissioned by another party)

-

Abstract

The objective of the project was to design and develop a browser extension that operates as a chatbot utilizing OpenAI's ChatGPT API, while documenting the outcomes in this thesis report. The extension aims to lower the barrier for easy adoption of the ChatGPT language model without the need for a log-in function. Simultaneously, it provides an enjoyable user experience and enhances efficiency and productivity.

The report briefly covers the features of web browsers, browser extensions, and language models, with the primary focus on the actual development work. The extension was created using HTML, CSS, and JavaScript browser technologies, targeting the three most used desktop browsers in Finland. As a result, a lightweight and straightforward browser extension was produced, including some entertaining additional features.

Keywords

browser, extension, language model, chatbot, AI, JavaScript, ChatGPT, programming

Sisällys

1 Johdanto ... 1
1.1 Tutkimuskysymys, tausta ja tavoitteet ... 1
1.2 Työn rakenne ... 2
2 Selaimet ja laajennukset ... 3
2.1 Verkkoselainten historia lyhyesti ... 3
2.2 Verkkoselaimet nykypäivänä ... 5
2.3 Selainlaajennukset ... 6
3 Laajat kielimallit ja ChatGPT ... 8
3.1 Kielimallit ... 8
3.2 OpenAI ja ChatGPT ... 9
4 Laajennuksen toteutuksessa käytetyt teknologiat ... 11
4.1 Verkkosivut ... 11
4.2 Ohjelmointi ... 12
5 Selainlaajennuksen toteuttaminen ... 14
5.1 Prototyyppi ja suunnittelu ... 14
5.2 Yhteys rajapintaan ... 15
5.3 Laajennuksen rakenne ja toiminnallisuus ... 18
5.3.1 Keskustelu-välilehti ... 18
5.3.2 Asetukset-välilehti ... 23
5.4 Laajennuksen käytettävyys ja viimeistely ... 27
5.5 Testaus ... 28
6 Yhteenveto ja jatkokehitys ... 31
6.1 Yhteenveto ... 31
6.2 Jatkokehitysideat ... 31
Lähteet ... 34

Liite 1. Linkki projektin Github-sivulle

Lyhenteet

API	Application Programming Interface. Ohjelmointirajapinta, jonka avulla eri ohjelmat voivat keskustella keskenään.
CORS	Cross-Origin Resource Sharing. Turvamekanismi, joka sallii tai estää web-sovellusten HTTP-pyyntöjen lähettämisen eri alkuperistä tuleviin resursseihin, kuten verkkosivustoihin tai palvelimiin.
CSS	Cascading Style Sheets. Tyylittelykieli, jota käytetään määrittämään ja muotoilemaan verkkosivujen ulkoasua, kuten asettelua ja värejä.
HTML	HyperText Markup Language. Standardoitu merkintäkieli, jota käytetään verkkosivujen rakenteen kuvaamiseen.
HTTP	Hypertext Transfer Protocol. Protokolla, jota käytetään tietojen siirtämiseen verkossa, erityisesti verkkoselaimen ja palvelimen välillä.
IIFE	Immediately Invoked Function Expression. JavaScriptissä käytetty koodirakenne, jossa funktio määritetään ja suoritetaan saman tien.
JS	JavaScript. Pääasiassa verkkoselaimissa suoritettava ohjelmointikieli, joka lisää verkkosivuille dynaamista sisältöä ja interaktiivisuutta.
JSON	JavaScript Object Notation. Kevyt datan esitysmuoto, jota käytetään tiedon tallentamiseen ja siirtämiseen erityisesti verkkosovellusten ja palvelimien välillä.
LLM	Large Language Model. Laaja kielimalli, joka on koulutettu käsittelemään ja kirjoittamaan ihmiskieltä tehokkaasti ja selkeästi.
NPM	Node Package Manager. Paketinhallintatyökalu, jonka avulla voidaan ladata erilaisia paketteja npm-rekisteristä ohjelmointiprojektiin.
REST	Representional state transfer. Ohjelmointirajapintojen kehittämiseen luotu arkkitehtuurimalli.
WWW	World Wide Web. Maailmanlaajuinen verkko, joka koostuu linkitetyistä verkkosivuista ja on saatavilla internet-selaimen kautta.

1 Johdanto

1.1 Tutkimuskysymys, tausta ja tavoitteet

Opinnäytetyön tavoitteena oli suunnitella ja toteuttaa omaan käyttöön selainlaajennus, joka toimii keskustelubottina hyödyntäen OpenAI:n ChatGPT-kielimallin rajapintaa. Tutkimuskysymyksenä on ”Miten voidaan toteuttaa tekoälyä hyödyntävä selainlaajennus?”. Kyseessä on toiminnallinen opinnäytetyö, jonka avulla pääsimme kehittämään omaa projektiosaamistamme iteratiivisen kehitysprojektin muodossa. Kehittämämme selainlaajennuksen tarkoituksena on pienentää ChatGPT:n käyttöönottamisen kynnystä ohittamalla kirjautumisvaatimus sekä lisäämällä keskustelubotti suoraan selaimen työkalupalkkiin helposti käyttäjän saataville.

ChatGPT:n räjähdysmäinen suosion kasvu on nostanut sen monissa yhteyksissä puheenaiheeksi ympäri maailmaa kuluneen vuoden aikana. Sen voidaan katsoa tänä päivänä olevan jo osa kulttuuria useissa yrityksissä ja oppilaitoksissa, mutta myös yksittäisten kuluttajien piirissä, koska sen avulla voidaan parantaa tuottavuutta ja ratkaista yleisiä ongelmia. Monilla yrityksillä on jo vuosien ajan ollut käytössään eri tarkoituksiin soveltuvia chatbotteja, jotka käyttäjän syötteen perusteella antavat yksinkertaisia vastauksia asiakasrajapinnassa esitettyihin yleisiin kysymyksiin, kuitenkin hyvin rajoitetuilla toiminnoilla. Kehittämämme laajennus sen sijaan hyödyntää OpenAI:n valtavaa tietokantaa ja näin pystyy tuottamaan vastauksia myös monimutkaisempiin syötteisiin.

Idea opinnäytetyölle syntyi molempien tekijöiden mielenkiinnosta ohjelmointia ja selainteknologioita kohtaan, minkä lisäksi tekoälyn ja kielimallien mahdollisuudet tuntuivat erittäin ajankohtaisilta. Päätimme yhdistää nämä aihealueet ja keskittyä projektissamme siihen, miten nykypäivän tekoälyn potentiaali voidaan tuoda lähemmäs tavallista käyttäjää selainlaajennuksen avulla. Halusimme luoda työkalun, joka olisi helppokäyttöinen ja tarjoaisi hauskan käyttökokemuksen siihen lisättyjen ominaisuuksien avulla. Työn tuloksena luotu laajennus on tällä hetkellä tarkoitettu vain tekijöiden omaan käyttöön, mutta sillä on kehityspotentiaalia, koska laajennuksen rakenne on yksinkertainen ja helposti muokattavissa erilaisiin käyttötarkoituksiin. Toteutuksen lähdekoodi ilman API-avainta löytyy Githubista, jonka linkki on liitteessä 1.

1.2 Työn rakenne

Opinnäyteraportin rakenne koostuu karkeasti kolmesta pääaihealueesta. Pääluvussa 2 tutustutaan ensin internetselainten historiaan ja niiden nykytilanteeseen, ja tämä luo pohjan sille, miten päädyimme valitsemaan laajennuksen kanssa yhteensopivat selaimet. Seuraavana on lyhyt katsaus selainlaajennuksiin ja niiden toimintaan yleisellä tasolla. Luvussa 3 puolestaan kerrotaan laajoista kielimalleista, pääpainona tässä työssä tärkeässä roolissa oleva ChatGPT. Nämä luvut kattavat raportin teoriaosuuden.

Tietoperustaan perehtymisen jälkeen pääluvussa 4 siirrytään laajennuksen toteutuksessa käytettyihin teknologioihin. Ne on jaettu verkosta löytyviin palveluihin sekä ohjelmointityökaluihin, joihin sisältyy esimerkiksi tekstieditori, ohjelmointikielet ja niiden kirjastot sekä versionhallintaohjelmisto. Luvun tarkoituksena on toimia tukena pääluvulle 5, joka on opinnäytetyön kannalta oleellisin. Luku 5 on toiminnallinen osuus, joka käsittelee selainlaajennuksen kehitystä vaihe vaiheelta aina prototyypin suunnittelusta sen viimeistelyyn ja testaukseen.

Viimeisenä päälukuna on yhteenveto ja jatkokehitys, jossa ensin pohditaan työn tuloksia sekä kehityksen aikana ilmenneitä haasteita ja ongelmia. Lopuksi käydään läpi tuotteen jatkokehitystä ajatellen muutamia mieleen tulleita kehitysideoita. Niitä ovat erilaiset lisäominaisuudet ja -toiminnallisuudet, joita laajennukseen olisi mahdollista toteuttaa, sekä myös käytettävyyttä ja toiminnallisuutta parantavat kehitysideat.

2 Selaimet ja laajennukset

2.1 Verkkoselainten historia lyhyesti

Alkuvuodesta 1993 Illinoisin yliopiston National Center for Supercomputing Applications eli NCSA-yksikkö julkaisi yhden ensimmäisistä graafisista verkkoselaimista nimeltään Mosaic, jota ajettiin tutkijayhteisöjen suosimassa X Window System -ikkunointijärjestelmässä. Mosaicin teki ainutlaatuiseksi se, että se osasi näyttää kuvat samassa ikkunassa tekstin kanssa. Samana vuonna 30. huhtikuuta CERN julkaisi Tim Berners-Leen kehittämän WWW:n lähdekoodin lisenssivapaasti yleiseen käyttöön. Lähdekoodin julkaisu edisti webin kasvua merkittävästi, ja loppuvuodesta -93 oli olemassa jo yli 500 tunnettua verkkopalvelinta. Vuoden 1994 lopussa verkkopalvelimia oli jo 10 000, joista 2000 oli kaupallisia ja käyttäjiä oli siihen mennessä 10 miljoonaa. (CERN).

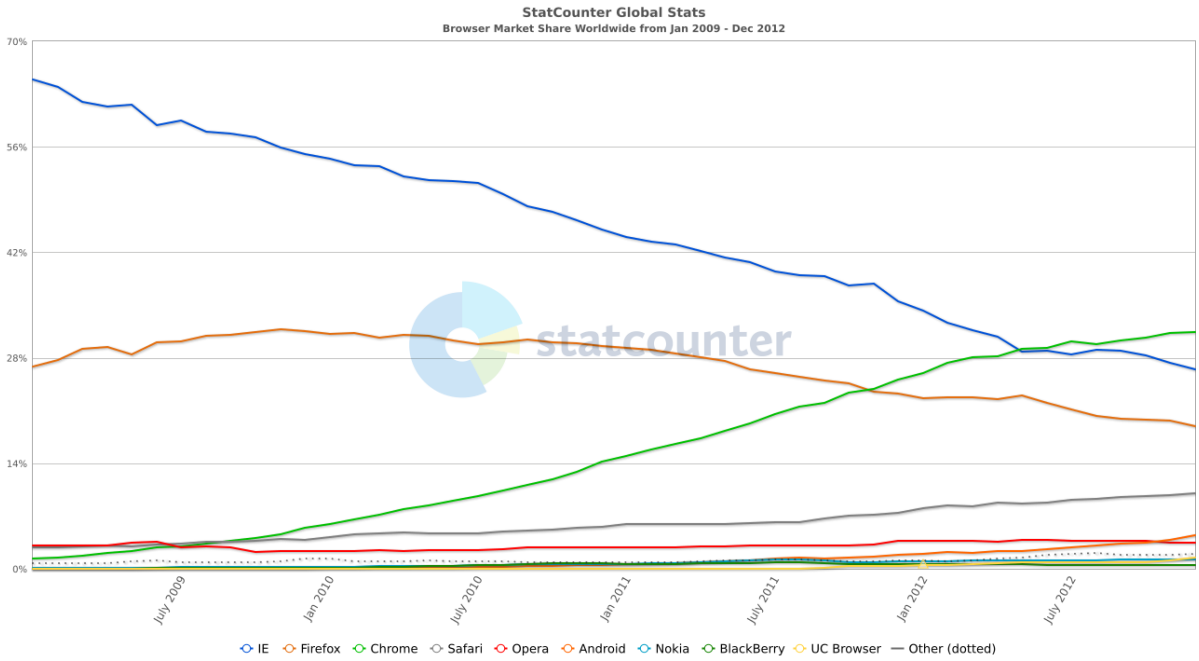
Keväällä 1994 NCSA:n jäsenten perustama yritys Mosaic Communications Corporation vaihtoi saman vuoden lopulla pääosin lakiteknisistä syistä nimekseen Netscape Communications ja toi markkinoille edellä mainittuun Mosaic-selaimeen pohjautuvan Netscape Navigator -verkkoselaimen. Seuraavana vuonna Microsoft julkaisi ensimmäisen version omasta Internet Explorer -selaimestaan, mikä johti näiden kahden yhtiön väliseen kilpailuun tuoreilla selainmarkkinoilla, ja se loi pohjan lopulta ensimmäiseen selainsotaan. Kyseisenä ajankohtana oli useimpien verkkosivujen kohdalla merkitystä käytetyllä selaimella, sillä sivujen käyttämät tekniikat eivät vielä olleet standardoituja. Tämän takia jotkin sivut toimivat paremmin riippuen käytetystä selaimesta. Selaimet olivat myös maksullisia, joten useimmissa tapauksissa käyttäjä joutui tekemään päätöksen verkkoselaimen ominaisuuksien perusteella. Vuonna 1996 Microsoft päätti sisällyttää oman Internet Explorer -selaimensa Windows 95 -käyttöjärjestelmään ilmaiseksi, mikä alkoi pian näkyä markkinatilanteessa. (Hoffmann 2017).

Vielä lokakuussa 1997 Netscapella oli hallussaan noin 70 prosenttia selainmarkkinoista, mutta jo vuoden päästä Internet Explorerin osuus oli 50 prosenttia, ja loppuvuodesta 1999 Microsoft hallitsi miltei 80 prosenttia selainmarkkinoista Internet Explorerin 5.0-versiollaan. Tämän myötä Netscapen valta-asema oli kukistettu ja vuoteen 2003 mennessä yhtiö lopetti virallisesti toimintansa siirtäen kuitenkin tuotemerkkinsä toiselle yritykselle, jossa kehitystyö jatkui hiljaiselossa useamman vuoden. (Hoffmann).

Internet Explorer hallitsi selainmarkkinoita ylivoimaisesti aina vuoteen 2004, jolloin Netscapen raunioille perustettu Mozilla Foundation toi markkinoille kehittämänsä Mozilla Firefox -

selaimen. Tämän myötä myös muut yhtiöt alkoivat huomata, että selainmarkkinoilla on kysyntää. Seuraavina vuosina esimerkiksi Apple-yhtiön Safari-selain saavutti suosiota valmistajan omissa laitteissa, mutta myös pienemmät kilpailijat kuten Opera saivat jalansijaa omissa käyttäjäkunnissaan. Vuonna 2008 Google julkaisi oman Chrome-selaimensa, minkä myötä Internet Explorerin suosio alkoi hiipua entistä nopeammin, eivätkä sen uudet versiopäivityksetkään enää riittäneet suosion kasvattamiseen. (Raghavan 2019).

Kuten kuviossa 1 (Statcounter 2023a) on nähtävissä, vajaassa neljässä vuodessa Internet Explorerin suosio markkinoilla romahti noin 65 prosentin osuudesta vuonna 2009 noin 27 prosenttiin vuoden 2012 loppuun mennessä. Google Chromen osuus puolestaan nousi samassa ajassa noin 1,5 prosentista reiluun 31 prosenttiin, ohittaen Internet Explorerin suosion toukokuun 2012 tienoilla. Kaavio kuvaa selainten käyttäjäosuuksia kaikilla laitteilla maailmanlaajuisesti, joten esimerkiksi mobiililaitteiden suosion kasvaminen on osaltaan voinut olla suuressa roolissa tapahtumien kulussa. Statcounter-sivustolta löytyvistä tilastoista on nähtävillä, että Chrome on alla näkyvän kaavion aikajanan jälkeen kasvattanut suosiotaan entisestään. Vuonna 2015 alkuperäisen Internet Explorerin korvasi Microsoftin uusi Edge-selain, ja vuonna 2018 se uudistettiin käyttämään eri selainmoottoria. Tämän myötä Internet Explorer ajettiin pikkuhiljaa alas ja sen tuki Windows 10 -käyttöjärjestelmässä päättyi kesällä 2022 (Baker & Courtright 2022).

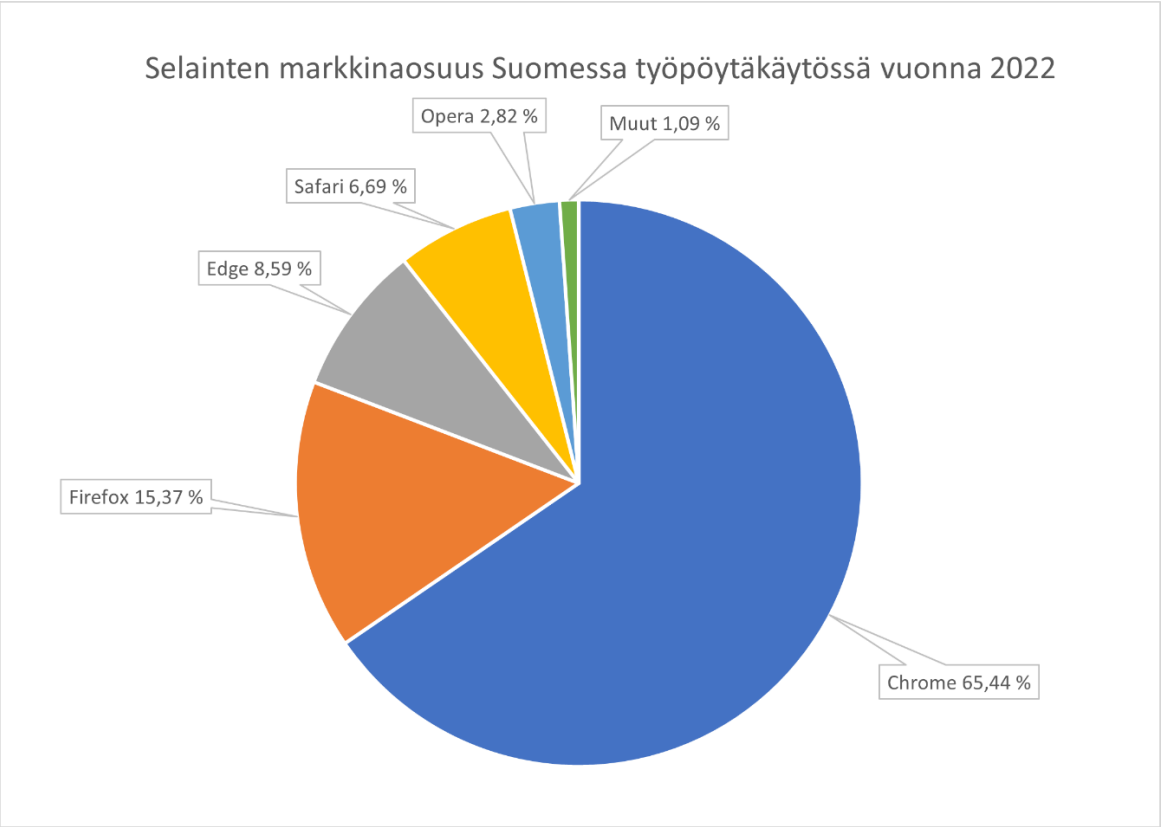


Kuvio 1. Verkkoselaimien markkinaosuudet vuosina 2009–2012 (Statcounter 2023a)

2.2 Verkkoselaimet nykypäivänä

Nykyään verkkoselaimet noudattavat pitkälti samoja standardeja ja kaikki ovat maksuttomia, joten käyttäjien näkökulmasta tärkeintä on tietoturva ja omat mieltymykset sekä monissa tapauksissa käytetty alusta, esimerkiksi PC, tabletti tai älypuhelin. Uusissa laitteissa on tänä päivänä aina verkkoselain sisällytettynä käyttöjärjestelmään, kuten Windowsin mukana tuleva Edge tai Applen iPhone-puhelimissa ja Mac-tietokoneissa Safari. Android-älypuhelimissa taas usein on valmiiksi asennettuna Google Chrome. (National Cyber Security Centre 2021).

Tämän opinnäytetyön aihe, eli selainlaajennuksen kehittäminen, huomioiden oli oleellista valita laajennuksen kanssa yhteensopivat selaimet. Laajennus on tarkoitettu nimenomaan PC-alustan verkkoselaimelle, käyttömaana Suomi, joten statistiikkaa tarkastelemalla käytetyimmät selaimet olivat vuoden 2022 otannan perusteella Google Chrome, Mozilla Firefox sekä Microsoft Edge (Statcounter 2023b). Kuviossa 2 on nähtävillä Suomen suosituimmat työpöytäkäytössä olevat selaimet, ja kolme edellä mainittua ovatkin käytössä jopa lähes 90 prosentilla käyttäjistä, minkä vuoksi juuri nämä valikoituivat työmme selainalustoiksi.



Kuvio 2. Selainten markkinaosuus Suomessa työpöytäkäytössä 2022 (Statcounter 2023b)

2.3 Selainlaajennukset

Selainlaajennukset ovat ohjelmia, jotka lisäävät verkkoselaimeen ominaisuuksia ja toimintoja käyttäen HTML-, JavaScript- ja CSS-teknologioita (MDN Web Docs 2023c). Kyseisiä ominaisuuksia voivat olla esimerkiksi mainosten estäminen, verkkosivujen ulkonäön tai toimintojen parantaminen, tai selaimeen lisätty työkalu, kuten kääntäjä tai kieliopintarkistaja. Ensimmäisiä tunnettuja selainlaajennuksia olivat Internet Exploreriin vuonna 1999 julkaistut Explorer Bar:it eli erilaiset työkalupalkit (Fedewa 2021).

Jokainen selainlaajennus sisältää toimintansa kannalta pakollisen manifest.json-tiedoston, joka niin sanotusti kertoo selaimelle, mitä laajennus tekee. Tiedosto sisältää ohjelman perustietoja, kuten versionumeron, laajennuksen nimen, sekä mahdolliset oikeudet tiettyjä toimintoja varten (MDN Web Docs 2023b). Kuvassa 3 on Chromen dokumentaatiosta otettu esimerkki manifest.json-tiedoston sisällöstä Reading Time -nimiseen laajennukseen.

```
{
  "manifest_version": 3,
  "name": "Reading Time",
  "description": "Add the reading time to Chrome Extension documentation articles",
  "version": "1.0",
  "icons": {
    "16": "images/icon-16.png",
    "32": "images/icon-32.png",
    "48": "images/icon-48.png",
    "128": "images/icon-128.png"
  },
  "content_scripts": [
    {
      "js": [
        "scripts/content.js"
      ],
      "matches": [
        "https://developer.chrome.com/docs/extensions/*",
        "https://developer.chrome.com/docs/webstore/*"
      ]
    }
  ]
}
```

Kuva 3. Reading Time -laajennuksen manifest.json-tiedosto (Chrome for Developers 2023)

Manifestissa voidaan myös tarvittaessa viitata muihin tiedostoihin, jotka ovat tärkeitä laajennuksen toiminnan kannalta. Kyseiset tiedostotyypit ja niiden selitteet löytyvät alla olevasta taulukosta 1. (MDN Web Docs 2023b).

Tiedostotyyppi	**Selite**
Taustaskripti (Background script)	Skripti, joka ”kuuntelee” selaimen tapahtumia.
Kuvake (Icon)	Laajennuksen kuvakkeet.
HTML-komponentti	Käyttöliittymäkomponentit, kuten erillinen asetusikkuna tai sivupalkki, joita laajennus voi käyttää käyttöliittymässään.
Sisältöskripti (Content script)	JavaScript-koodi, jota voidaan syöttää verkkosivuille toiminnallisuuden saavuttamiseksi.
Verkossa saavutettavat resurssit (Web accessible resources)	Pakatut kuvat tai HTML-, CSS- ja JavaScript-tiedostot, joita laajennuksen skriptit voivat hyödyntää.

Taulukko 1. Tiedostotyypit, joihin voidaan viitata manifest.json-tiedostossa (MDN Web Docs 2023b)

Kuten kaikessa tietotekniikkaan liittyvässä, myös selainlaajennuksissa on riskinsä. Laajennuksia löytyy nykypäivänä tuhansia, joten joukkoon mahtuu paljon haittaohjelmiksi tarkoitettuja, hyödyllisiksi naamioituja laajennuksia. Myös hyväntahtoisten kehittäjien hyödyllisiksi tarkoitetut laajennukset saattavat sisältää tietoturva-aukkoja, joita esimerkiksi hakkerit voivat hyödyntää tietojen urkkimiseen. Joissakin harvinaisissa tapauksissa Chromen Web Store on myös saattanut poistaa sivultaan vahingossa oikean laajennuksen ja korvannut sen samannimisellä haittaohjelmalla, tai laajennuksen kehittäjä on myynyt oikeudet ulkopuoliselle taholle, joka on muuttanut koodia sisältämään haittaohjelmia. Riskien välttämiseksi on siis kannattavaa tarkistaa aina laajennuksia asentaessaan niiden kehittäjien luotettavuus sekä laajennuksen saamat käyttäjäarvostelut, ja erityisesti tietotekniikasta vähemmän ymmärtävien tulisi aina asentaa selainlaajennukset kyseisen selaimen virallisesta ”laajennuskaupasta”. (Brave Software Inc. 2023).

3 Laajat kielimallit ja ChatGPT

3.1 Kielimallit

Laajat kielimallit, englanniksi Large Language Model, eli LLM:t ovat tekoälypohjaisia kielimalleja, jotka kykenevät imitoimaan ihmisen älykkyyttä keskustelua simuloiden (Priest 2023). Ensimmäiset modernit LLM:t ilmestyivät vuonna 2017 ja käyttävät transformer-malleja, eli neuroverkkoja, joita yleisesti kutsutaan transformereiksi tai muuntimiksi (Kerner 2023). Kielimallin kouluttamiseen käytetään valtavaa tekstitietojoukkoa hyödyntäen syväoppimista neuroverkkojen kautta (Dupont ym. 2023), ja dataa voidaan kerätä hyödyntämällä esimerkiksi kirjoja, artikkeleita tai verkkosivuja, joiden avulla se oppii kaavoja ja sanojen välisiä yhteyksiä (Priest 2023). Kielimalleja voidaan käyttää tekstin tuottamiseen, ohjelmointiin, piirtämiseen ja luovuuden sekä tuottavuuden parantamiseen (NVIDIA Corporation).

LLM:t käyttävät kielen käsittelyyn ja tuottamiseen tokeneita, jotka ovat perusyksiköitä tekstissä tai koodissa. Näitä voivat olla yksittäiset merkit, sanojen osat, kokonaiset sanat tai lauseiden osat. Tokeneille annetaan numeroita, jotka puolestaan muutetaan vektoriksi, josta tulee itse syöte LLM:n ensimmäiseen neuroverkkoon. Se, miltä tokenit näyttävät, riippuu käytetystä tokenisointijärjestelmästä, ja nyrkkisääntönä voidaan pitää sitä, että 1 000 tokenia on noin 750 sanaa englannin kielellä. (Witt 2023).

Tutkimme OpenAI:n sivuilta löytyvää token-laskuria, josta kävi ilmi erilaisten tekstien käyttämät token-määrät. Kuvassa 4 (OpenAI b) on nähtävillä selkeä ero käytetystä tokenien määrästä samaa asiaa tarkoittavilla suomen- ja englanninkielisillä lauseilla. Suomen kieli siis käyttää noin kaksinkertaisen määrän tokeneita kielimallissa, kuin englannin kieli. Tämä ei kuitenkaan käyttäjälle näy kielimalleja käyttäessä, eikä varsinaisesti vaikuta kyseisten palveluiden käyttöön muutoin, kuin rajaten syötteen ja vastauksen pituutta, mutta API-yhteyden kautta käytetystä kielimallista peritään tokenien perusteella rahallisia korvauksia, joista kerrotaan tarkemmin luvussa 4.

Kuva 4. Esimerkkitekstit OpenAI:n Tokenizer-palvelussa (OpenAI b)

Laajojen kielimallien ongelmapuoliin liittyy niiden taipumus ”hallusinoida”. Termillä tarkoitetaan ilmiötä, jossa tekoäly luo sille annettujen algoritmien perusteella vakuuttavan, mutta täysin keksityn vastauksen (Athaluri ym. 2023). Kyseessä ei siis ole ajatteleva yksilö tai joukko, vaan kielimalli yksinkertaisesti asettaa tokeneita toistensa perään todennäköisyyksiä laskien, yrittäen tuottaa uskottavan ja kielellisesti ”järkevän” vastauksen käyttäjälle (Tang ym. 2023). Tämä siten rajoittaa merkittävästi sen käyttömahdollisuuksia esimerkiksi tieteen ja tutkimuksen saralla. Lisäksi, vaikka lähtökohtaisesti LLM:t pyrkivät noudattamaan niille koulutettuja sääntöjä olla antamatta esimerkiksi ohjeita haitantekoon tai kiroiluun, ne ovat helposti manipuloitavissa tekemään juuri niin, mikäli kysymykset muotoillaan tietyllä tavalla (Malwarebytes).

3.2 OpenAI ja ChatGPT

OpenAI perustettiin vuonna 2015 voittoa tavoittelemattomana yhtiönä, jonka tarkoituksena oli kehittää turvallista ja hyödyllistä tekoälyä ihmiskunnalle. Vuonna 2019 yhtiö ilmoitti, että vaikuttavimmat tekoälyjärjestelmät sekä uudet algoritmit käyttävät niin paljon laskentatehoja, että heidän täytyy sijoittaa tulevina vuosina miljardeja dollareita pilvilaskentaan, osaavien ihmisten palkkaamiseen sekä supertietokoneiden rakentamiseen. Saman vuoden aikana Microsoft sijoitti yhtiöön miljardi dollaria tavoitteenaan auttaa kehittämään yleistä tekoälyä eli Artificial General Intelligence, lyhyesti AGI:a ja integroimaan sen Microsoftin Azure-pilviympäristöön. (Brockman 2019; Brockman & Sutskever 2019).

Vuoden 2022 marraskuun lopulla OpenAI julkisti suuren yleisön saataville ilmaisen ChatGPT-palvelunsa, joka käyttää edistynyttä GPT-3.5-kielimallia (OpenAI 2022). Nimi ChatGPT tulee sanoista Chat Generative Pre-trained Transformer (Lock 2022). Tammikuussa 2023 Microsoft ja OpenAI sopivat uudesta 10 miljardin dollarin sijoituksesta, joka toteutuisi usean vuoden kuluessa, ja sen tarkoituksena on jatkaa itsenäistä tutkimustyötä ja kehittää tekoälyä edelleen turvallisemmaksi, hyödyllisemmäksi ja tehokkaammaksi, luopumatta tavoitteesta hyödyttää ihmiskuntaa ja tinkimättä turvallisuudesta (OpenAI 2023b). ChatGPT saavutti 100 miljoonan aktiivisen käyttäjän rajapyykin tammikuussa 2023, vain kaksi kuukautta palvelun julkistamisen jälkeen, tehden siitä historian nopeimmin kasvavan kuluttajakäyttöön tarkoitetun sovelluksen (Hu 2023).

Helmikuussa 2023 OpenAI esitteli uuden ChatGPT Plus -palvelun, joka maksaa käyttäjälleen 20 dollaria kuukaudessa. Palvelu tarjoaa tilaajilleen pääsyn ChatGPT-palveluun jopa ruuhkaisina aikoina, on nopeampi tuottamaan vastauksia sekä tarjoaa ensisijaisen pääsyn

uusien ominaisuuksien ja parannusten pariin (OpenAI 2023b). Lisäksi alkaen 14. maaliskuuta ChatGPT Plus -palvelun käyttäjät saivat rajoitetun käyttöoikeuden uuteen GPT-4-malliin, joka kykenee ammattimaisella ja akateemisella tasolla ihmisen kaltaisiin suorituksiin ja on monitilainen, eli se osaa tulkita tekstin lisäksi myös kuvia (OpenAI 2023c).

Kuten laajoilla kielimalleilla yleensä, on myös ChatGPT:llä haittapuolensa. Aiemmin mainitun ”hallusinoinnin” lisäksi sillä on laaja pääsy käyttäjän dataan, mikä herättää huolta yksityisyydestä sekä tietoturvasta. Se myös saattaa olla joissakin asioissa puolueellinen, epäeettinen tai jopa rasistinen johtuen sen koulutuksessa käytetyistä kirjallisista materiaaleista, joissa aiheita käsitellään. Jotkut käyttäjät saattavat luottaa liikaa tekoälyn tuottamiin vastauksiin ja tämän takia yksilön oma ajatusmaailma voi muuttua. Tämä voi johtaa huonojen päätösten tekemiseen, esimerkiksi Kolumbiassa oikeuden tuomari teki autistisen lapsen vakuutusturvaa koskevan päätöksen ChatGPT:tä käyttäen. Koska kielimalli kykenee simuloimaan ihmisten välistä keskustelua todella tarkasti, voivat yksittäiset käyttäjät jopa luulla keskustelevansa oikean henkilön kanssa, mikä voi johtaa tietynlaiseen todellisuudentajun hämärtymiseen. Sitä voidaan myös hyödyntää pahantahtoisten ihmisten toimesta tuottamaan vihapuhetta, valeuutisia tai erittäin uskottavia huijausviestejä, koska se kykenee imitoimaan lähes ketä tahansa. Haittapuolistaan huolimatta ChatGPT on vienyt tekoälyn kehitystä merkittävästi eteenpäin ja sen kehitys jatkuu edelleen, minkä lisäksi haittoja pyritään tulevaisuudessa kitkemään muuttamalla lainsäädäntöä kielimallien osalta sekä vahvistamalla palvelun turvatoimia. (Ray 2023).

4 Laajennuksen toteutuksessa käytetyt teknologiat

4.1 Verkkosivut

Tämä oli ensimmäinen kerta, kun kumpikaan meistä oli suunnittelemassa ja kehittämässä selainlaajennusta, minkä vuoksi asiaan liittyvät dokumentaatiot ja ohjeet olivat tärkeässä roolissa työn kannalta. OpenAI:n rajapintaan yhteyden saamiseksi tuli luoda paikallinen palvelin, jonka toteuttamisessa käytimme apuna Express.js:n sekä OpenAI:n dokumentaatioita. Näiden lisäksi Mozillan ylläpitämä MDN Web Docs toimi pääasiallisena tiedonlähteenä JavaScript- HTML- ja CSS-koodin tuottamisessa, kuten myös joiltain osin W3Schoolsin tarjoamat ohjeet.

Selainlaajennuksen rautalankamallin suunnitteluun käytimme Figma-verkkosivusovelluksen ilmaisversiota. Figma on suosittu työkalu käyttöliittymän, käyttökokemuksen ja prototyyppien suunnitteluun, koska se mahdollistaa työryhmien reaaliaikaisen työskentelyn projektin parissa. Palvelun hyödyllisiin ominaisuuksiin kuuluu esimerkiksi suunnittelijan ja kehittäjän välisen yhteistyön mahdollistaminen, koska mm. palvelussa luodun prototyypin asettelu- ja väriasetukset ovat yhteensopivia yleisten web-standardien kanssa. (Figma).

Koska selainlaajennuksella on oltava työkalurivillä kuvaava logo tai kuvake, tarvitsimme luonnollisesti sellaisen myös omaan laajennukseemme. Teeman mukaisesti päätimme hyödyntää logon suunnittelussa tekoälyä, joten päädyimme käyttämään OpenAI:n DALLE-2-kuvageneraattoria, joka luo annettujen avainsanojen perusteella kuvia aina maalauksista fotorealistisiin, ”valokuvamaisiin” kuviin. Avainsanojamme olivat ”chatbot”, ”logo” ja ”no text”, ja näiden avulla saimme riittävän yksinkertaisen, mutta selkeän kuvan, jonka muokkauksen jälkeen asetimme laajennuksen kuvakkeeksi. Kuvan muokkaukseen ja siistimiseen käytimme internetistä löytyvää ilmaista kuvanmuokkausohjelmaa nimeltään Photopea, joka käyttöliittymältään muistuttaa Adoben Photoshop-kuvankäsittelyohjelmaa.

Laajennuksen ulkonäön toteuttamisessa käytimme väriteematyökaluna Color Hunt -sivustolta saatuja väripaletteja, jotka on suunniteltu käyttäjäystävällisiksi. CSS:n puolella toteutetun latausanimaation puolestaan löysimme CodePen-sivustolta ja muokkasimme sitä tarpeisiimme sopivaksi. Lisäksi viestinlähetyspainikkeen ikonin saimme suoraan Google Fonts -palvelusta.

4.2 Ohjelmointi

Ohjelmakoodin kirjoittamisessa käytimme työkaluna Microsoftin Visual Studio Code -tekstieditoria, joka on ilmainen, ohjelmoijille tarkoitettu avoimen lähdekoodin ohjelmisto. Kyseiseen sovellukseen on mahdollisuus asentaa tuhansia lisäosia, joiden avulla sovelluksella voi mm. tuottaa lähes mitä tahansa ohjelmointikieltä, mutta tarvitsemamme JavaScript-HTML- ja CSS-kielten tuki löytyy editorista oletuksena. Versionhallinnassa hyödynsimme GitHub-palvelua, joka toimii saumattomasti Visual Studio Coden kanssa, kun tietokoneelle on asennettuna git-versionhallintajärjestelmä. Tämän ansiosta laajennuksen tuorein versio oli aina saatavilla molemmille osapuolille, mikä oli hyödyksi esimerkiksi itsenäisen testaamisen toteuttamisessa.

Laajennuksen toiminnallisuuden kannalta tärkeässä roolissa oli Node.js, joka sallii koodin suorittamisen paikallisesti selaimen ulkopuolella, eli se toimii käytännössä paikallisena palvelimena. Laajennus vaati toimiakseen kirjastot, joita ovat OpenAI, Express, Body-parser sekä CORS. OpenAI:n kirjasto mahdollistaa pääsyn OpenAI REST-, eli Representational State Transfer -rajapintaan JavaScriptiä käyttäen (NPM 2023b). Express on nopea ja kevyt web-sovelluskehys, joka tarjoaa yksinkertaisia työkaluja web- ja mobiilisovellusten kehittämiseen ja tekee HTTP-pyyntöjen käsittelystä ja reitityksestä helpompaa (NPM 2022). Body-parser on puolestaan väliohjelma Expressille, ja se käsittelee saapuvien HTTP-pyyntöjen runkoa muuttamalla sen helposti käsiteltäväksi JavaScript-objektiksi (NPM 2023a). Myös CORS, eli Cross-Origin Resource Sharing on Expressin väliohjelma, joka mahdollistaa ristiin menevien resurssien, kuten fonttien, kuvien ja API-pyyntöjen käytön toiselta verkkotunnukselta (NPM 2018). Nämä kaikki ovat asennettavissa Noden tarjoaman paketinhallintajärjestelmän eli Node Package Managerin kautta.

Selainlaajennuksen rakenne luotiin käyttämällä HTML-merkintäkieltä, tyylimäärittely toteutettiin CSS:llä ja toiminnallisuudet JavaScript-ohjelmointikielellä. HTML-tiedosto sisältää erilaisia elementtejä, jotka kertovat selaimelle, mitä sisältöä näytetään käyttäjälle. Elementeissä voi olla attribuutteja, joilla voidaan määrittää tiettyjä ominaisuuksia tai tyylejä. Esimerkiksi tekstielementille voidaan antaa attribuutiksi ”class”, jonka arvo olisi ”warning-text”, eli kyseessä olisi varoitusteksti, jolloin CSS:n puolella voidaan viitata kyseiseen arvoon ja antaa sille ominaisuuksia, kuten punainen väri, lihavointi, fontti ja sen koko, tai kaikki edellä mainitut (MDN Web Docs 2023c). JavaScript on puolestaan verkkoselainten tulkitsema ohjelmointikieli, jolla verkkosivuille voidaan lisätä dynaamisia ja interaktiivisia ominaisuuksia,

kuten elementtien manipulointi, datan käsittely tai käyttäjän toimiin reagointi (MDN Web Docs 2023a).

Ohjelmoinnin tukena käytimme, kuitenkin suhteellisen harvoissa tapauksissa, ChatGPT-palvelua, jonka avulla saimme nopeasti tarkistettua esimerkiksi JavaScript-metodien oikeanlaisen syntaksin tai virheilmoitusten sisältämän virhekoodin selityksen. Joissakin tapauksissa pyysimme ChatGPT:tä kertomaan yksityiskohtaisesti joistakin JavaScriptin toiminnoista suomeksi, mikä auttoi ymmärtämään logiikan toiminnon takana. Vaikka kyseinen palvelu pystyy tuottamaan valmista ohjelmakoodia, sen käyttö rajoittui osaltamme edellä mainittuihin tapauksiin.

Kuten on jo mainittu, kehittämämme laajennus hyödyntää keskustelutoiminnoissaan ChatGPT-palvelua. Vaikka itse ChatGPT:n version 3.5 käyttäminen on maksutonta, vaatii tässä opinnäytetyössä luotu sovellus toimiakseen API-avaimen, eli yksilöllisen merkkijonon, jonka käytöstä peritään tokenien käyttömäärästä riippuvaisia rahasummia. Tämä johtuu siitä, että laajennusta ei voi yhdistää suoraan ChatGPT:n verkkosovellukseen kirjautumatta, vaan yhteys tulee ottaa rajapinnan kautta, minkä ansiosta erillistä kirjautumistoimintoa ei kyseisen selainlaajennuksen käyttöön vaadita. Syöteteksti ja vastauksen sisältämä teksti hinnoitellaan OpenAI:n sivuilta löytyvän hinnoittelutaulukon mukaisesti riippuen käytetystä mallista. Tässä opinnäytetyössä luomamme keskustelubotti voi käyttää OpenAI:n 3.5-turbo-4k- tai 3.5-turbo-16k-mallia, joiden hinnoittelu on nähtävillä taulukossa 2 (OpenAI a).

Kielimallin versio	**Syötteen hinta**	**Vastauksen hinta**
GPT-3.5-turbo-4k	$0,0015 / 1000 tokenia	$0,002 / 1000 tokenia
GPT-3.5-turbo-16k	$0,003 / 1000 tokenia	$0,004 / 1000 tokenia

Taulukko 2. Eri kielimallien rajapintojen hinnasto (OpenAI a)

5 Selainlaajennuksen toteuttaminen

5.1 Prototyyppi ja suunnittelu

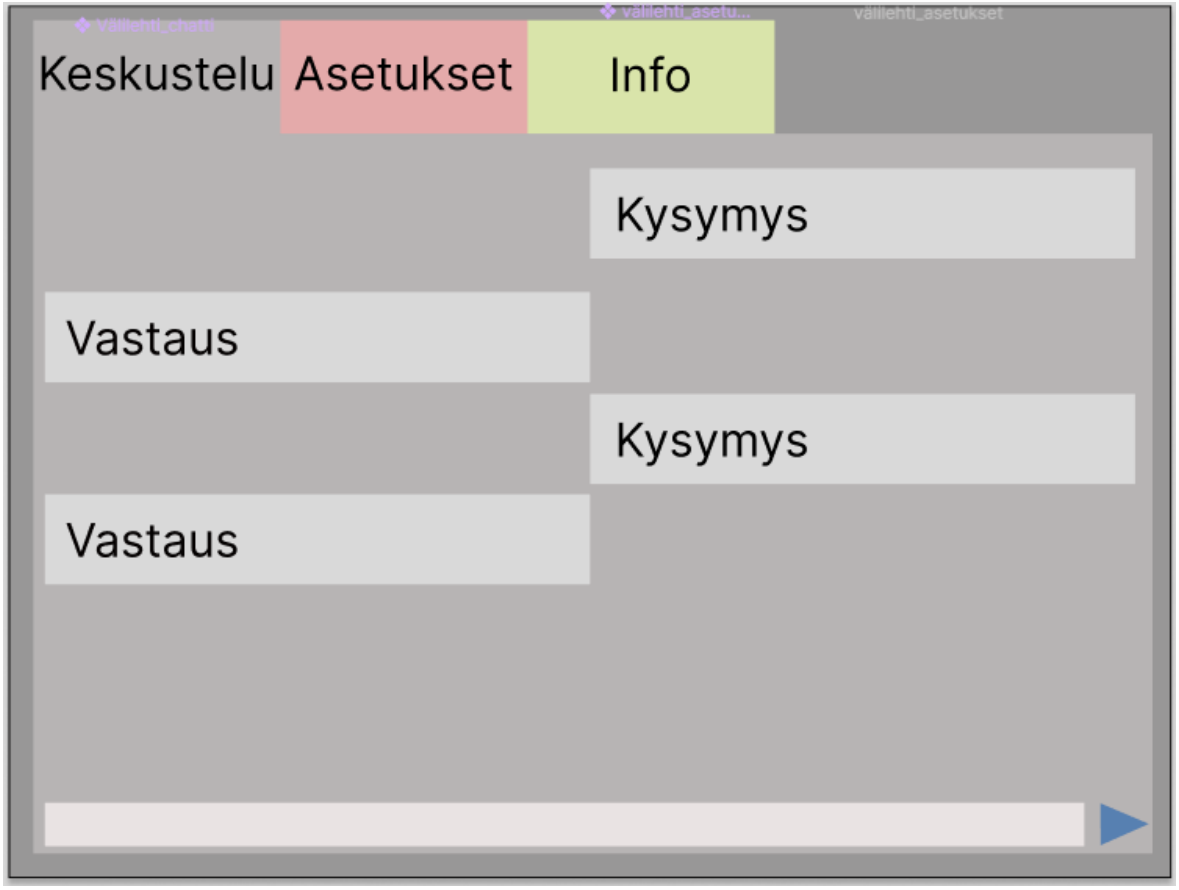
Projektin ensimmäinen vaihe oli luoda alkeellinen prototyyppi. Tutustuimme OpenAI:n dokumentaatioon ja loimme yhteisen käyttäjätunnuksen palveluun. Rajapinnan käyttö vaati maksutavan lisäämisen, joten vahinkojen estämiseksi asetimme turvarajaksi kolme dollaria. Tunnuksen luotuamme generoitiin käyttöömme API-avain. Koska rajapinnan käyttö vaatii palvelinta, asennettiin projektiin Node.js-ajoympäristö, joka sallii koodin suorittamisen paikallisella palvelimella. Projektiin asennettiin lisäksi tarvittavat kirjastot NPM:n kautta.

Aluksi toteutimme yksinkertaisen ohjelman, joka otti syötteen suoraan koodista ja palautti ChatGPT:n generoiman vastauksen rajapinnan kautta Visual Studio Coden konsoliin. Tämä onnistui luomalla tiedosto index.js, jonka toiminnallisuutta on selitetty tarkemmin luvussa 5.2. Ensimmäisen kokeilun onnistuttua toteutimme toisen testin yksinkertaisella HTML-sivulla (kuva 5). Verkkosivulla oli syöterivi, johon kirjoitettiin kysymys ChatGPT:lle, ja nappia painamalla palautettiin kielimallin vastaus tekstielementtinä. Testisivu vaati toimiakseen tiedoston app.js, jonka avulla syöteteksti lähetetään palvelimelle ja vastaus palautetaan selaimen ikkunaan luettavassa muodossa. Myös app.js:n toimintoja selostetaan tarkemmin seuraavissa alaluvuissa.

Kuva 5. Yksinkertainen prototyyppi.

Kun prototyypin toiminnallisuus oli varmistettu, aloitimme rautalankamallin suunnittelun Figmassa. Laajennuksessa tuli olla kolme välilehteä: Keskustelu, Asetukset ja Info. Laajennuksen mitoiksi päätimme alustavasti 800x600 pikseliä ja malli tehtiin näiden mittojen mukaisesti. Käytimme eri välilehdille eri värejä mallin selkeyttämiseksi (kuva 6), mutta lopullinen toteutus tulisi käyttämään yhtenäistä väriteemaa.

Tavoitteena oli, että Keskustelu-välilehti tulisi sisältämään tyhjän elementin botin kanssa käytävää keskustelua varten, sekä tekstinkentän ja lähetyspainikkeen. Asetukset-välilehti puolestaan antaa mahdollisuuden muuttaa laajennuksen väriteemaa tummaan tai vaaleaan, sisältää tekstikentän botille annettavaa roolia varten, sekä keskustelun tyhjennyspainikkeen. Info-välilehti ohjeistaa käyttäjää laajennuksen käytössä ja kertoo laajennuksen tekijöiden nimet.



Kuva 6. Rautalankamalli.

5.2 Yhteys rajapintaan

Tämän selainlaajennuksen tärkein toiminto on lähettää syöte OpenAI:n rajapintaan ja palauttaa kielimallin vastaus luettavassa muodossa käyttäjälle. Tiedosto index.js luo palvelimen, joka välittää käyttäjän syötteen rajapintaan ja palauttaa kielimallin vastauksen laajennukseen (kuva 7). Koska palvelin on paikallinen, täytyy index.js käynnistää komentoriviltä käyttäen komentoa ”node .\src\index.js” aina ennen laajennuksen käyttöä. Palvelin

hyödyntää express.js-verkkosovelluskehystä, sekä body-parser- ja CORS-nimisiä väliohjelmistoja, jotka edesauttavat HTTP-pyyntöjen käsittelyä.

```
import express from "express";
import { Configuration, OpenAIApi } from "openai";
import bodyParser from "body-parser";
import cors from "cors";

//Setup config for api
const configuration = new Configuration({
  organization: "[redacted]",
  apiKey: "[redacted]",
});
const openai = new OpenAIApi(configuration);

//Create express app with port 3000
const app = express();
const port = 3000;

//Deploy middleware
app.use(bodyParser.json());
app.use(cors());

//Send prompt to API and return answer
app.post("/", async (req, res) => {
  const { messages } = req.body;
  const completion = await openai.createChatCompletion({
    model: "gpt-3.5-turbo",
    messages: [
      ...messages,
    ],
  });
  res.json(
    completion: completion.data.choices[0].message,
  );
});

app.listen(port, () => {
  console.log(`Kuunnellaan porttia ${port}`);
});
```

Kuva 7. Tiedoston index.js koodi.

Laajennuksen lähetyspainiketta painettaessa app.js-tiedosto (kuva 8) suorittaa funktion, joka tallentaa tekstikentän viestin muuttujaan messageText. Tämä muuttuja sisällytetään user-roolilla newMessage-olioon, joka myöhemmin lisätään taulukkoon messages. Palvelimeen lähetetään fetch-pyyntö POST-metodilla. Pyynnön sisältönä on edellä mainittu messages-taulukko, joka on muutettu JSON-muotoon. Palvelin vastaanottaa pyynnön ja lähettää sen rajapintaan, minkä jälkeen rajapinta palauttaa vastauksen ja palvelin välittää sen

koodin käsiteltäväksi JSON-muodossa. Kielimallin vastaus sisällytetään assistant-roolin ominaisuudessa olioon, joka lisätään messages-taulukkoon, ja tämä lähetetään aina uuden viestin mukana kielimallille keskustelun kontekstin säilyttämiseksi. Samalla vastaus asetetaan parametrina printAnswer-funktioon, joka luo elementin ja tulostaa sen käyttäjän ruudulle laajennukseen luettavaan muotoon. Kuva 8 esittää app.js-tiedoston sisältämän, ainoastaan kielimallin toimivuuden kannalta oleellisimman koodin, joka laajennuksen kehittämisen edetessä sai lisää toimintoja, mutta niihin palataan tässä tekstissä myöhemmin.

```
let messages = [];
const message = document.getElementById("message");
const form = document.getElementById("send");

//Click listener for send button
form.addEventListener("click", (e) => {
  e.preventDefault();

  //Save prompt into variable and store it into object
  let messageText = `${message.value}`;
  const newMessage = { role: "user", content: `${messageText}` };

  //Push message to array and reset message value
  messages.push(newMessage);
  message.value = "";

  //Send array to localhost(API) and return answer
  fetch("http://localhost:3000", {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify({
      messages,
    }),
  })
    .then((res) => res.json())
    .then((data) => {
      //Push object into array for context
      let newAssistantMessage = {
        role: "assistant",
        content: `${data.completion.content}`,
      };
      messages.push(newAssistantMessage);
      //Print element with answer
      printAnswer(data.completion.content);
    });
});
```

Kuva 8. Tiedoston app.js viestien lähettämisen ja vastaanottamisen koodi.

5.3 Laajennuksen rakenne ja toiminnallisuus

Saatuamme onnistuneesti yhteyden rajapintaan, oli aika aloittaa varsinaisen laajennuksen rungon kehittäminen. Pohjana käytimme alkuperäistä prototyyppiä, johon lisäsimme ominaisuuksia. Aluksi kehitys tapahtui selaimessa verkkosivuna, joka rakentui neljästä tiedostosta: index.html toimii laajennuksen runkona, joka sisältää staattiset elementit, kuten laajennuksen välilehdet ja niiden valintapainikkeet, style.css määrittelee elementtien asettelun ja värit, sekä aiemmin mainitut index.js ja app.js, jotka lisäävät laajennukseen dynaamisuuden ja datan siirron.

Kun verkkosivun runko oli suunnitelman mukainen ja perustoiminnot olivat kunnossa, lisättiin projektiin vielä Chrome-selaimeen yhteensopiva manifest.json-tiedosto, joka mahdollistaa HTML-dokumentin avaamisen selaimen yläreunan laajennusvalikossa. Seuraavien alaotsikoiden alla avataan tarkemmin laajennuksen Keskustelu- ja Asetukset-välilehtien toimintoja. Info-välilehti sisältää ainoastaan staattisia elementtejä, joita ovat laajennuksen käyttöohjeet, käyttäjälle kerrottu varoitus yksityisten tietojen salassapidosta sekä tekijöiden nimet ja laajennuksen toteuttamisvuosi, joten se ei vaadi omaa alaotsikkoa.

5.3.1 Keskustelu-välilehti

Keskustelu-välilehdessä (kuva 9) on tyhjä div-elementti, johon on varattu tilaa käyttäjän ja botin keskustelun elementeille, jotka luodaan app.js-tiedostossa. Välilehdestä löytyy myös tekstikenttä sekä lähetyspainike. Tekstikentässä on oletuksena paikanvaraaja eli placeholder, joka on himmeämmällä värillä näkyvä teksti ohjeistaen käyttäjää kirjoittamaan viestinsä keskustelubotille. Tämä teksti korvautuu välittömästi käyttäjän kirjoittamalla, kirkkaamman värisellä tekstillä, kun ensimmäinen merkki on kirjoitettu kenttään.

Keskusteluruudun ollessa tyhjä, tulee ensimmäinen lähetetty viesti näkyviin ruudun oikeaan yläreunaan puhekuplaan ja botin vastaus puolestaan sen alapuolelle vasempaan reunaan. Kun viesti on lähetetty, tulostetaan välittömästi puhekuplaan näkyviin botin viestin latautumista kuvaava teksti ”Kirjoittaa…”, jossa näkyvät kolme pistettä on animoitu ilmaisemaan kielimallin ”kirjoittavan” vastausta. Kun botin viesti on saatu kokonaisuudessaan palvelimelta, korvataan latausta ilmaiseva puhekupla uudella, vastauksen sisältävällä puhekuplalla keskusteluikkunaan. Keskusteluruudun täyttyessä ilmestyy laajennuksen oikeaan reunaan vierityspalkki, ja näkymä pysyy aina uusimman viestin kohdalla.

Kuva 9. Näkymä keskustelusta, jossa botin viesti latautuu.

Kuten luvussa 5.2 mainitaan, lähetyspainiketta painamalla lähtee käyttäjän syöte palvelimelle. Tämän lisäksi lähetyspainikkeen addEventListener-metodi sisältää nyt if-ehtolauseen, joka tarkistaa, että viesti ei ole tyhjä. Jos viesti on tyhjä, käyttäjä saa alert-toiminnolla ilmoituksen, joka kehottaa tätä kirjoittamaan tekstikenttään jotakin. Mikäli lähetetty syöte ei ole tyhjä, suoritetaan ensimmäisenä funktio printPrompt (kuva 10), joka luo tyhjän div-elementin. Sille asetetaan luokat ”speech” ja ”right”, ja siihen sisällytetään parametrinä saatu käyttäjän tekstisyöte, joka on tallennettuna muuttujaan msgPrompt. Lopuksi elementti lisätään sen luokan perusteella laajennuksen oikeaan reunaan puhekuplana.

```
function printPrompt(prompt) {
  //Create div element and text node of prompt and insert text node into div
  const speechBubble = document.createElement("div");
  const msgPrompt = document.createTextNode(prompt);
  speechBubble.appendChild(msgPrompt);
  //Set classes for speech bubble and insert speech bubble into chat window
  speechBubble.setAttribute("class", "speech right");
  chatBox.appendChild(speechBubble);
}
```

Kuva 10. Funktio printPrompt app.js-tiedostossa.

Kun käyttäjän viestin sisältämä puhekupla on luotu, suoritetaan funktio loadingElement (kuva 11), joka ensimmäisenä luo tyhjän puhekuplan laajennuksen vasempaan reunaan kutsumalla printAnswer-funktiota parametrillä, joka on tyhjä merkkijono. Kyseisen printAnswer-funktion toimintoja avataan seuraavassa kappaleessa. Tämän jälkeen luodaan uusi div-elementti, jolle määrätään tunniste eli ID. Seuraavana luodaan tekstielementti, johon sisällytetään teksti ”Kirjoittaa”, sekä CSS:n puolella animoidut kolme pistettä, jotka viestivät käyttäjälle vastauksen latautumisesta. Tämä div-elementti asetetaan tyhjän puhekuplan sisään. Lopuksi vieritetään keskusteluikkuna alas viimeisimmän puhekuplan kohdalle, jolloin käyttäjä näkee aina viimeisimmän viestin.

```
function loadingElement() {
  //Create empty speech bubble
  printAnswer("");
  //Create div and set id
  const loadingDiv = document.createElement("div");
  loadingDiv.setAttribute("id", "loading");
  //Create <p> element and set class load-text
  const loadingText = document.createElement("p");
  loadingText.setAttribute("class", "load-text");
  //Create textnode and append it into <p> element
  const text = document.createTextNode("Kirjoittaa");
  loadingText.appendChild(text);
  //Append <p> element into div (loading element)
  loadingDiv.appendChild(loadingText);
  //Append loading div into speech bubble
  const speechBubble = document.getElementById("loader");
  speechBubble.appendChild(loadingDiv);
  //Scroll chat window to bottom
  chatBox.scrollTo(0, chatBox.scrollHeight);
}
```

Kuva 11. Funktio loadingElement app.js-tiedostossa.

Palvelimen palauttaessa kielimallin vastauksen, poistetaan loadingElement-funktion luoma puhekupla, joka sisälsi latausta kuvaavan tekstin. Heti perään kutsutaan aiemmin mainittu funktio printAnswer (kuva 12), jolle annetaan tällä kertaa parametriksi palvelimen palauttama viesti. Funktiossa luodaan tyhjä div-elementti, jolle lisätään luokat ”speech” ja ”left”. Mikäli funktion parametrina on tyhjä merkkijono, kuten aiemmassa kappaleessa mainittiin, lisätään puhekuplalle loader-tunniste, jonka ansiosta vastauksen saapuessa voidaan ”latauselementti” poistaa vaivattomasti. Muussa tapauksessa tunnistetta ei lisätä. Seuraavaksi puhekuplaan asetetaan muuttuja, joka luo tekstisolmun eli text noden annetusta

parametrista, joka on palvelimen lähettämä kielimallin vastaus. Lopuksi lisätään puhekuplaelementti keskusteluikkunaan ja vieritetään keskustelunäkymä alas.

```
function printAnswer(answerString) {
  //Create div element and save chat window into variable
  const speechBubble = document.createElement("div");
  //Add two classes, speech and left
  speechBubble.setAttribute("class", "speech left");
  //If answerString is empty, set loader id for removing element later
  if (answerString === "") {
    speechBubble.setAttribute("id", "loader");
  }
  //Create textnode and append it into speech bubble
  const msgString = document.createTextNode(answerString);
  speechBubble.appendChild(msgString);
  //Append speech bubble into chat box and scroll to bottom
  chatBox.appendChild(speechBubble);
  chatBox.scrollTo(0, chatBox.scrollHeight);
}
```

Kuva 12. Funktio printAnswer app.js-tiedostossa.

Keskustelua testatessa huomasimme, että keskusteluikkuna tyhjentyi, kun laajennuksesta poistui vahingossa tai tahallisesti. Tilanteen korjaamiseksi keksimme hyödyntää selaimen paikallista muistia eli local storagea viestihistorian säilyttämiseksi. Päätimme, että tallentaisimme lähetetyt ja vastaanotetut viestit omiin taulukkoihinsa, minkä ansiosta niiden tulostaminen olisi myöhemmin helpompaa. Taulukoille annettiin nimeksi sentHistory ja answerHistory, jotka alustetaan aina laajennuksen avaamisen yhteydessä. Taulukoiden tallentamiseksi selaimen paikalliseen muistiin loimme saveToHistory-funktion (kuva 13), joka tallentaa molemmat taulukot omiin avainarvoihinsa, eli key name:ihin setItem-metodilla. Lisäsimme saveToHistory-funktion lähetyspainikkeen addEventListener-metodin alle, jossa tallennetaan lähetetty merkkijono taulukkoon ja tämän jälkeen taulukko syötetään parametrinä saveToHistory-funktiolle. Sama toiminto toistetaan myös rajapinnan palauttamalle vastaukselle. Tällöin kummatkin taulukot tallentuvat selaimen paikalliseen muistiin omiin avainarvoihinsa.

```
//Check parameter for sent or received message and save it into local storage
function saveToHistory(arr) {
  if (arr === sentHistory) {
    localStorage.setItem("sentMessages", JSON.stringify(arr));
  } else if (arr === answerHistory) {
    localStorage.setItem("receivedMessages", JSON.stringify(arr));
  }
}
```

Kuva 13. saveToHistory-funktio, joka tallentaa viestit paikalliseen muistiin app.js-tiedostossa.

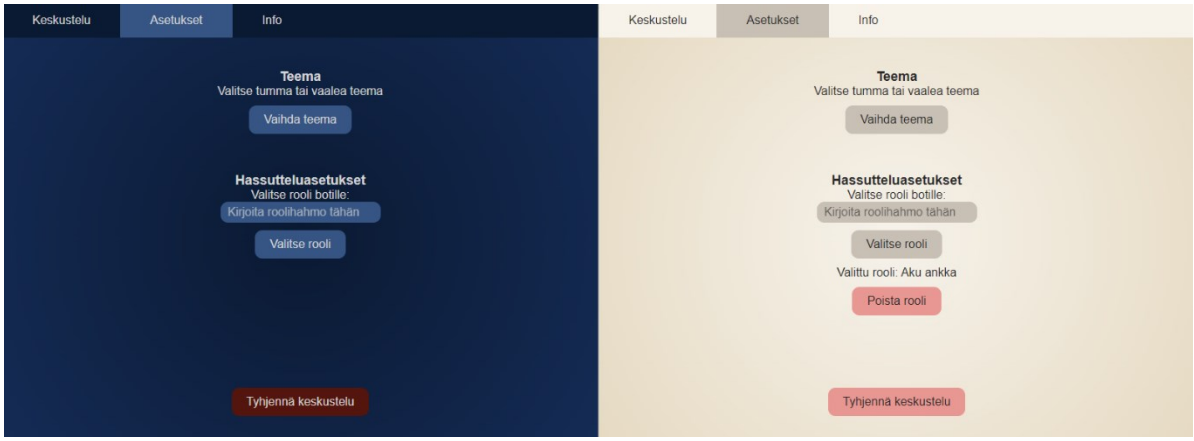
Viestihistorian palauttamiseksi laajennuksen avaamisen yhteydessä suoritetaan aina getHistory-funktio (kuva 14), joka tarkistaa, onko paikalliseen muistiin tallennettu aikaisempia viestejä. Mikäli paikallisesta muistista löytyy viestejä, eli viestihistoria ei ole tyhjä, tallennetaan ne sentHistory- ja answerHistory-taulukoihin. Tämän jälkeen kutsutaan printHistory-funktio, joka tulostaa taulukoissa olevat viestit keskusteluikkunan puhekupliin omille paikoilleen käyttäen for-silmukkaa.

```
//Create arrays for message history
let sentHistory = [];
let answerHistory = [];

//IIFE to save messages from local storage into arrays
(function getHistory() {
  //Save messages into arrays if local storage is not empty and print them
  if (localStorage.getItem("receivedMessages") !== null) {
    sentHistory = JSON.parse(localStorage.getItem("sentMessages"));
    answerHistory = JSON.parse(localStorage.getItem("receivedMessages"));
    printHistory();
  }
})();

function printHistory() {
  //Print messages into individual speech bubbles alternately
  for (let i = 0; i < sentHistory.length; i++) {
    printPrompt(sentHistory[i]);
    printAnswer(answerHistory[i]);
  }
}
```

Kuva 14. Funktiot viestihistorian palauttamiseen app.js-tiedostossa.

5.3.2 Asetukset-välilehti

Asetukset-välilehti sisältää tarkalleen kolme toimintoa, joita käyttäjä voi halutessaan käyttää. Näitä ovat tässä tekstissä aiemmin mainitut väriteeman vaihto, botin roolin valitseminen, sekä painike keskustelun tyhjentämiseksi. Kuvassa 15 on havainnollistettu vasemmalla puolella Asetukset-välilehden ulkonäkö tummalla teemalla ilman roolivalintaa, ja oikealla puolella vaalealla teemalla sama sivu roolivalinnan kanssa. CSS-tiedostossa painikkeiden selectorille, eli valitsimelle, on määritettynä hover-pseudoluokka eli pseudo-class, joka saa painikkeen värin muuttumaan, kun osoitin viedään sen päälle. Seuraavana on yksityiskohtaisemmin selitettynä välilehden sisältämiä toimintoja.



Kuva 15. Tumma teema ilman roolia sekä vaalea teema roolivalinnan kanssa.

Laajennuksen väriteeman vaihto on toteutettu käyttämällä CSS-muuttujia (kuva 16), mikä mahdollistaa teeman vaihtamisen yksinkertaisella funktiolla app.js-tiedostossa (kuva 17). Kun laajennus avataan, ajetaan automaattisesti IIFE-funktio (kuva 17), joka määrittää ensimmäisellä kerralla laajennusta käytettäessä teeman tummaksi ja tallentaa tiedon paikalliseen muistiin, minkä jälkeen se toimii seuraavilla käyttökerroilla ainoastaan tarkistusfunktiona eikä varsinaisesti tee mitään. ”Vaihda teema” -painiketta painamalla sille määritetty addEventListener-metodin if-ehtolause tarkistaa, onko selaimen paikalliseen muistiin tallennettu tumma vai vaalea teema. Jos teema on tumma, painikkeen klikkaaminen vaihtaa sen vaaleaksi tai mikäli se on vaalea, se vaihtuu tummaksi. Samalla uusi teema tallennetaan selaimen paikalliseen muistiin setTheme-funktion sisällä, ja se pysyy muistissa myös selaimen sulkeuduttua.

```
/*Variables for colors*/
.theme-dark {
  --background: #2a5586;
  --background-secondary: #0a2b57;
  --navbar: #051a34;
  --navbar-hover: #0d2c52;
  --navbar-active: #2a5586;
  --speech-left: #11345c;
  --speech-right: #2a5586;
  --font-color: #d6d6d6;
  --button-hover: #2c74b3;
  --dots: #11345c;
  --warning: rgb(92, 11, 11);
}
```

```
.theme-light {
  --background: #f7ecde;
  --background-secondary: #e9dac1;
  --navbar: #f8f3eb;
  --navbar-hover: #c4bcb0;
  --navbar-active: #cac0b4;
  --speech-left: #9ed2c6;
  --speech-right: #54bab9;
  --font-color: #292828;
  --button-hover: #999289;
  --dots: #9ed2c6;
  --warning: rgb(248, 145, 145);
}
```

Kuva 16. Värien muuttujat style.css-tiedostossa.

```
//Save button into a variable
const themeButton = document.getElementById("themeChange");

//On click, check theme from local storage and call setTheme function
themeButton.addEventListener("click", () => {
  if (localStorage.getItem("theme") === "theme-dark") {
    setTheme("theme-light");
  } else {
    setTheme("theme-dark");
  }
});

//Save theme in local storage and change theme in extension
function setTheme(themeName) {
  localStorage.setItem("theme", themeName);
  document.documentElement.className = themeName;
}

//IIFE to set the theme on initial load
(function () {
  if (
    localStorage.getItem("theme") === "theme-dark" ||
    localStorage.getItem("theme") === null
  ) {
    setTheme("theme-dark");
  } else {
    setTheme("theme-light");
  }
})();
```

Kuva 17. JavaScript-funktiot teeman vaihtamiseen app.js-tiedostossa.

Asetuksissa käyttäjällä on halutessaan mahdollisuus asettaa rooli keskustelubotille. Se voi esittää esimerkiksi elokuvan henkilöä tai omaksua pyydetyn tunnetilan tai tietyn käyttäytymistyylin. Käyttäjän tarvitsee vain kirjoittaa haluamansa rooli tekstikenttään ja painaa ”Valitse rooli” -painiketta. Syötetty rooli asetetaan muuttujaan role, minkä lisäksi se tallentuu paikalliseen muistiin. Kun rooli on valittuna, lähetettävään syötteeseen lisätään joka kerta lisäteksti, joka pyytää keskustelubottia esittämään valittua roolia (kuva 18). Roolinvalintakentän alapuolelle laajennukseen ilmestyy lisäksi teksti, joka ilmaisee sen hetkisen käytössä olevan roolin, sekä ”Poista rooli” -painike, jolla roolin voi poistaa käytöstä.

Kun rooli poistetaan, annetaan viestin syötteessä yhden kerran pyyntö lopettaa roolin esittäminen ja tämän jälkeen viestit lähetetään jälleen ilman lisäkehotteita. Mikäli roolivalintaa ei poisteta, se tallentuu selaimen paikalliseen muistiin, kuten aiemmin mainitut laajennuksen viestihistoria sekä teema. Näin käyttäjän ei tarvitse joka kerta asettaa roolia uudelleen.

```
//If role is not empty, modify prompt
if (role != "") {
  //If role is "no_role", modify it once for resetting role
  if (role === "no_role") {
    messageText = `Vastaa ilman roolia: ${message.value}`;
    role = "";
  } else messageText = `Vastaa kuin olisit ${role}: ${message.value}`;
}
```

Kuva 18. Lähetyspainikkeen addEventListener-metodin sisällä oleva koodi roolille app.js-tiedostossa.

Laajennuksen avautuessa tarkistetaan, onko aiemmalla käyttökerralla tallentunut roolia paikalliseen muistiin (kuva 19). Jos roolia ei löydy, alustetaan roolimuuttuja tyhjäksi merkkijonoksi. Mikäli paikallisesta muistista löytyy rooli edelliseltä istunnolta, asetetaan se nykyiseksi rooliksi muuttujaan.

```javascript
//Create role variable and save elements into variables
let role;
const roleName = document.getElementById("chosenRole");
const removeRoleBtn = document.getElementById("removeRole");

//Check local storage for role
if (localStorage.getItem("role") === null) {
  role = "";
} else {
  //Role found = role name and button visible
  role = localStorage.getItem("role");
  removeRoleBtn.style.visibility = "visible";
  roleName.innerText = `Valittu rooli: ${role}`;
}

//Save button into variable and add event listener
const roleBtn = document.getElementById("chooseRole");
roleBtn.addEventListener("click", (e) => {
  e.preventDefault();
  //Grab role name and save it into local storage
  role = document.getElementById("rolename").value;
  localStorage.setItem("role", role);
  //If input is empty, show alert
  if (role === "") {
    alert("Kirjoita jokin rooli!");
  } else {
    /*If role is chosen, make role removal button visible,
    print chosen role and reset input field*/
    removeRoleBtn.style.visibility = "visible";
    roleName.innerText = `Valittu rooli: ${role}`;
    document.getElementById("rolename").value = "";
  }
});
//Listener for role removal button
removeRoleBtn.addEventListener("click", () => {
  /*Remove role from local storage
  hide button, set no_role as role, empty printed role*/
  localStorage.removeItem("role");
  removeRoleBtn.style.visibility = "hidden";
  role = "no_role";
  roleName.innerText = ``;
});
```

Kuva 19. Koodi roolin asettamiselle ja poistamiselle app.js-tiedostossa.

Käyttäjä voi tyhjentää käydyn keskustelun klikkaamalla ”Tyhjennä keskustelu” -painiketta. Tällöin laajennus varmistaa confirm-metodilla, että keskustelu halutaan varmasti tyhjentää tai vaihtoehtoisesti peruutetaan tyhjennyspyyntö klikkaamalla varoitusikkunan ”Peruuta”-painiketta. Jos pyyntö varmistetaan, keskusteluikkunasta poistuvat kaikki näkyvät viestit ja paikallisesta muistista poistetaan lähetetyt sekä vastaanotetut viestit (kuva 20).

```
//Save button and chatbox into variables and add event listener
const emptyButton = document.getElementById("emptyChat");
const chatContainer = document.getElementById("chatting");

emptyButton.addEventListener("click", () => {
  //Ask user to confirm message history deletion
  if (confirm("Haluatko varmasti poistaa kaikki viestit?")) {
    //Remove all child nodes from chatbox and empty arrays
    removeAllChildNodes(chatContainer);
    sentHistory = [];
    answerHistory = [];
    //Clear localstorage messages
    localStorage.removeItem("sentMessages");
    localStorage.removeItem("receivedMessages");
  } else {
    console.log("Viestejä ei poistettu :)");
  }
});

//Function to loop through child elements and remove them all
function removeAllChildNodes(parent) {
  while (parent.firstChild) {
    parent.removeChild(parent.firstChild);
  }
}
```

Kuva 20. Koodi keskusteluhistorian poistamiselle app.js-tiedostossa.

5.4 Laajennuksen käytettävyys ja viimeistely

Kun laajennus alkoi olla perustoimintojen osalta valmis, oli aika kiinnittää huomiota yksityiskohtiin käytettävyyden kannalta. Koska kyseessä on chatbot-laajennus, on käytettävyys huomioiden oleellista, että laajennus käynnistyy aina avatessa ensimmäisenä Keskustelu-välilehdelle. Aluksi näin ei kuitenkaan ollut, vaan käynnistettäessä ei yksikään välilehti ollut aktiivisena ja se piti valita ylävalikosta. Ongelma korjattiin muuttamalla Keskustelu-välilehden display-ominaisuus eli property aloituksessa arvoon ”block”, joka oli alun perin ”none”

(kuva 21). Nyt laajennuksen käynnistyessä Keskustelu-välilehti aukeaa aina automaattisesti ensimmäisenä. Lisäksi asetetaan ”tablinks” ja ”active” -luokat painikkeelle, jonka tunniste on ”chat”, minkä ansiosta välilehtipainike muuttuu näkymältään aktiiviseksi ilmaisten ylävalikossa käyttäjälle aktiivisena olevan välilehden.

```
//Always open chat-tab first
const chatTab = document.getElementById("tab-chatting");
const tabButton = document.getElementById("chat");
chatTab.style.display = "block";
tabButton.setAttribute("class", "tablinks active");
```

Kuva 21. Keskusteluikkunan aktivointi aloituksessa app.js-tiedostossa.

Testauksen aikana huomasimme, että noin 20 viestin jälkeen keskustelubotti meni ”jumiin”, koska token-raja tuli täyteen. Tämä johtui siitä, että käytössä oli kielimalli gpt-3.5-turbo-4k, jonka raja on 4097 tokenia. Rajaa saatiin nostettua helposti vaihtamalla index.js-tiedostoon kielimalliksi kalliimpi gpt-3.5-turbo-16k, joka sallii 16 385 tokenia, eli noin nelinkertaisen määrän verrattuna halvempaan kielimalliin. Myös tehokkaammalla kielimallilla voi tulla raja vastaan, mutta se sallii kuitenkin huomattavasti enemmän tekstiä. Rajan täyttyessä tulee palvelin käynnistää komentoriviltä uudestaan.

Selainlaajennusta kehitettiin aluksi pelkällä Chrome-selaimella, mutta koska Microsoft Edge käyttää samaa selainmoottoria, laajennus toimi moitteetta myös sillä. Halusimme kuitenkin saada sen toimimaan myös Mozilla Firefox -selaimella, ja tätä varten muokkasimme manifest.json-tiedostoa niin, että se oli lopulta yhteensopiva jokaisen kolmen selaimen kanssa. Kun laajennus avattiin Firefox-selaimella, huomasimme joitakin ulkonäöllisiä eroja Chrome-versioon verrattuna. Tämä johtui selaimien omista CSS-säännöistä, jotka aiheuttavat poikkeamia selaimien välillä. Korjasimme ongelman muokkaamalla tyylitiedostoa yhteensopivaksi kummallekin selainalustalle, ja lopulta sama laajennusversio oli ulkonäöllisesti ja toiminnallisesti yhtenäinen kaikissa kolmessa halutussa selaimessa.

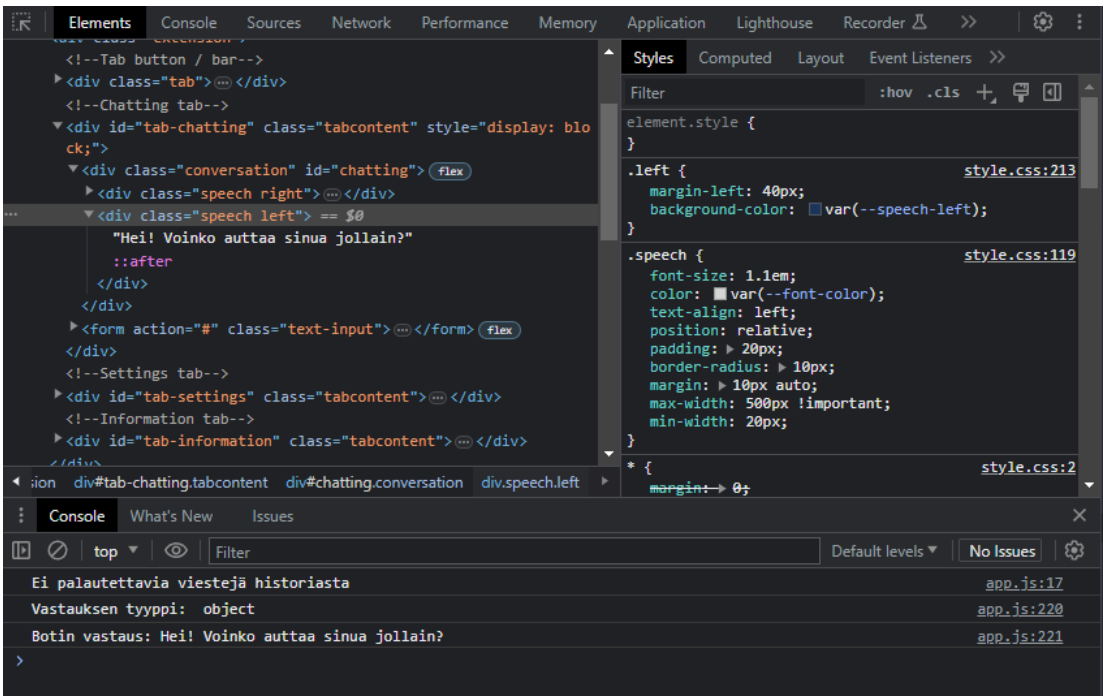
5.5 Testaus

Selainlaajennusta kehitimme alkuperäisen suunnitelman mukaisesti, ja ominaisuuksia lisäsimme yksi kerrallaan. Jokainen lisätty toiminto testattiin ja tarpeen mukaan niitä korjattiin tai paranneltiin. Esimerkiksi silloin, kun jokin uusi toiminto aiheutti koodissa virheen tai ”rikkoi” jonkin toisen osan laajennuksesta, selvitimme mistä kyseinen ongelma aiheutui, ja

kirjasimme sen ylös. Tämä auttoi kehityksessä huomattavasti, koska olimme jatkuvasti tietoisia mahdollisista ongelmista, joita saattoi ilmetä tietyn toiminnon yhteydessä.

Testausta suoritimme ohjelmointivirheiden varalta manuaalisesti kokeilemalla laajennuksen jokaista perustoimintoa sekä aina uutta lisättyä ominaisuutta. Osa virheistä olivat syntaksivirheitä, jotka aiheuttivat suoria ongelmia laajennuksen toiminnassa, kun taas semanttiset virheet johtuivat huonosta ohjelmointilogiikasta, joitten takia laajennus toimi, mutta ei toivotulla tavalla.

Debuggaus eli virheenjäljittäminen suoritettiin käymällä koodia läpi rivi riviltä. Jäljittämistä helpotti tekstieditori, joka ilmaisi syntaksivirheet välittömästi, kun niitä ilmeni. Useimmissa tapauksissa hyödynsimme console.log-metodia, jota käytimme laajennuksen toimintojen eri vaiheiden seuraamiseen sekä muuttujien arvojen ja tyyppien tarkasteluun. Nämä tulostuivat selaimen kehittäjätyökalun lokiin. Kyseisestä työkalusta pystyimme myös seuraamaan HTML-elementtien muutoksia laajennuksen käytön aikana, sekä hienosäätämään CSS-arvoja oikeanlaisen asettelun löytämiseksi. Kuvassa 22 on ylempänä vasemmalla näkyvissä elementit, oikealla valitun elementin tyylimäärittely, sekä alareunassa console.log-metodilla tulostetut tiedot viestihistorian palauttamisesta, keskustelubotin vastauksen tyypistä sekä vastauksen sisällöstä. Oikeassa alareunassa on lisäksi näkyvillä tiedosto, tässä tapauksessa app.js, ja koodirivi, jossa console.log-metodi on suoritettu.



Kuva 22. Chrome-selaimen kehittäjätyökalun näkymä.

Kaikki ohjelmointivirheet eivät tulleet ilmi normaalin testauksen aikana, vaan niitä löytyi jonkin verran satunnaisesti ”vahingossa” tiettyjen tapahtumaketjujen tuloksena. Esimerkkinä mainittakoon ”Tyhjennä keskustelu” -painikkeen virheellinen toiminta. Keskustelu tyhjentyi painiketta painamalla normaalisti, mutta mikäli tyhjentämisen jälkeen jatkettiin botin kanssa keskustelua, ja tyhjentämättä suljettiin laajennus, ilmestyivät seuraavalla avaamiskerralla keskusteluruutuun normaali keskusteluhistoria, minkä lisäksi kaikki ennen tyhjentämistä lähetetyt ja vastaanotetut viestit tulostuivat myös. Ongelman syyksi paljastui lopulta logiikkavirhe koodissa, ja samassa yhteydessä huomasimme asettaneemme viestien tallentamiseen tarpeettomat, ylimääräiset muuttujat. Korjattuamme logiikkavirheen ja karsittuamme turhat muuttujat, oli ohjelmakoodi siistimpi ja tyhjennyspainike toimi vihdoin halutulla tavalla.

Toinen mainitsemisen arvoinen, vahingossa löytynyt logiikkavirhe aiheutti tietyn käyttäjän virheen seurauksena koko laajennuksen kaatumisen. Ongelman aikaansaaminen edellytti laajennuksen virheellisesti avaamisen ilman, että paikallinen palvelin käynnistetään. Tällöin, mikäli käyttäjä painoi lähetyspainiketta ja oli lähettävinään keskustelubotille viestin, se ei lähtenyt eteenpäin, vaan vastaustoiminto jäi pyörittämään pelkkää lataustekstiä, koska data ei liikkunut laajennuksen ja rajapinnan välillä. Mikäli tämän jälkeen laajennus suljettiin tyhjentämättä keskusteluhistoriaa, se meni seuraavalla käynnistyskerralla täysin ”jumiin”. Syynä tähän oli viestihistorian palauttamisen tarkistusfunktio, joka aina laajennuksen avaamisen yhteydessä tarkisti, löytyykö paikallisesta muistista lähetettyjä viestejä. Koska yksi viesti oli lähetetty, mutta mitään ei ollut vastaanotettu, oli vastaustaulukko tyhjä ja tämä aiheutti sovelluksen kaatavan virheen, jolloin mikään ei toiminut. Ongelmaan löytyi helppo ratkaisu, joka oli muuttaa tarkistusfunktion tarkistettava taulukko lähetettyjen viestien sijaan vastaanotettujen viestien taulukoksi. Nyt tarkistus ei siis huomioi, onko paikallisessa muistissa lähetettyjä viestejä, vaan sen sijaan vastaanotettujen viestien olemassaolon, mikä estää saman ongelman toistumisen, koska laajennuksen toimiessa oikein jokaista lähetettyä viestiä kohden on aina yksi vastaus.

Kun kaikki löytämämme virheet ja ”bugit” oli korjattu ja laajennus toimi kuten piti, annoimme sen kolmen ulkopuolisen käyttäjän testattavaksi sekä Chrome- että Firefox-selaimilla. Tässä vaiheessa uusia ongelmia ei tullut enää ilmi, eikä käyttäjäpalautteen perusteella koehenkilöillä ollut ongelmia laajennuksen toimintojen ymmärtämisessä.

6 Yhteenveto ja jatkokehitys

6.1 Yhteenveto

Projektin tavoitteena oli luoda toimiva selainlaajennus ja avata sen toimintoja opinnäyteraportin lukijalle mahdollisimman tarkasti ja selkeästi. Halusimme saada laajennuksen toimimaan kolmella suosituimmalla työpöytäselaimella, missä lopulta onnistuimme oletettua paremmin, koska sama kehitysversio toimii jokaisessa näistä selaimista. Myös kaikki laajennukseen alun perin kaavaillut lisäominaisuudet saatiin toteutettua suunnitelman mukaisesti. Näitä olivat rooli keskustelubotille, käyttöliittymän väriteeman vaihto sekä mahdollisuus keskusteluhistorian tyhjentämiseen. Lisäksi keksimme käyttää selaimen paikallista muistia, jossa valitut asetukset pysyvät voimassa myös selaimen sulkeuduttua. Iteratiivisen kehitystyön ansiosta saimme myös korjattua kaikki löytämämme virheet ja ongelmat, joita prosessin aikana ilmeni.

Alun perin ideana oli kehittää toimiva prototyyppi, minkä jälkeen varsinaisen laajennuksen ohjelmointi oli tarkoitus aloittaa puhtaalta pöydältä. Olimme kuitenkin käyttäneet prototyypin toteuttamiseen niin paljon aikaa, että päätimme lopulta rakentaa kokonaisuuden sen päälle. Siihen mennessä kirjoitettu koodi oli valitettavasti hyvin sekavaa ja esimerkiksi muuttujien ja elementtien nimet eivät kuvanneet lainkaan sen toimintaa. Rakennetta olisi siis pitänyt miettiä tarkemmin alusta alkaen, koska jouduimme myöhemmin käyttämään paljon aikaa koodin läpikäymiseen ja siistimiseen. Lisäksi optimointi ja tehokkaampien toimintatapojen etsintä voisi parantaa ohjelmakoodin luettavuutta ja tehokkuutta, sillä vaikka laajennus palvelee tarkoitustaan, joillekin sen kehitysvaiheessa esiin tulleille haasteille saattaa löytyä parempia ratkaisumalleja. Näiden suhteen olemme kuitenkin pyrkineet projektin aikana toimimaan niin laadukkaasti, kuin omat taitomme sallivat.

6.2 Jatkokehitysideat

Vaikka projekti on käyttöliittymän ja ominaisuuksien puolesta valmis ja sille asetetut tavoitteet on saavutettu, saimme kehityksen aikana paljon ideoita myös mahdollisista lisätoiminnallisuuksista ja parannuksista. Yksi mielenkiintoisimmista ideoista oli, että laajennusikkunaan lisättäisiin uusi välilehti, joka sisältäisi kuvageneraattorin hyödyntäen OpenAI:n DALL-E 2 -palvelua. Näin käyttäjä voisi laajennuksen avulla tuottaa myös kuvia antaen tekoälylle sanoja, joiden pohjalta grafiikkaa piirretään. Tähän liitettäisiin mahdollisuus ladata generoidut kuvat talteen omalle tietokoneelle. Myös keskustelun tallentamiseen voisi lisätä

erillisen painikkeen, jolla botin kanssa käyty keskustelu olisi mahdollista tallentaa esimerkiksi tekstitiedostona käyttäjän laitteelle. Tämänkaltaisten lisäominaisuuksien avulla laajennus voisi lisätä tuottavuutta entisestään, koska sen käytöstä olisi mahdollista saada jotakin konkreettista sisältöä talteen myös itse laajennuksen ulkopuolelle.

Käyttökokemusta olisi mahdollista parantaa tekemällä käyttöliittymään pieniä lisäyksiä, jotka tekisivät selainlaajennuksen käytöstä mukavampaa. Nykyinen versio ei osaa muotoilla esimerkiksi keskustelubotin luomia listoja tai koodinpätkiä erikseen, vaan kaikki viestit tulevat näkyviin yhtenäisenä tekstinä ilman kappalejakoja tai rivien erottelua. Tämä olisi korjattavissa käsittelemällä rajapinnan palauttama merkkijono ennen sen tulostamista, mutta emme selvitystyöstä huolimatta rajallisen ajan takia keksineet tähän ratkaisua. Toinen käytettävyyttä parantava ominaisuus voisi olla puhekupliin lisättävä painike, jolla kyseisen puhekuplan sisältämän viestin voisi kopioida helposti leikepöydälle. Tällöin viestejä voisi helpolla ja nopealla tavalla liittää esimerkiksi johonkin ulkopuoliseen palveluun, kuten foorumeille, sähköposteihin tai viestisovelluksiin.

Asetukset-välilehdellä keskustelubotille asetetut roolit eivät tällä hetkellä tallennu muistiin, pois lukien kyseisellä hetkellä käytössä oleva rooli. Kokemusta voitaisiin parantaa tallentamalla roolit muistiin ja lisäämällä aiemmat roolit pudotusvalikkoon roolinvalintakenttään. Näin käyttäjän ei tarvitsisi kirjoittaa aiemmin käytettyjä rooleja uudelleen. Käyttöliittymä on tällä hetkellä ainoastaan suomen kielellä, mutta tekemällä koodiin pieniä muutoksia, olisi mahdollista lisätä käyttökieleksi myös muita kieliä. Tämä tarjoaisi sujuvan käyttökokemuksen myös muille, kuin suomenkielisille käyttäjille. Käyttökielen valinnan voisi toteuttaa esimerkiksi laajennusikkunan oikeaan yläreunaan lisättävällä pudotusvalikolla.

Selainlaajennus olisi mahdollista julkaista selaimien laajennuskaupoissa, kuten Chromen Web Store tai Mozillan Firefox Add-Ons. Se kuitenkin käyttää luomaamme henkilökohtaista API-avainta, joten julkaisu ei nykyisellään olisi missään määrin järkevää rajanpinnan maksullisuuden takia. Ongelma olisi korjattavissa antamalla käyttäjälle mahdollisuus lisätä oma API-avain, tai laajennukseen voisi integroida kirjautumismahdollisuuden OpenAI-tunnuksella, mutta se vaatisi rakenteellisesti isoja muutoksia ohjelmakoodissa. Lisäksi paikallisen palvelimen sijasta tarvittaisiin palveluntarjoaja, joka tarjoaisi verkkoisännöinti-, eli web hosting -palvelua. Potentiaalisia vaihtoehtoja tähän voisi olla esimerkiksi Heroku, Netlify tai Microsoftin Azure. Tämänkaltaiset palvelut ovat kuitenkin useimmiten myös itsessään maksullisia, mutta niiden piiristä löytyy joitakin ilmaisia, joskin rajoitettuja vaihtoehtoja.

Vaikka laajennus ei sellaisenaan ole toistaiseksi julkaisukelpoinen, sen yksinkertaisen rakenteen ansiosta sitä olisi mahdollista jalostaa moniin, myös kaupallisiin käyttötarkoituksiin. Sen voisi esimerkiksi helposti muokata toimimaan laajennusominaisuuden sijasta osana verkkosivustoa, mikä poistaisi vaatimuksen asentaa sitä erikseen laitteelle, koska kyseessä ei silloin olisi enää selainlaajennus. Näin se voitaisiin räätälöidä ulkonäöllisesti käytettävään sivustoon soveltuvaksi osana muuta kokonaisuutta. Syötteitä on myös mahdollista tietyissä määrin rajata esimerkiksi käsittelemään haluttuja aihealueita, kuten ruokareseptejä, mutta tämän osalta kuitenkaan GPT:n 3.5-versio ei ole tällä hetkellä täydellisen luotettava.

Laajennuksen kehitysvaiheessa ei tietoturvan kannalta olla otettu huomioon Info-välilehden varoitusta lukuun ottamatta mahdollisia puutteita. Se ei varsinaisesti ollut toiminnan kannalta oleellista, koska alkuperäinen käyttötarkoitus huomioiden ei laajennuksen oikeanlainen käyttö aiheuta tietoturvariskejä. Kaupallisessa käytössä kuitenkin tulisi kiinnittää tähän osa-alueeseen erityistä huomiota, sillä tietotekniikan saralla väärinkäyttömahdollisuuksia on monia. Tällöin olisikin tapauskohtaisesti mahdollisten käyttöönottajien vastuulla tehdä lopullinen päätös keskustelubotin hyödyntämisestä verkkosivuillaan, ja tietoturvaa parannettaisiin tarpeen mukaan.

Lähteet

Athaluri S., Manthena S., Kesapragada, K., Yarlagadda, V., Dave, T. & Duddumpudi, R. 2023. Exploring the Boundaries of Reality: Investigating the Phenomenon of Artificial Intelligence Hallucination in Scientific Writing Through ChatGPT References. Cureus. Viitattu 18.10.2023. Saatavissa https://assets.cureus.com/uploads/original_article/pdf/148687/20230511-14808-1wokz88.pdf

Baker, T. & Courtright, A. 2022. Internet Explorer 11 desktop application ended support for certain operating systems. Microsoft. Viitattu 28.9.2023. Saatavissa https://learn.microsoft.com/en-us/lifecycle/announcements/internet-explorer-11-end-of-support

Brave Software Inc. What are browser extensions, and are they safe? 2023. Viitattu 2.10.2023. Saatavissa https://brave.com/learn/what-are-web-browser-extensions/

Brockman, G. 2019. Microsoft invests in and partners with OpenAI to support us building beneficial AGI. Viitattu 5.10.2023. Saatavissa https://openai.com/blog/microsoft-invests-in-and-partners-with-openai

Brockman, G. & Sutskever, I. 2019. OpenAI LP. Viitattu 5.10.2023 Saatavissa https://openai.com/blog/openai-lp

CERN. A short history of the Web. Viitattu 25.9.2023. Saatavissa https://www.home.cern/science/computing/birth-web/short-history-web

Chrome for Developers. 2023. Manifest file format. Viitattu 2.10.2023. Saatavissa https://developer.chrome.com/docs/extensions/mv3/manifest/

Dupont, E., Fehr, T. & Maniar, T. 2023. Usein kysytyt kysymykset Copilotin tietosuojasta ja yksityisyydestä Dynamics 365:ssä. Viitattu 3.10.2023. Saatavissa https://learn.microsoft.com/fi-fi/dynamics365/faqs-copilot-data-security-privacy

Fedewa, J. 2021. What Is a Browser Extension? How To Geek. Viitattu 2.10.2023. Saatavissa https://www.howtogeek.com/718676/what-is-a-browser-extension/

Figma. About. Viitattu 11.10.2023. Saatavissa https://www.figma.com/about/

Hoffmann, J. 2017. The History of the Browser Wars: When Netscape Met Microsoft. The History of The Web. Viitattu 26.9.2023. Saatavissa https://thehistoryoftheweb.com/browser-wars/

Hoffmann, J. Timeline. The History of the Web. Viitattu 26.9.2023. Saatavissa https://thehistoryoftheweb.com/timeline/?date_from=all

Hu, K. 2023. ChatGPT sets record for fastest-growing user base - analyst note. Reuters. Viitattu 5.10.2023. Saatavissa https://www.reuters.com/technology/chatgpt-sets-record-fastest-growing-user-base-analyst-note-2023-02-01/

Ibiblio. The Story of the Netscape Browser. Viitattu 26.9.2023. Saatavissa http://ibiblio.org/team/history/evolution/browser.html

Kerner, S. 2023. What are large language models (LLMs)? Viitattu 3.10.2023. Saatavissa https://www.techtarget.com/whatis/definition/large-language-model-LLM

Lock, S. 2022. What is AI chatbot phenomenon ChatGPT and could it replace humans? The Guardian. Viitattu 5.10.2023. Saatavissa https://www.theguardian.com/technology/2022/dec/05/what-is-ai-chatbot-phenomenon-chatgpt-and-could-it-replace-humans

Malwarebytes. What is ChatGPT? Viitattu 6.10.2023. Saatavissa https://www.malwarebytes.com/cybersecurity/basics/chatgpt-ai-security

MDN Web Docs. 2023a. JavaScript basics. Viitattu 11.10.2023. Saatavissa https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/JavaScript_basics

MDN Web Docs. 2023b. Anatomy of an extension. Viitattu 10.10.2023. Saatavissa https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/Anatomy_of_a_WebExtension

MDN Web Docs. 2023c. Web technology for developers. Viitattu 12.10.2023. Saatavissa https://developer.mozilla.org/en-US/docs/Web

MDN Web Docs. 2023d. <style>: The Style Information element. Viitattu 13.10.2023. Saatavissa https://developer.mozilla.org/en-US/docs/Web/HTML/Element/style

National Cyber Security Centre. 2021. Managing web browser security. Viitattu 29.9.2023. Saatavissa https://www.ncsc.gov.uk/collection/device-security-guidance/policies-and-settings/managing-web-browser-security

NPM. 2018. cors. Viitattu 12.10.2023. Saatavissa https://www.npmjs.com/package/cors

NPM. 2022. express. Viitattu 12.10.2023. Saatavissa https://www.npmjs.com/package/express

NPM. 2023a. OpenAI Node API Library. Viitattu 12.10.2023. Saatavissa https://www.npmjs.com/package/openai

NPM. 2023b. body-parser. Viitattu 12.10.2023. Saatavissa https://www.npmjs.com/package/body-parser

NVIDIA Corporation. Large Language Models Explained. Viitattu 3.10.2023 Saatavissa https://www.nvidia.com/en-us/glossary/data-science/large-language-models/

OpenAI. 2022. Introducing ChatGPT. Viitattu 5.10.2023. Saatavissa https://openai.com/blog/chatgpt

OpenAI. 2023a. OpenAI and Microsoft extend partnership. Viitattu 5.10.2023. Saatavissa https://openai.com/blog/openai-and-microsoft-extend-partnership

OpenAI. 2023b. Introducing ChatGPT Plus. Viitattu 6.10.2023. Saatavissa https://openai.com/blog/chatgpt-plus

OpenAI. 2023c. GPT-4. Viitattu 6.10.2023. Saatavissa https://openai.com/research/gpt-4

OpenAI a. Pricing. Viitattu 4.10.2023. Saatavissa https://openai.com/pricing

OpenAI b. Tokenizer. Viitattu 4.10.2023. Saatavissa https://platform.openai.com/tokenizer

Priest, M. 2023. What are large language models and how do they work? Boost.ai. Viitattu 3.10.2023. Saatavissa https://www.boost.ai/blog/llms-large-language-models

Raghavan, R. 2019. The History of Browser Wars. Viitattu 28.9.2023. Saatavissa https://acodez.in/browser-wars/

Ray, P. 2023. ChatGPT: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope. Sikkim University. Viitattu

19.10.2023. Saatavissa https://www.sciencedirect.com/science/article/pii/S266734522300024X

Statcounter. 2023a. Browser Market Share Worldwide. Viitattu 27.9.2023. Saatavissa https://gs.statcounter.com/browser-market-share#monthly-200901-201212

Statcounter. 2023b. Desktop Browser Market Share Finland. Viitattu 27.9.2023. Saatavissa https://gs.statcounter.com/browser-market-share/desktop/finland/#monthly-202201-202212-bar

Tang, R., Chuang Y. & Hu, X. 2023. The Science of Detecting LLM-Generated Texts. Department of Computer Science, Rice University. Viitattu 19.10.2023. Saatavissa https://arxiv.org/pdf/2303.07205.pdf

Witt, T. 2023. How To Understand, Manage Token-Based Pricing of Generative AI Large Language Models. Acceleration Economy. Viitattu 4.10.2023. Saatavissa https://accelerationeconomy.com/ai/how-to-understand-manage-token-based-pricing-of-generative-ai-large-language-model-costs/

Liite 1. Linkki projektin GitHub-sivulle

https://github.com/521311840/AiAssistant