Tampereen ammattikorkeakoulu

Gophertype

Tekstipohjaisen käyttöliittymän toteuttaminen Go:lla

Oskari Lindroos

OPINNÄYTETYÖ
Toukokuu 2025

Tietotekniikan tutkinto-ohjelma
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma
Ohjelmistotekniikka

LINDROOS, OSKARI:

Gophertype
Tekstipohjaisen käyttöliittymän toteuttaminen Go:lla

Opinnäytetyö 27 sivua, joista liitteitä 1 sivu
Toukokuu 2025

Näppäimistöllä kirjoittaminen on taito, joka voi parantaa tuottavuutta ja tehokkuutta, erityisesti henkilöillä, jotka kirjoittavat tietokoneella paljon päivittäin. Opinnäytetyössä kehitettiin Gophertype-sovellus, joka tarjoaa mahdollisimman yksinkertaisen ja häiriöttömän ympäristön, jossa näppäimistöllä kirjoittamista voi harjoitella.

Sovellus toteutettiin tekstipohjaisena käyttöliittymänä, sillä se mahdollistaa sovelluksen ajon suoraan tietokoneen komentoriviltä ilman ulkoisia riippuvuuksia, kuten selainta, internetyhteyttä tai graafista ympäristöä. Sovellus kehitettiin Go-ohjelmointikielellä käyttäen Bubble Tea -ohjelmistokehystä. Kirjoittamisen harjoittelu perustuu kirjoitustestiin, jossa käyttäjän tulee kirjoittaa satunnaisia sanoja mahdollisimman nopeasti ja tarkasti. Testin lopussa sovellus kertoo käyttäjän kirjoitusnopeuden ja tarkkuuden.

Työn lopputuloksena saatiin ensimmäinen versio sovelluksesta, johon toteutettiin sovelluksen tärkeimmät toiminnot, eli kirjoitusnopeuden ja tarkkuuden mittaus. Sovelluksessa on kuitenkin paljon mahdollisuuksia jatkokehitykselle. Kehityskohteita ovat muun muassa numeroiden, symbolien ja muiden kuin englannin kielen sanojen harjoittelu. Projektiin valitut teknologiat osoittautuivat erinomaisiksi valinnoiksi.

Asiasanat: go, golang, gophertype, kirjoitustesti, sovellus, tekstipohjainen käyttöliittymä

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in ICT Engineering
Software Engineering

LINDROOS, OSKARI:
Gophertype
Implementation of a Text-Based User Interface in Go

Bachelor's thesis 27 pages, appendices 1 pages
May 2025

Typing on a keyboard is a skill that can improve productivity and efficiency, especially for individuals who type a lot on a computer daily. In this thesis, the Gophertype application was developed to provide a simple and distraction-free environment for practicing keyboard typing.

The application was implemented as a Text-Based User Interface, allowing it to run directly on a computer's terminal without external dependencies such as a browser, internet connection, or graphical environment. The application was developed using the Go programming language and the Bubble Tea framework. The typing practice is based on a typing test where the user must type random words as quickly and accurately as possible. At the end of the test, the application reports the user's typing speed and accuracy.

This thesis resulted in the development of the first version of the application which implements the core functionality of measuring the user's typing speed and accuracy. The application offers numerous opportunities for further development. The areas for improvement include the ability to practice typing numbers, symbols, and words in languages other than English. The technologies chosen for the project proved to be excellent choices, as they make the application fast and easily extendable.

Key words: go, golang, gophertype, software, text-based user interface, tui, typing test

SISÄLLYS

ERITYISSANASTO

Bubble Tea	Go-kielen ohjelmistokehys tekstipohjaisten käyttöliittymien kehittämiseen.
Git	Versionhallintatyökalu.
CLI	Command Line Interface, komentorivikäyttöliittymä.
GUI	Graphical User Interface, graafinen käyttöliittymä.
CSS	Cascading Style Sheets. Tyylitiedostokieli, jolla määritellään verkkosivujen ulkoasu.
Lip Gloss	Go-kielen ohjelmistokirjasto, jota käytetään tekstipohjaisten käyttöliittymien tyylittelyyn.
skripti	Lyhyt ohjelmakoodi, jolla voidaan automatisoida erilaisia tehtäviä.
SSH	Secure Shell. Protokolla, jonka avulla voidaan muodostaa salattu etäyhteys toiseen tietokoneeseen.
TUI	Text-Based User Interface, tekstipohjainen käyttöliittymä.
WPM	Words Per Minute. Yksikkö, jolla mitataan kirjoitusnopeutta.

1 JOHDANTO

Opinnäytetyön tavoitteena on kehittää Gophertype-sovellus. Sovellus mahdollistaa näppäimistöllä kirjoittamisen harjoittelun suoraan tietokoneen komentorivillä, ilman graafista käyttöliittymää. Sovellus tarjoaa käyttäjille häiriöttömän ja minimaalisen ympäristön, joka on ihanteellinen kirjoittamisen harjoitteluun.

Kirjoittamisen harjoittelu tapahtuu kirjoitustestin avulla, jossa käyttäjän on kirjoitettava satunnaisia sanoja mahdollisimman nopeasti ja tarkasti. Testin lopussa sovellus ilmoittaa kirjoitusnopeuden WPM (words per minute) -yksikössä ja tarkkuuden prosentteina.

Sovellus toteutetaan Go-ohjelmointikielellä, ja Bubble Tea -ohjelmistokehyksellä. Bubble Tea mahdollistaa nykyaikaisen tekstipohjaisen käyttöliittymän luomisen, joka tekee sovelluksesta käyttäjäystävällisen ja helposti laajennettavan. Sovelluksen kehittämisessä käytetään myös Lip Gloss -ohjelmistokirjastoa, joka mahdollistaa tekstin muotoilun ja värityksen komentorivillä. Lip Glossin avulla sovelluksen käyttöliittymästä saadaan visuaalisesti miellyttävämpi.

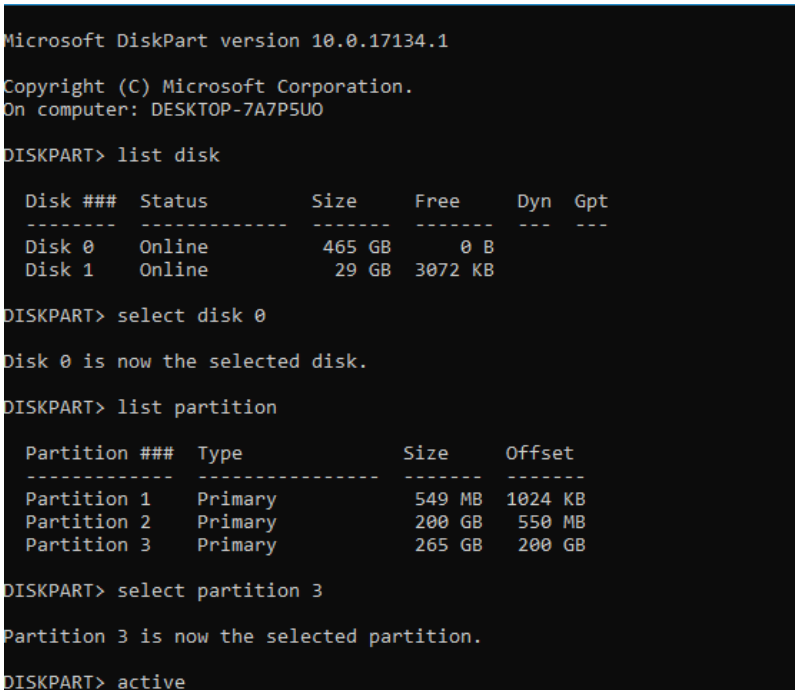
2 TAUSTA

2.1 Tekstipohjaiset käyttöliittymät

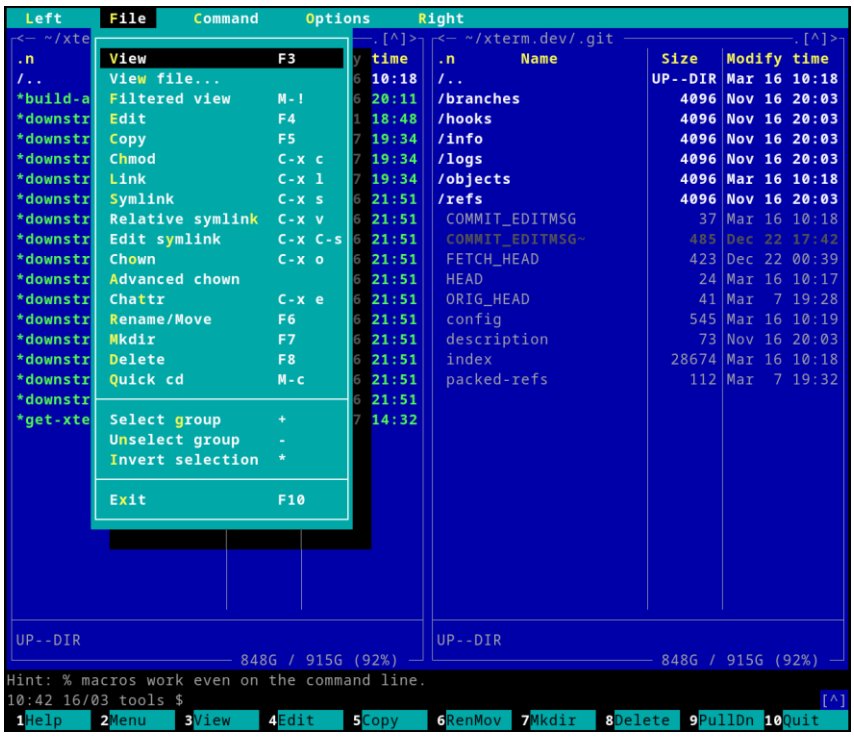
2.1.1 Määritelmä

Tekstipohjaiset käyttöliittymät (engl. TUI, Text-based User Interface) ovat käyttöliittymiä, jotka käyttävät pelkkää tekstiä visualisoidakseen käyttöliittymäkomponentteja. Tekstipohjaisia käyttöliittymäsovelluksia ajetaan suoraan tietokoneen komentoriviltä. Ne olivat ensimmäisiä käyttöliittymiä, joita tietokoneet käyttivät, ennen kuin graafiset käyttöliittymät yleistyivät.

Ero komentorivisovellusten (engl. CLI, Command Line Interface) ja TUI-sovellusten välillä on se, että CLI-sovelluksissa käyttäjä kirjoittaa itse komentoja komentoriville, kun taas TUI-sovelluksissa käyttäjä voi käyttää näppäimistöä navigoidakseen sovelluksessa ja suorittaakseen toimintoja. TUI-sovellukset siis jäljittelevät graafisten käyttöliittymien toimintaa. Kuvissa 1 ja 2 on esitetty esimerkit CLI- ja TUI-sovelluksista.



KUVA 1. Kuvankaappaus CLI-sovelluksesta (Diskpart).



KUVA 2. Kuvankaapaus perinteisestä TUI-sovelluksesta (Midnight Commander).

Nykypäivänä TUI:t ovat suurimmaksi osaksi korvattu graafisilla käyttöliittymillä (engl. GUI, Graphical User Interface). Graafiset käyttöliittymät tarjoavat peruskäyttäjille helpomman tavan käyttää sovelluksia, sillä ne mahdollistavat hiiren käytön näppäimistön lisäksi. Lisäksi graafisten sovellusten asennus ja käyttö on peruskäyttäjille helpompaa.

On kuitenkin olemassa joukko tehokäyttäjiä, jotka edelleen suosivat TUI-sovelluksia graafisten sijaan. Tällaiset käyttäjät ovat yleensä kokeneita tietokoneen käyttäjiä, jotka käyttävät TUI-sovelluksia, koska ne tarjoavat nopeamman ja tehokkaamman tavan suorittaa tehtäviä verrattuna graafisiin käyttöliittymiin. TUI-sovellukset ovat erityisen suosittuja ohjelmoijien, järjestelmänvalvojien ja muiden teknisten käyttäjien keskuudessa.

Syitä käyttää tekstipohjaisia käyttöliittymiä graafisten sijaan ovat muun muassa:

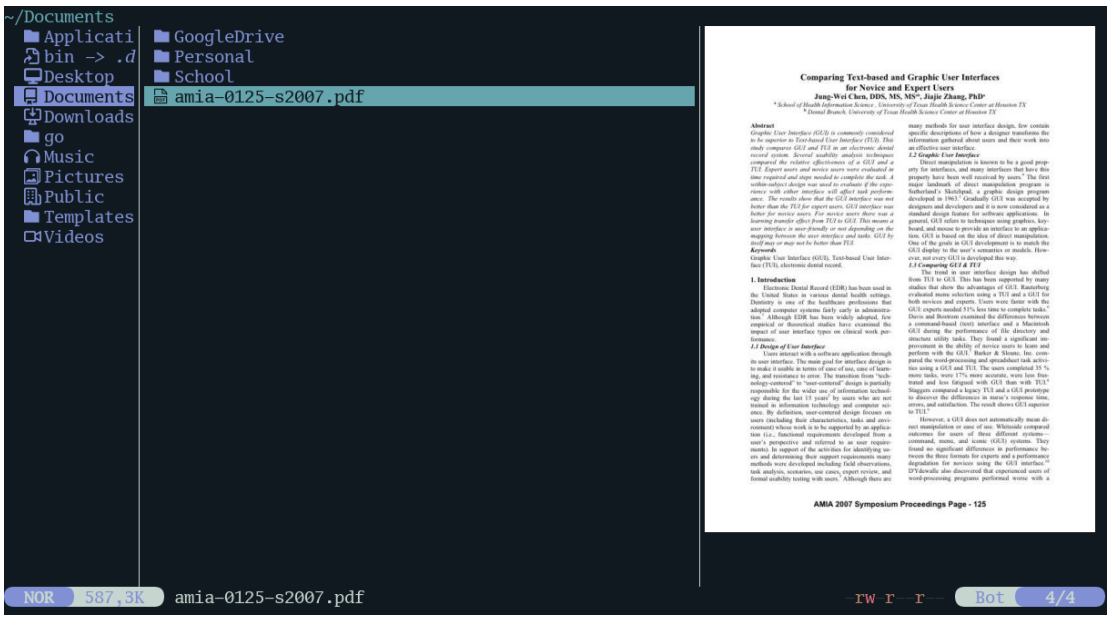
- Nopeus.
    - TUI-sovellukset eivät yleensä sisällä turhia animaatioita, jotka hidastavat siirtymiä näkymistä toisiin.
- Ergonomisuus.

    - Koska TUI-sovellukset ovat suunniteltu käytettäväksi pelkällä näppäimistöllä, käyttäjän ei tarvitse siirtää käsiään toistuvasti hiiren ja näppäimistön välillä, mikä saattaa aiheuttaa ergonomisia ongelmia ja hidastaa työskentelyä.
- Resurssien käyttö.
    - TUI-sovellukset kuluttavat yleensä vähemmän järjestelmän resursseja kuin GUI-sovellukset, mikä on hyödyllistä vanhemmilla tai vähemmän tehokkailla laitteilla.
- Etäkäyttö.
    - Koska TUI-sovellukset ajetaan tietokoneen komentorivillä, niitä voidaan käyttää etänä toiselta tietokoneelta helposti SSH-yhteyden avulla.
- Automatisointi.
    - TUI-sovelluksia on helpompi automatisoida skriptien avulla, sillä niiden komennot voidaan usein suorittaa suoraan komentoriviltä ilman graafisen käyttöliittymän väliintuloa.

2.1.2 Modernit tekstipohjaiset käyttöliittymät

Nykyään monet TUI-sovellukset tarjoavat graafisten käyttöliittymien kaltaisia ominaisuuksia, kuten värejä, animaatioita, hiiren käyttöä ja monimutkaisempia käyttöliittymäkomponentteja. Tällaisia ominaisuuksia sisältäviä tekstipohjaisia sovelluksia voidaan kutsua moderneiksi TUI-sovelluksiksi. TUI-sovellusten kehitystä on edistänyt erityisesti terminaaliemulaattoreiden kehittyminen, sillä ne tukevat vuosi vuodelta yhä monipuolisempia käyttöliittymäelementtejä – kuten kuvien ja PDF-tiedostojen näyttämistä suoraan komentorivillä. Monet modernit TUI-sovellukset hyödyntävät myös responsiivista suunnittelua, jonka avulla käyttöliittymäkomponentit skaalautuvat eri kokoisilla näytöillä ja ikkunoilla.

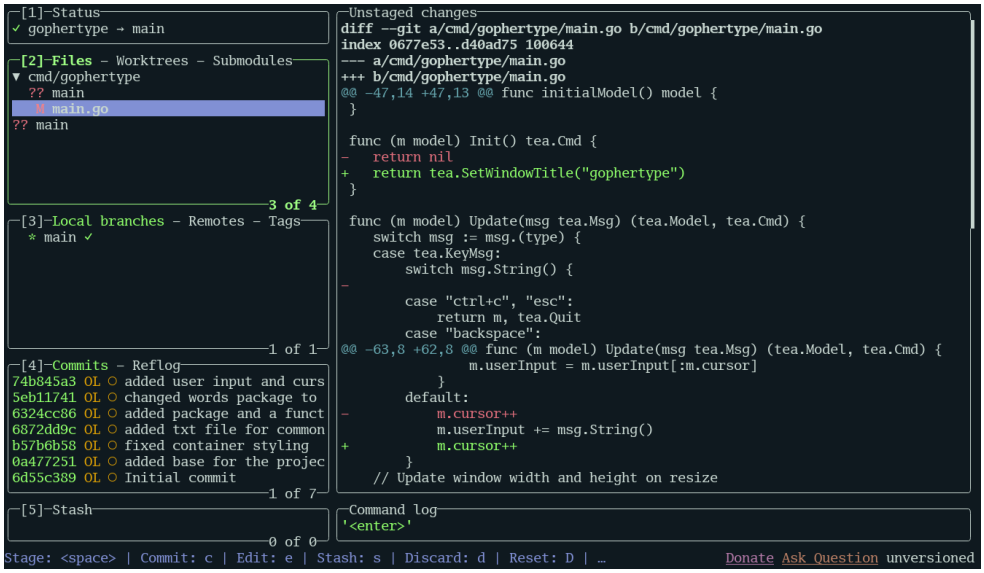
Yksi esimerkki suositusta modernista TUI-sovelluksesta on Yazi, joka on Rust-kielellä toteutettu tiedostonhallintaohjelma. Se hyödyntää edistyneempiä terminaaliominaisuuksia, kuten kuvien ja PDF-tiedostojen esikatselua (kuva 4).



KUVA 3. Moderni tekstipohjainen tiedostonhallintaohjelma (Yazi).

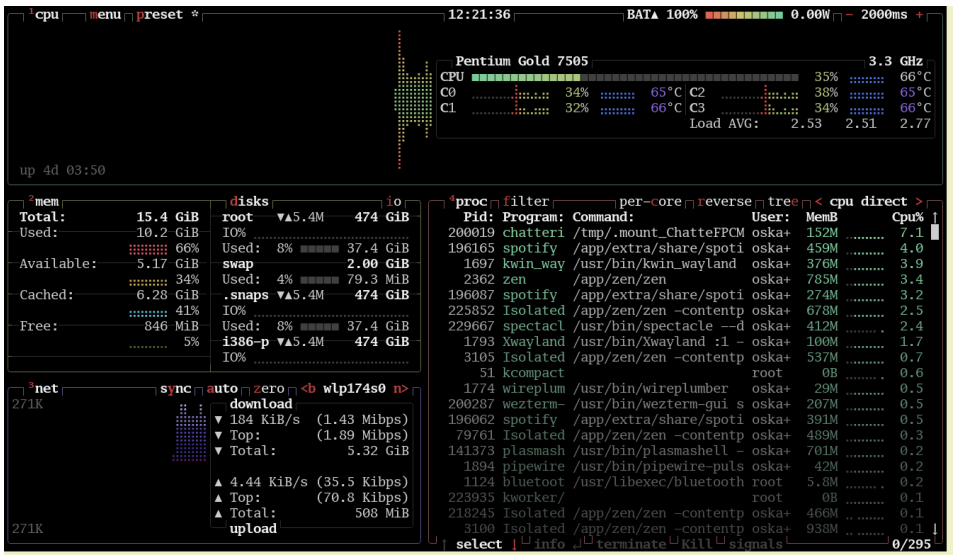
Muita suosittuja moderneja TUI-sovelluksia ovat muun muassa:

- Lazygit.
    - Git-työkalu, joka tarjoaa yksinkertaisen ja helpon tavan suorittaa Git-komentoja.



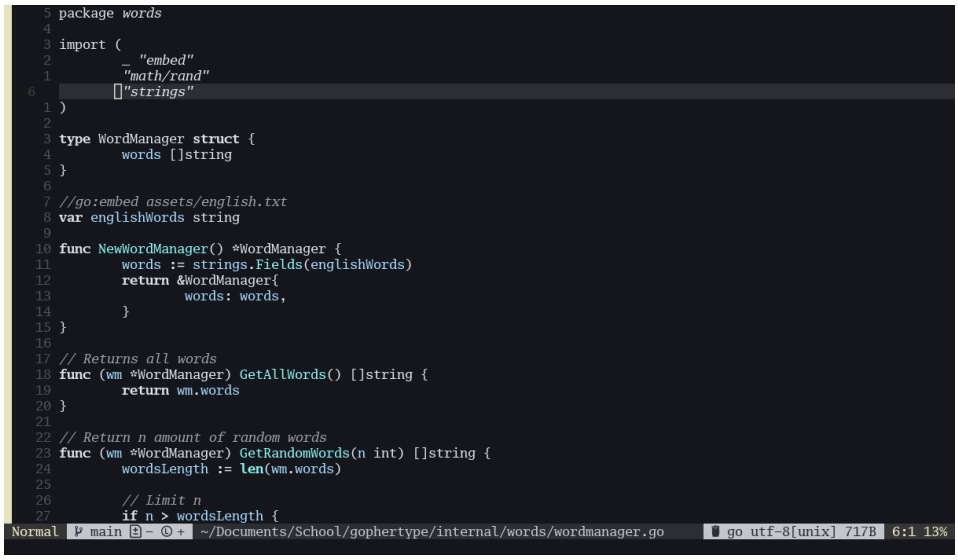
KUVA 4. Kuvankaappaus Lazygit-sovelluksesta.

- Htop
    - Prosessinhallintaohjelma, joka näyttää järjestelmän prosessit ja resurssit graafisessa muodossa.



KUVA 5. Kuvankaappaus Htop-sovelluksesta.

- Neovim
  - Laajennettu versio Vim-editorista, joka on komentorivillä toimiva tekstieditori.



KUVA 6. Kuvankaappaus Neovim-sovelluksesta.

2.2 Go-ohjelmointikieli

Go-ohjelmointikieli (tunnetaan myös nimellä Golang) sai alkunsa Googlen toimistossa vuonna 2007, jolloin siellä työskentelevät Robert Griesemer, Rob Pike ja Ken Thompson alkoivat hahmotella uuden ohjelmointikielen tarpeita valkotaululle. Heidän turhautumisensa olemassa oleviin ohjelmointikieliin johti ideaan uudesta kielestä, joka yhdistäisi tehokkaan kääntämisen, tehokkaan suorituksen ja ohjelmoinnin helppouden – ominaisuudet, joita ei löytynyt yhdestäkään saatavilla olevasta kielestä. Vuonna 2009 Google julkaisi Go-ohjelmointikielen avoimena lähdekoodina, jonka jälkeen sen suosio kehittäjien keskuudessa kasvoi nopeasti. (Go FAQ n.d.)

Go-kielen keskeisiä suunnitteluperiaatteita ovat sen yksinkertaisuus ja tehokkuus. Yksinkertaisuus näkyy Go-kielen syntaksissa, johon on otettu vaikutteita C-kielestä, mutta se on yksinkertaisempi ja helpompi oppia. Tehokkuus puolestaan tarkoittaa, että Go-kieli on suunniteltu erityisesti suurten ja monimutkaisten ohjelmistojen kehittämiseen. Go-kieli on myös suunniteltu erityisesti rinnakkaiseen ohjelmointiin, mikä tekee siitä erinomaisen valinnan esimerkiksi paljon laskentaa vaativiin sovelluksiin ja verkkopalveluihin.

Go-kielen keskeisiä ominaisuuksia ovat:

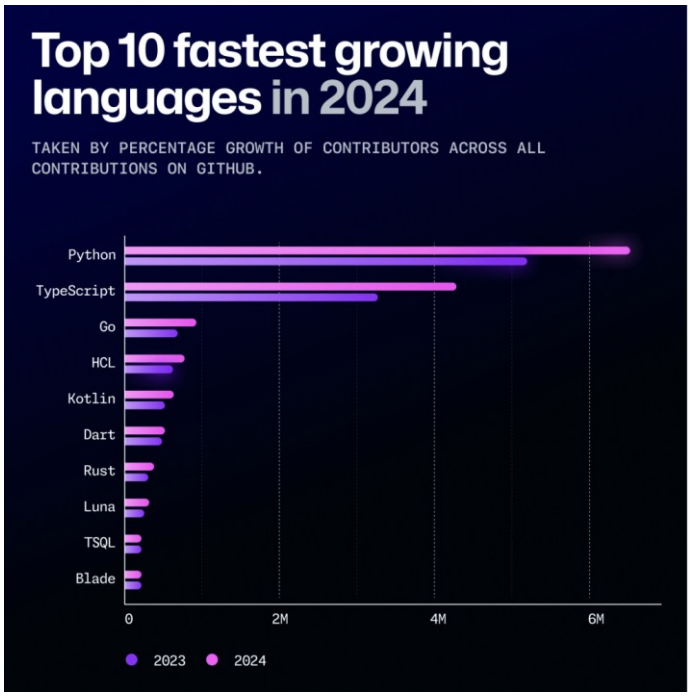
- Yksinkertainen syntaksi.
- Hyvä suorituskyky ja nopea käännös.
- Kiinteä tyyppijärjestelmä (muuttujien tyypit määritellään koodissa).
- Rinnakkaisuus ja monisäikeisyys Go-rutiinien ja kanavien avulla.
- Laaja standardikirjasto, joka sisältää monia valmiita toimintoja ja työkaluja.
- Gofmt-työkalun avulla koodin muotoilu on helppoa ja johdonmukaista.
- Helppo pakettiriippuvuuksien hallinta 'go mod' -komennolla.

Go-kielessä on myös monia muita ominaisuuksia, kuten automaattinen roskienkeruu, joka helpottaa muistinhallintaa, ja erinomainen virheiden käsittely, joka auttaa kehittäjiä löytämään ja korjaamaan virheitä koodissa. Go-kielessä ei ole perinteisiä luokkia, vaan se käyttää C-kielen tapaisesti rakenteita (structs) ja rajapintoja (interfaces) tietorakenteiden määrittämiseen.

Go-kieli valittiin tämän projektin ohjelmointikieleksi erityisesti sen vahvuuksien vuoksi TUI-sovellusten kehittämisessä. Go-kielellä on saatavilla useita kirjastoja ja ohjelmistokehyksiä, jotka helpottavat ja nopeuttavat TUI-sovellusten kehittämistä.

2.2.1 Go-kielen kasvu ja kehitysnäkymät

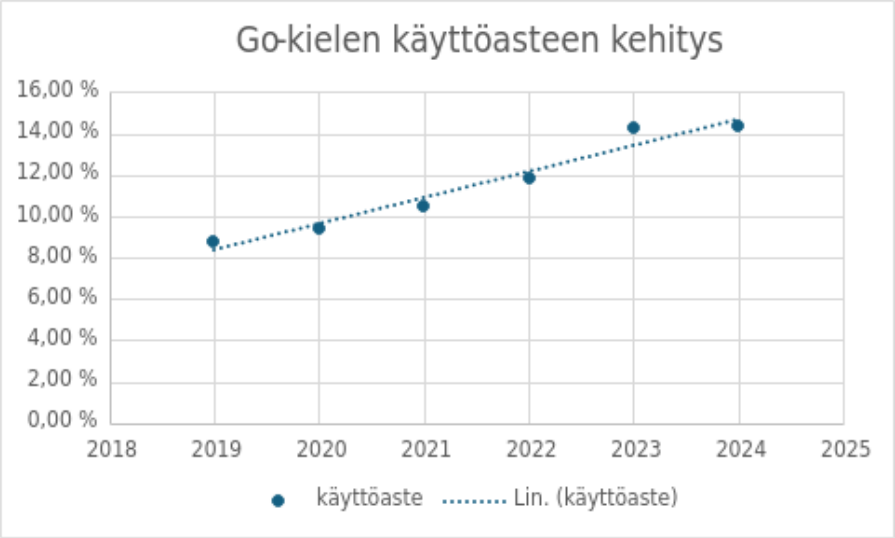
Go-kieli oli vuoden 2024 GitHub Octoverse –raportin mukaan kolmen nopeimmin kasvavan kielten joukossa avoimen lähdekoodin projektien aktiivisuuden osalta. GitHub Octoverse on GitHubin julkaisema vuosittainen raportti, joka tarkastelee avoimen lähdekoodin tilaa ja kehittäjäyhteisön toimintaa GitHub-alustalla. Kuvassa 7 on esitelty Github Octoverse -raportin tulokset kymmenen nopeimmin kasvavien kielten osalta vuonna 2024. (Octoverse: AI leads Python to top language as the number of global developers surges 2024)



KUVA 7. Go-ohjelmointikieli kolmen nopeimmin kasvavien kielten joukossa (Octoverse: AI leads Python to top language as the number of global developers surges 2024).

Go-kielen suosion kasvu näkyy myös Stack Overflow –sivuston kehittäjäkyselyn tuloksissa (kuvio 1). Sivuston teettämässä vuosittaisessa kyselyssä kysytään ammattiohjelmoijilta kysymyksiä, kuten mitä ohjelmointikieliä, kehitystyökaluja ja

-ympäristöjä he käyttävät työssään. (Stack Overflow Developer Survey 2019–2024)



KUVIO 1. Go-kielen käyttöasteen kehitys ammattikehittäjien keskuudessa (Stack Overflow Developer Survey 2019–2024).

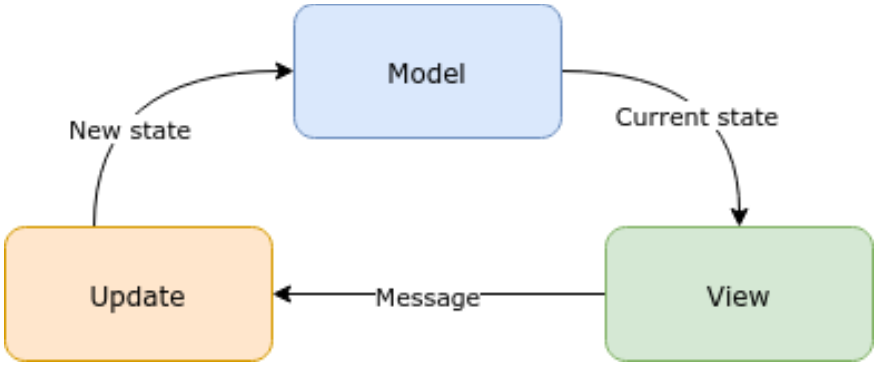
Maaliskuussa vuonna 2025 Anders Hejlsberg, TypeScript-kielen kehittäjä, ilmoitti Microsoftin blogissa, että TypeScript tullaan muuttamaan natiiviksi käännöstyökaluksi, mikä tarkoittaa, että sen ydin tullaan kirjoittamaan uudelleen suorituskykyisemmällä ohjelmointikielellä. Kieleksi valittiin Go-kieli. Tämä muutos tulee todennäköisesti kasvattamaan Go-kielen suosiota entisestään, varsinkin webkehittäjien keskuudessa. (Hejlsberg 2025)

2.2.2 Bubble Tea -ohjelmistokehys

Bubble Tea on Go-kielelle kehitetty avoimen lähdekoodin ohjelmistokehys, jonka avulla tekstipohjaisten käyttöliittymien ohjelmointi on helpompaa. Bubble Tea on osa Charm-kirjastoa, joka on kokoelma työkaluja tekstipohjaisten käyttöliittymien kehittämiseen.

Bubble Tea:llä kehitetyt sovellukset noudattavat Elm-arkkitehtuuria, joka on ohjelmointimalli, joka jakaa sovelluksen kolmeen pääkomponenttiin: Model, View ja Update:

- Model on sovelluksen tila. Se voi sisältää erilaisia tietotyyppien tietoja, kuten merkkijonoja, kokonaislukuja tai muita tietorakenteita. Sen avulla voidaan hallita sovelluksen tilaa ja sen muutoksia.
- View on sovelluksen käyttöliittymä, joka määrittelee, miltä sovellus näyttää käyttäjälle. TUI-sovelluksissa View esitetään merkkijonosta, joka muodostaa näytettävän näkymän komentorivillä.
- Update on sovelluksen logiikka, joka määrittelee, miten sovelluksen tila muuttuu käyttäjän syötteen tai muiden tapahtumien seurauksena. Se on siis funktio, joka vastaanottaa viestejä ja päivittää sovelluksen tilan niiden perusteella. (Bubble Tea -dokumentaatio n.d.)



KUVIO 2. Elm-arkkitehtuurin toimintaperiaate.

Elm-arkkitehtuuri on erityisen suosittu reaktiivisessa ohjelmoinnissa, ja se mahdollistaa selkeän erottelun sovelluksen tilan, käyttöliittymän ja logiikan välillä. Arkkitehtuuri sai alkunsa Elm-ohjelmointikielestä, mutta se on sittemmin levinnyt muihin kieliin. Sitä käytetään laajasti monissa ohjelmointikielissä ja -kehyksissä, ja se on varsinkin suosittu web-kehityksessä, jossa reaktiivisuus käyttäjäsyötteisiin on tärkeää. (The Elm Architecture n.d.)

2.2.3 Lip Gloss -ohjelmistokirjasto

Lip Gloss on Go-kielelle kehitetty ohjelmistokirjasto, joka mahdollistaa tekstin muotoilun ja värityksen komentorivillä. Se on erityisesti suunniteltu käytettäväksi yhdessä Bubble Tea -ohjelmistokehyksen kanssa, mutta sitä voidaan käyttää myös erikseen. Lip Gloss tarjoaa helpon tavan lisätä värejä, taustoja ja muita muotoiluominaisuuksia tekstipohjaisiin käyttöliittymiin.

Sen syntaksi muistuttaa paljon CSS:ää, mikä tekee siitä helpon oppia ja käyttää. Lip Glossin avulla kehittäjät voivat luoda visuaalisesti miellyttäviä tekstipohjaisia käyttöliittymiä, jotka erottuvat perinteisistä tekstipohjaisista sovelluksista. (Lip Gloss -dokumentaatio n.d.)

2.3 Kirjoitustestit

2.3.1 Määritelmä

Kirjoitustesteillä mitataan käyttäjän kirjoitusnopeutta ja tarkkuutta. Ne tarjoavat käyttäjille mahdollisuuden harjoitella näppäimistöllä kirjoittamista erilaisilla teksteillä.

Suurin osa kirjoitustesteistä perustuu satunnaisiin sanoihin, jotka valitaan sanalistasta. Sanalista voi olla joko satunnainen tai se voi perustua tiettyyn aiheeseen tai teemaan. On esimerkiksi olemassa ohjelmoijille suunnattuja kirjoitustestejä, joissa sanalista koostuu ohjelmointikielen avainsanoista ja symboleista. Kirjoitustesteissä käyttäjälle annetaan yleensä tietty aika, jonka kuluessa hänen on kirjoitettava mahdollisimman monta sanaa oikein. Testin aikana käyttäjä näkee kirjoitettavat sanat ja hänen on yritettävä kirjoittaa ne mahdollisimman nopeasti ja tarkasti. Testin lopussa ilmoitetaan käyttäjän kirjoitusnopeus ja tarkkuus.

2.3.2 Kirjoitusnopeuden laskeminen

Kirjoitusnopeus mitataan yleensä sanoina minuutissa (engl. WPM, words per minute), ja se kertoo, kuinka monta sanaa käyttäjä pystyy kirjoittamaan minuutissa. Kirjoitusnopeus voidaan ilmoittaa bruttokirjoitusnopeutena (engl. Gross WPM) tai nettokirjoitusnopeutena (engl. Net WPM). Bruttokirjoitusnopeudessa ei huomioida virheitä, kun taas nettokirjoitusnopeudessa otetaan huomioon virheet. Nettokirjoitusnopeus kuvaa paremmin käyttäjän todellista kirjoitusnopeutta, ja se onkin yleisemmin käytetty kirjoitustesteissä. (How to Calculate Typing Speed (WPM) and Accuracy n.d.)

Bruttokirjoitusnopeus lasketaan seuraavalla kaavalla:

$$Brutto\ WPM = \frac{kirjoitetut\ merkit}{5 \cdot aika\ minuutteina} \quad (1)$$

Ja nettokirjoitusnopeus:

$$Netto\ WPM = Brutto\ WPM - \frac{virheelliset\ sanat}{aika\ minuutteina} \quad (2)$$

Laskukaavassa muunnetaan kirjoitetut merkit sanoiksi jakamalla ne viidellä. Luku viisi perustuu siihen, että keskimääräinen sana englannin kielessä koostuu viidestä merkistä.

Kirjoitusnopeuden lisäksi kirjoitustesteissä mitataan myös kirjoittamisen tarkkuutta. Tarkkuus lasketaan seuraavalla kaavalla:

$$Tarkkuus\ = \frac{oikein\ kirjoitetut\ sanat}{kirjoitetut\ sanat} \cdot 100 \quad (3)$$

Tarkkuus ilmoitetaan prosentteina, ja se kertoo, kuinka monta prosenttia kirjoitetuista sanoista kirjoitettiin oikein. (How to Calculate Typing Speed (WPM) and Accuracy n.d.)

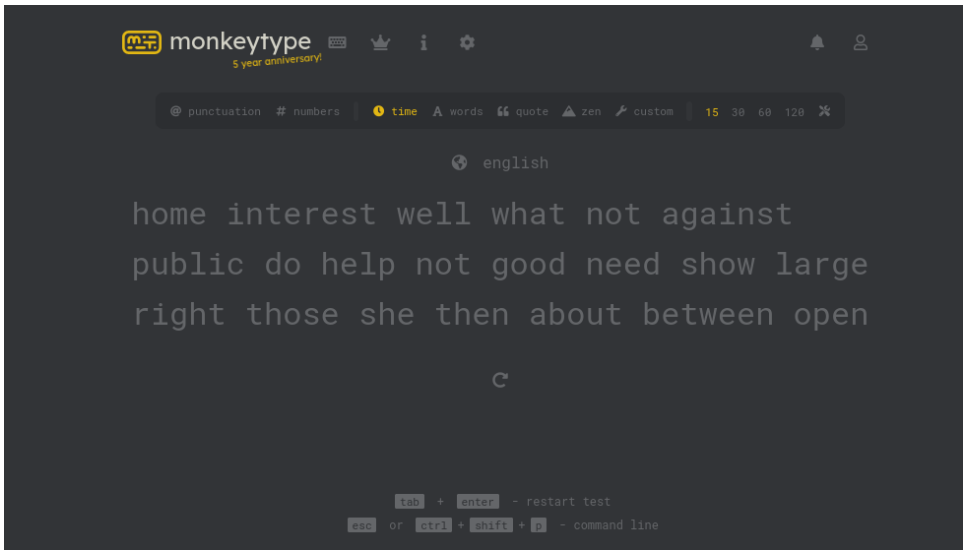
2.3.3 Vastaavanlaiset sovellukset

On olemassa monia erilaisia kirjoitustestisovelluksia, jotka tarjoavat käyttäjille mahdollisuuden harjoitella kirjoittamista. Suurin osa sovelluksista on toteutettu graafisina käyttöliittyminä, jotka toimivat selaimessa ja vaativat internetyhteyden. Esimerkkejä tällaisista sovelluksista ovat Keybr ja Monkeytype.

Keybr ja Monkeytype ovat edistyneempiä verrattuna tässä opinnäytetyössä toteutettavaan sovellukseen, sillä ne tarjoavat käyttäjille mahdollisuuden harjoitella kirjoittamista monella erilaisella tekstillä, kuten ohjelmointikoodilla ja monella eri kielellä. Niihin on myös toteutettu edistyneempi kirjoitusnopeuden kehityksen seuranta, joka mahdollistaa käyttäjien seuraavan omaa kehitystään ja verrata suorituksiaan muihin käyttäjiin.

Ero näiden kahden suositun sovelluksen välillä on se, että Keybr on suunniteltu erityisesti kymmensormijärjestelmän oppimiseen, kun taas Monkeytype on enemmänkin kirjoitusnopeuden ja tarkkuuden mittaamiseen keskittyvä sovellus. (Monkeytype n.d.; Keybr n.d.)

Tässä opinnäytetyössä keskitytään toteuttamaan juuri Monkeytype:n kaltainen sovellus, joka toimisi ilman riippuvuutta graafisesta ympäristöstä tai internetyhteydestä (kuva 8).

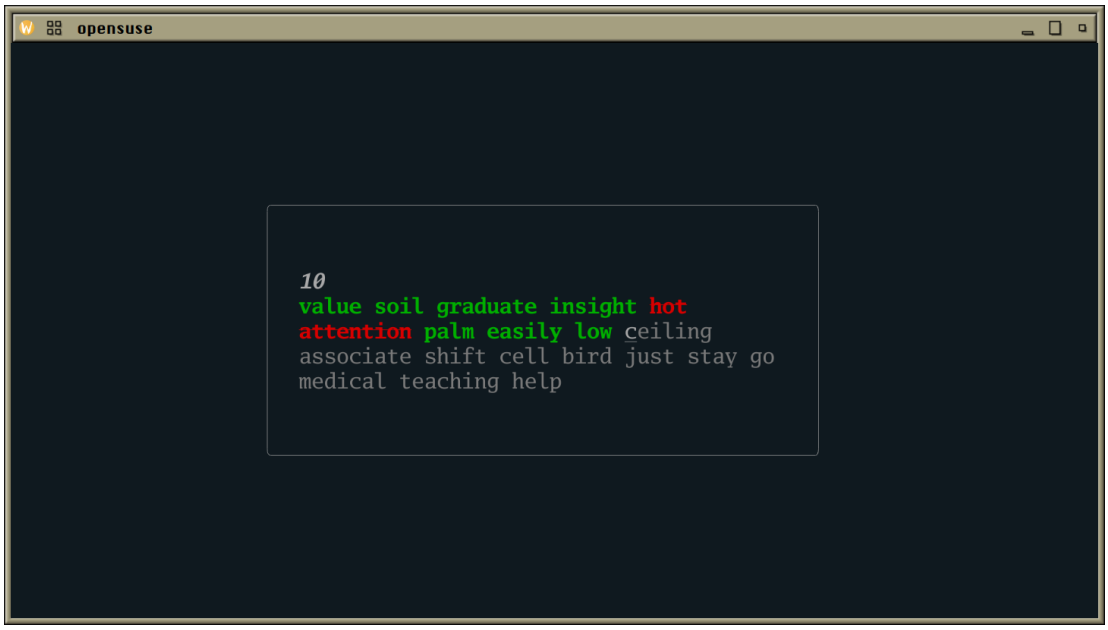


KUVA 8. Kuvankaappaus Monkeytype-verkkosivuston kirjoitustestistä.

3 SOVELLUKSEN TOTEUTUS

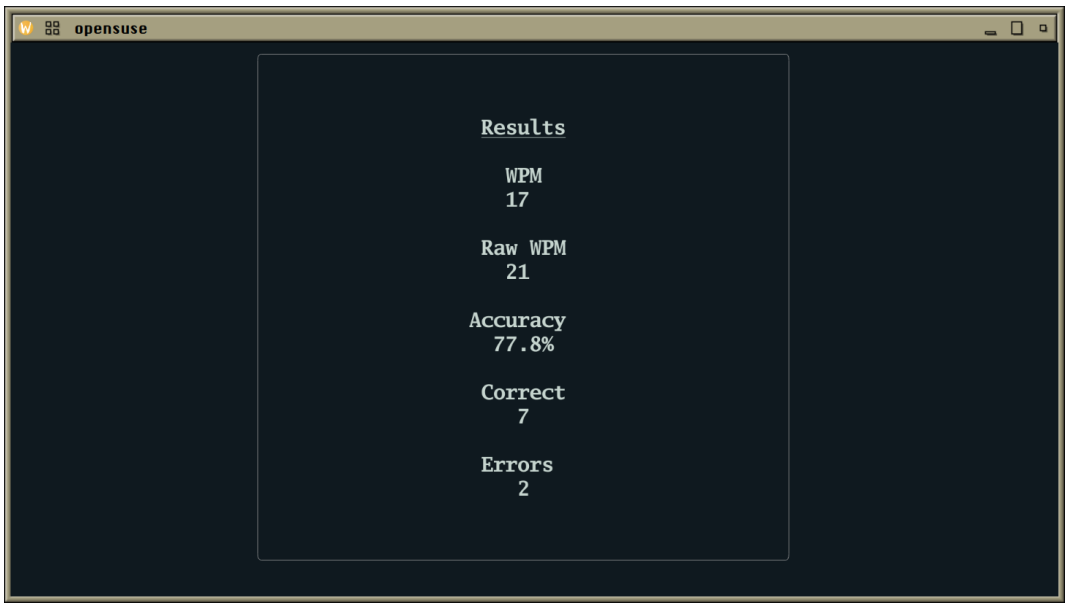
3.1 Käyttöliittymä

Gophertype-sovelluksen käyttöliittymä on mahdollisimman yksinkertainen ja häiriötön, jotta käyttäjä voi keskittyä kirjoittamiseen. Käyttöliittymässä näkyy kirjoitettavat sanat, käyttäjän kirjoittamat merkit ja kirjoitusnopeus sekä tarkkuus testin lopussa. Käyttöliittymä on keskitetty keskelle näyttöä, ja se käyttää Lip Gloss -kirjastoa tekstin muotoiluun ja väritykseen. Käyttöliittymä antaa myös käyttäjälle reaaliaikaista palautetta kirjoitusvirheistä värittämällä virheelliset merkit ja sanat punaisiksi (kuva 9).



KUVA 9. Kuvankaappaus Gophertype-sovelluksesta kirjoitustestin aikana.

Käyttöliittymä sisältää myös ruudun, jossa kirjoitustestin tulokset näytetään testin lopussa. Tuloksissa näytetään käyttäjän kirjoitusnopeus brutto- ja nettonopeuksina, tarkkuus, sekä oikein ja väärin kirjoitettujen sanojen määrät (kuva 10).



KUVA 10. Kuvankaappaus Gophertype-sovelluksesta testin päätyttyä.

Käyttöliittymän ulkoasun inspiraationa toimii pääasiassa Monkeytype-sovelluksen yksinkertainen käyttöliittymä (kuva 8).

3.2 Sanalistan hallinta

Sovellus tarvitsee tavan hallita sanalistaa, josta se voi valita satunnaisia sanoja kirjoitustestin aikana. Sanalistan hallinta toteutetaan erillisessä paketissa, joka sisältää kaikki tarvittavat toiminnot sanojen käsittelemiseksi.

Sanalistan hallintaa varten luodaan erillinen paketti nimeltä words, joka sisältää WordManager-tietorakenteen. WordManagerille luodaan funktiot sanalistan lataamiseen ja satunnaisten sanojen valitsemiseen. Sanalista ladataan tekstitiedostosta, joka sisältää sanoja rivinvaihdolla erotettuna. Sovelluksen ensimmäisen version sanalista koostuu kolmesta tuhannesta englanninkielisestä sanasta, mutta tulevaisuudessa sanalistaa voidaan laajentaa myös muihin kieliin.

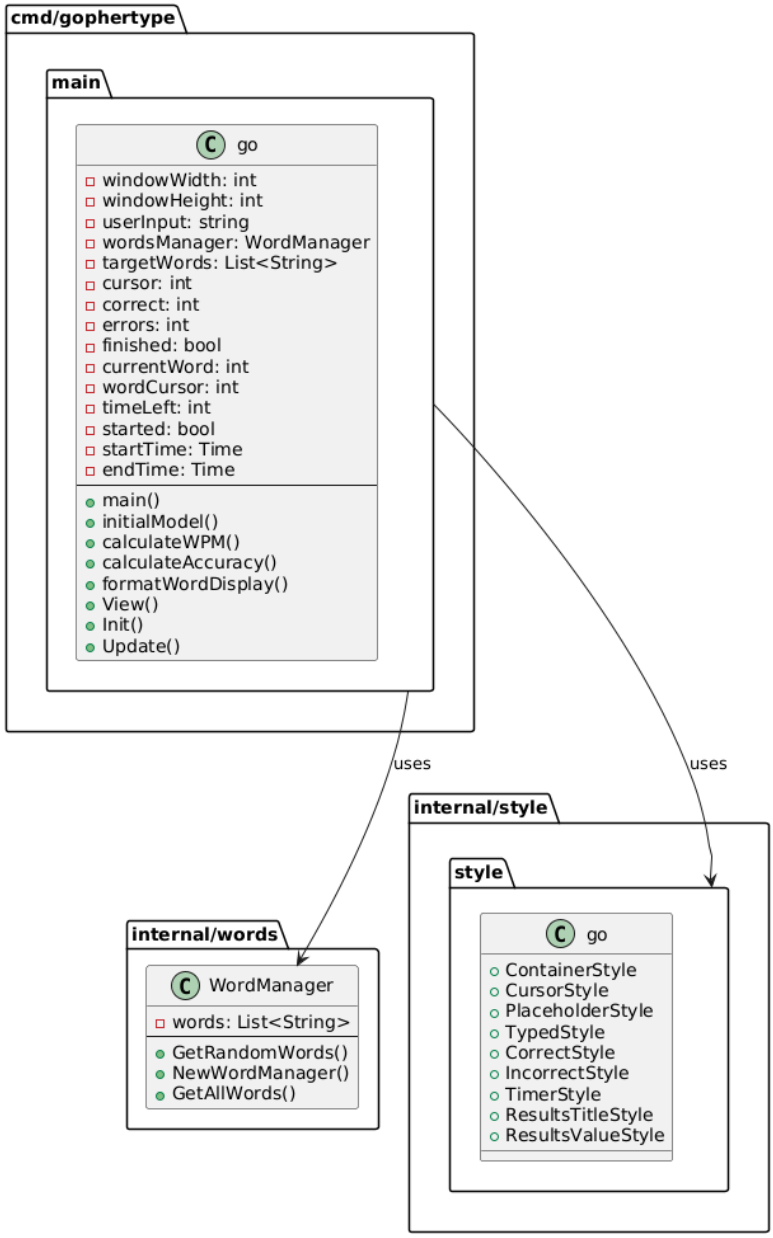
Sanalista saadaan upotettua WordManager-rakenteeseen näppärästi Go-kielen standardikirjastoon kuuluvalla Embed-ominaisuudella (kuva 11).

```
13      //go:embed assets/english.txt
14      var englishWords string
```

KUVA 11. Sanalistan sisältävän tekstitiedoston upotus koodiin.

3.3 Sovelluksen rakenne

Koska sovellus toteutetaan Bubble Tea -ohjelmistokehyksellä, sovelluksen rakenne noudattaa Elm-arkkitehtuuria. Alla olevassa kaaviossa on esitetty sovelluksen rakenne UML-kaaviona.



KAAVIO 3. UML-kaavio sovelluksen rakenteesta.

Sovelluksen koodi on jaettu kolmeen pakettiin: main, words ja style. Main-paketti sisältää sovelluksen pääohjelman ja kaikki kirjoitustestiin ja käyttöliittymään

näyttämiseen liittyvän logiikan. Words-paketti sisältää sanalistan hallintaan liittyvän logiikan, ja style-paketti sisältää käyttöliittymän tyylityksen eli Lip Gloss koodin.

3.3.1 Model

Model on sovelluksen tila, joka sisältää kaikki tarvittavat tiedot kirjoitustestin aikana. Model toteutetaan tietorakenteena (engl. struct), joka sisältää tietokenttiä, kuten kirjoitettavat sanat, käyttäjän kirjoittamat merkit, virheet ja jäljellä olevan ajan. Model-tietorakenteen koko sisältö on kuvattu alla olevassa koodipätkässä, jossa se määritellään (kuva 12).

```
26  type model struct {
27          windowWidth  int
28          windowHeight int
29          userInput    string
30          wordsManager *words.WordManager // Word manager dependency for generating words
31          targetWords  []string
32          cursor       int       // Cursor position in the input
33          correct      int       // Number of correctly typed words
34          errors       int       // Number of incorrectly typed words
35          finished     bool      // Indicates if the test is finished
36          currentWord  int       // Index of the current word
37          wordCursor   int       // Cursor position within the current word
38          timeLeft     int       // Time left in seconds
39          started      bool      // Indicates if the test has started
40          startTime    time.Time // Start time of the test
41          endTime      time.Time // End time of the test
42  }
```

KUVA 12. Model-tietorakenteen sisältö koodissa.

3.3.2 Update

Update on funktio, joka saa syötteenä Bubble Tea -viestin (engl. message) ja palauttaa päivitetyn Model-rakenteen. Tämän sovelluksen tapauksessa viestit ovat erilaisia tapahtumia, kuten näppäimen painalluksia, ikkunan skaalauksen tai ajastimen päivityksiä. Bubble Tea -ohjelmistokehys hoitaa Update-funktioon saapuvien viesteihin reagoinnin switch-lauseen avulla. Jokaiselle viestityypille on oma tapaus (engl. case), joka määrittelee, mitä tapahtuu, kun kyseinen viesti vastaanotetaan.

Alla olevasta koodista näkyy, miten esimerkiksi Ctrl+C ja Escape-näppäinten painalluksiin reagoidaan koodissa.

```
66  func (m model) Update(msg tea.Msg) (tea.Model, tea.Cmd) {
67          switch msg := msg.(type) {
68          case tea.KeyMsg:
69                  // Handle quit signals first
70                  if msg.String() == "ctrl+c" || msg.String() == "esc" {
71                          return m, tea.Quit
72                  }
```

KUVA 13. Ctrl+C- ja Escape -näppäinten käsittely koodissa.

Update-funktiossa käsitellään myös kirjoitustestin ajastin. Ajastimen käsittely toimii siten, että kun ajastin lähettää Update-funktiolle viestin, tarkistetaan, onko kirjoitustesti käynnissä. Jos näin on, jäljellä olevaa aikaa vähennetään yhdellä sekunnilla. Testi päättyy, kun ajastimen aika loppuu, jolloin Model-tietorakenteen finished-kenttä asetetaan todeksi. Tämän jälkeen sovellus näyttää käyttäjälle tulosruudun (kuva 14).

```
148         case tickMsg:
149                 if m.started && !m.finished {
150                         m.timeLeft--
151                         if m.timeLeft <= 0 {
152                                 m.finished = true
153                                 m.endTime = time.Now() // Stop the timer
154                                 return m, nil
155                         }
156                         return m, tea.Tick(time.Second, func(t time.Time) tea.Msg {
157                                 return tickMsg(t)
158                         })
159                 }
```

KUVA 14. Ajastimeen liittyvä koodi.

Käyttäjän syötteen käsittely toimii siten, että kun käyttäjä kirjoittaa merkkejä, ne tallennetaan muuttujaan. Kun käyttäjä painaa välilyöntiä, tarkistetaan, onko sana kirjoitettu oikein verrattuna mallisanaan. Tämän jälkeen siirrytään seuraavaan sanaan, päivitetään oikeiden ja virheellisten sanojen laskurit, ja tarvittaessa luodaan lisää sanoja testin jatkumiseksi.

3.3.3 View

View on funktio, joka saa parametrina sen hetkisen Model-rakenteen ja palauttaa käyttöliittymän merkkijonona. Merkkijonoa myös muotoillaan Lip Gloss -kirjaston avulla, ennen kuin se palautetaan käyttöliittymään.

Sovelluksessa View-funktio kutsuu calculateWPM- ja calculateAccuracy apufunktioita, jotka laskevat kirjoitusnopeuden ja tarkkuuden. Funktiot käyttävät aiemmassa osiossa määritettyjä laskukaavoja (1, 2 ja 3), laskeakseen kirjoitusnopeuden ja tarkkuuden (kuva 15).

```
165  func calculateWPM(userInput string, timeElapsed time.Duration, errors int) (grossWPM int, netWPM int) {
166          minutes := timeElapsed.Minutes()
167          if minutes == 0 {
168                  minutes = minTimeElapsed
169          }
170
171          gross := float64(len(userInput)) / charsPerWord / minutes
172          net := max((float64(len(userInput))/charsPerWord-float64(errors))/minutes, 0)
173
174          return int(gross), int(net)
175  }
176
177  func calculateAccuracy(correct, errors int) float64 {
178          if correct+errors == 0 {
179                  return 100.0
180          }
181          return float64(correct) / float64(correct+errors) * 100
182  }
```

KUVA 15. Kirjoitusnopeuden ja tarkkuuden laskeminen koodissa.

Kun kirjoitustesti on käynnissä, View-funktio näyttää käyttäjälle kirjoitettavat sanat, käyttäjän kirjoittamat merkit ja ajastimen. Kun testi päättyy, View-funktio näyttää käyttäjälle tulosruudun, jossa näkyvät kirjoitusnopeus ja tarkkuus.

4 POHDINTA

Sovellus saatiin toteutettua suunnitellusti, ja se toimii odotetulla tavalla. Kaikkia projektin alkuvaiheessa ideoituja ominaisuuksia ei kuitenkaan ehditty toteuttaa, ja sovellus vaatii vielä paljon jatkokehitystä ollakseen vertailukelpoinen muiden saatavilla olevien kirjoitustestien kanssa. Sovellukseen toteutettiin kuitenkin kaikista tärkeimmät toiminnot, eli kirjoitusnopeuden ja tarkkuuden mittaus. Käyttöliittymästä saatiin myös juuri sellainen, mitä suunniteltiin, eli mahdollisimman yksinkertainen ja häiriötön.

Jatkokehitysideoita sovellukselle ovat esimerkiksi mahdollisuus harjoitella eri kielten sanoja, symboleita sekä ohjelmointikielissä esiintyviä termejä. Lisäksi sovellukseen voitaisiin lisätä ominaisuus, jonka avulla käyttäjä voisi seurata omaa kehitystään ja verrata suorituksiaan muiden käyttäjien tuloksiin. Tämä tekisi sovelluksesta kilpailukykyisemmän verrattuna muihin vastaaviin kirjoitustestisovelluksiin.

Sovelluksen koodiin kannattaisi myös lisätä yksikkötestejä, erityisesti kirjoitusnopeutta ja -tarkkuutta mittaaviin funktioihin. Näin voitaisiin varmistaa, että mittaukset toimivat luotettavasti ja oikein.

Uskon, että sovellukselle on käyttöä vielä pitkälle tulevaisuuteen – ainakin omalla kohdallani. Kirjoitusnopeus on kuitenkin taito, joka säilyy relevanttina, ellei kehitetä kokonaan uutta ja parempaa tapaa syöttää tekstiä tietokoneelle.

LÄHTEET

Bubble Tea -dokumentaatio. n.d. Github. Verkkosivu. Viitattu 9.5.2025. https://github.com/charmbracelet/bubbletea/

Go FAQ. n.d. Go. Verkkosivu. Viitattu 13.4.2025. https://go.dev/doc/faq/

Hejlsberg, A. 2025. A 10x Faster TypeScript. Microsoft Dev Blogs. Verkkosivu. Viitattu 17.5.2025. https://devblogs.microsoft.com/typescript/typescript-native-port/

How to Calculate Typing Speed (WPM) and Accuracy. n.d. Speed Typing Online. Verkkosivu. Viitattu 16.5.2025. https://www.speedtypingonline.com/typing-equations

Keybr. n.d. Learn to Type Faster. Verkkosivu. Viitattu 18.5.2025. https://www.keybr.com/

Lip Gloss -dokumentaatio. n.d. Github. Verkkosivu. Viitattu 9.5.2025. https://github.com/charmbracelet/lipgloss/

Monkeytype. n.d. About. Verkkosivu. Viitattu 18.5.2025. https://monkeytype.com/about

Octoverse: AI leads Python to top language as the number of global develop-ers surges. 2024. Github. Verkkosivu. Viitattu 26.3.2025. https://github.blog/news-insights/octoverse/octoverse-2024/

Stack Overflow Developer Survey. 2019–2024. Stack Overflow. Verkkosivu. Viitattu 26.3.2025. https://survey.stackoverflow.co/

The Elm Architecture. n.d. Elm-lang. Verkkosivu. Viitattu 8.5.2025. https://guide.elm-lang.org/architecture/

LIITTEET

Liite 1. Linkki Gophertype-sovelluksen lähdekoodiin

https://github.com/oskarilindroos/gophertype/