Spraakherkenning in software

Dewin van Zaanen, N&T, H5C
Robin Walthuis, N&G, A6A
Datum van inleveren: 21/12/2023
Kim Blankendaal, Don Bosco College

1. Samenvatting

In dit onderzoek is gekeken naar de mogelijkheid om in een programma effectief gebruik te maken van spraakherkenning. Hiervoor is in Python een programma ontworpen dat door middel van spraakherkenning een digitale agenda bestuurt. Om dit te bereiken moet eerst een module gebruikt of gemaakt worden die spraak herkent en interpreteert tot tekst, in dit geval OpenAI Whisper. Deze tekst moet dan gecontroleerd worden op keywords. Deze keyword moet geïnterpreteerd worden naar een functie en deze moet de rest van de context kunnen gebruiken om de informatie in de kalender te zetten of deze aan te passen. Voor dit onderzoek is het van belang dat de opgeslagen data op meerdere apparaten beschikbaar is, en dus is het nodig om een server te hosten waarop ingelogd kan worden op een gebruikersaccount waarop de data opgeslagen is. Hiervoor moet alle data gecategoriseerd worden, zodat de server het later terug kan vinden en sturen. Dan is het ook belangrijk om wachtwoorden op een veilige manier op te slaan, zodat het niet makkelijk is de inloggegevens van iemand anders te vinden. Dit kan gemakkelijk over het hoofd gezien worden, waardoor veel kleine bedrijven “data leaks” hebben waar alle gegevens publiek worden gesteld door een vaak heel kleine fout. Als laatste moet de informatie visueel gecommuniceerd worden met de gebruiker via een GUI. Ook is het voor toegankelijkheid handig om een manier te hebben de functionaliteit van spraakherkenning te gebruiken zonder echt te spreken. Dit is belangrijk voor wanneer het programma wordt gebruikt in drukke ruimtes, of wanneer de gebruiker geen werkende microfoon heeft aangesloten. Hiervoor is een terminal pagina gemaakt waarin de opgeschreven tekst op dezelfde manier geïnterpreteerd wordt als bij spraakherkenning. De demo-app die in dit onderzoek gemaakt is, DaySphere, is hier te downloaden. Dit programma is echter niet zo goed als het had kunnen zijn als niet gebruik gemaakt was van Tkinter.

2. Inhoudsopgave

3. Inleiding

Spraakherkenning wordt steeds meer en meer geavanceerd, maar het wordt in zeer weinig programma's echt gebruikt. Een van de bekendste toepassingen van spraakherkenning op Windows systemen, Cortana, wordt sinds augustus niet meer onderhouden door Microsoft. Het probleem hiermee is dat Cortana door veel mensen gebruikt wordt om handsfree hun notities op te schrijven, reminders te zetten en algemene agenda functies te gebruiken.

Als vervanger voor Cortana zou dus een ander programma nodig zijn dat op Windows systemen spraakinput kan gebruiken om bijvoorbeeld een agenda te beheren. De onderzoeksvraag is daarom: Op welke manier kan Python gebruikt worden om effectief gebruik te maken van spraak herkennende technologie?

Voor een programma gefocust op spraakherkenning is het vooral belangrijk dat het instructies uit spraak kan halen. De eerste deelvraag is dan dus ook: "Hoe wordt een stem geïnterpreteerd naar uitvoerende functies?"

Al is het niet per se nodig om data op te slaan op een server is dat wel aangeraden. Zo kan dezelfde informatie op meerdere apparaten beschikbaar worden gesteld zonder dat een externe harde schijf of USB nodig is. Met deze methode komen veel moeilijkheden, die verwerkt zijn in de deelvragen; op welke manier communiceren clients met de server? En op welke manier wordt de opgeslagen data gecategoriseerd?

Ook moet het programma er goed uit zien en gebruikersvriendelijk zijn. Dit kan makkelijk lijken, maar met design moet er veel nagedacht worden over hoe de gebruiker zal interageren met het programma. Dat moet dan tot uiting komen in code, wat vooral in Python het niet makkelijk is. Dit wordt daarom uitgewerkt in de deelvraag: "Hoe wordt informatie gecommuniceerd met de gebruiker?"

4. Theoretisch kader

4.1. Python

Python is een programmeertaal die door miljoenen programmeurs wordt gebruikt. Dat komt voor een groot deel door de simpliciteit van Python. Het wordt dus ook steeds meer gebruikt door scholieren op de middelbare school. Veel van deze begrippen zijn niet alleen toepasbaar in de context van Python, maar staan in deze codeertaal in elk geval centraal.

Code	De code is de instructies voor de computer uitgeschreven naar woorden. Deze worden wanneer het programma start door de compiler omgezet tot basisinstructies, en dan naar binair. Dat is hoe de computer het uitvoert.
Syntax	Syntax is de structuur van code.
Functie	Een functie is een set instructies die eerst gedefinieerd en later pas gebruikt wordt. Zo wordt het makkelijker om een grote hoeveelheid instructies meerdere keren te gebruiken wanneer een ander deel van het programma bereikt wordt. Het uitvoeren van een functie wordt soms een callback genoemd.
String	Een string is een type variabele dat een tekenreeks bevat. De nummers in een string worden gezien als het letterlijke symbool voor dat nummer, niet een getal.
Bytelike Object	Een bytelike object werkt in toepassing bijna identiek aan een string, maar in een bytelike object worden tekens opgeslagen als een code in plaats van het letterlijke teken. Hoe de code wordt gegenereerd en welke code voor welk symbool staat verschilt per encoding, maar het blijft binnen de encoding altijd hetzelfde. Strings en bytelike objects zijn makkelijk in elkaar om te zetten door gebruik van de encode() en decode() functies.
Integer	Een integer is een type variabele dat een getal bevat. Met dit type variabele kunnen wiskundige formules worden toegepast wat niet kan met de nummers in een string.
Boolean	Een boolean is een type variabele die alleen waar of onwaar kan zijn. Dit wordt vaak ook gezien als aan of uit, of 1 of 0.
Class	Een class wordt net als een functie eerst gedefinieerd en later pas gebruikt. Dit kan echter ook een type variabele zijn. In dat geval wordt gezegd dat de variabele een “Object” is van het type van de class. Daarnaast kan een Class ook nog een Class bevatten en/of van een Class overerven oftewel Inheriten. Dit houdt in dat de Class dezelfde structuur en informatie erft als de “Inheritee”, dus alle informatie en functies. Dit is anders dan een Class in een Class hebben, aangezien “Inheritance” eerder een copy van de originele maakt.
OOP	Object-Oriented-Programming (OOP) is een ontwerpfilosofie. Het principe is dat er geprogrammeerd wordt in Objecten, waardoor je niet je code hoeft te herhalen. In de praktijk zullen veel instanties van “Inheritance” en call-backs te zien zijn.
Module	Een hoop code dat op zichzelf meestal niets doet maar wel functies bevat die door een ander programma geïmporteerd en gebruikt kunnen worden. Deze wordt meestal aan het begin van een programma geïmporteerd, waardoor de functies overal in het bestand beschikbaar worden.
List	Een type variabele waarin meerdere andere variabelen in zitten.
Dictionary	Een type variabele waarin informatie gekoppeld aan een sleutel wordt opgeslagen en achterhaald kan worden zolang dezelfde sleutel gebruikt wordt.

Self	Self is een begrip dat uitsluitend te vinden is in Classes. Self verwijst hier naar de Class waarin het is genoemd.
Lambda	Een Lambda is een anonieme Functie. Dat betekent dat het een Call-Back kan uitvoeren zonder dat het gedefinieerd wordt als een functie. Bij CustomTkinter wordt het voornamelijk gebruikt om Call-Backs te gebruiken bij een actie zoals op een knop drukken.
Front-End	Front-End is alles wat er visueel gebeurt, dus dit houdt in de tekst en knoppen op het scherm die informatie overbrengt naar de gebruiker. Front-End wordt meestal gebruikt in de context van ontwerpen. Het eindproduct daarvan noemen we GUI, maar Front-End is bijna een synoniem daarvan.
Back-End	Back-End is alles wat op de achtergrond gebeurt zonder gebruiker interactie. Een voorbeeld hiervan is de Client-Server Communication.
GUI	Afkorting voor Graphical User Interface. Dit is het deel van het programma dat op het scherm wordt geprojecteerd waarmee de gebruiker kan interageren.
Initialiseren	Als een variabele wordt geïnitialiseerd wordt het gedefinieerd op het huidige niveau zonder waarde. Een class kan ook geïnitialiseerd worden. In dat geval wordt het net als een functie gestart en wordt de code in de __init__() functie van de class uitgevoerd.
Fork	Een Fork is een versie van het originele programma dat verder ontwikkeld kan worden zonder dat het origineel beïnvloed wordt. Dit kan ook gedaan worden door een externe gebruiker die niets aan het origineel gecontribueerd heeft.
Python standard library	Een lijst modules die al beschikbaar is te gebruiken met de installatie van Python. Alle andere modules moeten door een package-management system gedownload worden. Voor Python heet dit systeem Pip.

Tabel 1 Algemene Python begrippen

4.2. CustomTkinter

[Official Documentation And Tutorial | CustomTkinter (tomschimansky.com)]

CustomTkinter (CTk) is een Fork van Python's Tkinter. Tkinter is door Python ontwikkeld om simpele GUIs te maken. CustomTkinter wordt hier gebruikt in plaats van Tkinter omdat CustomTkinter een meer modern uiterlijk heeft en beter aanpasbaar is. CustomTkinter was gemaakt met OOP in gedachte en geeft makkelijk de mogelijkheid om complexere GUIs te maken. Dit maakt het ontwikkelen van de Front-End wordt ook sneller en gemakkelijker gemaakt door standaard functies beschikbaar te maken die in Tkinter anders zelf gemaakt zouden moeten worden.

Command	Command is de naam die binnen CTk gebruikt wordt voor een callback.
CTk	De CTk Class is de applicatie in zijn meest pure vorm en deze staat bovenin in de hiërarchie. Daarom Inherit de App Class deze Class. Dit is waar alle logica plaatsvindt binnen de applicatie, en ook wat de schermen toonbaar maakt.

CTkButton	De standaard knop van CTk, waarvan de syntax er zo uit ziet:
	`customtkinter.CTkButton(self, text="CTkButton",`
	`command=lambda: button_event())`

CTkCheckBox	De standaard checkbox van CTk, waarvan de syntax er zo uit ziet: `customtkinter.CTkCheckBox(app, text="CTkCheckBox", command=lambda: checkbox_event(), variable=check_var, onvalue="on", offvalue="off")`
CTkEntry	De CTkEntry is een tekstvak waar tekst ingevoerd kan worden. De syntax ziet er zo uit: `customtkinter.CTkEntry(self, placeholder_text="CTkEntry")`

CTkFrame	Een CTkFrame is voornamelijk een Class bedoeld om de hiërarchie op te bouwen. Dit wordt gebruikt door net als bij de CTk een andere Class te maken die de CTkFrame class inherit. Dit wordt gedaan met deze syntax:
	
	`class NewCTkFrame(customtkinter.CTkFrame):`
	`  def __init__(self, master):`
	`  customtkinter.CTkFrame.__init__(self, master)`

CTkLabel	CTkLabel geeft alleen tekst weer. De syntax ziet er zo uit: `customtkinter.CTkLabel(self, text="CTkLabel", fg_color="transparent")`
CTkOptionMenu	CTkOptionMenu geeft een lijst van opties om uit te kiezen. De syntax ziet er zo uit: `customtkinter.CTkOptionMenu(self, values=["option 1", "option 2"], command=lambda: optionmenu_callback, variable=optionmenu_var)`
CTkStringVar	CTkStringVar is een CTk Class die een String bevat met CTk functionaliteit, zoals bijvoorbeeld Trace. Deze Strings worden in CTk classes gebruikt in plaats van Strings zodat functies uitgevoerd kunnen worden wanneer er een verandering plaatsvindt in de string. De syntax voor het definiëren van een CTkStringVar is: `NewCTkStringVar = customtkinter.StringVar(self, "Value")`

grid_rowconfigure & grid_columnconfigure	Dit zijn opties binnen CTkFrames, die de hoeveelheid ruimte dat een CTkObject binnen de CTkFrame neemt. Dit wordt gedaan door specifieke indexes een “weight” te geven. De syntax ziet er zo uit:
	
	`self.grid_rowconfigure(0, weight=0)`
	`self.grid_columnconfigure([0, 1, 2, 3, 4, 5, 6], weight=1)`

Master	Een Master is de Class die in de Hiërarchie boven de huidige Class zit. Dus ieder CTk Class bevat een Master behalve de allerbovenste, de CTk Class instantie.
Trace	Een Trace is een Functie van de CTkStringVar class. Het voegt de functionaliteit toe aan een CTkStringVar om wanneer een verandering in de String plaatsvindt een externe functie uit te voeren. Zo kan data bijvoorbeeld dynamisch worden opgeslagen wanneer de String verandert. De syntax hiervoor is: `NewCTkStringVar.trace_add("write", lambda x, y, z: callback())`

Tabel 2 Algemene CustomTkinter begrippen

4.3. Pillow

[Pillow · PyPI]

Pillow is een Fork van de Python Imaging Library (PIL) module, deze Fork wordt gebruikt omdat Pillow nog geüpdatet wordt door programmeurs, en PIL niet meer. Inmiddels is de Python Imaging module zo erg verouderd dat het niet meer werkt op huidige versies van Python. Deze module maakt het mogelijk om een afbeeldingsbestand te gebruiken in projecten, en CustomTkinter maakt hier ook gebruik van.

Om Pillow te gebruiken wordt ook de Python standard library module “os” gebruikt. Dit is omdat os nodig is om toegang te krijgen tot bestandsystem van de computer, waardoor het nodig is om een afbeelding uit een bestand te halen.

4.4. Hashlib

[hashlib — Secure hashes and message digests — Python 3.12.0 documentation]

Hashlib is een zeer simpele module met maar één echte toepassing. Het voegt alle types van hashing toe in de vorm van functies. Omdat een hash een apart type object is heeft Hashlib ook een functie toegevoegd om een hash-object om te zetten naar een string of bytelike object.

Hashing	Een manier van het mixen van de data in een string zodat de originele niet meer terug te vinden is. Als de functie meerdere keren met dezelfde invoervariabele gedaan wordt zal de uitvoervariabele echter altijd hetzelfde zijn. Er bestaan verschillende soorten hashing, waarvan de veiligheid en specificaties verschillen, maar het doel blijft hetzelfde.
Salting	Als het gebruikt wordt voor cryptografie kan één stap voor hashing een willekeurige string, een salt, gegenereerd en geplakt worden aan de voorkant van de string. Het doel hiervan is om ervoor te zorgen dat als 2 accounts hetzelfde wachtwoord hebben, de uiteindelijke hash alsnog verschilt. Deze salt moet dan wel worden opgeslagen, want als het doel is om dezelfde hash te genereren zullen de invoervariabelen exact hetzelfde moeten zijn. Salting wordt niet gedaan door Hashlib zelf, maar het heeft alleen betekenis in deze context.

Tabel 3 Hashlib begrippen

4.5. Secrets

secrets — Generate secure random numbers for managing secrets — Python 3.12.0 documentation

Secrets voegt een aantal functies toe die gebruikt worden om zo willekeurig mogelijke variabelen te genereren. Dit wordt gebruikt voor de salt en voor de willekeurige priemgetallen die gebruikt worden in RSA. Voor normale willekeurige getallen wordt meestal os.urandom() gebruikt, maar aangezien computers niet volledig willekeurige variabelen kunnen genereren is dit niet veilig genoeg voor cryptografie.

RSA	RSA is een type encryptie dat door gebruik van 2 willekeurige priemgetallen 3 verschillende nummers genereert; een publieke sleutel, privé sleutel en een extra getal. Met de publieke sleutel en het extra getal kan een bericht gecodeerd worden dat alleen door gebruik van de privé sleutel en het extra getal gedecodeerd kan worden.

Tabel 4 Secrets begrippen

4.6. Datetime

datetime — Basic date and time types — Python 3.12.0 documentation

Datetime is een Python module van de Python standard library, die alles met kan weergeven. Ook al wordt op Calendar Page voornamelijk gebruik gemaakt van zelfgemaakte code, wordt Datetime gebruikt om de naam van de maand of dag te krijgen. Verder zorgt het ervoor dat de huidige dag actueel blijft en niet achterloopt.

4.7. Socket

socket — Low-level networking interface — Python 3.12.0 documentation

Socket is de module die communicatie tussen server en client mogelijk maakt. Het omvat alles wat nodig is voor communicatie: een socket openstellen via het IP en een openstaande poort, verbinden aan een al openstaande socket, berichten versturen en binnenkrijgen etc.

Socket	“an endpoint of communication to which a name may be bound” (Singh, 2019). Oftewel, een eindpunt van communicatie waaraan een naam kan worden gebonden.

Tabel 5 Socket begrippen

4.8. RegEx

Regular Expression HOWTO — Python 3.12.0 documentation

RegEx, volledig uitgeschreven Regular Expressions, is een module van de Python standard library die patronen uit een string haalt. Er zijn heel veel toepassingen voor, maar in dit onderzoek wordt RegEx alleen gebruikt om in de naar tekst geïnterpreteerde spraak te zoeken naar keywords zonder de string aan te passen.

4.9. Spraakherkenning

Spraakherkenning is het opnemen van geluid en dit daarna te analyseren om vanuit de opname de gesproken woorden als tekst te representeren. Voor dit onderzoek is het de bedoeling de opname en analyse tegelijk doen, zodat zo min mogelijk wordt gemist.

Keyword	Een woord of korte zin die een verstaanbare betekenis bevat voor een programma. Een voorbeeld hiervan is “schedule”. Dit laat het programma weten dat de data die erna komt opgeslagen moet worden op een gespecificeerde datum.

Tabel 6 Spraakherkenning begrippen

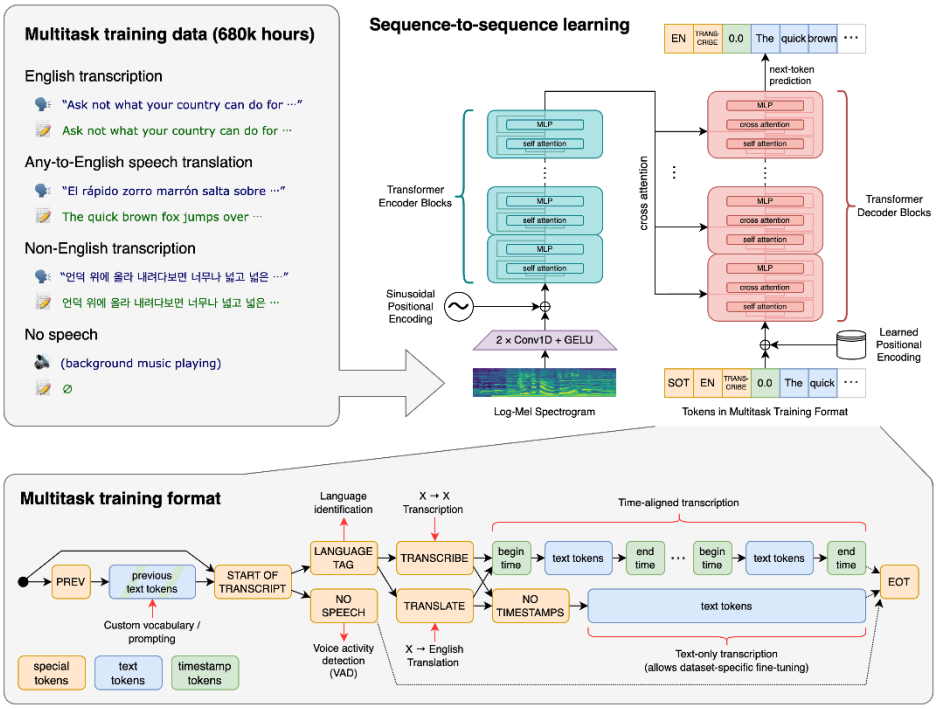
4.9.1. Python Speechrecognition

SpeechRecognition · PyPI

Spraakherkenning wordt geregeld door de SpeechRecognition module. De grote reden om dit te gebruiken in plaats van simpelweg de herkenningsmodule is omdat SpeechRecognition meerdere verschillende spraakherkenners gebruiken kan en dit ook kan verbinden aan een alternatieve audio-invoer.

4.9.2. OpenAI Whisper

GitHub - openai/whisper: Robust Speech Recognition via Large-Scale Weak Supervision



Figuur 1: Een visualisatie van het spraakherkenningsproces van OpenAI Whisper (Radford, Kim, Xu, Brockman, McLeavey & Sutskever, 2023)

Spraakherkenning is zeer ingewikkeld, dus voor meer dan een basis-level uitleg zal gekeken moeten worden naar de pdf. Het idee is dat de audiodata veranderd wordt in een Log-Mel Spectrogram. Dit is een manier van het weergeven van geluidsgolven. Deze worden genormaliseerd op een schaal van -1 tot 1. De encoder verwerkt dit met een korte stam die bestaat uit 2 “convolution layers” met een filter wijdte van 3 en de “GELU” *(Hendrycks & Gimpel, 2016)* activatie functie waar de 2^{e} convolution layer een “stride” heeft van 2. “Sinusoidal position embeddings” worden daaraan toegevoegd, waarna het de “encoder Transformer blocks” toepast. Deze transformer gebruikt “pre-activation residual blocks” *(Child, Gray, Radford & Sutskever, 2019)*, en een laatste laag normalisatie wordt gebruikt op de uitgang van al dit. De decoder gebruikt “learned position embeddings” en “tied input-output token representations” *(Press & Wolf, 2017)*. De encoder en decoder hebben dezelfde wijdte en hoeveelheid transformers. Figuur 1 is een visuele samenvatting van dit hele proces. *(Radford, Kim, Xu, Brockman, McLeavey & Sutskever, 2023)*

4.10. JSON

[json — JSON encoder and decoder — Python 3.12.0 documentation]

JSON, oftewel JavaScript Object Notation is een veelvoorkomend formaat om data op te slaan. Dit formaat wordt voornamelijk gebruikt met dictionaries als het type variabele, zodat één bestand meerdere variabelen kan opslaan en er niet meerdere bestanden nodig zijn voor andere variabelen. Een andere soort variabele waarmee dit ook kan is een lijst, maar het nadeel hiervan is dat alleen de positie in de lijst bepaalt naar welke variabele gekeken wordt, dus het kan alleen gebruikt worden als aangenomen kan worden dat de lijst altijd gesorteerd is.

4.11. Threading

[threading — Thread-based parallelism — Python 3.12.0 documentation]

Threading is een module van de Python standard library. Dit is de meest gebruikte module voor thread-gebaseerd parallelisme. Parallelisme is in deze context het gebruik van parallel processing om meerdere dingen tegelijk te doen. Er zijn verschillende soorten parallellisme in Python, afhankelijk van het abstractieniveau, het type uitvoering en het communicatiemodel. De meest voorkomende opties zijn:

Proces-gebaseerd parallellisme: dit omvat het creëren van meerdere processen die onafhankelijk draaien en communiceren via gedeeld geheugen of het doorgeven van berichten. De multiprocessing-module in de Python standard library is de meest gebruikte module voor dit type parallelisme. Deze wordt niet gelimiteerd door de GIL. Het creëren en schakelen tussen processen kan echter enige problemen veroorzaken.

Thread-gebaseerd parallellisme: hierbij worden meerdere threads gemaakt die gelijktijdig worden uitgevoerd en dezelfde geheugenruimte delen. Thread-gebaseerd parallellisme is vooral goed voor dingen zoals netwerk- of schijfbewerkingen, die niet veel CPU-tijd vereisen. Dit type parallellisme kan vanwege de GIL echter niet profiteren van meerdere CPU's of kernen en kan ook problemen veroorzaken wanneer meerdere threads proberen toegang te krijgen tot dezelfde variabele(n).

Coroutine-gebaseerd parallellisme: dit omvat het creëren van meerdere coroutines die samenwerken en de controle aan elkaar overgeven. Hiervoor wordt de asyncio-module gebruikt om coroutines gemakkelijk te maken en beheren. Deze soort parallelisme is echter niet volledig parallel, aangezien maar één coroutine tegelijk bezig is. Toch wordt het gezien als een type parallelisme omdat het wel gebruikt kan worden om met een functie verder te gaan voordat een andere functie klaar is.

Gespecialiseerde bibliotheken: Er zijn veel modules die gespecialiseerde oplossingen bieden voor parallellisme in Python, zoals numpy, scipy, pandas, scikit-learn, dask, joblib, numba, cython en meer. Deze maken gebruik van optimalisaties op een laag niveau die in standaard Python niet beschikbaar zijn. Zo kan 1 functie meerdere processen tegelijk uitvoeren om ervoor te zorgen dat hij sneller klaar is. Omdat ze zo gespecialiseerd zijn kunnen ze echter alleen voor heel specifieke, meestal wiskundige toepassingen gebruikt worden.

Global Interpreter Lock (GIL)	De GIL is een veiligheidsmaatregel die voorkomt dat meerdere CPU-kernen tegelijkertijd Python-code uitvoeren.

Thread	Een thread is vrij letterlijk een thread op de CPU van een computer. Deze threads kunnen allemaal tegelijkertijd verschillende processen uitvoeren, maar 1 programma kan ook meerdere threads gebruiken. Als iets op daemon thread wordt uitgevoerd zal het originele programma niet stoppen.
Lock	Een lock is een type object die maar door 1 thread tegelijk gebruikt kan worden. Dit wordt gedaan door de functie lock.acquire(). Als hierna een andere thread probeert dezelfde lock te gebruiken moet deze wachten totdat de thread die de lock op dat moment in bezit heeft lock.release() gebruikt.
Local	Een local in deze context is een type object waarin verschillende variabelen in opgeslagen kunnen worden. Deze variabelen zijn echter uniek per thread. Dus als thread 1 bijvoorbeeld local.schaap als “Beeh” definieert zal het voor thread 2 nog niet gedefinieerd zijn.

Tabel 77 Threading begrippen

5. Methode

5.1. App

In de eerste regel van Figuur 2 wordt de app variabele gedefinieerd als een App class-object. Hierna wordt de mainloop van de app instantie gestart. De simpele uitleg daarvoor is dat deze zichtbaar wordt op het scherm. Vanaf hier wordt de basisinformatie van het programma ingesteld zoals te zien is in Figuur 3.

```
app = App()
app.mainloop()
```

Figuur 2 GitHub

```
with open("settings.json") as file:
    settings = load(file)

if settings["remember_me"]:
    remembered = login()
else: remembered = False

class App(customtkinter.CTk):
    def __init__(self):
        super().__init__()
        global settings
        try: customtkinter.set_appearance_mode(settings["appearance"])
        except: pass

        self.iconpath = ImageTk.PhotoImage(file=os.path.join("assets","logo@4x.png"))
        self.wm_iconbitmap()
        self.iconphoto(False, self.iconpath)

        self.title("DaySphere")

        self.geometry("1300x975")

        self.minsize(1300, 975)

        self._view = None

        self.protocol("WM_DELETE_WINDOW", lambda: [save(), close_program(), self.destroy()])

        if remembered : self.switch_view(MainPage)
        else : self.switch_view(LoginPageFrame)
```

Figuur 3 GitHub

Hier worden ten eerste de instellingen (settings) van een extern bestand ingeladen in de eerste twee regels. Ten tweede wordt gekeken naar de boolean “remember_me” in instellingen. Als deze waar is zal de functie login() van client-server communication uitgevoerd worden. Deze zal vervolgens proberen in te loggen met de gegevens die in de instellingen gevonden waren. Als dit lukt zal de boolean “remembered” ook waar worden. Als “remember_me” onwaar was wordt “remembered” ook onwaar en wordt er niet geprobeerd in te loggen.

Na deze backend functies wordt de App class gedefinieerd en inherit het dezelfde functionaliteit als een CustomTkinter app. Er wordt gekeken naar welke informatie er in de instellingen staat over de thema, als deze “light” voor een licht thema of “dark” voor een donker thema. Als hier geen instelling voor wordt gevonden zal het de standaard instelling van je system gebruiken. Hierna wordt het icoon van het programma gekozen, waarvoor de code zoekt naar een bestand genaamd “logo@4x.png” in de “assets” folder op dezelfde plek als het bestand dat het programma start. 4x in de bestandnaam staat voor een afbeelding met een resolutie van 256 bij 256 pixels. Daarna wordt de titel beslist; “DaySphere”, de naam van het programma. Ook wordt de grootte van de applicatie ingesteld bij het opstarten met “self.geometry()”, hierbij wordt gegeven dat de applicatie 1300 bij 975 pixels zal zijn bij opstarten. Dit wordt ook gelijk gebruikt als minimumgrootte, zodat het programma nooit klein genoeg wordt dat het formaat niet meer werkt. Hierna wordt de privé variabele “_view” geïnitialiseerd. Deze variabele staat voor welke pagina op dat moment weergegeven wordt. Ook wordt er een protocol ingesteld die, wanneer het programma op de normale manier gesloten wordt, de nog niet opgeslagen data opslaat en de verbinding sluit. Als laatste wordt met de boolean “remembered” besloten welke pagina wordt afgebeeld door middel van de App class functie “switch_view()”. Als remembered waar is wordt het de “Main Page”, zo niet de “Login Page”.

In Figuur 4 is de definitie van de App class functie switch_view() te zien. Het doel van deze functie is om de pagina die weergegeven wordt, oftewel self._view, te veranderen. De functie neemt één variabele die naar de class van een pagina wijst, dit zal dus een van de 7 pagina’s zijn: Login Page, Account Creation Page, Main Page, Terminal Page, Calendar Page, Settings Page en de Day View. Deze nieuwe variabele heet als eerst “view”, maar wordt meteen omgezet naar de variabele “new_view”, die gedefinieerd wordt als een geïnitialiseerde instantie van view met de huidige App class als parent. Daarna kijkt de functie of op dat moment al een pagina weergegeven wordt. Zo ja wordt het scherm geleegd om ruimte te maken voor het nieuwe scherm. Hierna wordt self._view veranderd naar het scherm in “new_view”. Als laatst wordt dit nieuwe scherm op de hele applicatie gebruikt in plaats van maar een deel daarvan, dit wordt gedaan door de laatste drie regels code. De eerste twee regels nemen het de hele applicatie in beslag voor de weergave, en de laatste regel zorgt ervoor dat deze in het midden van het scherm blijft, en niet op een willekeurig plek in de applicatie.

```
def switch_view(self, view):
        new_view = view(self)
        if self._view is not None:
            self._view.destroy()
        self._view = new_view

        self.grid_rowconfigure(0, weight=1)
        self.grid_columnconfigure(0, weight=1)

        self._view.grid(sticky="nesw")
```

Figuur 4 GitHub

5.2. Login Page

De Login Page bevat alle benodigdheden om in een account in te loggen. Voornamelijk de entries voor een gebruikersnaam en wachtwoord (Figuur 5).

```
class LoginPage(customtkinter.CTkFrame):
    def __init__(self, master):
        customtkinter.CTkFrame.__init__(self, master)

        self.user_entry = customtkinter.CTkEntry(self, placeholder_text="")
        self.user_entry.grid(row=3, column=0, padx=20, pady=(0, 8))
        self.user_entry.bind("<Return>", command=lambda x: self.pass_entry.focus_set())
        self.user_entry.focus_set()

        self.pass_entry = customtkinter.CTkEntry(self, placeholder_text="", show='*')
        self.pass_entry.grid(row=5, column=0, padx=20, pady=(0, 8))
        self.pass_entry.bind("<Return>",
                             command=lambda x: [master.master.switch_view(MainPage) if
                                                login(self.user_entry.get(),
                                                self.pass_entry.get(), box_state)
                                               else master.error_label.grid(row=1, column=0)])
```

Figuur 5 GitHub

Naast de mogelijkheid om tekst in de entries te kunnen plaatsen is het belangrijkste hier zijn de binds. De binds van de entries voor gebruikersnaam en wachtwoord zijn voornamelijk voor toegankelijkheid. De bind van de gebruikersnaam entry zorgt het ervoor dat wanneer enter ingedrukt wordt terwijl de focus in de entry zit dat deze naar de wachtwoord entry springt. De bind van de wachtwoord entry zorgt ervoor dat de enter knop dezelfde functionaliteit heeft als de “Login” knop.

“self.hide_pass_button” wordt gebruikt om je wachtwoord te laten zien of weer verstoppen wanneer er een tweede keer op gedrukt wordt. Hiervoor wordt de functie “toggle_hidden_login(self)” gebruikt (Figuur 6).

```
self.hide_pass_button = customtkinter.CTkButton(self, text="", width=0,
                                                 command=lambda: toggle_hidden_login(self),
                                                 image= customtkinter.CTkImage(
                                                 light_image=Image.open("assets/hidden_white.png"),
                                                 dark_image=Image.open("assets/hidden_black.png")),
                                                 fg_color="transparent")
self.hide_pass_button.grid(row=4, column=0, padx=12, pady=0, sticky="e")
```

Figuur 6 GitHub

Ook heeft het een checkbox die wanneer ingelogd wordt ervoor zorgt dat wanneer hij aan staat de inloggegevens opgeslagen worden en “remember_me” in de instellingen omschakelt (Figuur 7).

```
if box_state: check_box_state = customtkinter.StringVar(value="on")
else: check_box_state = customtkinter.StringVar(value="off")
self.remember_me_checkbox = customtkinter.CTkCheckBox(self, text="Remember me",
                                                       command=checkbox_event,
                                                       variable=check_box_state,
                                                       onvalue="on",
                                                       offvalue="off")
self.remember_me_checkbox.grid(row=7, column=0, padx=20, pady=12, sticky="w")
```

Figuur 7 GitHub

Bij het inloggen wordt de informatie in de entries samen met de staat van de checkbox naar het client bestand gestuurd met de login() Functie. (Figuur 8).

```
self.login_button = customtkinter.CTkButton(self, text="Log in",
                                             command=lambda: [master.master.switch_view(MainPage)
                                             if login(self.user_entry.get(), self.pass_entry.get(),
                                                   box_state)
                                             else master.error_label.grid(row=2, column=1)])
self.login_button.grid(row=7, column=0, padx=20, pady=8)
```

Figuur 8 GitHub

Voor het geval dat de gebruiker nog geen account heeft is er een knop om naar de Account Creation Page te navigeren (Figuur 9).

```
self.acc_creation_menu_button = customtkinter.CTkButton(self, text="Create account", command=lamba:
                                                         master.master.switch_view(AccountCreation
                                                         PageFrame), fg_color="transparent",
                                                         hover=False, text_color="black"
                                                         if master.master._get_appearance_mode() ==
                                                             "light" else "white")
self.acc_creation_menu_button.grid(row=9, column=0, padx=20, pady=(12, 20))
```

Figuur 9 GitHub

5.3. Account Creation Page

De Account Creation Page bevat drie entries, één voor de gebruikersnaam van je nieuwe account.

```
class AccountCreationPage(customtkinter.CTkFrame):
    def __init__(self, master):
        customtkinter.CTkFrame.__init__(self, master)

        self.username = customtkinter.StringVar()
        self.username.trace_add("write", lambda x, y, z: password_verification(self))

        self.user_entry = customtkinter.CTkEntry(self, placeholder_text="",
                                                 textvariable=self.username)
        self.user_entry.grid(row=2, column=0, padx=20, pady=(0, 8))
        self.user_entry.bind("<Return>", lambda x: self.pass_entry.focus_set())
```

Figuur 10 GitHub

En twee andere om je wachtwoord te beslissen en verifiëren door hetzelfde wachtwoord nog een keer in te voeren. Dit wordt gedaan om het probleem te vermijden dat de gebruiker een typfout maakt tijdens het beslissen van zijn wachtwoord daardoor later niet meer kan inloggen.

```
self.pass_entry = customtkinter.CTkEntry(self, placeholder_text="Password",
                                         show='*', textvariable=self.password)
self.pass_entry.grid(row=4, column=0, padx=20, pady=(0, 8))
self.pass_entry.bind("<Return>", lambda x: self.pass_ver_entry.focus_set())

self.pass_ver_label = customtkinter.CTkLabel(self, text="Verify password",
                                             font=customtkinter.CTkFont(self, size=12))
self.pass_ver_label.grid(row=5, column=0)

self.pass_ver_entry = customtkinter.CTkEntry(self, placeholder_text="Verify password",
                                             show='*', textvariable=self.ver_password)
self.pass_ver_entry.grid(row=6, column=0, padx=20, pady=(0, 8))
```

Figuur 11 GitHub

Hiernaast bevat het nog drie knoppen, een hide passwordknop net als die op de Login Page, een “Create Account” knop en een terug knop. De “Create Account” knop staat standaard uit, maar kan op gedrukt worden wanneer alle entries zijn ingevuld.

```
self.hide_pass_button = customtkinter.CTkButton(self, text="", width=0, command=lambda:
                                                 toggle_hidden_creation(self), image=
                                                 customtkinter.CTkImage(light_image=
                                                 Image.open("assets/hidden_white.png"),
                                                 dark_image=Image.open("assets/hidden_black.png")),
                                                 fg_color="transparent")
self.hide_pass_button.grid(row=7, column=0, padx=12, pady=4, sticky="e")

self.create_account_button = customtkinter.CTkButton(self, text="Create Account", state="disabled",
                                                      command=lambda:
                                                      [master.master.switch_view(MainPage)
                                                      if create_account(self.user_entry.get(),
                                                          self.pass_entry.get(), box_state)
                                                      else [master.error_label.grid(row=2, column=0,
                                                            pady=(0, 12)),
                                                            master.offset_label.grid(row=0,
                                                                                     column=0),]])
self.create_account_button.grid(row=8, column=0, padx=20, pady=8)

self.go_back_to_login_button = customtkinter.CTkButton(self, text="Go back", command=lambda:
                                                        master.master.switch_view(LoginPageFrame),
                                                        fg_color="transparent", hover_color="red",
                                                        text_color="black"
                                                        if master.master._get_appearance_mode() ==
                                                        "light"
                                                        else "white")
self.go_back_to_login_button.grid(row=9, column=0, padx=20, pady=(12,20))
```

Figuur 12 GitHub

Daarnaast is er ook en een remember_me checkbox die dezelfde functionaliteit heeft als de checkbox op de Login Page.

```
self.pass_ver_entry = customtkinter.CTkEntry(self, placeholder_text="Verify password", show='*',
                                             textvariable=self.ver_password)
self.pass_ver_entry.grid(row=6, column=0, padx=20, pady=(0, 8))

if box_state: check_box_state = customtkinter.StringVar(value="on")
else: check_box_state = customtkinter.StringVar(value="off")
self.remember_me_checkbox = customtkinter.CTkCheckBox(self, text="Remember me",
                                                      command=checkbox_event,
                                                      variable=check_box_state, onvalue="on",
                                                      offvalue="off")
self.remember_me_checkbox.grid(row=7, column=0, padx=20, pady=12, sticky="w")
```

Figuur 13 GitHub

5.4. Main Page & Sidebar

De Main Page is de eerste pagina die wordt laten zien wanneer ingelogd is. Deze pagina bevat zelf niets, maar het laat wel voor het eerst de “Sidebar” zien. De Sidebar bevat knoppen die toegang geven naar alle schermen (behalve de Login Page, Account Creation Page en de Day View). Dit is waarmee de gebruiker zal navigeren door het programma.

```
class Sidebar(customtkinter.CTkFrame):
    def __init__(self, master):
        customtkinter.CTkFrame.__init__(self, master, bg_color="black", fg_color="black")

        from GUI.main import MainPage, SettingsPageFrame
        from GUI.terminal import TerminalPageFrame
        from GUI.calendar import CalendarPage, mass_update_information, selected_date
        self.grid_rowconfigure(3, weight=1)
```

Figuur 14 GitHub

De bovenste knop met als plaatje het logo van DaySphere, brengt je naar de Main Page.

```
self.home_button = customtkinter.CTkButton(self,
command=lambda:[mass_update_information(master.master._view.day_view,
          selected_date), master.master.switch_view(MainPage)]
          if ".!dayviewpage" in str(master.master._view)
          else master.master.switch_view(MainPage), hover_color="black",
          image=customtkinter.CTkImage(light_image=Image.open("assets/logo@4x.png"),
                                       dark_image=Image.open("assets/logo@4x.png"),
          size=(80, 80)), height=120, fg_color="black", bg_color="black", text="")
self.home_button.grid(row=0, column=0)
```

Figuur 15 GitHub

Daaronder staat de knop om naar de Terminal Page te gaan.

```
self.terminal_button = customtkinter.CTkButton(self,
command=lambda:[mass_update_information(master.master._view.day_view,
          selected_date), master.master.switch_view(TerminalPageFrame)]
          if ".!dayviewpage" in str(master.master._view)
          else master.master.switch_view(TerminalPageFrame), hover_color="black",
          image=customtkinter.CTkImage(light_image=Image.open("assets/terminal_white.png"),
                                       dark_image=Image.open("assets/terminal_white.png"),
          size=(40, 40)), height=80, fg_color="black", bg_color="black", text="")
self.home_button.grid(row=1, column=0)
```

Figuur 16 GitHub

Daarna is de knop voor de kalender, die gaat naar Calendar Page.

```
self.calendar_button = customtkinter.CTkButton(self, command=lambda:
[mass_update_information(master.master._view.day_view,
           selected_date), master.master.switch_view(CalendarPage)]
           if ".!dayviewpage" in str(master.master._view)
           else master.master.switch_view(CalendarPage), hover_color="black",
           image=customtkinter.CTkImage(light_image=Image.open("assets/calendar_white.png"),
                                        dark_image=Image.open("assets/calendar_white.png"),
           size=(45, 45)), height=80, fg_color="black", bg_color="black", text="")
self.calendar_button.grid(row=2, column=0)
```

Figuur 17 GitHub

Als laatste staat aan de onderkant een knop die je naar de Settings Page brengt. Om de knop onderin te zetten wordt een offset_label gebruikt die alle ruimte tussen de andere knoppen en deze knop inneemt.

```
self.offset_label = customtkinter.CTkLabel(self, text="")
self.offset_label.grid(row=3, column=0)

self.settings_button = customtkinter.CTkButton(self, command=lambda:
[mass_update_information(master.master._view.day_view,
           selected_date), master.master.switch_view(SettingsPageFrame)]
           if ".!dayviewpage" in str(master.master._view)
           else master.master.switch_view(SettingsPageFrame), hover_color="black",
           image=customtkinter.CTkImage(light_image=Image.open("assets/gear_white.png"),
                                        dark_image=Image.open("assets/gear_white.png"),
           size=(40, 40)),  height=80, fg_color="black", bg_color="black", text="")
self.settings_button.grid(row=4, column=0, pady=(0, 10))
```

Figuur 18 GitHub

5.5. Terminal Page

De Terminal Page is waar alle functionaliteit binnen DaySphere bereikbaar is. Voor het huidige ontwerp is dat alleen de kalender en de instellingen. Deze functionaliteit is toepasbaar op twee manieren, spraakherkenning en een entry. Spraakherkenning is hier de belangrijkste, maar de entry is ook nuttig om bijvoorbeeld in drukke ruimtes nog steeds dezelfde functionaliteit te hebben. Op de Terminal Page zelf staan maar twee CTk Classes; een CTkButton en een CTkEntry. De CTkButton wordt gebruikt om spraakherkenning te starten of te stoppen. Nadat het is aangezet blijft de spraakherkenning ook actief op andere pagina's, totdat het door deze knop weer wordt uitgezet.

```
class TerminalPage(customtkinter.CTkFrame):
    def __init__(self, master):
        customtkinter.CTkFrame.__init__(self, master, fg_color="transparent")

        global app_instance, dev_tools
        app_instance = self
        dev_tools = False

        self.columnconfigure(0, weight=1)

        from speechrecognition.main import VR

        self.mic_toggle_button = customtkinter.CTkButton(self, text="", command=lambda:
                                                         [toggle_mic(self), VR.main()],
                                                         image=customtkinter.CTkImage(
                                                         light_image=Image.open
                                                         ("assets/mic_closed.png"),
                                                         dark_image=Image.open
                                                         ("assets/mic_closed.png")))
        self.mic_toggle_button.grid(row=10, column=1, padx=20, pady=20)

mic_state = False
def toggle_mic(app_instance):
    global mic_state
    if mic_state:
        mic_state = False
        app_instance.mic_toggle_button
        .configure(image=customtkinter.CTkImage(light_image=Image.open
        ("assets/mic_closed.png"), dark_image=Image.open("assets/mic_closed.png")))
    else:
        mic_state = True
        app_instance.mic_toggle_button
         .configure(image=customtkinter.CTkImage(light_image=Image.open
         ("assets/mic_open.png"), dark_image=Image.open("assets/mic_open.png")))
```

Figuur 19 GitHub

De CTkEntry heeft dezelfde functionaliteit als spraakherkenning, behalve dat het niet zoekt naar de initiële keyword “DaySphere”. Ook bevat deze entry autocorrect functionaliteit, voor de gebruikers die niet zeker weten hoe een keyword gespeld hoort te worden.

```
self.terminal_entry = customtkinter.CTkEntry(self, width=800, height=40,
                                             textvariable=terminal_command,
                                             font=customtkinter.CTkFont(self, size=20))
self.terminal_entry.grid(row=10, column=0, padx=0, pady=40)
self.terminal_entry.bind("<Tab>", command=lambda x: [auto_completion(self),
self.terminal_entry.focus_set()])
self.terminal_entry.bind("<Return>", command=lambda x:
                         feedback(self.terminal_entry.get(), False)
                         if auto_completed_text == ""
                         else auto_completion(self))
self.terminal_entry.focus_set()

def auto_completion(app_instance):
    global auto_completed_text
    global updating
    updating = True
    if auto_completed_text == "": return
    else:
        app_instance.preview_label.configure(text="")
        app_instance.terminal_entry.delete(0, END)
        print(auto_completed_text)
        app_instance.terminal_entry.insert(0, auto_completed_text)
        auto_completed_text = ""
        updating = False

def feedback(terminal_command, voice: bool):
    global app_instance
    if terminal_command == "": return
    app_instance.output_label.configure(text=runfromstring(terminal_command))
    app_instance.command_label.configure(text=terminal_command)
    if not voice:
        app_instance.terminal_entry.delete(0, END)
```

Figuur 20 GitHub

5.6. Calendar Page

De Calendar Page bevat alle informatie uit de kalender die correspondeert aan de huidige gebruiker. Deze informatie wordt per maand weergeven. Per maand worden 5 weken weergegeven om ervoor te zorgen dat elke maand past, zelfs als deze begint op een zondag. Wanneer de Calendar Page inlaadt stuurt het een verzoek naar de server om de data te sturen. De server weet al welk account het is dat de data aanvraagt, aangezien dit al is opgeslagen wanneer ingelogd werd. Zo wordt het moeilijk om het programma te laten denken dat ingelogd is op een account dat niet van jou is en zo iemands data te stelen. De hiërarchie van de Calendar Page is als volgt; Calendar > Calendar Weeks > Calendar Days. De Calendar Days bevatten de daadwerkelijke informatie, de rest wordt gebruikt om deze te formatteren.

```
class Calendar(customtkinter.CTkFrame):
    def __init__(self, master):
        customtkinter.CTkFrame.__init__(self, master, fg_color="black"
                                        if master._get_appearance_mode() == "dark"
                                        else "white")

        self.grid_rowconfigure([0, 1, 2, 3, 4], weight=1)
        self.grid_columnconfigure(0, weight=1)

        self.calendar_week1 = CalendarWeek(self)
        self.calendar_week1.grid(row=1, column=0, columnspan=3)

        self.calendar_week2 = CalendarWeek(self)
        self.calendar_week2.grid(row=2, column=0, columnspan=3)

        self.calendar_week3 = CalendarWeek(self)
        self.calendar_week3.grid(row=3, column=0, columnspan=3)

        self.calendar_week4 = CalendarWeek(self)
        self.calendar_week4.grid(row=4, column=0, columnspan=3)

        self.calendar_week5 = CalendarWeek(self)
        self.calendar_week5.grid(row=5, column=0, columnspan=3)
```

Figuur 21 GitHub

De fg-color beslist de kleur van de voorgrond van de CTkFrame. Dit wordt daarom verbonden met het huidige thema van de applicatie die te vinden is in de appearance_mode variabele. Hierna worden de Calendar Weeks geïnitialiseerd en naar het scherm geschreven binnen een raster. De reden dat de weergave op deze manier wordt gedaan is omdat er altijd exact 5 weken tegelijk afgebeeld worden. Hierdoor zijn ook de “weights” van grid_rowconfigure en grid_columnconfigure makkelijk in te stellen.

Binnen de Calendar Weeks zitten alle Calendar Days.

```
class CalendarWeek(customtkinter.CTkFrame):
    def __init__(self, master):
        customtkinter.CTkFrame.__init__(self, master, fg_color="black")

        self.grid_rowconfigure(0, weight=1)
        self.grid_columnconfigure([0, 1, 2, 3, 4, 5, 6], weight=1)

        self.calendar_day1 = CalendarDay(self)
        self.calendar_day1.grid(row=0, column=0)

        self.calendar_day2 = CalendarDay(self)
        self.calendar_day2.grid(row=0, column=1)

        self.calendar_day3 = CalendarDay(self)
        self.calendar_day3.grid(row=0, column=2)

        self.calendar_day4 = CalendarDay(self)
        self.calendar_day4.grid(row=0, column=3)

        self.calendar_day5 = CalendarDay(self)
        self.calendar_day5.grid(row=0, column=4)

        self.calendar_day6 = CalendarDay(self)
        self.calendar_day6.grid(row=0, column=5)

        self.calendar_day7 = CalendarDay(self)
        self.calendar_day7.grid(row=0, column=6)
```

Figuur 22 GitHub

Weer kunnen de posities dagen hier vast gedefinieerd worden in een raster omdat één week nooit meer of minder dan zeven dagen heeft. Hetzelfde geldt ook weer voor de weights.

```
class CalendarDay(customtkinter.CTkButton):
    def __init__(self, master):
        customtkinter.CTkButton.__init__(self, master, hover=False, fg_color="black"
                                         if master.master.master.master._get_appearance_mode() == "dark"
                                         else "white", border_width=3, border_color="black"
                                         if master.master.master.master._get_appearance_mode() == "light"
                                         else "white",
                                         corner_radius=0, text="", width=254, height=201, border_spacing=0,
                                         command=lambda: [switch_day_view(self.date),
                                         master.master.master.master.switch_view(DayViewPage)])

        self.date = get_calendar_day()

        global original_month, information

        self.info_label1 = customtkinter.CTkButton(self, text="", font= customtkinter.CTkFont(size=16),
                                                   fg_color="transparent", hover=False, command=lambda:
                                                   [switch_day_view(self.date),
                                                   master.master.master.master.switch_view(DayViewPage)],
                                                   text_color="white"
                                                   if master._get_appearance_mode() == "dark"
                                                   else "black")
        self.info_label1.grid(row=1, column=0, padx=(12, 0), sticky="nw")

        self.info_label2 = customtkinter.CTkButton(self, text="", font=customtkinter.CTkFont(size=16),
                                                   fg_color="transparent", hover=False, command=lambda:
                                                   [switch_day_view(self.date),
                                                   master.master.master.master.switch_view(DayViewPage)],
                                                   text_color="white"
                                                   if master._get_appearance_mode() == "dark" else "black")
        self.info_label2.grid(row=2, column=0, padx=(12, 0), sticky="nw")

        self.info_label3 = customtkinter.CTkButton(self, text="", font=customtkinter.CTkFont(size=16),
                                                   fg_color="transparent", hover=False, command=lambda:
                                                   [switch_day_view(self.date),
                                                   master.master.master.master.switch_view(DayViewPage)],
                                                   text_color="white"
                                                   if master._get_appearance_mode() == "dark" else "black")
        self.info_label3.grid(row=3, column=0, padx=(12, 0), sticky="nw")

        self.more_info_label = customtkinter.CTkButton(self, text="", font=customtkinter.CTkFont(size=16),
                                                       fg_color="transparent", hover=False, command=lambda:
                                                       [switch_day_view(self.date),
                                                       master.master.master.master.switch_view(DayViewPage)],
                                                       text_color="white"
                                                       if master._get_appearance_mode() == "dark"
                                                       else "black")
        self.more_info_label.grid(row=4, column=0, padx=(12, 0), sticky="nw", pady=(0, 12))

        get_information(self, self.date)
```

Figuur 23 GitHub

In de definitie van Calendar Day in Figuur 23 is te zien dat Calendar Day de CTkButton class inherit. Dit wordt gedaan omdat het de bedoeling is dat wanneer op een dag in de kalender gedrukt wordt deze de pagina moet verwisselen naar een waarin de details van iedere activiteit te zien en aan te passen is. Iedere keer dat een Calendar Day wordt initialiseert definieert het een “date” variabele die de dag, maand en jaar representeert in een Date class-object die gekregen wordt door get_calendar_day().

```
class Date():
    def __init__(self):
        global month_offset, original_month, original_year
        self.month = datetime.datetime.now().month
        self.year  = datetime.datetime.now().year
        if self.month + month_offset > 12:
            self.year += divisible(self.month + month_offset, 12)
            original_year = self.year
            self.month = (self.month + month_offset) % 12
            original_month = self.month
        else:
            original_year = self.year
            self.month += month_offset
            original_month = self.month
        self.day = get_starting_day(self.month, self.year)
        if self.day != 0:
            self.month -= 1
```

Figuur 24 GitHub

De Date class bevat zelf alleen de huidige datum zoals gekregen van datetime.now() en wat logica voor als de afgebeelde maand verwisseld moet worden. Het is na deze logica ook belangrijk om te bepalen welke datum de maandag is van de week waarin de huidige maand begint. Hiervoor wordt get_starting_day() gebruikt.

```
def get_starting_day(month, year):
    weekday = datetime.datetime(year, month, 1).weekday()
    if weekday == 0:
        return 0
    else:
        if month % 2 == 0:
            return 30 - weekday
        else:
            return 31 - weekday
```

Figuur 25 GitHub

Dit is belangrijk om ervoor te zorgen dat de kalender begint op een maandag, en eindigt op een zondag.

get_calendar_day() gebruikt dan de dag, maand en jaar zoals gedefinieerd in de Date class om dynamisch bij te houden welke datum de volgende dag is, wat ingewikkelder is dan het lijkt omdat niet iedere maand even lang is.

```
counter = 35

def get_calendar_day():
    global date, counter
    if counter == 35:
        date = Date()
        counter = 0
    if date.month == 2:
        if date.year % 4 == 0:
            if date.day + 1 > 29:
                date.month += 1
                date.day = 1
            else:
                date.day += 1
        else:
            if date.day + 1 > 28:
                date.month += 1
                date.day = 1
            else:
                date.day += 1
    elif date.month % 2 == 0:
        if date.day + 1 > 31:
            if date.month + 1 > 12:
                date.year += 1
                date.month = 1
                date.day = 1
            else:
                date.month += 1
                date.day = 1
        else:
            date.day += 1
    else:
        if date.day + 1 > 30:
            date.month += 1
            date.day = 1
        else:
            date.day += 1
    counter += 1
    return [date.day, date.month, date.year]
```

Figuur 26 GitHub

Er bestaan veel elegantere oplossingen voor dit probleem, maar voor simpliciteit is het hier gedaan door een beetje pure rekenkunde.

Nadat get_calendar_day() de datum heeft teruggestuurd naar de Calendar Day instantie worden alle variabelen die nodig zijn om de informatie over de dag in hierin te tonen gedefinieerd, maar nog niet naar het scherm schreven omdat de Calendar Day de daadwerkelijke activiteiten nog niet kent. Dit wordt pas aan het einde van de Calendar Day allemaal tegelijk toegevoegd met get_information(). Aan deze functie worden de Calendar Day class self en de datum die het bevat gegeven, zodat de informatie door deze externe functie veranderd kan worden.

```
def get_information(day_instance, date):
    global information
    view_param = 0
    if day_instance.master.master.master.master.state() == "zoomed":
        view_param = 11
    try:
        for day_information in information[f"{date[0]}/{date[1]}/{date[2]}"]:
            if len(day_information) > 10 + view_param:
                day_information = day_information[:10 + view_param] + "..."
            if (day_instance.info_label1._text == ""
            or day_instance.info_label1._text == day_information):
                    day_instance.info_label1.configure(text=day_information)
            elif (day_instance.info_label2._text == ""
            or day_instance.info_label2._text == day_information):
                    day_instance.info_label2.configure(text=day_information)
            elif (day_instance.info_label3._text == ""
            or day_instance.info_label3._text == day_information):
                    day_instance.info_label3.configure(text=day_information)
            else:
                day_instance.more_info_label.configure(text="More info...")
    except: pass
```

Figuur 27 GitHub

Deze functie verandert de al geïnitialiseerde variabele binnen Calendar Day met de .configure() functie van deze variabele. Hiervoor gebruikt het de globale variabele information, wat correspondeert aan de informatie gegeven door de server. Als deze variabele geen waarde heeft voor die datum zal het bij de try except statement het try: gedeelte overslaan en in plaats daarvan naar except gaan en stoppen met veranderingen maken. Als het wel informatie bevat zullen de eerste drie planningen voor die dag weergegeven worden. In het geval dat er meer dan drie activiteiten op die dag gepland zijn is dit te zien aan de “More info…” die eronder komt te staan. Ook wordt gekeken naar view_param, dit is om zeker te maken dat de tekst niet te groot is om in een Calendar Day weer te geven.

Binnen Calendar Day zelf wordt ook gecheckt of er activiteiten op die datum staan met een try except statement. Dit wordt gedaan om een • naast de datum te zetten wanneer die informatie bestaat, zodat er direct gezien kan worden dat de Calendar Day informatie bevat.

```
try:
    self.day_label = customtkinter.CTkButton(self, text=f"{self.date[0]} •"
    if information[f"{self.date[0]}/{self.date[1]}/{self.date[2]}"][0]
    else self.date, font=customtkinter.CTkFont(size=30), fg_color="transparent",
    text_color="#1976D2" if datetime.datetime.now().day == self.date[0]
    and datetime.datetime.now().month == self.date[1]
    and datetime.datetime.now().year == self.date[2]
    else "white" if master.master.master.master._get_appearance_mode() == "dark"
    else "black", hover=False, width=30, command=lambda: [switch_day_view(self.date),
    master.master.master.master.switch_view(DayViewPage)])
except:
    self.day_label = customtkinter.CTkButton(self, text=self.date[0],
    font=customtkinter.CTkFont(size=30), fg_color="transparent", text_color="#1976D2"
    if datetime.datetime.now().day == self.date[0] and
    datetime.datetime.now().month == self.date[1] and
    datetime.datetime.now().year == self.date[2]
    else "white" if master.master.master.master._get_appearance_mode() == "dark"
    else "black", hover=False, width=30, command=lambda: [switch_day_view(self.date),
    master.master.master.master.switch_view(DayViewPage)])
        self.day_label.grid(row=0, column=0, sticky="nw", padx=(12, 0), pady=(8, 0))
```

Figuur 28 GitHub

Het belangrijkste hier is de “text”, waar self.date wordt gebruikt om de dag van de maand aan te geven. Verder wordt gekeken of deze dag buiten de maand van de weergave valt. In dit geval wordt de kleur van de tekst grijs. Zo kan er onderscheid worden gemaakt tussen de dagen die net buiten de huidige maand vallen maar nog wel te zien zijn en de dagen echt binnen deze maand.

De Calendar Page bevat ook functionaliteit om van maand te wisselen zodat verschillende maanden weergegeven kunnen worden.

```
if self.master.master.state() == "zoomed":
    self.view_button_left = customtkinter.CTkButton
                            (self, width=35, height=35, text="", bg_color="black" if
                             master._get_appearance_mode() == "dark" else "white",
                             corner_radius=12, command=lambda: switch_months(self, -1),
                             image=CustomTkinter.CTkImage(light_image=Image.open
                             ("assets/arrow_left.png"),
                             dark_image=Image.open("assets/arrow_left.png")))
else:
    self.view_button_left = customtkinter.CTkButton
                            (self, width=20, height=20, text="", bg_color="black" if
                             master._get_appearance_mode() == "dark" else "white",
                             corner_radius=6, command=lambda: switch_months(self, -1),
                             image=customtkinter.CTkImage(light_image=Image.open
                             ("assets/arrow_left.png"),
                             dark_image=Image.open("assets/arrow_left.png")))

self.view_button_left.grid(row=0, column=1, sticky="ne", padx=(10, 0), pady=10)

if self.master.master.state() == "zoomed":
    self.view_button_right = customtkinter.CTkButton
                            (self, width=35, height=35, text="", bg_color="black" if
                             master._get_appearance_mode() == "dark" else "white",
                             corner_radius=12, command=lambda: switch_months(self, 1),
                             image=customtkinter.CTkImage(light_image=Image.open
                             ("assets/arrow_right.png"),
                             dark_image=Image.open("assets/arrow_right.png")))
else:
    self.view_button_right = customtkinter.CTkButton
                            (self, width=20, height=20, text="", bg_color="black" if
                             master._get_appearance_mode() == "dark" else "white",
                             corner_radius=6, command=lambda: switch_months(self, 1),
                             image=customtkinter.CTkImage(light_image=Image.open
                             ("assets/arrow_right.png"),
                             dark_image=Image.open("assets/arrow_right.png")))

self.view_button_right.grid(row=0, column=2, sticky="ne", padx=10, pady=10)
```

Figuur 29 GitHub

Bij initialisatie wordt gekeken of de applicatie het hele scherm opneemt of niet. Dit wordt gedaan om de grootte van de knoppen te bepalen. Wanneer de één van de twee knoppen wordt ingedrukt, links of rechts, wordt switch_months() gebruikt.

```
def switch_months(calendar_instance, switch):
    global month_offset
    month_offset += switch
    calendar_instance.master.master.switch_view(CalendarPage)
```

Figuur 30 GitHub

switch_months() verandert een globale variabele month_offset, die bepaalt hoe ver de huidige maand verwijderd is van de maand gegeven door Datetime. Om de kalender te verversen wordt het opnieuw geïnitialiseerd.

5.7. Day View

In totaal zijn in de Calendar Page dus 35 verschillende dagen tegelijk te zien. Deze dagen in het maandoverzicht zijn zoals eerder genoemd ook knoppen die, wanneer erop gedrukt wordt de gebruiker naar de Day View brengen.

```
class DayView(customtkinter.CTkFrame):
    def __init__(self, master):
        customtkinter.CTkFrame.__init__(self, master, fg_color="transparent")

    try:
        self.entry_trace1 = customtkinter.StringVar(self,
             information[f"{selected_date[0]}/{selected_date[1]}/{selected_date[2]}"][0])
        (…)
        self.entry_trace25 = customtkinter.StringVar(self,
            information[f"{selected_date[0]}/{selected_date[1]}/{selected_date[2]}"][24])
        except: pass
```

Figuur 31 GitHub

De Day View is een nieuw scherm dat 26 activiteiten voor een dag kan bevatten. Dit is een vaste hoeveelheid, ook al zou het beter zijn als het dynamisch was. De (…) representeert alle entry_traces tussen 1 en 25. Wanneer een entry_trace wordt gedefinieerd wordt binnen de information variabele gezocht bijbehorende waarde van de key van de datum. Als dit niet bestaat stoppen de entry_traces met gedefinieerd worden, belangrijk is om ervoor te zorgen dat het voor de gebruiker lijkt dat het aantal entries en het opslaan van informatie dynamisch gedaan wordt. Trace nummer 26 wordt last_entry_trace genoemd, want deze werkt iets anders dan de rest. Nadat deze ingevuld is moet namelijk niet net als bij de rest een nieuwe entry verschijnen waarin de volgende activiteit geschreven kan worden.

```
self.last_entry_index = 1
try:
    self.entry1 = customtkinter.CTkEntry(self, width=800, height=30, placeholder_text="",
                                         textvariable=self.entry_trace1,
                                         font=customtkinter.CTkFont(size=20),
                                         fg_color="#1D1E1E"
                                         if master.master._get_appearance_mode() == "dark"
                                         else "white", border_color="Black")

     self.entry1.grid(row=1, column=1, sticky="nw")
     self.entry1.bind("<Return>", command=lambda x: update_information
                      (f"{selected_date[0]}/{selected_date[1]}/{selected_date[2]}",
                      self.entry1.get(), 0))
     self.last_entry_index = 2
     (…)
     self.entry25 = customtkinter.CTkEntry(self, width=800, height=30, placeholder_text="",
                                         textvariable=self.entry_trace1,
                                         font=customtkinter.CTkFont(size=20),
                                         fg_color="#1D1E1E"
                                         if master.master._get_appearance_mode() == "dark"
                                         else "white", border_color="Black")

     self.entry25.grid(row=1, column=1, sticky="nw")
     self.entry25.bind("<Return>", command=lambda x: update_information
                      (f"{selected_date[0]}/{selected_date[1]}/{selected_date[2]}",
                      self.entry1.get(), 0))
     self.last_entry_index = 26
except: pass
```

Figuur 332 GitHub

In Figuur 32 wordt bij het definiëren van een variabele gekeken of de corresponderende entry_trace van de entry te vinden is. Als er geen entry_trace wordt gevonden stopt het definiëren van entries. Dit is ook waarom tussen iedere entry variabel de last_entry_index handmatig veranderd wordt zodat het weet wat de laatste entry op de pagina is. Dit is nodig om te bepalen waar deze geplaatst moet worden.

```
try:
    self.last_entry_trace = customtkinter.StringVar(self, information
  [f"{selected_date[0]}/{selected_date[1]}/{selected_date[2]}"][self.last_entry_index - 1])
except:
    self.last_entry_trace = customtkinter.StringVar(self)
    self.last_entry = customtkinter.CTkEntry(self, width=800, height=30,
                                             placeholder_text="",
                                             textvariable=self.last_entry_trace,
                                             font=customtkinter.CTkFont(size=20),
                                             fg_color="#1D1E1E" if
                                             master.master._get_appearance_mode() == "dark"
                                             else "white", border_color="Black")

    self.last_entry.grid(row=self.last_entry_index, column=1, sticky="nw")
    self.last_entry.bind("<Return>", command=lambda x: update_information
                         (f"{selected_date[0]}/{selected_date[1]}/{selected_date[2]}",
                         self.last_entry.get(), self.last_entry_index - 1))
```

Figuur 33 GitHub

Hier probeert de last_entry_trace te achterhalen of er op plek 26 al informatie staat. Als dit niet zo is, wordt last_entry onder de laatst definieerde entry geplaatst als ruimte voor nieuwe informatie.

Alle entries gebruiken de update_information() functie om veranderde informatie op te slaan.

```
def update_information(date, new_information, index = None):
    global information
    try:
        information[date][index] = new_information
    except:
        try:
            information[date].append(new_information)
        except:
            information[date] = [new_information]
    save_data(information, "Calendar")
```

Figuur 34 GitHub

Update information kijkt als eerste of er al informatie is geschreven op de gespecificeerde datum. Als dat er al wel is, is het heel simpel om de informatie in de dictionary te vervangen met nieuwe informatie. Zo niet wordt een nieuwe sleutel aan de informatie van die datum toegevoegd. Hierna wordt alle informatie van de kalender naar de server gestuurd om opgeslagen te worden met de save_data() functie.

Er zijn instanties waar de data niet door deze functie wordt opgeslagen, bijvoorbeeld als van scherm gewisseld wordt of de applicatie sluit. Om te voorkomen dat informatie in die gevallen verloren gaat wordt de functie mass_update_information() gebruikt.

```
def mass_update_information(day_instance, date):
    global information
    indexes = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
               21, 22, 23, 24, 25,]
    new_information = []
    try:
        new_information.append(day_instance.entry1.get())
        (…)
        new_information.append(day_instance.entry25.get())
    except: pass

    try:
        for index in indexes:
            information[f"{date[0]}/{date[1]}/{date[2]}"][index] = new_information[index]
    except: pass
        if day_instance.last_entry.get() != "":

    try:
        information[f"{date[0]}/{date[1]}/{date[2]}"][day_instance.last_entry_index -1] =
          day_instance.last_entry.get()
    except:
        information[f"{date[0]}/{date[1]}/{date[2]}"] = [day_instance.last_entry.get()]

    save_data(information, "Calendar")
```

Figuur 35 GitHub

Deze functie pakt alle informatie van alle entries in een dag en stuurt ze naar de server om opgeslagen te worden. Dit wordt gedaan door eerst alle informatie in een List genaamd “new_information” te zetten. Daarna wordt deze List op de juiste datum in information gezet en wordt alle informatie naar de server gestuurd om opgeslagen te worden.

Het dag overzicht bevat ook een “Back” en “+” knop. Dit laat de gebruiker teruggaan naar het maandoverzicht, of een extra entry toevoegen om meer informatie te noteren.

```
self.back_button = customtkinter.CTkButton
                   (self, width=60, height=60, corner_radius=20, text="←",
                    fg_color="transparent", font=customtkinter.CTkFont(size=60),
                    hover=False, command=lambda: [mass_update_information(self,
                    selected_date), master.master.switch_view(CalendarPage)],
                    text_color="white"
                    if master._get_appearance_mode() == "dark" else "black")
self.back_button.grid(row=0, column=0, sticky="w", padx=(60, 20), pady=(35, 20))

self.add_button = customtkinter.CTkButton
                  (self, text="+", font=customtkinter.CTkFont(size=40), height=50,
                   width=50, command=lambda: [mass_update_information(self,
                   selected_date), master.master.switch_view(DayViewPage)])
self.add_button.grid(row=1, column=0, padx=10, pady=10, rowspan=5)
```

Figuur 36 GitHub

De back_button verandert de view naar de Calendar Page, en de add_button initialiseert de Day View opnieuw. Dit werkt op deze manier omdat de initialisatie altijd alle entries met informatie heeft en 1 extra entry toevoegt. Dit betekent wel dat er niet meerdere entries worden toegevoegd door meerdere keren de add_button te drukken zonder iets in de entries te schrijven.

5.8. Settings Page

Settings Page heeft als functie om je instellingen te veranderen. Als DaySphere meer functionaliteit had dan de demo-functies voor dit onderzoek zou deze pagina veel groter zijn, maar in de huidige versie is alleen het thema van het programma te veranderen. Hier kan gekozen worden voor een licht of een donker thema.

```
class SettingsPage(customtkinter.CTkFrame):
    def __init__(self, master):
        customtkinter.CTkFrame.__init__(self, master)

        self.theme_label = customtkinter.CTkLabel(self, text="Theme", font=customtkinter
                                                  .CTkFont(self, size=20))
        self.theme_label.grid(row=1, column=0, padx=20, pady=10)

        self.variable = customtkinter.StringVar(self, theme_interprator(
                                                theme_interprator(customtkinter.
                                                AppearanceModeTracker.get_mode())))
        self.variable.trace_add("write", lambda x, y, z:
                                change_theme(theme_interprator(self.theme_menu.get())))

        self.theme_menu = customtkinter.CTkOptionMenu(self, values=["Light mode","Dark
                                                      mode"], variable=self.variable)
        self.theme_menu.grid(row=2, column=0, padx=20, pady=(0, 10))
```

Figuur 37 GitHub

Ook staat de log uit knop in de Settings Page.

```
global settings

self.logout_button = customtkinter.CTkButton(self, text="Log out", command=lambda:
                                                [master.master.switch_view
                                                (LoginPageFrame), logout()])
self.logout_button.grid(row=3, column=0, padx=20, pady=20)
```

Figuur 38 GitHub

5.9. Client -Server Communication

Het eerste wat gebeurt in het client bestand is dat er een ander bestand, “settings.json”, geopend wordt. Hierin staan een aantal variabelen in dictionary formaat. Deze worden opgeslagen in de globale variabele “settings”. Daarna wordt de establish_connection functie gedefinieerd.

Hier wordt als eerste de “connection_socket” variabele globaal gemaakt. Het doel van deze functie is om een nieuwe socket te initialiseren, die in deze variabele opgeslagen wordt. Daarna wordt deze socket verbonden aan de gespecificeerde combinatie van IP en poort.

```
def establish_connection():
    global connection_socket
    connection_socket = socket(AF_INET, SOCK_STREAM)
    connection_socket.connect((settings["host"],
```

Figuur 39 GitHub

Dan wordt login gedefinieerd zoals te zien is in Figuur 40.

```
def login(user:str = "", password:str = "", remember:bool = False) -> bool:
    """Attempts a login"""
    global settings
    if settings["remember_me"] == True:
        encoded = settings['encpass']
        user = settings["user"]
        remember=True

    else:
        connection_socket.sendall(f"func->Any\nAccounts.request_key(\"{user}\")".encode('UTF-8'))
        try: keys = list(map(int, eval(receive())))
        except ValueError: return False
        encoded = encoder(keys,password)

    connection_socket.sendall(f"func->Any\nAccounts.login(\"{user}\",{encoded},{remember})"
                                                        .encode("UTF-8"))
    reply = receive()
    if reply == "True": return True
    try:
        settings["encpass"] = list(map(int, eval(reply)))
        settings["user"] = user
        settings["remember_me"] = True
        dump(settings, open("settings.json", 'w'))
        return True
    except:
        settings["remember_me"] = False
        dump(settings, open("settings.json", 'w'))
        return False
```

Figuur 40 GitHub

Eerst wordt gecheckt of remember_me in de settings variabele waar is. Zo ja kunnen het gecodeerde wachtwoord en de gebruikersnaam daarvandaan verkregen worden. Ook wordt “remember” als waar gezet zodat deze zelfde inlogmethode nog een keer werkt de volgende keer dat ingelogd wordt. Als remember_me niet waar was wordt aan de server gevraagd wat de RSA-sleutels zijn. De publieke sleutel en het extra getal zijn niet beveiligd, omdat deze alleen gebruikt kunnen worden om een bericht te coderen, niet decoderen. Het zou kunnen dat het programma hier een ValueError

tegenkomt; dat betekent dat het account niet bestaat. In dat geval wordt de login poging gestopt en doorgegeven dat de poging mislukt is. Zolang dat niet gebeurt wordt het wachtwoord gecodeerd en gaat het programma verder naar de volgende stap. Hier wordt aan de server gevraagd om in te loggen door de functie door een bericht te sturen waarin staat dat er een poging wordt gemaakt in te loggen op een specifieke gebruikersnaam met een bijpassend gecodeerd wachtwoord en een boolean die bepaalt of de server een nieuw gecodeerd wachtwoord terug moet sturen.

```
def login(User:str, encpass:list, remembered:bool=False) -> bool:
    """Attempts a login."""

    global userpass, userpasslock
    global Threadlocalvars
    userpasslock.acquire()
    try:
        if sha256((userpass[User][3] + f"{encpass}".translate(listpunc))
                  .encode('UTF-8')).hexdigest() == userpass[User][4]:
            Pass = RSA.decoder(userpass[User][0:3:2], encpass)
            Accounts.__savepass(User, Pass)
            Threadlocalvars.Username = User
            if remembered == True:
                reencpass = RSA.encoder(userpass[User][1:3:1], Pass)
                userpasslock.release(); return reencpass
            else:
                userpasslock.release(); return True
         else:
            userpasslock.release(); return False
    except:
        userpasslock.release(); return False
```

Figuur 41 GitHub

In de login functie op de server worden een eerst een aantal globale variabelen gedefinieerd. Userpass is de dictionary met gebruikers en wachtwoorden die van te voren uit een bestand opgehaald zijn. Userpasslock is de lock van userpass, en Threadlocalvars is een local die op elke thread gebruikt wordt. Als tweede wordt userpasslock in bezit genomen, want als dat niet gedaan wordt kan het gebeuren dat wanneer 2 mensen tegelijkertijd inloggen of een account aanmaken hun data gedesynchroniseerd wordt, en het account van de persoon die als 1e klaar is verwijderd worden. Hierna wordt het binnengekregen gecodeerde wachtwoord niet gedecodeerd, maar juist nog erger gecodeerd. Het wachtwoord wordt namelijk niet letterlijk op de server opgeslagen, omdat dat het makkelijk zou maken om alle wachtwoorden van iedere gebruiker in een grote lijst terug te vinden. Om dat tegen te gaan wordt de meest moderne cryptografie technologie toegepast; salting en hashing. Voor het hashen wordt sha256 gebruikt, en voor salting wordt een willekeurige string gegenereerd met de functie token_hex van secrets. Als dit extra gecodeerde wachtwoord hetzelfde is als wat in het bestand op de server is opgeslagen betekent dat dat het wachtwoord klopt. In dat geval wordt het wachtwoord gedecodeerd, worden er nieuwe sleutels voor de RSA-encryptie en een nieuwe salt gegenereerd door gebruik van de privéfunctie __savepass en worden alle encryptiestappen opnieuw uitgevoerd, zodat het resultaat op de server kan worden opgeslagen. Threadlocalvars.Username wordt veranderd naar de gebruikersnaam waarmee zojuist is ingelogd, zodat de rest van het programma ook weet dat de gebruiker ingelogd is. In het geval dat remembered

waar was wordt het wachtwoord RSA-gecodeerd teruggestuurd naar de client, waar het opgeslagen zal worden tot de volgende login.

5.10. Spraakherkenning

Wanneer spraakherkenning vanuit de terminal page gestart wordt, wordt main() uitgevoerd. De definitie van main is te zien in Figuur 42.

```
def main():
    global listening_thread, listening

    if not listening:
        with VR.mic as source:
            history("Adjusting")
            VR.recog.adjust_for_ambient_noise(source, 0.5)


        listening_thread = VR.recog.listen_in_background(VR.mic, VR.Recognize)
        history("Listening...")
    else: listening_thread()
    listening = not listening
```

Figuur 42 GitHub

Eerst worden listening_thread en listening als globaal gedefinieerd. Dit wordt gedaan zodat de volgende keer dat deze functie weer gebruikt wordt deze variabelen hetzelfde zijn als de vorige keer, zodat het programma weet of het de spraakherkenning aan of uit moet zetten. Listening is de boolean die daarvoor gebruikt wordt. Listening thread is echter gedefinieerd als de functie die het luisteren zal stoppen. Deze wordt teruggegeven door listen_in_background wanneer deze gestart wordt. Het grootste deel van deze functie gebeurt dus alleen wanneer niet al geluisterd wordt. In dat geval wordt naar de terminal page gestuurd dat begonnen is met het aanpassen van de achtergrondgeluid filter. Deze wordt automatisch ingesteld door VR.recog.adjust_for_ambient_noise. Source is hier de microfoon, en 0.5 staat voor de tijdsduur van de achtergrondgeluidscheck. Daarna wordt listening_thread gedefinieerd en start het programma met het luisteren op de achtergrond. Wanneer het herkent dat gestopt is met praten wordt VR.Recognize uitgevoerd zoals gedefinieerd in Figuur 42.

```
def Recognize(recognizer: Recognizer, audio):
    history("Processing...")
    try:
        text=recognizer.recognize_whisper(audio, "small.en", False, None, "english", False)
        nopunc = str(text).lower().translate(str.maketrans('', '', punctuation))

        for keyword in commands.keywords:
            if keyword in nopunc.replace(" ",""):
                feedback(terminal_command=nopunc[find_with_spaces(keyword, nopunc)+1:], voice=True)
                break

        history(nopunc)

    except UnknownValueError:
        print("Speech recognition could not understand audio")
    except RequestError as e:
        print("Could not request results from the speech recognition service; {0}".format(e))
```

Figuur 43 GitHub

Hier wordt eerst naar de terminalpage gestuurd dat het herkenningsproces bezig is. Daarna wordt geprobeerd om de audio te herkennen met het “small.en” model in het Engels. Vervolgens wordt het resultaat hiervan omgezet naar een string, worden hoofdletters klein gemaakt en wordt alle punctuatie weggehaald. Dan wordt gecheckt, voor elke keyword in een lijst in een ander bestand of deze in de versie van deze tekst zonder spaties staat. De spaties moeten weggehaald worden omdat een keyword soms spaties heeft op onverwachte plekken. In de meeste gevallen zou maar 1 keyword gecheckt hoeven worden, maar omdat DaySphere op zichzelf niet een Engels woord is maar een naam, herkent OpenAI Whisper het niet goed. Daarom is het handig een lijst te hebben met meerdere keywords in de buurt van het correcte. Wanneer een keyword is gevonden wordt de tekst met een RegEx functie nog een keer gecheckt voor de keyword, maar deze keer zonder de spaties weg te halen, zodat de tekst die doorgegeven wordt aan de terminal page wel spaties kan bevatten. Deze string moet doorgegeven worden aan de Terminal Page, omdat de terminal entry op dezelfde manier functies uitvoert als de spraakherkenning. De feedback functie in de terminal page past dan runfromstring toe, een functie uit het andere bestand van spraakherkenning.

```python
def runfromstring(inputstr:str):
    global function_dictionary
    for lenak in function_dictionary.keys():
        try: function = function_dictionary[lenak][inputstr[:lenak].lower()]
        except: continue

        if type(function) == str: function = [function, "str"]

        if function[1] == "str":
            return eval(function[0] + f"(\"{inputstr[lenak:].lower()}\")")
        elif function[1] == "var":
            return eval(function[0] + f"(*{tuple(inputstr[lenak:].lower().split(' ')[1:])})")
        else:
            return eval(function[0] + f"(*{tuple(inputstr[lenak:].lower().split(' ')[1:function[1]+1])})")
```

Figuur 44 GitHub

Runfromstring neemt een “inputstr” waar de tekst die geïnterpreteerd zal worden binnen komt. Ook gebruikt het de globale variabele function_dictionary, een aangepaste versie van de dictionary waarin alle keywords gelijk staan aan functies. Deze dictionary moet aangepast worden omdat het gesorteerd op lengte van de keyword moet staan. Dit helpt het programma want normaal weet het niet welk deel van de inputstr het moet vergelijken met de keyword, maar de dictionary gesorteerd is op lengte kan het gewoonweg de sleutel als lengte aannemen. Hierna wordt dus gecheckt of de string van de eerste paar karakters in function_dictionary staat, wat herhaald wordt voor elke lengte van keyword. In de function_dictionary staat ook voor elke functie de verwachtte invoervariabele, waarvan de standaard “str” is. Als een match is gevonden wordt de function variabele gedefinieerd als deze lijst van functie en type. Als laatste wordt deze functie uitgevoerd en worden de ingangsvariabelen doorgestuurd gebaseerd op wat in de het type variabele dat verwacht was door de functie.

6. Resultaten

6.1. Back-End

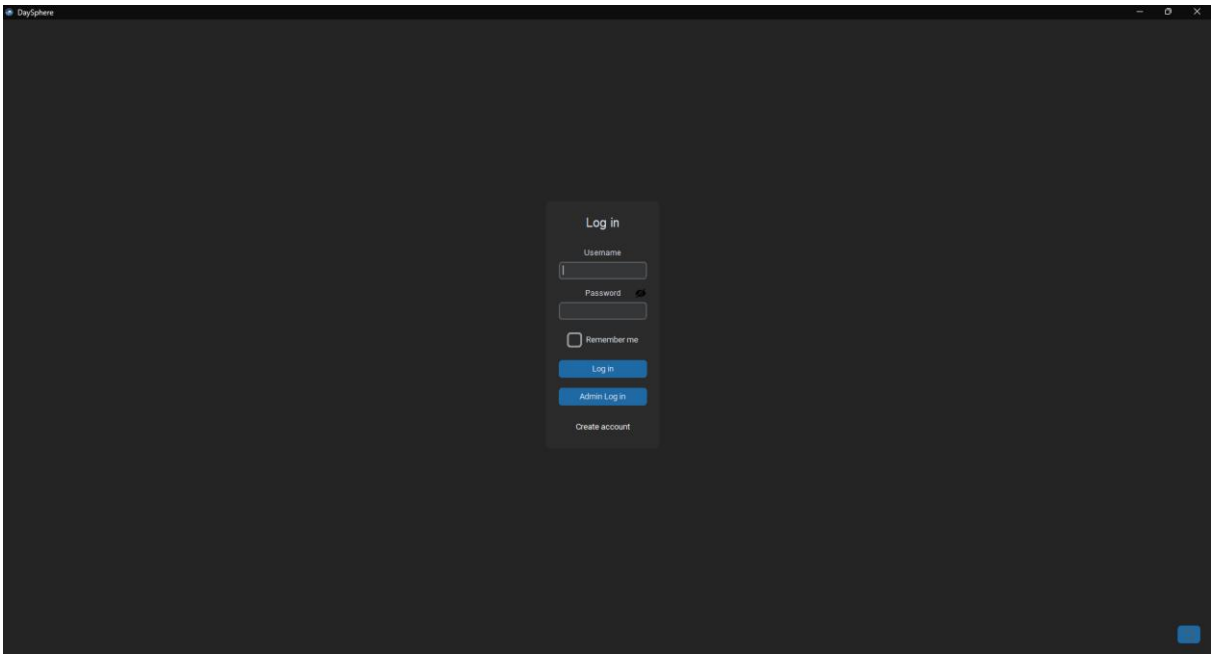
Voor spraakherkenning wordt een dictionary gebruikt om keywords zoals “schedule” tegenover functies te zetten. Als een andere functie checkt of een van deze keywords overeenkomt met de herkende tekst uit spraakherkenning kan de ingebouwde eval() functie de corresponderende functie uitvoeren met invoervariabelen die geformatteerd zijn zoals staat in de dictionary. De functie die op deze manier uitgevoerd wordt moet de herkende tekst verder kunnen interpreteren tot echte data, zoals bijvoorbeeld een datum of de daadwerkelijke activiteit op die datum.

DaySphere heeft een server waaraan de applicatie wanneer het opstart direct probeert te verbinden via port 5050. Hierna kunnen alle berichten via deze verbinding als bytelike objects worden verstuurd. Deze berichten beginnen met een code die de server laat weten wat de client als respons verwacht. Daarna komt een separator, in dit geval “\n”, wat staat voor een nieuwe witregel, en dan de functie en variabelen om aan die functie te geven. De server stuurt op een soortgelijke manier een bericht terug naar de client, die daarna verder gaat met de rest van het programma.

De server heeft een variabele, Datatypes, die alle datatypes die opgeslagen zullen worden bevat. Als een account gemaakt wordt, wordt er in de userdata folder op de server een nieuwe sleutel toegevoegd aan de dictionary met alle gebruikers. De waarde van deze dictionary is nog een dictionary, waarin alle datatypes als sleutels staan. De waardes van die dictionaries zijn nog meer dictionaries, waarin alle data van de gespecificeerde categorie is opgeslagen. Om toegang te krijgen naar deze data wordt gekeken naar een variabele die voor elke verbinding verschillend is, namelijk Threadlocalvars.Username. Als iemand inlogt of een account aanmaakt wordt deze veranderd naar de gebruikersnaam van hun account. Als iemand informatie aanvraagt zonder ingelogd te zijn zal dit een error opgeven en wordt alleen een Boolean onwaar teruggestuurd.

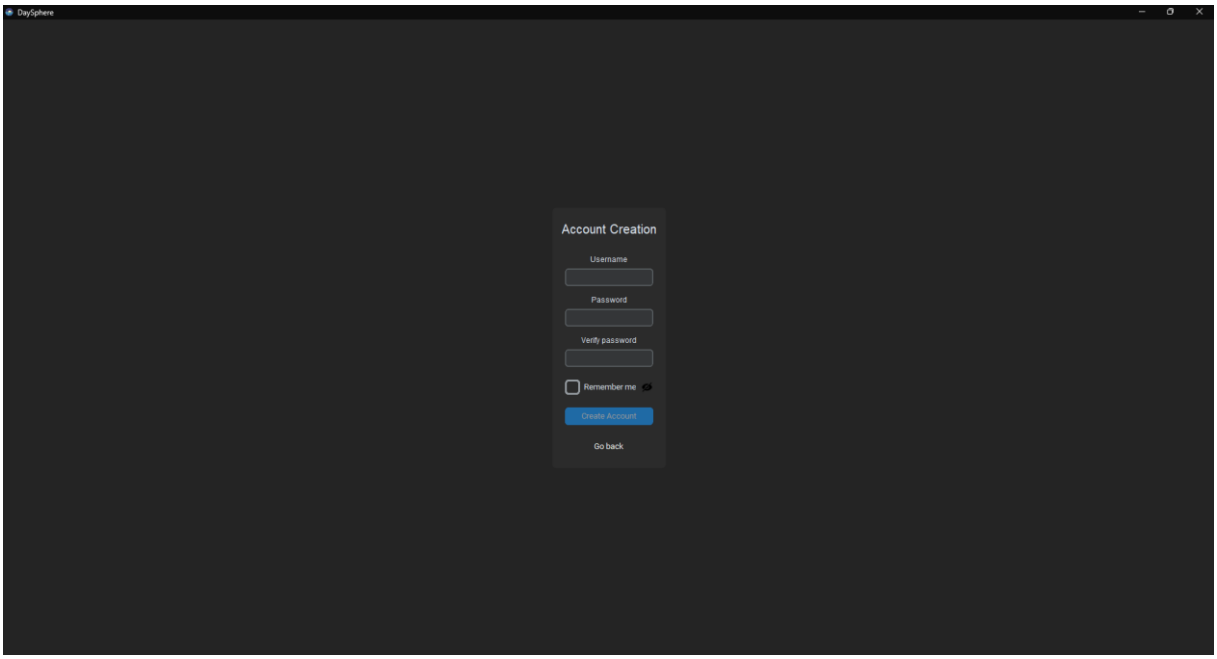
Wachtwoorden worden niet op dezelfde manier opgeslagen. Hier wordt encryptie toegepast, in dit geval RSA, Salting en Hashing. Wanneer een account wordt gemaakt worden sleutels voor RSA en een willekeurige salt gegenereerd. De publieke sleutel en het extra getal worden naar de client gestuurd wanneer deze probeert in te loggen. De client codeert dan het wachtwoord en stuurt deze door naar de server. De server slaat het voor een korte tijd op en verandert het naar een string, voegt de salt eraan toe en hasht het. Hierna wordt gecheckt of het hetzelfde is als wat opgeslagen is op de server. Zo niet gebeurt er niets en wordt teruggestuurd dat het inloggen gefaald is. Als het wel klopt wordt weer gekeken naar het originele bericht. Hieruit wordt het wachtwoord gedecodeerd. Daarna worden nieuwe encryptiesleutels en een nieuwe salt gegenereerd, die gelijk gebruikt worden om het wachtwoord te coderen. Ook wordt het opnieuw gehasht. Dit is het resultaat wat uiteindelijk weer opgeslagen wordt op de server. Omdat hashing een onomkeerbare functie is, is het onmogelijk om vanaf de harde schijf van de server de wachtwoorden van gebruikers af te lezen.

6.2. Front-end



Figuur 45 Login Page

Midden in het scherm staat een menu. Bovenaan staat een username CTkEntry en Password CTkEntry met daarnaast een knop om het wachtwoord te tonen of verstoppen. Onder de CTkEntries staat een CTkCheckBox met de tekst “Remember me”. Daaronder staan drie CTkButtons waar van boven naar beneden op staat “Log in” “Admin Log in” en “Create Account”. Buiten het menu staat helemaal rechtsonder in het scherm knop om de verbinding te verversen.

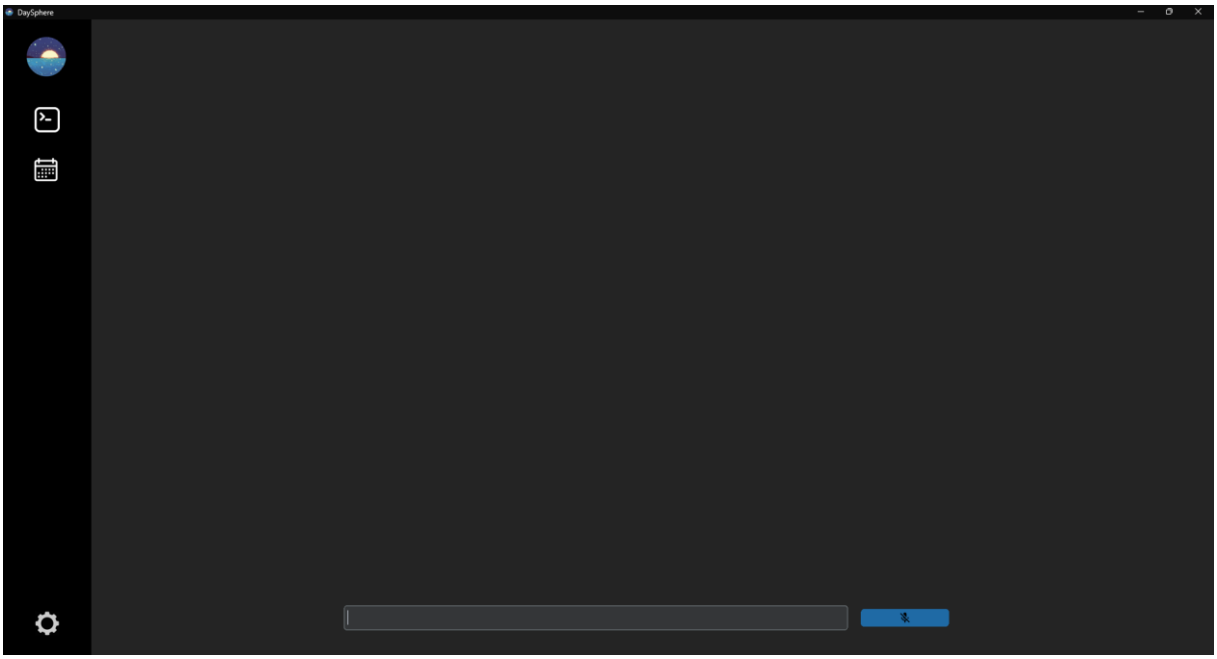


Figuur 46 Account Creation Page

Midden in het scherm staat een menu. Bovenaan staat een username CTkEntry, een Password CTkEntry en een Password verificatie CTkEntry. Daar onder staat een CTkCheckBox met de tekst “Remember me”, en naast de CTkCheckBox staat een knop om het wachtwoord te tonen of verstoppen in beide wachtwoord entries. Hieronder staat een CTkButton waarvop staat: “Create Account”. De tekst van deze knop is grijs wanneer ongeldige gegevens zijn ingevuld. In alle andere gevallen is deze wit.

Figuur 47 Main Page

Op deze pagina zelf staat niks. Links is de Sidebar te zien. De Sidebar is een dunne balk met 4 iconen die corresponderen aan andere pagina's. Het bovenste icoon is het logo van het programma, die brengt de gebruiker naar de Main Page. Onder het logo staat een "terminal" icoon, dit brengt de gebruiker naar de Terminal Page. Daaronder zit het "Calendar" icoon, wat de gebruiker naar de Calendar Page stuurt. Helemaal onderaan de balk staat een "Settings" icoon, dit brengt de gebruiker naar de Settings Page.

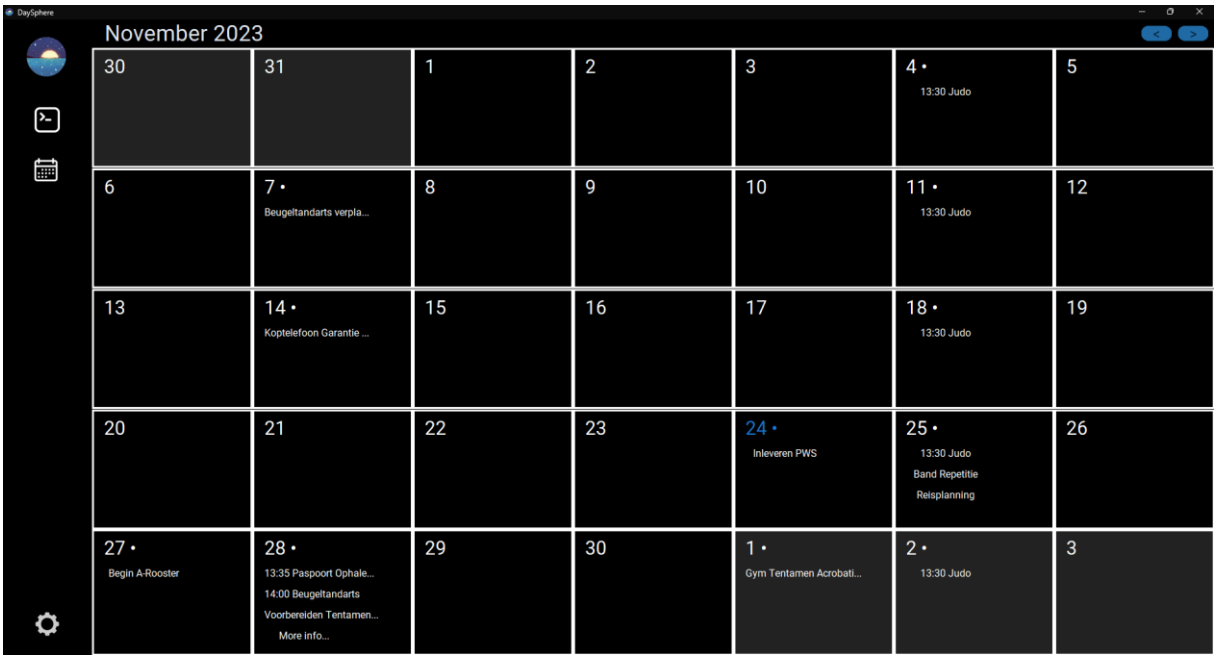


Figuur 48 Terminal Page

De Terminal page bevat alleen één CTkEntry en één knop. De knop activeert spraakherkenning, de entry is hiervoor een substitutie. Wanneer er een functie wordt uitgevoerd wordt het resultaat hiervan afgebeeld worden op het scherm.

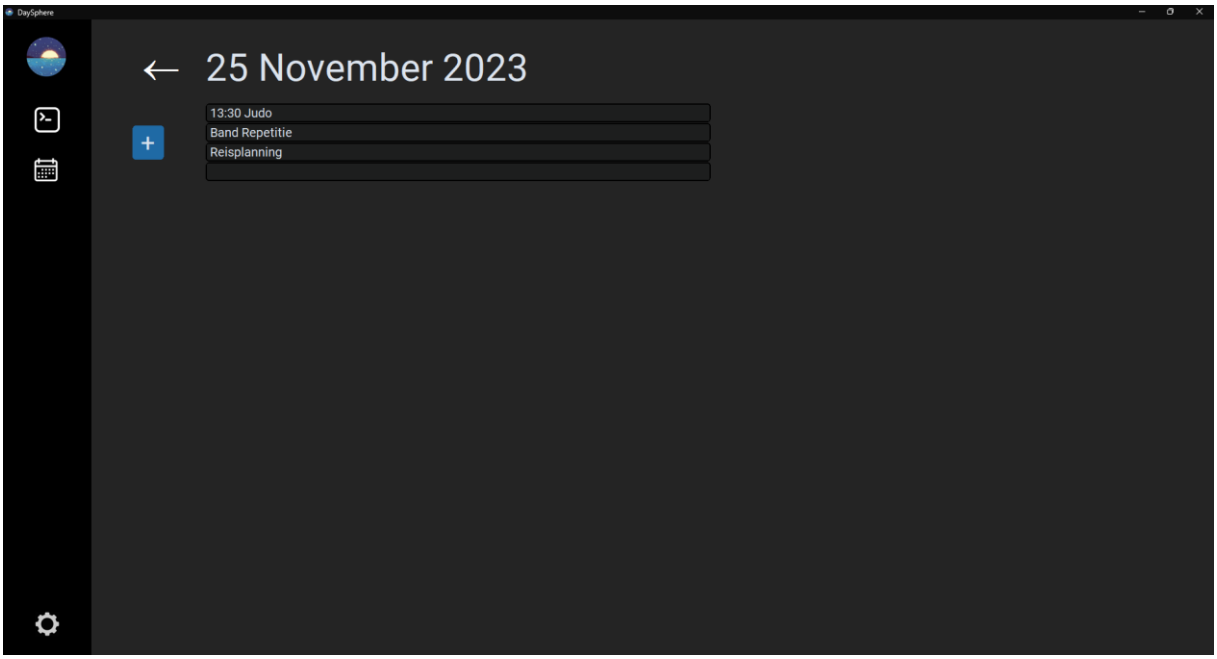
Figuur 49 Calendar Page Maandoverzicht (Geen gebruikersinformatie)

De Calendar Page geeft linksboven de geselecteerde maand en het jaar weer. Helemaal rechtsboven staan twee knoppen om één maand naar voren of naar achteren te gaan. Onder deze 3 schermelementen staat de kalender. De dagen buiten de geselecteerde maand zijn grijs afgebeeld, en het nummer van de huidige dag is blauw gekleurd.



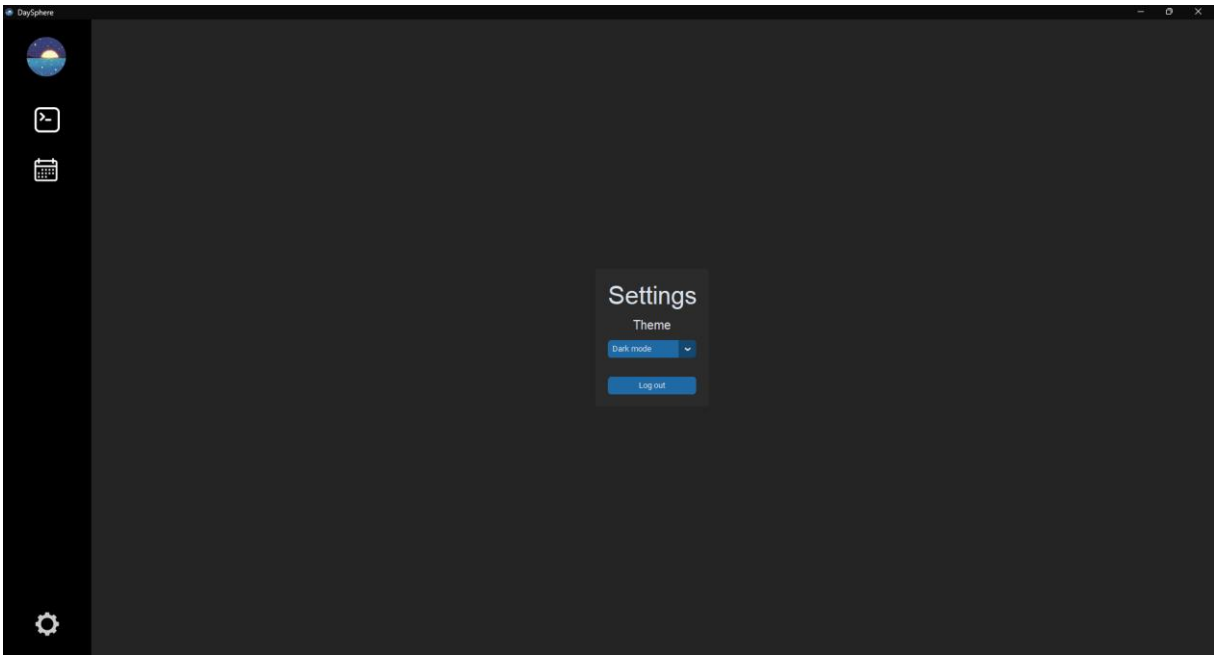
Figuur 50 Calendar Page Maandoverzicht

De kalender zelf bevat activiteiten, en er wordt voor iedere dag afgebeeld of er minimaal 1 activiteit is met de · naast de datum. In de vakken van de dagen staan rijen tekst die de eerste 3 activiteiten weergegeven. Als de tekst te lang is om weer te geven wordt het afgekort met "…", en als er meer dan drie planningen op een dag zitten wordt dit weergeven met "More info…" onderaan het vlak.



Figuur 51 Calendar Page Dag Overzicht

Bovenin wordt de datum weergegeven in het formaat Dag Maand Jaar. Links hiervan staat een knop met een pijl naar links die de gebruiker terug brengt naar de Calendar Page. Onder de knop staat nog een knop met een plus als symbool. Deze voegt één nieuwe entry toe om meer informatie over de dag neer te kunnen zetten. Rechts van deze knop staat een rij entries waar alle informatie over de dag wordt weergegeven. De informatie in deze entries kunnen worden veranderd, en de data hierin zal opgeslagen worden op het gebruikte account.



Figuur 52 Settings Page

In het midden zit een menu, hierin zit een CTkOptionMenu en knop. Het option menu geeft de mogelijkheid om het thema van het programma te wisselen tussen zwart en wit. Onder de option menu zit een knop waarop staat “Log out” die de gebruiker uitlogt en terug naar de Login Page brengt.

7. Conclusie

Het doel van dit onderzoek is om een programma te ontwerpen dat door gebruik van spraakherkenning te besturen is, en anderen te leren hoe dat te doen. Er is veel behoefte aan meer van dit soort apps, maar de meeste grote bedrijven stoppen met dit soort functies ondersteunen of verder ontwikkelen. Om dit voor elkaar te krijgen is in dit onderzoek gekeken naar de letterlijke stemherkenning en hoe het werkt, de communicatie tussen client en server, de veilige opslag van data en wachtwoorden en het bouwen van een demoprogramma dat hier gebruik van maakt.

7.1. Hoe wordt een stem geïnterpreteerd naar uitvoerende functies?

Spraak wordt herkend door OpenAI Whisper en veranderd in tekst. In deze tekst wordt door RegEx gezocht naar keywords die in een dictionary tegenover functies staan. Deze worden door de ingebouwde eval() functie uitgevoerd met een hoeveelheid invoervariabelen die gedefinieerd is in dezelfde dictionary.

7.2. Op welke manier communiceren clients met de server?

Clients communiceren met de server door een Socket. Wanneer een client hieraan verbonden is kunnen berichten heen en weer gestuurd worden in de vorm van bytelike strings. De server kan in deze berichten vinden wat de client verwacht als respons en wat de functie en variabelen zijn waarmee het uitgevoerd moet worden.

7.3. Op welke manier wordt de opgeslagen data gecategoriseerd?

Data wordt op de server opgeslagen in een dictionary waar per gebruiker voor iedere categorie alle data in staat. Als de client een verzoek naar data stuurt gebruikt de server een local om te vinden wie de ingelogde gebruiker is en stuurt de gespecificeerde data van deze gebruiker terug naar de client. Wanneer de client een verzoek stuurt om data op te slaan wordt hetzelfde gedaan, maar dan andersom.

7.4. Hoe wordt informatie gecommuniceerd met de gebruiker?

CustomTkinter wordt gebruikt om pagina's te maken en daar schermelementen op te zetten waarmee de gebruiker kan interageren. Dit werd beperkt tot knoppen en entries in een vaste stijl. Door dit te beperken krijgt de gebruiker een overzichtelijke weergave zonder dat ze worden afgeleid, waardoor het GUI gemakkelijk is te gebruiken en de informatie effectiever gecommuniceerd wordt.

7.5. Op welke manier kan Python gebruikt worden om effectief gebruik te maken van spraak herkennende technologie?

Python kan gebruikt worden om een programma te maken dat effectief gebruik maakt van spraak herkennende technologie door gebruik te maken van een spraakherkenningsmodule om spraak te veranderen in tekst en deze tekst te interpreteren naar functies. Deze functies kunnen data opslaan op een server via een socket verbinding. Deze data moet daar gesorteerd opgeslagen worden zodat het later teruggevonden kan worden wanneer de client dit verzoekt. Alle informatie kan met de gebruiker gecommuniceerd worden met de CustomTkinter module. Met deze module kan een makkelijk te gebruiken GUI gemaakt worden.

8. Discussie

Dit onderzoek was origineel als JavaScript programma bedoeld, maar nadat ontdekt was dat continue spraakherkenning niet mogelijk zou zijn werd overgestapt naar Dart. Deze codeertaal wordt gebruikt met het framework genaamd Flutter. Hierin zou de Python SpeechRecognition module gebruikt worden, maar de dart module die dat mogelijk zou maken werkte uiteindelijk niet. Op dit punt kon uit 2 dingen gekozen worden: Opnieuw beginnen in een nieuwe codeertaal, of met Python verder werken wetende dat Tkinter gebruikt zou moeten worden en het eindproduct niet zo hoge kwaliteit zou zijn. Vanwege een gebrek aan tijd werd de 2^{e} keuze genomen, maar in een vervolgonderzoek wordt aangeraden om in plaats daarvan keuze 1 te kiezen. Een goed voorbeeld voor een alternatieve codeertaal is C#. Hierin bestaan meerdere speechrecognition modules, waaronder de OpenAI Whisper module die in dit onderzoek gebruikt werd. Ook wordt aangeraden dat als een programma wordt gemaakt dat gefocust is op 1 onderdeel, in dit geval spraakherkenning, dat zeker wordt gemaakt dat dat ook echt werkt in de codeertaal die gebruikt wordt voordat aan de Front-End begonnen wordt. Dit probleem is meerdere keren opgekomen tijdens dit onderzoek, vooral wanneer van codeertaal gewisseld moest worden werd alle vooruitgang aan de Front-End gewist. Vervolgonderzoeken kunnen verder ingaan op de specificaties van andere soorten programma's en hoe deze met spraakherkenning interageren. Er bestaat al een recente ongerelateerde studie naar spraakherkenning in combinatie met moderne chat AI om in een code editor automatisch code te schrijven met spraakherkenning. In de toekomst kan men misschien alles op zijn of haar computer besturen met de stem.

9. Literatuurlijst

Type bron		Vermelding volgens APA
Boek	Een boek over client-server communicatie met sockets in Python.	Singh, A. (2019). Socket Programming With Python
Website	Het onderzoek naar GELUs.	Hendrycks, D., & Gimpel, K. (2016). Gaussian error linear units (gelus). arXiv preprint arXiv:1606.08415.
Website	Het onderzoek naar een snellere versie van transformers in AI-modellen.	Child, R., Gray, S., Radford, A., & Sutskever, I. (2019). *Generating long sequences with sparse transformers*. arXiv preprint arXiv:1904.10509.
Website	Een onderzoek naar het gebruiken van grammaticale regels om automatisch gegenereerde tekst te verbeteren.	Using the Output Embedding to Improve Language Models, Press & Wolf, EACL 2017
Website	Het onderzoeksverslag van Openai Whisper.	Radford, A., Kim, J. W., Xu, T., Brockman, G., McLeavey, C., & Sutskever, I. (2023, July). Robust speech recognition via large-scale weak supervision. In International Conference on Machine Learning (pp. 28492-28518). PMLR.
Website	De officiële documentatie van CustomTkinter module.	Official Documentation And Tutorial | CustomTkinter (tomschimansky.com)
Website	De officiële documentatie van Pillow module.	Pillow · PyPI
Website	De officiële Python documentatie van secrets.	secrets — Generate secure random numbers for managing secrets — Python 3.12.0 documentation
Website	De officiële Python documentatie van hashlib	hashlib — Secure hashes and message digests — Python 3.12.0 documentation
Website	De officiële Python documentatie van Datetime.	datetime — Basic date and time types — Python 3.12.0 documentation
Website	De officiële Python documentatie van socket.	socket — Low-level networking interface — Python 3.12.0 documentation
Website	De officiële documentatie van SpeechRecognition	SpeechRecognition · PyPI
Website	De officiële documentatie van OpenAI Whisper.	GitHub - openai/whisper: Robust Speech Recognition via Large-Scale Weak Supervision
Website	De officiële Python documentatie van json.	json — JSON encoder and decoder — Python 3.12.0 documentation
Website	De officiële Python documentatie van threading.	threading — Thread-based parallelism — Python 3.12.0 documentation
Website	De officiële Python documentatie van RegEx.	Regular Expression HOWTO — Python 3.12.0 documentation

Tabel 8 8Literatuurlijst

10. Bijlagen

10.1. Eindproduct

Het eindproduct is hier te downloaden.

10.2. Logboek

Task name	Individu	Tijd besteed	Onderwerp
#0000 IDE Setup	Dewin van Zaanen	16/04/23 20:30 - 21:30 Totaal: 1:00	Setup
#0001 Front-End Start	Dewin van Zaanen	16/04/23 21:30 - 23:00 17/04/23 19:00 - 19:30 03/05/23 15:30 - 16:15 10/05/23 12:30 - 14:15 11/05/23 10:00 - 11:00 11/05/23 23:45 - 12/05/23 01:15 Totaal: 7:00	Front-End
#0002 Voice Control Start	Robin Walthuis	16/04/23 20:50 - 23:35 14/05/23 14:30 - 18:00 04/06/23 11:50 - 14:10 06/06/23 14:30 - 18:00 25/06/23 15:00 - 17:30 01/07/23 14:10 - 17:00 04/07/23 20:00 - 21:15 07/07/23 16:40 - 17:50 11/07/23 02:30 - 04:05 11/07/23 09:15 - 15:45 25/07/23 19:10 - 20:30 26/08/23 13:20 - 16:30 Totaal: 32:25	Back-End
#0003 Front-End Conceptual Understanding	Dewin van Zaanen	03/06/23 00:15 - 02:15 03/06/23 09:45 - 12:30 04/06/23 00:00 - 01:00 04/06/23 09:15 - 14:15 04/06/23 15:45 -16:15 07/06/23 17:00 - 18:00 27/06/23 23:00 - 28/06/23 02:15 11/07/23 09:15 - 16:00 12/07/23 04:15 - 06:15 10/08/23 00:00 - 07:15 10/08/23 07:45 - 16:30 Totaal: 32:15 + 25:00	Front-End
#0004 Python Log-In Page	Dewin van Zaanen	26/08/23 13:15 - 17:00 28/08/23 15:00 - 17:30 28/08/23 18:15 - 20:15 06/09/23 08:30 - 12:45 06/09/23 13:15 - 14:00 Totaal: 13:15	Front-End

#0005 Python SpeechRecognition	Robin Walthuis	26/08/23 16:30 - 17:00 26/08/23 17:45 - 19:45 28/08/23 15:30 - 20:00 29/08/23 16:00 - 18:00 29/08/23 18:45 - 20:30 Totaal: 10:45	Back-End
#0006 User accounts	Robin Walthuis	06/09/23 08:30 - 12:45 07/09/23 16:30 - 17:45 08/09/23 00:30 - 02:00 Totaal: 7:00	Back-End
#0007 GUI Views and User Accounts integration	Dewin van Zaanen	06/09/23 14:30 - 15:00 07/09/23 21:15 - 08/09/23 04:45 Totaal: 8:00	Front-End
#0008 Client-Server Communication	Robin Walthuis	10/09/23 11:45 - 12:00 10/09/23 12:15 - 14:00 10/09/23 14:15 - 17:30 Totaal: 5:15	Back-End
#0009 Function Page Templates/Calendar	Dewin van Zaanen	08/09/23 19:15 - 19:45 09/09/23 15:45 - 16:45 10/09/23 11:45 - 12:00 10/09/23 12:15 - 14:00 10/09/23 14:15 - 17:30 11/09/23 15:45 - 17:30 11/09/23 18:45 - 22:45 12/09/23 13:00 - 16:00 22/09/23 16:00 - 16:45 22/09/23 17:15 - 19:45 23/09/23 00:00 - 3:15 23/09/23 15:45 - 18:15 23/09/23 19:15 - 20:45 28/09/23 10:15 - 11:00 30/09/23 12:00 - 12:45 01/10/23 11:30 - 13:15 03/10/23 13:30 - 14:15 03/10/23 15:00 - 17:45 03/10/23 18:30 - 19:30 06/10/23 12:00 - 12:45 06/10/23 13:00 - 14:00 07/10/23 09:30 - 12:45 07/10/23 15:45 - 16:00 09/10/23 09:45 - 11:00 17/10/23 14:45 - 16:15 27/10/23 13:30 - 16:15 11/11/23 15:45 - 17:00 11/11/23 17:15 - 18:00 12/11/23 15:30 - 18:00 16/11/23 15:45 - 17:30 16/11/23 18:00 - 19:30 17/11/23 16:00 - 16:45 19/11/23 21:45 - 00:00	Front-End

		21/11/23 18:30 - 19:45 21/11/23 20:30 - 22/11/23 00:45 23/11/23 05:15 - 11:00 23/11/23 16:00 - 17:00 Totaal: 69:30	
#000A Connection rework	Robin Walthuis	12/09/23 10:45 - 11:00 12/09/23 12:45 - 13:30 12/09/23 15:15 - 16:00 12/09/23 19:00 - 19:45 14/09/23 12:45 - 13:30 14/09/23 15:15 - 16:00 14/09/23 19:15 - 22:15 14/09/23 23:15 - 23:45 23/09/23 15:30 - 16:45 23/09/23 17:15 - 20:45 Totaal: 12:15	Back-End
#000B Prime selection	Robin Walthuis	25/09/23 19:15 - 21:00 Totaal: 1:45	Back-End
#000C Reformatting old tickets	Robin Walthuis	28/09/23 11:00 - 11:45 Totaal: 00:45	Verslag Schrijven
#000D Linking Speech recognition to new program	Robin Walthuis	03/10/23 15:00 - 17:45 03/10/23 18:30 - 19:30 04/10/23 23:00 - 23:00 Totaal: 3:45	Back-End
#000E Voice Recognition Cleanup	Robin Walthuis	17/10/23 15:00 - 16:15 Totaal: 1:15	Back-End
#000F Report Writing	Dewin van Zaanen, Robin Walthuis	Dewin 17/10/23 16:15 - 16:45 23/11/23 17:30 - 23:15 24/11/23 12:00 - 14:15 24/11/23 15:45 - 18:30 24/11/23 19:00 - 25/11/23 00:00 18/12/23 20:00 - 19/12/23 00:00 19/12/23 10:00 - 11:30 19/12/23 22:15 - 20/12/23 02:30 20/12/23 13:00 - 13:30 21/12/23 00:00 - 04:15 21/12/23 04:30 - 06:00 Totaal: 32:15 Robin 17/10/23 16:15 - 16:45 11/11/23 15:45 - 16:45 11/11/23 17:00 - 17:45 12/11/23 16:30 - 18:00 19/11/23 21:45 - 00:00 20/11/23 22:15 - 23:00 21/11/23 14:30 - 15:45 21/11/23 19:00 - 23:30 21/11/23 23:45 - 01:00 22/11/23 19:15 - 01:45	Verslag Schrijven

		23/11/23 16:00 - 18:30 23/11/23 18:45 - 21:15 23/11/23 21:45 - 23:15 24/11/23 11:30 - 13:15 24/11/23 15:45 - 18:30 24/11/23 18:45 - 25/11/23 00:00 18/12/23 12:45 - 13:15 18/12/23 19:00 - 20:15 18/12/23 20:45 - 21:45 19/12/23 10:00 - 11:00 19/12/23 22:15 - 00:15 20/12/23 00:30 - 02:30 20/12/23 08:45 - 09:00 20/12/23 12:15 - 14:00 21/12/23 00:30 - 06:00 Totaal: 51:30	
#0010 Redoing undone progress	Robin Walthuis	14/11/23 16:15 - 17:45 Totaal: 1:30	Back-End
#0011 Server side password encryption	Robin Walthuis	14/11/23 17:45 - 18:00 14/11/23 18:30 - 20:30 Totaal: 2:15	Back-End
#0012 Reworking threading	Robin Walthuis	16/11/23 15:45 - 18:00 Totaal: 2:15	Back-End
Logo	Dewin van Zaanen	Totaal: 2:00	Design
Totaal	Dewin van Zaanen	204:30	
Totaal	Robin Walthuis	132:40	
Totaal		**337:10**	

Tabel 99 Logboek