Amir Qurbani
CLO24 – Campus Mölndal
Examensarbete HT 2025

Lokal AI-assistent för filhantering

Lokal AI-assistent för filhantering

1. Inledning

Många har svårt att hitta rätt dokument på sin dator, särskilt när man har många pdf och Markdown-filer utspridda i olika mappar. Gamla filer ligger ofta kvar utan att användas, och det är svårt att veta vilka man kan rensa bort. Detta gjorde att jag ville bygga ett verktyg som hjälper användaren att förstå sina filer bättre och hitta rätt dokument snabbare.
Jag har därför utvecklat en lokal AI-assistent som skannar en vald mapp, läser in pdf och Markdown-filer och sparar informationen i en SQLite-databas. Med hjälp av en liten AI-modell (Mini LM) kan appen söka efter ämnen och visa de dokument som passar bäst, tillsammans med en sammanfattning och en matchnings poäng. Användaren kan också se vilka filer som är gamla och välja att rensa dem på ett säkert sätt.
Den här rapporten beskriver hur appen är uppbyggd, hur AI-sökningen fungerar och vilka tekniska val jag gjort som nybörjare för att skapa en enkel men smart filhanterings-assistent.

---

2. Systemöversikt

När appen startar möts användaren av en meny med olika val, till exempel att skanna en mapp, söka efter dokument eller rensa gamla filer. Användaren börjar vanligtvis med att skriva in en sökväg, alltså den mapp på datorn där appen ska arbeta.

När en mapp har valts skannar appen alla PDF- och Markdown-filer som finns där. För varje fil hämtas information som filnamn, storlek, filtyp och när filen senast ändrades. All denna information sparas i en lokal SQLite-databas, i en tabell som appen använder senare för att visa och söka bland filerna.

Sökfunktionen gör det möjligt att snabbt hitta dokument baserat på ett ord eller ett ämne. Appen använder både en bas-score och en AI-baserad likhet poäng för att hitta de fem filer som matchar bäst. Varje gång användaren söker sparas sökningen i databasen. Om användaren söker efter liknande saker flera gånger får resultatet en bonuspoäng, vilket gör att appen gradvis lär sig vad användaren är intresserad av och kan visa bättre träffar nästa gång.

Appen har också en funktion som visar gamla filer. Användaren anger hur många dagar en fil får vara gammal, till exempel 65 eller 95 dagar. Appen visar då alla filer i databasen som är äldre än det värdet, och användaren kan välja att ta bort dem eller flytta dem till en “trash”-mapp.

En viktig del av systemet är loggningen. Varje gång appen rensar, flyttar eller tar bort en fil sparas en rad i loggfilen i mappen **/data/log.txt**. Loggen innehåller vilken fil som påverkades, vilken åtgärd som gjordes och när det skedde. Detta gör det möjligt att följa alla ändringar i efterhand och ger en extra trygghet om något skulle bli fel.

Sammanfattningsvis arbetar appen i fyra steg:

1. Skanna och spara filer i databasen.
2. Söka smart bland dokument med hjälp av AI.
3. Rensa gamla filer genom att ta bort eller flytta dem.
4. Logga alla viktiga händelser så användaren kan se vad som har gjorts.

---

3. Tekniker och bibliotek

Python 3

Appen är byggd i Python 3 eftersom språket är enkelt att läsa och har många inbyggda funktioner för filhantering, databaser och AI. Python passar bra för små lokala verktyg som ska vara snabba att utveckla.

pathlib

Jag använder pathlib för att hantera sökvägar och filer på ett enkelt och säkert sätt. Det gör det lättare att gå igenom mappar, hitta filer och läsa information om dem utan att behöva skriva komplicerade strängar.

os

os används när jag behöver kommunicera direkt med operativsystemet, till exempel för att kontrollera om en fil finns, skapa mappar eller ta bort filer. Det är ett grundläggande bibliotek för att hantera filer och mappar.

pypdf

pypdf används för att läsa text från PDF-filer. Det gör att appen kan öppna PDF-dokument, plocka ut texten och använda den senare för sökning och AI-matchning.

markdown-parser

Markdown-filer behöver göras om till vanlig text. Jag använder en enkel Markdown-parser för att ta bort Markdown-format och bara behålla ren text som appen kan analysera och söka i.

sqlite3

sqlite3 används för att skapa och hantera den lokala databasen där all information sparas. SQLite är perfekt för min app eftersom den inte kräver en server och allt lagras i en enda fil på datorn.

AI-modell (MiniLM)

Jag använder en liten embedding-modell som heter MiniLM, som kommer från Sentence-Transformers. Den gör om text till siffror (embedding) så att appen kan jämföra dokument och hitta liknande innehåll. Modellen är snabb, gratis och går att köra helt lokalt utan internet.

logging

logging används för att spara loggar över vad appen gör, till exempel när filer tas bort, flyttas eller analyseras. Detta gör appen tryggare, eftersom man kan se historiken om något går fel.

4.AI-modell, embeddings och AI-minne

I min app använder jag en AI-modell som heter MiniLM. Jag är nybörjare i Python och AI, så jag behövde en modell som är liten, snabb och enkel att komma igång med. När jag började undersöka hur man kan hitta liknande dokument frågade jag också en AI-assistent om tips. Jag fick då rekommendationen att använda embeddings och att MiniLM är en bra modell för nybörjare. När jag läste mer såg jag att många använder MiniLM för just enkla och lokala projekt, så den passade min app perfekt. MiniLM är också lätt att installera. Det enda jag behövde göra var att köra:

pip install sentence-transformers

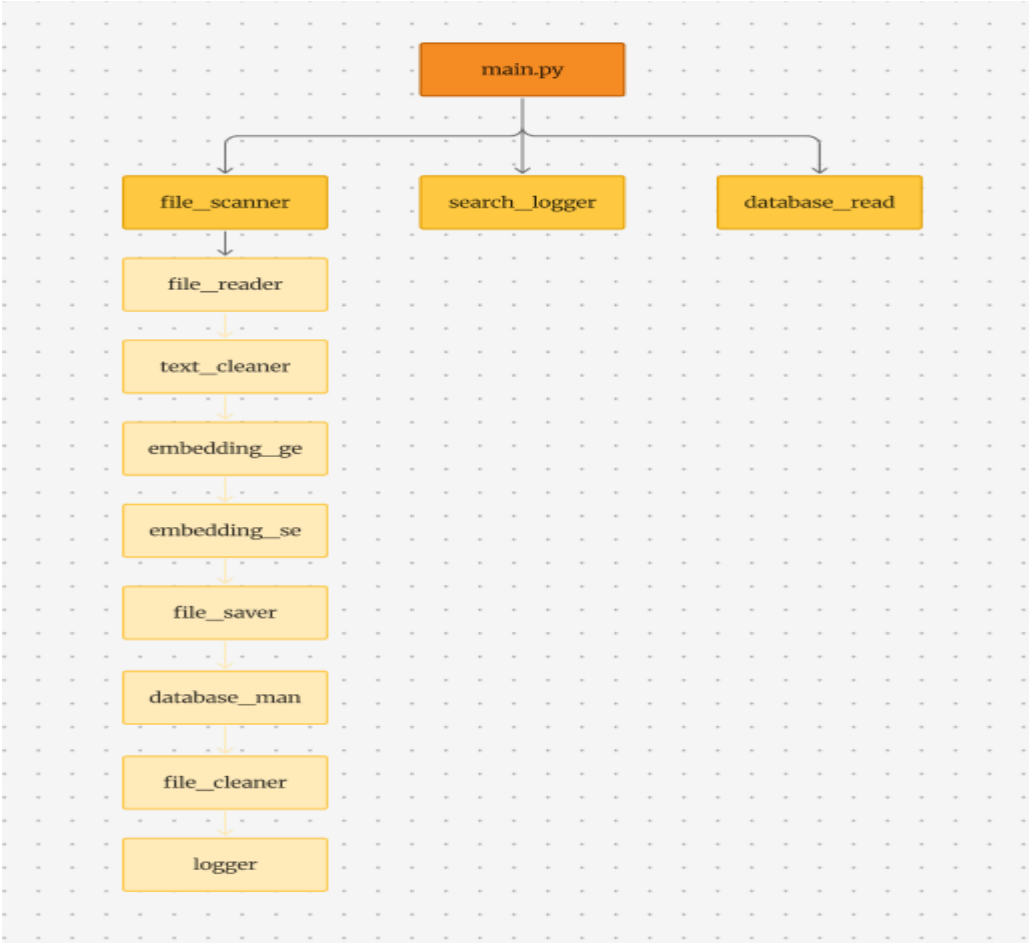
Sedan kan jag ladda modellen direkt i koden. Modellen laddas automatiskt första gången och sparas på datorn, så jag behöver inte göra något mer. MiniLM gör om text till siffror, så kallade embeddings. Dessa siffror visar vad texten betyder. När jag söker på ett ord gör appen om frågan till siffror och jämför den med siffrorna som finns sparade från dokumenten. Ju mer siffrorna liknar varandra, desto bättre matchning.
Jag kör modellen helt lokalt, eftersom appen jobbar med personliga dokument. Då skickas ingen text till internet, och allt stannar på datorn. MiniLM fungerar bra utan molntjänster och utan stark hårdvara, så den är ett bra val för en enkel lokal AI-assistent.

Det finns större modeller som BERT, RoBERTa eller GPT-modeller, men de är mycket större, långsammare och kräver ofta moln-AI eller en kraftfull GPU. Min app behöver bara kunna hitta liknande dokument, inte generera text, och därför valde jag en liten modell som MiniLM.
Min app använder också ett litet AI-minne. Det betyder att appen kommer ihåg vilka ord jag har sökt på tidigare. Den använder bonus-poäng för att göra framtida sökningar bättre. Bonusen tittar bland annat på:

- vilka ord jag har sökt förut
- hur många gånger jag sökt dem
- om orden finns i filens text
- om orden finns i sammanfattningen
- direkta matchningar i texten

Bonus-poängen läggs ovanpå den vanliga likhetspoängen. Till exempel, om jag söker på “azure” flera gånger får dokument om Azure lite extra poäng nästa gång jag söker. På så sätt lär sig appen vad jag är intresserad av och visar bättre resultat över tid.

5. Arkitektur / Mappstruktur



database_manager.py

Denna modul skapar databasstrukturer och de tabeller som används av resten av appen. Jag använder SQLite så att allt är lokalt och enkelt att hantera. Varje tabell har sin egen funktion, och många delar av appen är beroende av den här modulen.

file_scanner.py

Den här modulen tar emot en mappsökväg och går igenom hela mappen. Den hittar alla Markdown- och PDF-filer och samlar information om varje fil. Resultatet returneras som en lista som andra moduler kan arbeta vidare med.

file_saver.py

file_saver tar emot resultatet från file-scanner och sparar all metadata i databasen. Den använder “INSERT OR REPLACE” för att databasen alltid ska vara uppdaterad. På så sätt har AI-assistenten alltid korrekt information om alla filer.

file_reader.py

Denna modul har som uppgift att läsa textinnehållet från olika dokument. Den är nödvändig för AI-delen eftersom modellen behöver texten för att kunna skapa meningar och förstå innehållet. Utan file_reader har appen ingen text att analysera.

text_cleaner.py

text_cleaner fungerar som en städare i AI-flödet. När text extraheras från olika filer finns det ofta konstiga tecken, radbrytningar och onödiga whitespace. Den här modulen rengör texten så att AI får ren och strukturerad data, vilket ger bättre embeddings och bättre sökresultat.

embedding_generator.py

Denna modul gör två saker:

1. Hämtar text från varje fil.

2. Skapar en AI-vektor (embedding) som representerar textens betydelse.

Dessa embeddings sparas i databasen och används för att hitta liknande dokument när användaren gör en sökning. Utan denna modul skulle appen sakna AI, relevanssökningar och intelligens.

embedding_search.py

Detta är som appens sökmotor. Den tar emot användarens fråga, skapar en embedding av frågan och jämför den med alla embeddingar i databasen. Resultatet blir en lista av filer som är liknande. Modulen lägger också till en bonuspoäng baserad på AI-minnet och visar de fem bästa träffarna.

search_logger.py

Denna modul ansvarar för att logga alla sökningar i databasen. Varje gång man söker sparas frågan, tidpunkten, vilken fil som hade högsta score och hur stark matchningen var.

Den visar även loggning i terminalen med färgkodning så att man enkelt kan se hur bra matchningen blev.

file_cleaner.py

Detta är städmodulen i min AI-assistent. Den ansvarar för att hitta gamla filer och flytta dem till en “trash”-mapp på ett säkert sätt, utan att radera något permanent. Användaren får välja hur gamla filerna ska vara.

log_service.py

Används för att skriva ned viktiga händelser i appen i loggfilen data/log.txt. Den hjälper mig att hålla koll på vad som händer i systemet, speciellt när filer raderas eller flyttas.

main.py

Detta är appens kontrollcenter och meny. Den innehåller ingen AI-logik, ingen databaslogik och ingen städlogik. I stället anropar den alla andra moduler beroende på vad användaren väljer i menyn. Den skannar, sparar till DB, söker med AI, visar loggar och städar – men gör inget av det själv.

database_reader.py

database_reader är ett adminverktyg som läser filer från databasen och visar gamla filer eller filer från en viss mapp. Den ändrar inget i databasen, den visar bara information.

file_analyzer.py

file_analyzer räknar filstorlekar, beräknar hur gamla filerna är och visar statistik. Den gör inga ändringar i databasen och används mest för analys och debugging.

model_loader.py

Denna modul ansvarar för att ladda AI-modellen endast en gång. Sedan återanvänds samma modell i hela programmet. Det fungerar som en modell-cache och gör appen snabbare.

content_preview.py

content_preview visar de första cirka 300 tecknen av några filer i databasen. Detta gör att användaren snabbt kan se vad filerna innehåller utan att öppna dem.

---

6. Databasdesign (SQLite)

Jag använder SQLite eftersom min app är helt lokal och inte behöver en stor databasserver. SQLite sparas i en enda fil på datorn och är enkelt, snabbt och passar perfekt för små AI-appar. Det gör projektet lätt att installera och köra utan molntjänster eller extra konfiguration.

Mina tabeller i databasen

1. files (lagrar filinformation)

Denna tabell fylls när jag skannar en mapp. Här sparas metadata om varje fil.

Kolumner:

- id – unikt ID
- name – filnamn
- path – full sökväg
- size – filstorlek
- ext – filtyp (pdf, md)
- modified – senast ändrad datum
- text – den rensade texten från filen

Texten sparas som vanlig text (string). AI-delen använder denna text när den ska göra embeddings- och sökningar.

2. embeddings (AI-vektorer)

Här sparas embedding-vektorer som skapas från varje fil.

Embeddings är siffror som visar betydelsen av texten så att AI kan jämföra dokument.

Kolumner:

- file_id – koppling till files
- vector – embedding-vektorn (lagras som text/BLOB)

3. search_log (sökhistorik)

Här sparas alla sökningar som användaren gör.

Kolumner:

- query – vad användaren sökte
- timestamp – tidpunkt för sökningen
- top match – fil som matchade bäst
- score – hur stark matchningen var

Detta hjälper med historik, debugging och förbättringar.

4. user memory (AI-minne)

Denna tabell hjälper AI att förstå vad användaren brukar söka på.

Kolumner:

- keyword – ord som användaren sökt på
- bonus – extra vikt som påverkar framtida sökningar

Det gör att appen lär sig användarens intressen och kan ge bättre och mer personliga resultat över tid.

Hur appen hämtar data

Appen använder vanliga SQL-frågor (SELECT) för att hämta:

- metadata från *files*
- embeddings från *embeddings*
- historik från *search_log*
- bonus/AI-minne från *user_memory*

SQLite fungerar bra i Python via biblioteket sqlite3, och jag hämtar databasen som Python-objekt som sedan används i AI-sökningen eller rensningsfunktionen.

---

7. Funktioner

Funktionen är uppdelad i fyra delar: filskanning, sökning, rensning och loggning.

7.1 Filskanning

Appen börjar med att fråga efter en sökväg (path) till en mapp. När användaren anger mappen går appen igenom alla PDF- och Markdown-filer. Den samlar in metadata som:

- filnamn
- storlek
- sökväg
- filtyp
- senaste ändringsdatum

Appen läser även text från varje fil och skapar en kort sammanfattning. All metadata och text sparas i SQLite-databasen.

7.2 Sökning

När användaren skriver in ett ord, till exempel “azure”, gör appen:

1. om ordet till en embedding
2. jämför det med alla dokument embeddings
3. räknar ut base score
4. lägger på bonus från AI-minnet
5. visar de fem bästa resultaten med sammanfattning

Sökningen sparas även i loggen.

7.3 Rensning av gamla filer

Appen frågar hur många dagar gamla en fil ska vara för att räknas som gammal, till exempel 100 dagar. Den hämtar alla filer som inte ändrats på den tiden och visar listan.

Användaren väljer sedan om filerna ska:

- flyttas till trash-mappen

- eller lämnas kvar

Inget raderas direkt.

7.4 Loggar & historik

Appen sparar varje sökning i databasen. Loggen innehåller:

- vad användaren sökte
- när sökningen gjordes
- vilken fil som var top-resultat
- score
- om bonus användes

Det gör det lätt att se historik och förbättra framtida sökningar.

---

8. Installation och körning

Min applikation är helt lokal och kräver ingen server eller molninstallation. För att köra verktyget behöver användaren bara ha Python 3.12 eller senare installerat på datorn. Alla bibliotek som används är lätta att installera och fungerar direkt på Windows, Mac eller Linux.

När projektet laddas ner från GitHub öppnar användaren terminalen i projektmappen och installerar nödvändiga bibliotek med:

```
Python
pip install -r requirements.txt
```

Detta installerar bland annat PyPDF2, sentence-transformers, sqlite3-stöd och andra paket som behövs för AI-sökning och filhantering.
När installationen är klar går användaren in i `src/`-mappen och startar programmet med:

```
Python
python main.py
```

Programmet öppnas då i terminalen och visar huvudmenyn. Användaren kan direkt:

- skanna en valfri mapp
- söka efter dokument
- visa sökhistorik
- rensa gamla filer
- avsluta programmet

Ingen ytterligare installation krävs, och allt körs helt lokalt utan internetanslutning. Databasen (data/database.db) och loggarna (data/log.txt) skapas automatiskt om de inte redan finns.

9. Felhantering och loggning

För att undvika att appen kraschar har jag lagt in flera kontroller och try/except-block. Det finns många saker som kan gå fel när man arbetar med filer, och jag försöker fånga de vanligaste problemen.

Ett vanligt fel är att en fil kan vara tom, skadad eller låst av ett annat program. I sådana fall hoppar appen över filen i stället för att krascha. PDF-filer kan ibland vara trasiga eller svåra att läsa, och då visar appen ett felmeddelande och fortsätter med nästa fil.

Om databasen inte finns skapar appen en ny automatiskt. Samma sak gäller för trash-mappen. Om användaren har raderat den tidigare skapas en ny innan filerna flyttas, så att inget går förlorat av misstag.

Alla viktiga händelser loggas. Det kan vara:

- skannade filer
- sparade dokument
- sökningar och resultat
- fel som uppstod
- flyttade filer vid rensning

Loggarna lagras i filen data/log.txt.
I loggvisningen kan användaren se tidigare sökningar, tidpunkt, vilket ämne som söktes och vilket dokument som var träff nummer ett. Detta gör felsökning och uppföljning enklare och mer tydlig.

---

10. Tester

Jag har testat alla huvudfunktioner i appen manuellt för att se att allt fungerar som det ska. Jag använde både små mappar med några få filer och större mappar med många PDF- och Markdown-dokument. På så sätt kunde jag se att appen hanterar olika mängder filer utan att krascha.

9.1 Testning av filskanning

Jag testade att skanna flera olika mappar. Appen hittade rätt filer, läste metadata och sparade allt i databasen. Jag testade även att lägga in en trasig PDF och tomma filer för att se att appen inte kraschar, och det fungerade bra.

9.2 Testning av sökningen

Jag testade sökningen genom att skriva olika ord, till exempel “azure”, “test”, “python”. Appen visar alltid de fem bästa dokumenten och gav en base score. Jag testade också att söka samma ord flera gånger för att se att AI-minnet gav bonuspoäng, och det fungerade som tänkt.

9.3 Testning av rensning

Jag testade rensningfunktionen genom att söka efter filer äldre än t.ex. 30 dagar och 100 dagar. Appen visade rätt filer och kunde flytta dem till min trash-mapp utan att radera något permanent. Om trash-mappen saknades skapade appen en ny automatiskt.

9.4 Test av användare (klasskamrat)

Jag lät också en klasskamrat testa appen. Funktionerna fungerade för honom också, men han hade problem i början eftersom han saknade flera Python-paket. Efter att han installerade de saknade paketen fungerade appen som den skulle. Detta hjälpte mig förstå att jag behövde förklara installationen bättre.

---

11. Begränsningar och framtida förbättringar

Min app fungerar bra för det jag ville göra, men eftersom jag är nybörjare finns det flera saker jag inte hunnit eller kunnat göra ännu. Här är några begränsningar som finns idag:

Begränsningar

- Appen har bara terminal-meny och ingen grafisk design.
- Jag stödjer bara PDF och Markdown eftersom det var enklast för mig att börja med.

- AI-modellen jag använder är liten, men jag valde den för att den skulle funka på min dator.
- Om man har väldigt många filer kan det ta lite längre tid att söka.
- Appen rensar bara filer baserat på antal dagar, inget mer avancerat.

Framtida förbättringar

Eftersom jag fortfarande lär mig Python finns det saker jag gärna skulle vilja lägga till senare:

- Göra ett enklare GUI så att appen blir lättare att använda.
- Lägg till stöd för fler filtyper, t.ex. Word-dokument.
- Förbättra AI-delen när jag har mer kunskap.
- Göra rensningen smartare, kanske sortera filer efter kategorier.
- Lägg till automatisk backup eller schemalagd skanning.

---

12. Källor

- Python – officiell dokumentation
  https://docs.python.org/
- SQLite – officiell dokumentation
  https://www.sqlite.org/docs.html
- PyPDF2 / pypdf dokumentation
  https://pypdf.readthedocs.io/
- Sentence-Transformers (MiniLM-modellen)
  https://www.sbert.net/

- Stack Overflow – lösningar på mindre problem
  https://stackoverflow.com/

- ChatGPT – använt för förklaringar, lösning och tips under utvecklingen