

Feature: Smart File Finder (Human-Language Search)

Goal

User types:

“The PPT I made before placements”

System finds:

Placement_Presentation_Final.pptx (92% match)



Core Idea Behind This

Instead of searching by filename...

We search by **meaning**.

This is called:

Semantic Search



High-Level Architecture

```
User Query
      ↓
Convert to Embedding (Vector)
      ↓
Compare with File Embeddings
      ↓
Compute Similarity Score
      ↓
Rank Results
      ↓
Return Top Matches + Confidence %
```



Complete Tech Stack

Since you are comfortable with Python and ML:

◆ Backend

- FastAPI (recommended)
- Python

◆ Embedding Model

Option 1 (Best for Hackathon):

- sentence-transformers
- Model: all-MiniLM-L6-v2

Option 2 (Cloud-based):

- OpenAI embeddings API

For hackathon → use local model.

◆ Vector Database

You need fast similarity search.

Option 1 (Simple)

- FAISS (Facebook AI Similarity Search)

Option 2 (Production-level)

- Pinecone
- Weaviate

For hackathon → FAISS is perfect

◆ File Text Extraction

You need to convert files into text.

File Type Library

PDF	PyMuPDF
DOCX	python-docx
PPTX	python-pptx

File Type Library

TXT built-in

Code read as text



How This Works Step by Step

STEP 1 : Extract File Meaning

Don't just embed filename.

You embed:

- Filename
- Folder name
- Extracted text content
- Creation time context

Example:

For file:

Placement_Presentation_Final.pptx

You create description like:

Filename: Placement Presentation Final
Folder: 2026/Placements
Content: Slides about resume tips, interview prep...
Created: Jan 2026

Combine into one text block.

STEP 2 : Convert Files into Embeddings

```
from sentence_transformers import SentenceTransformer  
  
model = SentenceTransformer('all-MiniLM-L6-v2')  
  
embedding = model.encode(file_description)
```

Now each file becomes:

384-dimensional vector

Store this.

STEP **3** : Store in FAISS

```
import faiss
import numpy as np

dimension = 384
index = faiss.IndexFlatL2(dimension)

vectors = np.array(file_embeddings).astype('float32')
index.add(vectors)
```

Now you have vector database.

STEP **4** : User Query Handling

User types:

"The PPT I made before placements"

Convert to embedding:

```
query_vector = model.encode([query])
```

Search:

```
D, I = index.search(query_vector, k=5)
```

- I = indices of top files
 - D = distance
-

STEP **5** : Convert Distance to Confidence Score

Similarity:

```
similarity = 1 / (1 + distance)
confidence = similarity * 100
```

Now show:

Placement_Presentation_Final.pptx – 92% match



Folder Structure

```
smart-file-finder/
  └── backend/
      ├── main.py
      ├── scanner.py
      ├── embedding_engine.py
      ├── vector_store.py
      ├── search_engine.py
      └── database.py
  └── frontend/
      └── React UI
  └── data/
      └── file_metadata.db
```



Complete Backend Flow

When App Starts:

1. Scan folder
 2. Extract text
 3. Generate embeddings
 4. Store in FAISS
-

When User Searches:

1. Convert query → embedding
 2. Search FAISS
 3. Rank results
 4. Return JSON
-

🔥 Advanced Improvements (Judges Will LOVE This)

Add Time Understanding

If query contains:

- “before placements”
- “last semester”
- “during hackathon”

You detect time keywords.

Add time boost score.

Example:

```
if "before placements" in query:  
    boost files created before Jan 2026
```

Now it feels intelligent.

Add File Type Boost

If user says:

- “PPT”
- “PDF”
- “code”

Boost matching extensions.

Hybrid Ranking Formula

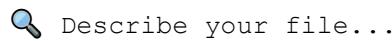
Final Score:

$$0.6 * \text{semantic_similarity} + 0.2 * \text{recency_score} + 0.2 * \text{filetype_match}$$

This gives better results than raw similarity.

Frontend UI Design

Search bar like:



Results show:

File	Match	Last Opened
Placement_Presentation_Final.pptx	92%	3 days ago
Resume_Template.pptx	61%	2 months ago



Execution Plan (Hackathon Friendly)

Phase 1 – Basic Version (MUST HAVE)

- File scanning
- Embeddings
- FAISS search
- Return ranked results

Phase 2 – Smart Boosting

- File type detection
- Time context
- Folder boost

Phase 3 – WOW Layer

- Explain why result matched
- Highlight matching phrases

Example:

“Matched because this file contains slides about placements.”



Why This Will Impress SanDisk

Because it turns storage into:

Intelligent Memory System

Traditional search:

- Exact match

Your system:

- Meaning match

That is a big difference.



Do You Need Deep Learning?

No.

Sentence transformers are pre-trained.

You are just using embeddings.



Important Design Decision

Where will files be?

Option A:

- Scan local user system

Option B:

- Upload files to web app

For hackathon demo:

👉 Create a demo folder with sample files.



Final Recommended Stack For YOU

Since you've worked with:

- Flask
- ML
- IBM Cloud

Use:

```
FastAPI  
SentenceTransformers  
FAISS  
SQLite  
React
```

That's clean and professional.



Final Summary

You are building:

1. File → text
2. Text → embedding
3. Store embeddings
4. Query → embedding
5. Compare
6. Rank
7. Return confidence

That's it.
