**💡 Enki’s Ideas**

(Enki, please list your ideas below using the format: E1, E2, etc.)

**Idea ID**: E1

**Project Title**: Social Media App With AI Driven Profile Customization

**Proposed Features**:

* Login/Sign Up page with OAuth and/or JWT driven authentication system.
* Ability for users to add a profile picture, background image, personal info, and bio to their profile
* Ability for users to type into a prompt box on their profile to request an AI bot to alter their profile page to look any sort of way they’d like it to. The AI bot will then go into the HTML/CSS of the page and directly alter it to follow the directives of the users prompt.
* Ability for users to connect their Spotify account to their profile so other users visiting their profile can see what kind of music they’re into, as well as maybe see what song they’re currently listening to.
* Ability for users to make posts. These posts can be text based, contain photos/videos, or even audio recorded by the user. The posts users make will then show up on their profile, and in the feeds of users they’re friends with. People can then like and comment on their posts. Users can adjust the privacy of their posts to be viewable by anyone who visits their profile or to only be viewable by people they’re friends with.
* Ability for users to add other users as a friend, as well as receive friend requests from other users, which will show up in a notifications tab.
* Ability for users to block other users, so that a blocked user can no longer interact with or see their profile or posts.
* Ability for users to message each other. Messages will be sent and  received in real-time through the use of web-sockets. I propose using [Socket.io](http://socket.io) for this feature.
* Page where users can view all the conversations they’ve had with various users over time. They should be able to go into any of the listed message threads on the page in order to continue the conversation with that user. Message threads containing a message the user has not yet opened and looked at should be visually distinguishable from message threads the user has already opened. Users should also be able to make group message threads, where multiple users can all be involved in a single thread. Users should also be able to leave group message threads. And delete message threads between themselves and an individual
* There should be an option for users to enable or disable read receipts for any given message thread.
* Users should be able to delete individual messages they send in a message thread.
* Users should be able to add “reactions”, such as a heart or thumbs up to messages in message threads.
* Users should be given the option to receive push notifications for things such as receiving a message, someone sending them a friend request, someone liking or commenting on one of their posts, etc.
* A feed page where users should be able to see and interact with posts made by other users they are friends with
* On the feed page the app should also showcase new friend suggestions to the user. We could create a machine learning algorithm that suggests new friends to the user based on having friends in common, location in common, interests in common, etc.
* A page where a user can view a list of all the other users they’re friends with
* We can potentially think up a specific niche to market the app towards, such as musicians for example, and create additional features oriented towards that, so that the app is less generic. I don’t have any specific ideas on that as of yet though.

**💡 Guli’s Ideas**

(Guli, please list your ideas below using the format: G1, G2, etc.)

Idea ID: G1

Project Title: Creating a KPI system for assessing employee performance by forming criteria.

Proposed features:

* Many companies struggle to figure out what performance criteria to use for evaluating employees. Our tool makes this easy: managers enter basic info about their company, and **AI automatically suggests the most relevant criteria** for assessment. This saves a lot of time.
* **Set Criteria with AI:** Managers describe the company (industry, goals, team roles), and AI suggests tailored performance criteria (e.g., meeting deadlines, teamwork, innovation).
* **Collect Data:**
  + Managers or team leads input employee results (like project completions or sales numbers).
  + The system can integrate with existing tools (e.g., Trello, Jira, or timesheets) to pull data automatically.
  + Employees can self-report progress if needed.
* **Score and Analyze:** The system calculates performance scores based on the selected criteria and visualizes progress with charts, dashboards, or KPI meters.
* **Generate Insights:** AI identifies trends (e.g., “Team A is improving in collaboration but falling behind on deadlines”) and suggests areas for improvement.
* **Reports:** Monthly or quarterly performance summaries are created for managers to use during reviews.
* And about tools, I honestly do not know. React for frontend, python for backend, Firebase for database

**💡 Shashank’s Ideas**

(Shashank, please list your ideas below using the format: S1, S2, etc.)

**Idea ID:** S1

**Project Title:** Sustainable Shopping Assistant – AI-Powered Eco-Friendly Product Recommender

**Proposed Features:**

* **Product Input via Search or Scan**: Users can enter product names manually, scan barcodes, or use OCR to extract product text from labels or packaging.
* **Sustainability Scorecard**: After identifying the product, the system provides a visual breakdown of its sustainability metrics, such as:  
    
  + Environmental impact (carbon footprint, biodegradability)
  + Ethical sourcing (labor conditions, fair trade)
  + Packaging recyclability and plastic content
  + Vegan or cruelty-free status (where applicable)
* **Green Alternative Recommendations**: Based on the product's sustainability score, the app uses embeddings and semantic similarity to suggest greener alternatives available on the market. These are also scored and explained.
* **AI-Powered Justifications (RAG Pipeline)**: The assistant explains why certain products are more sustainable using a Retrieval-Augmented Generation (RAG) approach. It pulls supporting text from sustainability blogs, open databases, and certification descriptions (e.g., “Rainforest Alliance”).
* **Educational Q&A Panel (Optional)**: Users can ask freeform questions like “Why is palm oil unsustainable?” or “How do carbon credits work?” The assistant answers using a hybrid of LLM and retrieved content.
* **Simple and Transparent UX**: The interface is designed to be intuitive for non-technical users. For each product, it clearly shows:  
    
  + A traffic-light sustainability rating
  + Explanations for the score
  + Eco-badges and certifications
  + Link to better alternatives
* **User Profile (Stretch Feature)**: Allows users to:  
    
  + Track eco-friendly purchases over time
  + Set sustainability preferences (e.g., prioritize low water usage or vegan-only products)
  + Save favorite green alternatives
* **Multi-language Support (Stretch Feature)**: Localized results and explanations for users in different countries or regions (e.g., EU eco-labels, Indian BIS eco-certifications).
* **Admin/Contributor Dashboard (Stretch Feature)**: Verified users can suggest or rate product entries and sustainability sources to grow the database over time.
* **Tech Stack Proposal**:  
    
  + **Frontend**: Streamlit or React (depending on scope)
  + **Backend**: Python, FastAPI, LangChain
  + **Embedding Search**: Qdrant or FAISS
  + **LLM**: OpenAI or Hugging Face for scoring + summarization
  + **OCR/Barcode API**: Tesseract OCR, Open Food Facts, or UPCItemDB
  + **Data Sources**: Open Food Facts, EWG, Sustainability blogs, Product label databases
* **Target Audience**: Eco-conscious consumers, especially Gen Z and Millennials; can also target sustainability influencers, zero-waste communities, or even small eco-friendly product businesses.

**Idea ID:** S2

**Project Title:** Local Semantic Search Engine for Personal Document Archives

**Proposed Features:**

* **Folder-Based Document Indexing**: The system monitors a designated folder (e.g., /AI\_docs/) and automatically processes any .pdf or .docx files added to it.
* **Local Vector Store Creation**: Text content from the documents is parsed, chunked, and embedded using a local embedding model (e.g., all-MiniLM-L6-v2 from SentenceTransformers). Embeddings are stored in a local vector database (such as **Qdrant**, **Chroma**, or **FAISS**) for offline access.
* **Semantic Search Capability**: Users can input questions or search queries in natural language. Instead of returning just filenames, the engine retrieves the most relevant **document chunks** and:  
    
  + Displays a **preview snippet** from the text
  + Provides a **clickable file path** to open the full document in its native app (e.g., Word, PDF reader)
* **Offline-First Design**: The entire system is designed to work without an internet connection, making it ideal for personal knowledge archives, air-gapped environments, or data-sensitive workspaces.
* **Optional Local LLM Integration**: A lightweight, locally-run LLM (such as ggml versions of **TinyLlama** or **Mistral-7B-Instruct**) can optionally be used to:  
    
  + Rephrase user queries into optimized embedding queries
  + Generate brief summaries of retrieved chunks
  + Offer smart suggestions based on the retrieved material
* **Incremental Updates**: A lightweight scheduler or filesystem watcher monitors the source folder. When 5–10 new files are added or updated, the vector database is **automatically updated** with the new embeddings, without reprocessing the entire archive.
* **Simple UI Options**:  
    
  + Option 1: A local web interface (Flask or Streamlit) to input search queries and display results with **“Open Document” buttons**.
  + Option 2: A minimal local desktop launcher (.bat file) that opens the search interface.
  + Option 3: CLI-based interface for advanced users with filters (e.g., by filename, date, or tag).
* **Use Case Motivation**:  
   Designed by a Hyperskill AI Engineering Bootcamp learner to manage and search through a growing archive of AI/NLP documents. Keyword search fails to deliver **content-level precision**, and LLMs like ChatGPT can't access local files or work offline — hence, the need for a **personal AI search engine**.
* **Tech Stack Proposal**:  
    
  + **Parsing**: PyMuPDF, python-docx
  + **Embeddings**: SentenceTransformers
  + **Vector DB**: Qdrant or Chroma (local setup via Docker or Python API)
  + **LLM (optional)**: llama.cpp or Ollama for TinyLlama, Mistral, or Phi-2
  + **UI**: Streamlit or Flask (launchable via .bat or shell shortcut)
  + **Auto-update Trigger**: watchdog (Python lib) or cron job
  + **File Access**: Return full document path with each result; clicking opens file via OS default application
* **Stretch Features**:  
    
  + File-level tagging or metadata indexing (author, topic)
  + Semantic summarization dashboard (e.g., “What topics did I read most about last month?”)
  + Exportable results or shareable answer cards
  + Optical character recognition (OCR) for scanned PDFs

**Idea ID:** S3

**Project Title:** CV-Driven Job Matcher & Personal ATS Companion

**Proposed Features:**

* **CV-Centric Input System**: Users upload one or more .pdf or .docx CVs. The system uses a **hybrid parsing pipeline** to extract structured information:  
    
  + spaCy (or pyresparser) is used to extract base fields like name, contact, education, and raw skills.
  + Simultaneously, the **semantic content** of work experience, projects, and skill descriptions is converted into embeddings for flexible, meaning-aware comparison.
* **Job Discovery Assistant**:  
    
  + Users enter desired job titles, locations, and optional filters (e.g., remote-only, min salary).
  + The system performs **Google-based job search queries** and scrapes or aggregates job listings from LinkedIn, Google Jobs, and Indeed.
  + This search works as a **fallback discovery layer**, especially useful when no structured APIs are available.
* **Job Ad Parsing and Semantic Matching**:  
    
  + Each job ad's text is parsed into structured and semantic layers:  
      
    - spaCy: extracts required skills, job title, location, and qualifications.
    - SentenceTransformers: embeds the full job description.
  + Each CV is semantically compared with each job ad using **cosine similarity**.
  + A **fit score** is calculated using a combination of:  
      
    - Embedding similarity (major weight)
    - Rule-based overlap (e.g., missing required skills penalize the score)
* **Match Dashboard**:  
    
  + Outputs a sortable, filterable table with:  
      
    - Job title, company, location, salary, deadline
    - Application link (clickable)
    - **Semantic fit score** for each CV
  + Clicking the result opens the job link for fast application
* **Multi-CV Plug-and-Play Support**:  
    
  + Users can upload multiple CVs and instantly see which version performs better for each job search.
* **Tailored CV Suggestions (Stretch Feature)**:  
    
  + The tool uses an LLM (e.g., GPT or TinyLlama) to suggest skills or phrases to add to a CV to improve scores.
  + Users can download a revised, job-specific CV generated via LLM.
* **Simple Frontend**:  
    
  + Streamlit or Flask app interface to:  
      
    - Upload CVs
    - Enter job preferences
    - View results
    - Download reports or matched CVs

**Use Case Motivation**:  
 Most job seekers waste time searching manually and tailoring resumes for jobs they aren't truly aligned with. This tool makes **the CV the entry point**, semantically matches job ads from across the web, and enables plug-and-play comparison of different CV versions — all while minimizing the effort needed to track and apply to the best-fit roles.