**HACKATON – AI AGENTER 2025.**

**Fokus: Boosta dina studier**

**5 ideér att hämta inspiration från.**

**1. Text Summarization Crew**

**Agents**

1. **Key Points Agent**
   * **Function:** Takes a block of text and produces short bullet-point summaries.
   * **Possible Implementation:**
     + *OpenAI API:* Call the GPT-3.5 (or similar) completion endpoint with a prompt that requests bullet points.
     + *Hugging Face Transformers:* Use a pre-trained summarization model (e.g., [BART](https://huggingface.co/facebook/bart-large-cnn), [T5](https://huggingface.co/t5-base)).
2. **Chapter Outline Agent**
   * **Function:** Identifies section headers and main subpoints from text.
   * **Possible Implementation:**
     + *NLP Library (e.g., spaCy, NLTK):* Parse text into sentences/paragraphs, and then attempt to detect headings or generate a simple outline by identifying topic sentences.
     + *OpenAI / LLM prompting:* Ask the model to split the text into logical sections and label them.

**Structure & Data Flow**

* **User Input:** Plain text (e.g., a copied textbook chapter).
* **Agent Pipeline:**
  1. **Key Points Agent** → Summaries
  2. **Chapter Outline Agent** → Outline
* **Output:** Concise bullet-point list + Outline or headings.

**API & Infrastructure Needs**

* **LLM or Summarization Model Endpoints:**
  + **OpenAI API** (REST-based; pass text in JSON, get summary in response).
  + **Hugging Face Inference API** or self-hosted model in a simple Python microservice (Flask/FastAPI).
* **Data Handling:** Basic JSON or text file I/O.
* **Deployment:** Can be a lightweight container or serverless function receiving text input and returning summarized text.

**2. Basic Writing & Grammar Crew**

**Agents**

1. **Grammar Checker Agent**
   * **Function:** Finds spelling/grammar mistakes and suggests corrections.
   * **Possible Implementation:**
     + *LanguageTool API:* Open-source grammar checking library with a ready-made API.
     + *Proprietary Services (e.g., Grammarly Business API, if accessible)*
     + *OpenAI GPT-3.5 Approach:* A carefully crafted prompt to detect and correct errors.
2. **Style Editor Agent**
   * **Function:** Highlights long or repetitive sentences; suggests simpler phrasing.
   * **Possible Implementation:**
     + *LanguageTool / Custom NLP Routines:* Check sentence length, repeated words, or passive voice.
     + *LLM Approach:* Provide the text to an LLM with instructions like “Identify and improve any awkward or overly long sentences.”

**Structure & Data Flow**

* **User Input:** Draft text (copied/pasted or typed).
* **Agent Pipeline:**
  1. **Grammar Checker Agent** → returns corrected suggestions
  2. **Style Editor Agent** → returns style improvement suggestions
* **Output:** A corrected and polished text (or a set of suggestions the user can accept/reject).

**API & Infrastructure Needs**

* **Grammar Checking API/Library:** LanguageTool (self-hosted or SaaS).
* **Style Checking:** Could be done with the same library or an LLM prompt.
* **Data Storage:** Possibly none needed beyond ephemeral text.
* **Front-End:** Could be a simple text box on a web page, or an editor plugin.

**3. Flashcard & Quiz Crew**

**Agents**

1. **Flashcard Generator Agent**
   * **Function:** Takes a list of Q&A pairs (or terms/definitions) and formats them into flashcards.
   * **Possible Implementation:**
     + *Static Approach:* Simple script that converts text or CSV input into a JSON structure for flashcards.
     + *LLM Approach:* If text is less structured, use an LLM to extract Q&As from paragraphs.
2. **Quick Quiz Agent**
   * **Function:** Creates short quizzes (multiple-choice, true/false, fill-in-the-blank) from the same data.
   * **Possible Implementation:**
     + *Simple Script:* Use Q&A data to automatically generate multiple-choice distractors.
     + *OpenAI or Hugging Face LLM:* Given a topic, ask the model to propose multiple-choice questions with plausible incorrect answers.

**Structure & Data Flow**

* **User Input:** A CSV file or text with terms and definitions.
* **Agent Pipeline:**
  1. **Flashcard Generator Agent** → outputs structured flashcards.
  2. **Quick Quiz Agent** → uses flashcard data to generate quizzes.
* **Output:** A set of flashcards (digital or printable), plus short quizzes.

**API & Infrastructure Needs**

* **Minimal ML:** Could be purely rule-based with optional LLM usage to generate distractor answers.
* **Data Format:** CSV → JSON for flashcards/quizzes.
* **Front-End:** A simple web page or a chatbot interface to display flashcards and quiz questions.

**4. Goal-Setter & Reminder Crew**

**Agents**

1. **Goal Planner Agent**
   * **Function:** Lets students input specific goals (in text) and sees if they are SMART (Specific, Measurable, Achievable, Relevant, Time-bound).
   * **Possible Implementation:**
     + *Simple Regex / NLP Check:* Ensure the goal has a date, a clear target, etc.
     + *LLM Approach:* Evaluate user’s goal statement and give feedback on how to refine it.
2. **Deadline Reminder Agent**
   * **Function:** Sends text/email/SMS reminders about upcoming deadlines.
   * **Possible Implementation:**
     + *Basic Scheduling (e.g., cron job or serverless function):* Check if the deadline is within X days, then trigger a reminder.

**Structure & Data Flow**

* **User Input:** Text-based goals and deadlines (“Finish Chapter 3 by Friday”).
* **Agent Pipeline:**
  1. **Goal Planner Agent** → formats goals into a structured list with deadlines.
  2. **Deadline Reminder Agent** → triggers notifications.
* **Output:** Regular reminders or “to-do” lists for the student.

**API & Infrastructure Needs**

* **Scheduling System:** Cron jobs or something like AWS Lambda scheduled events for sending reminders.
* **Notification Channels:** Simple email (SMTP) or SMS API (Twilio).
* **Goal Checking (optional):** Small LLM usage or rule-based template.

**5. Basic Q&A Crew**

**Agents**

1. **FAQ Agent**
   * **Function:** Stores a small knowledge base of common Q&A pairs (e.g., course info, assignment details).
   * **Possible Implementation:**
     + *Simple Database or JSON File:* Each question is a key, and each answer is a value.
     + *Vector Search Option (Optional):* For short text retrieval, you could store embeddings in a local vector database (e.g., [Faiss](https://github.com/facebookresearch/faiss), [Pinecone](https://www.pinecone.io/)).
2. **Answer Finder Agent**
   * **Function:** When a student asks a question (text-based), the agent looks up the closest match in the FAQ or knowledge base.
   * **Possible Implementation:**
     + *Keyword Matching:* A straightforward approach (find the best keyword overlap).
     + *LLM Embeddings:* Convert queries and FAQ answers to embeddings, then find the top similarity score.

**Structure & Data Flow**

* **User Input:** A text question (e.g., “When is homework due?”).
* **Agent Pipeline:**
  1. **FAQ Agent** (static knowledge)
  2. **Answer Finder Agent** (search & retrieve)
* **Output:** Best matching Q&A entry.

**API & Infrastructure Needs**

* **Data Storage:** Could be a JSON file with Q&A pairs or a small DB table.
* **Search Logic:** Either simple string matching or a minimal vector database approach.
* **Front-End:** Basic text input box, returns an answer in text.