

PROCESS.md: Meeting Summarization Tool

AI במהלך הפיתוח: עבדתי לפי 5 עקרונות הליבה של גוגל לעבודה נכונה עם AI בנוספ', שאלתם כיצד השתמשתי ב (עברית) קורס שלהם בזמןו מטעם מקום העבודה הקודם (הקודם שלו)

עקרונות הליבה 1

גוגל מבנה את כל שלב פרומפטינג סביב 5 עקרונות ליביה: **משמעות, קשר, דוגמאות, הערכתה ואיתרציה**.

1. (Task) משימה.

יעשה. לא הנושא הכללי, אלא הפלט המדוקש שאתה צריך AI-הבסיס הוא ה-**משמעות**: פשוט מה אתה רוצהše

- "**משמעות גורעה**": "עזר לי עם המיל"
- "**משמעות טוביה**": "נסח מחדש המשפט כדי לכתוב מיל לצוות חדר הקשר שלי בזוויג לשינוי בלוח הזמנים"

2. (Context) הקשר

מגדירה את ה"עדשה" (אוצר מילים ולוגיקה: **פרסונה**).

- דוגמה: לבקש תוכנית אימון זה בסדר, אבל הוספה "פעל **כפייזוטרפייט**" מבטיחה שתתקבל טיפים
 - לבתיות ומיקוד אנטומי, ולא סתם רשימת תרגילים

מגדיר את מבנה הפלט כדי להימנע מגושי טקסט גנריים: **פורמט**.

- JSON-קובץ ב, או Markdown דוגמאות: רשימת בולטים (נקודות), טבלת
 - ...

3. (References) דוגמאות.

"**יצטרך לנחש פחות AI**-נתוני רקו שמקווים את המודל. **"כלל שתספק יותר מידע, כך זה"**

דוגמה לתרכיש: טקסט לדף נחיתה

- "**פורמט גנרי**": כתוב טקסט לדף נחיתה לאתר שלך
 - פורמט עשיר בהקשר: "אני בונה כלי לניהול פרויקטים עבור מעצבים פרילנסרים. קהל היעד הוא גילאי שהם מוככים מדי. המוצר שלי מתמקד בצריכי זמן ויזואליים Asana עד 40 והם מותוסכים מכליים כמו וורדלים ללקחות. שמור על טון מקצועי, אבל חם"

4. (Evaluate) הערכתה

למה אתה מכון (אווירה, מבנה, סגנון). דוגמאות הופכות הוראות מעורפלות ליעדים AI-דוגמאות שמראות ל מוחשיים."

- תיאורי מוצר**: הדבק שלושה מהתיאורים הטוביים ביותר שלך ואמור למודל: "כתב תיאור חדש באותו סגנון כמו".
"הדוגמאות האלה"
לנתח למה הם עבדו -> בקש ממנו AI-**מדיה חברתית**: אין לו את ה פוסטים המכיצלים שלך -> אמור ל לייצר פוסטים חדשים שעוקבים אחר אותה תבנית בדיק

5. (Summarize) סיכום

בדוק באופן שיטתי אם הפלט תואם **משמעות**, פוגע בטון הנכון ומסתמך על **נתונים מדויקים**. אל תשתקפק ב"מספיק". טוב.

5. איטרציה (Iterate)

פרומפטינג הוא לולהה: בקש -> בדוק -> התאם -> בקש שוב.

שיטת לתיקון פרומפט שכור 4:

חוזר למסגרת: חזר להתחלה. האם פספסת הקשר? האם שכחת את הפרסונה?

פצל למשפטים פשוטים יותר: הימנע מהוואות ארוכות ומסורכבות.

הכל בנשימה (KPIs) עם תקציבים מודיעי ביצוע Z-רע: "צורך אסטרטגיה לרבעון הראשון המכוננת לדור ה... אחת"

כלול תקציב. הוסף מודיעי ביצוע Z-טוב: "צורך אסטרטגיה לרבעון הראשון. כוון לדור ה..."

השתמש במשמעות מקובלות: החלף את המסגרת/המודול המנטלי.

דוגמה: אם "כתב הצעה עסקית" נותן תוצאות יבשות, נסה **"כתב טיעון משכנע לשוטפות"**.

הוסף אילוצים: כופה יצירתיות.

דוגמה (רעיון וידאו): "חייב להיות מתחת ל-90 שניות, חייב להתמקד בטיפ יחיד, חייב להתחיל בשאלת "

2. Pre-Development Planning (How I Designed the System)

Before writing a single line of code, I spent time decomposing the brief into a modular, risk-mitigated execution plan. My goal was to move from a simple script to a production-grade asynchronous API. I broke the development into 8 distinct phases:

- **Phases 0-1 (Foundations):** I mapped out the API contract and the directory structure to separate business logic from AI orchestration.
- **Phases 2-3 (Data & Transcription):** I focused on the "Truth Source"—building a robust ingestion system for audio and integrating the Whisper API to ensure high-fidelity raw text. על החלק הזה אני-אשכח מהרeric אם תרצו
- **Phase 4 (The AI Brain):** I designed the System Prompt governance, treating prompts as version-controlled code in a dedicated directory.
- **Phases 5-7 (Output & UI):** I implemented the Word exporter and a reactive UI with polling to handle the long-running nature of AI tasks.
- **Phase 8 (Iteration):** I performed final quality control, testing for hallucinations and structural integrity.

3. The "Smart Use of AI" Strategy

I utilized AI not just for code generation, but for architectural decision-making and quality governance.

A. Prompt Chaining & Modular Architecture

I avoided the common mistake of a monolithic prompt. Instead, I chained the processes: Whisper-1 provides the transcription (Phase 2), and GPT-4o provides the analysis (Phase 3). This isolation allows me to verify the transcript before it is summarized.

B. Implementation Examples

I used specific prompts to scaffold my backend and frontend layers.

- **Example Prompt I used:** "Act as a Senior Python Developer. Based on my Design Spec, implement the POST /upload route using FastAPI and Motor. Ensure it saves with a unique ID and triggers a background

task."

4. The Winning System Prompt

I built the following Expert Persona prompt to drive the summarization logic:

```
# PERSONA
Senior Executive Assistant and Business Analyst (15 years exp).
# CONTEXT & CONSTRAINTS
- ACCURACY: Use ONLY the provided transcript.
- BRAIN OVER PATTERN: Do not hallucinate. Use null/[] for missing info.
- REFERENCE: (Few-Shot example included in the prompt code)
- FORMAT: Strict JSON only.
```

Why I built it this way:

- **Few-Shot Learning:** I included examples of a perfect summary to turn vague instructions into concrete targets, significantly reducing "Hallucinations".
- **Zero-Inference Rule:** I instructed the AI to prioritize accuracy over completeness, using null instead of inventing participants.

5. Independent Technical Wins

- **Asynchronous Flow:** I chose FastAPI BackgroundTasks and MongoDB (Motor) to ensure the server remains responsive while handling 5-minute transcription jobs.
- **Environment Problem Solving:** I successfully navigated port conflicts and MSYS2 environment restrictions by configuring custom virtual environments and port mappings (8001 for Backend, 5174 for Frontend).
- **Status Safety Gates:** I implemented a logic gate in the Word exporter to prevent downloads before a job status is COMPLETED.

6. example of User Prompts i used and & System Prompt

User Prompt

1. **Architecture & Planning:** "Act as a Senior Software Architect. I need to build a Meeting Summarization tool using FastAPI, React, and OpenAI. I created the following system design myself, using your experience in architecture as a senior software engineer, give me suggestions and improvements to the system design. my design:

```
/root └── backend/ | └── app/ | └── main.py # FastAPI entry point | | └── api/ # Endpoints (upload, status, download) | | └── services/ # Business Logic | | | └── transcription.py # Whisper API integration | | | └── ai_analyst.py # LLM Orchestration | | | └── exporter.py # Word document generation | | └── prompts/ # System Prompts library (Version controlled) | | └── database/ # MongoDB models and session | └── uploads/ # Raw audio files | └── outputs/ # Generated Word files └── frontend/ | └── src/ | └── components/ # UI: Upload, Progress, ResultsView | | └── hooks/ # usePolling for job status | └── services/ # API client └── PROCESS.md # Critical documentation
```

2. **Backend API Implementation:** "Act as a Senior Python Developer. i have provided you with a template i made for fastApi help me Implement the FastAPI POST endpoint `/upload` that accepts an audio file, saves it to a local directory, creates a record in MongoDB with a unique ID, and triggers a background task for transcription." then let me review it and if needed make changes' i will change it myself
3. **Frontend State Management:** "Act as a Frontend Engineer. Create a React component using Tailwind CSS that handles file uploads. After upload, it should start polling a GET `/status/{job_id}` endpoint every 5 seconds until the status is 'COMPLETED' or 'FAILED'." - Give me to review every step of the process and follow my Generic Frontend Template i provided for you
4. **Word Document Export Logic:** "Write a Python function using the `python-docx` library that takes a structured JSON summary (including participants, agenda, and action items) and converts it into a professionally formatted .docx file with bold headers and bullet points." - Give me to review every step of the process. remember that it is the first time i work with this library. so we review every step of the process

System Prompt

```
SYSTEM_PROMPT = """
```

PERSONA

Act as a Senior Executive Assistant and Business Analyst with 15 years of experience in corporate governance. Your goal is to transform messy meeting transcripts into high-fidelity, actionable executive summaries.

TASK

Analyze the provided meeting transcript. Extract and structure the information into a valid JSON format. Follow the provided schema strictly.

CONTEXT & CONSTRAINTS

- ACCURACY: Use ONLY the provided transcript. If a detail (like a name or deadline) is not mentioned, use null or [].
- BRAIN OVER PATTERN: Do not hallucinate. If the transcript is cut off or unclear, prioritize accuracy over completeness.
- TONE: Professional, concise, and executive-level.
- LANGUAGE: Output in the same language as the transcript (Hebrew or English).

REFERENCE (Few-Shot Example)

Input Transcript: "Dan: We need to fix the login bug by Tuesday. Sarah: I'll take it. Mike: Agreed, and let's use the new API." Output: { "summary": "The team discussed a critical login bug and agreed on a fix using the new API.", "participants": ["Dan", "Sarah", "Mike"], "decisions": ["Fix login bug using the new API"], "action_items": [{ "task": "Fix login bug", "owner": "Sarah", "due_date": "Next Tuesday", "notes": "Must use the new API" }] }

OUTPUT SCHEMA (JSON ONLY)

```
{ "summary": "string (1-2 concise paragraphs)", "participants": ["string"], "decisions": ["string"], "action_items": [ { "task": "string", "owner": "string or null", "due_date": "string or null", "notes": "string or null" } ] }
```

FINAL EVALUATION

- Ensure JSON is valid and parsable.
- No markdown code blocks (`json ...`).
- No conversational filler before or after the JSON. """"

Modular User Prompt for clear data separation

USER_PROMPT_TEMPLATE = """ Below is the transcript for analysis: {transcript} """ BTW, i would like to add here please that the use of XML-style tags () in the prompt is a highly effective way to provide Context and prevent the model from confusing its instructions with the meeting data -tamir

7. Challenges & Solutions

- **The Hallucination Hurdle:** During early testing, the AI occasionally "invented" participants or deadlines to fill a complete report.
 - **Solution:** Implemented a strict "**Zero-Inference Rule**" in the System Prompt, instructing the model to return `null` or `[]` for missing data rather than guessing.
- **Large Audio & Timeout Risks:** Processing long meetings created the risk of API timeouts and context limits.
 - **Solution:** Chose **Prompt Chaining**—breaking the process into sequential steps (Transcription → Analysis) to ensure the "truth source" (the transcript) is saved before analysis begins.
- **The Environment "Wall":** Developing in a restricted MSYS2/Windows environment caused pathing and port conflicts.
 - **Solution:** Configured a custom virtual environment and mapped the backend to **Port 8001** and the frontend to **Port 5174** to bypass system restrictions.

8. Actual Timeline & Run Instructions

Total Time Taken: 4.5 hours (including setup, architecture, and testing).

How to Run:

Prerequisites

- Python 3.10+
- Node.js 18+
- MongoDB (Running locally on default port 27017)

1. Environment Setup

Create a `.env` file in the `Meeting_Sum` root directory:

`#.env:`

i will add the file here because this is a test- in real life i will add it to the gitignore

```
MONGODB_URL=mongodb://localhost:27017
OPENAI_API_KEY=sk-proj-dgOvYMNtV5KuIUyVtjjXZLXy15JiQw1koRKd3C1q0LsSQPGinEQM-
ni71fANBslrSw70yiytljT3BlbkFJkYzpsUiY5y5qc6ZnG-d1w70xeWa-GDPPzVKGk8GyPjDlcJ-
oLzVeWODAs9JhexG2xj-w12_SYA
WAVESPEED_API_KEY=cc8df955e4f04fbe675f97983d0a30429866de0a1567f272a89416dd0580f59
```

2. Clone Repo & Navigate to `Meeting_Sum`.

3. Environment: Add `OPENAI_API_KEY` to `.env`.

4. Backend: `python -m uvicorn app.main:app --host 127.0.0.1 --port 8001`

5. Frontend: `cd frontend && npm run dev` (Runs on port 5173 or 5174).