

Schriftliche Arbeit

Zen AI

Jugend Forscht Junior

Teilnehmende: Bennet Wegener

Ort: Adolf-Reichwein-Gymnasium, Heusenstamm

Projektbetreuender: Andreas Schickedanz

Name des Projekts: Zen AI

Fachgebiet: Informatik

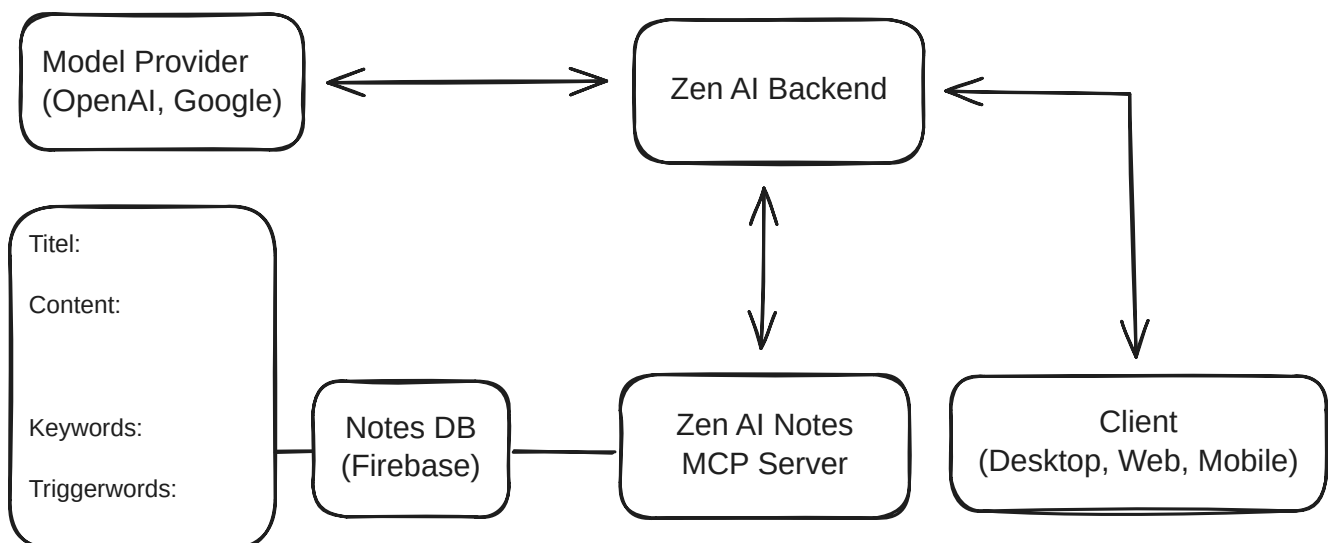
Wettbewerbsspalte: Jugend Forscht Junior

Bundesland: Hessen

Wettbewerbsjahr: 2026

Projektüberblick:

Ich verbessere die Effizienz von KI-Modellen, indem ich Probleme wie die Speicherung von Informationen über den Nutzer und wie man sie mit der KI teilt, verbessere. Hierzu erstelle ich keine neuen KI-Modelle, sondern baue auf bestehenden auf, indem ich ein sogenanntes MCP-Tool gebaut habe.

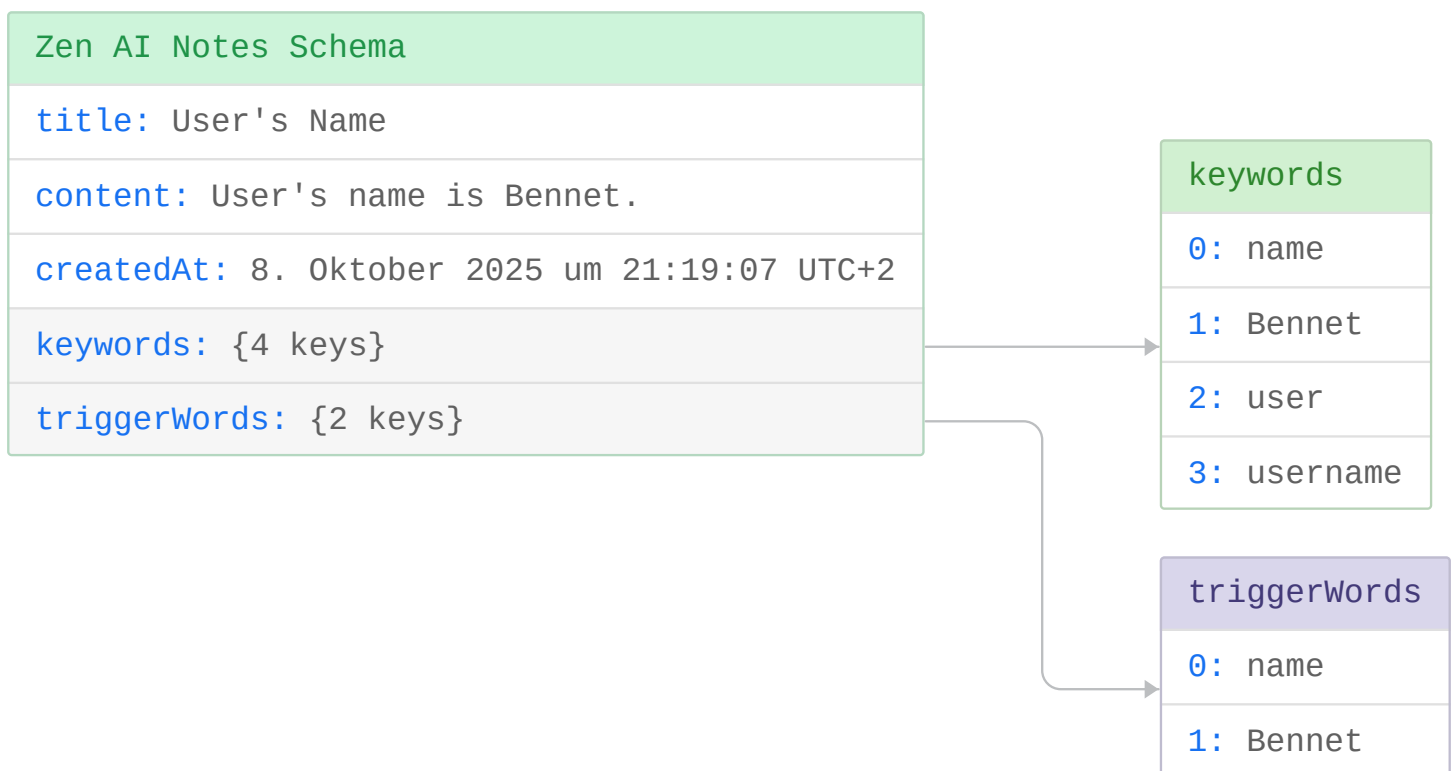


Inhaltsverzeichnis

Projektüberblick	1
Inhaltsverzeichnis	2
Fachliche Kurzfassung	3 - 4
Motivation und Fragestellung	4
Grundlagen	4
Vorgehensweise	4 - 7
Ergebnis	8
Ausblick	8 - 9
Anmerkungen	10

Fachliche Kurzfassung:

Das Ziel von Zen AI ist es, die bereits bestehenden Large Language Models (kurz LLMs) drastisch zu verbessern. Ein großes Problem für KI-Agenten ist, sich Informationen über den Nutzer zu merken. Dies stellt für LLMs ein großes Problem dar, denn wenn man ihnen zu viel Kontext mitgibt, entsteht das sogenannte „[Needle-in-a-Haystack](#)“-Problem. Dieses beschreibt, dass ein LLM bei zu viel Kontext ungenauer wird, da es die Nadel (die relevante Information zur Beantwortung der Frage) im Heuhaufen (dem gesamten Kontext) suchen muss. Ich habe dieses Problem gelöst, indem für jede Information über den Nutzer eine sogenannte Note angelegt wird. Eine Note enthält einen Titel, Content (die eigentliche Information), Keywords und Triggerwörter. Wenn ein Triggerwort vom Nutzer erwähnt wird, wird die entsprechende Note automatisch mit der Anfrage an die KI gesendet.



Mit diesem System werden nur Informationen an die KI gegeben, die wichtig für die Frage sind. Doch dieses Feature alleine ist problematisch, denn nicht immer sind die Triggerwörter in der Anfrage des Nutzers, sondern ergeben sich erst durch die Fragestellung. Wenn man die KI z.B. nach dem Erstellen einer E-Mail fragt, wäre es gut, wenn die KI den Namen des Nutzers kennen würde, doch in der Anfrage wurde nie das Triggerwort "Name" oder "Bennet" erwähnt. Dies lässt sich zum Glück leicht lösen, indem ich einen MCP-Server hinzugefügt habe. Über den MCP-Server kann das LLM folgende Aktionen ausführen:

- Suchen
- Erstellen
- Bearbeiten
- Lesen
- Löschen

Motivation und Fragestellung

Die Idee eines Künstlichen Assistenten existiert schon länger als die Technologie selbst. Doch “der” oder “die” perfekte Assistent*in existiert noch nicht. Ein großes Problem ist, wie oben schon angedeutet, das Token-Limit bei LLMs. Aufgrund dessen kann ein Sprachmodell nicht unendlich Informationen verarbeiten und sich damit auch nicht an alles erinnern.

GPT-5 (Open Als neuestes model) hat ein context window von [128.000 token](#). Ein Token sind ca. [4 Zeichen](#). Deutsche Wörter haben im Durchschnitt [6 Zeichen](#), dass heißt GPT-5 könnte ca. 85.000 Wörter verarbeiten, was ca. [5.000 Sätze](#) wären. (Diese Zahl ist sehr variierend basierend auf vielen Faktoren)

Meine Lösung ist ein Konzept, wie man dieses Problem löst und somit dem Traum eines digitalen allwissenden Assistenten einen Schritt näher kommt.

Grundlagen

Das Projekt wurde auf der [Gemini API](#) aufgebaut, ist theoretisch allerdings austauschbar mit jedem anderen Betreiber (ich arbeite gerade am Migrieren zu [Openrouter](#), um Zugang zu allen Modellen zu bekommen).

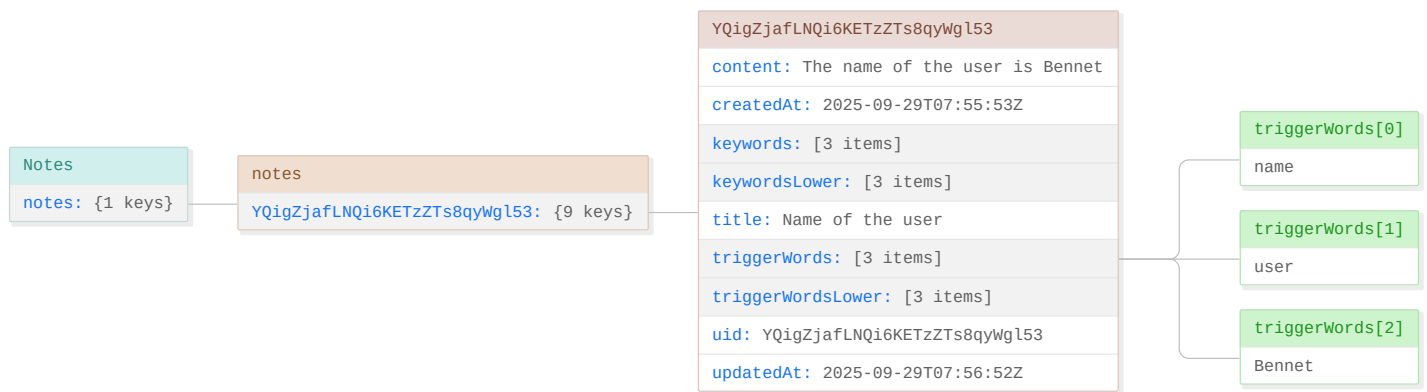
Warum Gemini? Weil es als eine der einzigen APIs [kostenlos](#) ist...

Als BaaS nutze ich [Firebase](#) und speichere dort alles bis auf Datei-Uploads, da dies ein kostenpflichtiges Feature ist.

Vorgehensweise

Ich startete mit dem Backend da es die Grundlage für dieses Projekt werden sollte. Dort baute ich einen simplen Prototyp der Auth-Routen, welche zu diesem Zeitpunkt nur E-Mail und Passwort unterstützten. Als dies stand, baute ich die Desktop app und verknüpfte sie mit dem Backend. Ich baute eine simple Chat-App nach dem Vorbild von t3.chat, ohne ein einziges Feature, welches dieses Projekt ausmacht. Erst als ich die Grundlagen hatte fing ich an das notes

system einzubauen. Hierzu erstellte ich in Firestore eine neue Column, in welcher die Informationen der Note sowie die UID des dazugehörigen Nutzers gespeichert sind.



Um die Effizienz zu steigern, habe ich zudem einen Index erstellt. Ein Index hilft dabei, schneller und effizienter Datenbankabfragen zu stellen.

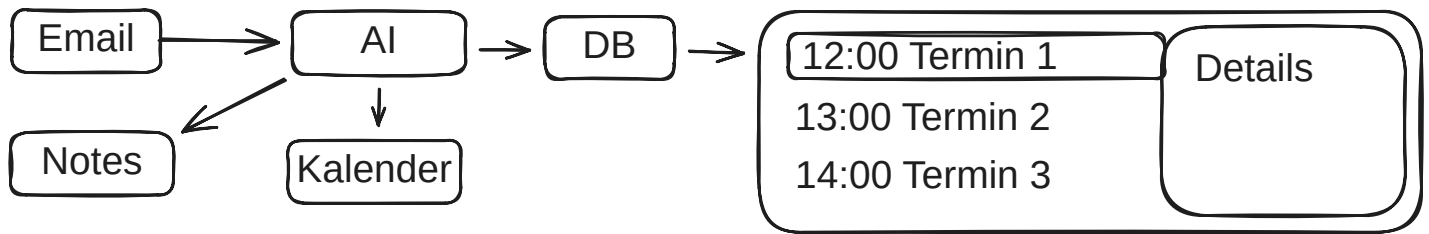
Dies war noch einfach doch damit die KI auch auf diese notes zugreifen kann musste ich einen MCP (Model Context Protocol) Server bauen. Dies war mit Abstand das Schwierigste an dem Projekt, da MCP noch relativ neu ist (und fast schon wieder abgestorben ist) gab es nur die offiziellen [Anthropic docs](#).

🌐 Notes MCP Server

Obwohl die KI bereits in der Lage ist, auf Notizen zuzugreifen und diese zu bearbeiten, existieren ähnliche Konzepte schon länger (es wäre nichts neues). Die grundlegende Idee besteht darin, der KI nur die wichtigsten Informationen bereitzustellen. Zu diesem Zweck entwickelte ich eine Logik, die ermöglicht, dass, wenn der Nutzer eine Nachricht an das Backend sendet, diese Nachricht an eine Funktion übergeben wird, welche nun überprüft, ob Triggerwörter aus den Notizen des Nutzers in der Nachricht vorkommen. Wird eine entsprechende Notiz gefunden, wird sie der KI als Kontext zur Verfügung gestellt.

Das Ganze klingt einfacher, als es tatsächlich ist. Ich benötigte mehrere Monate und etwa 34.826 Zeilen Code, um es zum Laufen zu bringen. Danach musste ich viele Feinjustierungen vornehmen und den Prompt optimieren. Damit war die Grundidee implementiert.

Doch dann verpasste ich einen relativ wichtigen Termin weil ich es mir nicht in meinen Kalender eingetragen habe. Ich habe bis heute keine gute software gefunden für Kalender und Email. Also habe ich mich entschieden es selber zu bauen. Die Grund Idee war all diese Komponenten Kalender, email und chat miteinander zu verbinden.



Ich habe also begonnen meinen Google Kalender einzubauen. Dies war relativ (trotzdem ca. 2.000 Zeilen Code) einfach doch der E-Mail-Part war deutlich schwieriger. Zum Glück kam zu diesem Zeitpunkt [glm 4.7](#) raus welches einen großen teil der gmail Integration übernommen hat. Doch wie wir alle wissen ist der Großteil des Vibecoden[*] Debuggen. Doch einfach nur die Emails anzeigen wie man es in gmail auch machen kann ist ja langweilig, der spannende Teil ist ja das AI review, doch dies hat gedauert. Von HTML parsing bis zu API Endpoints nichts schien zu funktionieren. Doch nach genügend Nächten endlosem Debuggen hatte ich es hingekriegt. Es gibt zwei Methoden seine E-Mail zu verbinden.

Option A) Gmail, wenn jemand einen Gmail Account mit Zen AI verbindet wird ein Webhook zwischen meinem Backend und Google Cloud erstellt. Wenn nun eine E-Mail an gmail gesendet wird sendet Google eine Nachricht an unser backend mit dem Inhalt.

Option B) SMTP und IMAP, um nicht nur leute [gefangen](#) im Google Ökosystem system zu supporten kann man bei fast (jeder normale dienst) jedem provider sich mit seinen IMAP (Empfangen von emails) und SMTP (Senden von emails) creds anmelden und so die AI features nutzen. Der Nachteil ist das es dieses Webhook system nicht gibt (es gibt es, **aber** es ist extrem kompliziert und nicht so weit verbreitet als das es sich lohnt) weshalb jede stunde das Backend manuell alle IMAP server die registriert sind nach neuen Nachrichten durchsucht.

Wenn nun also eine E-Mail im Backend ankommt, macht die KI [folgendes](#):

1. Zu allererst soll die KI die email auf einer Skala von 1 bis 10 nach der Wichtigkeit einordnen wobei 1 am wenigsten (z.B. ein Scam) wichtig ist und 10 eine Extrem wichtige Nachricht wie z.B. eine Deadline für ein Projekt ist.
2. Ein weiterer Vorteil von KI ist, dass es deutlich einfacher ist Emails zu Kategorisieren und genau dies tut sie in diesem Schritt. Aktuell sind es einfach nur statische presets doch das system ist eigentlich dazu gebaut, dass der Nutzer diese bearbeiten und neue erstellen kann.
3. Absender verifizieren. Ein großes Problem mit AI Agents ist das sie sehr einfach exploitet werden können (siehe [Googles Antigravity](#) welche .env

aufgrund von Prompt injection auf websites an dritte sendet) mit [prompt injection](#). Um dies vorzubeugen soll die KI verifizieren das es wirklich mail@Amazon.com und nicht mail@ammazon.com ist (funfact: die meisten großen firmen kaufen alle möglichen domains die so ähnlich wie ihre aussehen). Und wenn es nicht so aussieht wie ein scam versuch der adresse einen guten namen gibt z.B. Amazon.

4. Jetzt wird die email zusammengefasst in wenige kurze Sätze. Einfach, aber wohl das Wichtigste.
5. In Schritt fünf entscheidet die KI ob sie einen Notiz erstellen möchte.

Sie können sich den Prompt gerne im public repo ansehen unter:

[🌐 zen_ai_public/backend/zen_backend/email/analyzer.py at main · joan-code6...](#)

Nachdem die Backend-Funktionen erfolgreich implementiert worden waren, hatte ich mit der Integration in den neuen Client begonnen. Zuvor hatte ich ausschließlich digitale Clients für Web, Mobilgeräte und Desktop entwickelt. Das größte Problem bei diesen Kalender- und E-Mail-Lösungen war, dass sie alle digital waren. Man musste aktiv das Handy oder den Laptop öffnen und dem Drang widerstehen, eine App zu nutzen, die darauf ausgelegt ist, süchtig zu machen. Erst dann erhielt man die wichtigen Informationen.

Eine Lösung bestand darin, einfach Benachrichtigungen zu versenden, um die wichtigsten Informationen schnell zu übermitteln. Allerdings war mein Benachrichtigungszentrum immer überfüllt mit unwichtigen Mitteilungen und Werbung. Selbst wenn ich dort eine wichtige Information entdeckte, vergaß ich sie innerhalb weniger Stunden und schaffte es nicht, sie rechtzeitig zu notieren.

Die Antwort auf dieses Dilemma lag darin, den Nutzer außerhalb des digitalen Bereichs zu erreichen. Ich hatte ein E-Ink-Display mit einem ESP32-Entwicklerboard kombiniert und einer custom C++-Firmware sowie ein individuelles Interface und Backend-Routen entwickelt. Dadurch konnte man einfach über die Zen AI Mobile App eine Verbindung zu dem in der Nähe befindlichen Display herstellen und es ins lokale Netzwerk einbinden, um sie mit seinem Account zu verknüpfen.

Die Benutzeroberfläche (UI) wurde so konzipiert, dass Überschriften von Kalendereinträgen und E-Mails auch aus einer Entfernung von etwa drei Metern gut sichtbar sind. Die Entscheidung für das Display fiel mir relativ leicht. Ich wollte kein herkömmliches LCD oder OLED verwenden, da deren Licht oft als störend empfunden wird. E-Ink-Displays hingegen sind energieeffizient, da sie nur beim Aktualisieren Strom verbrauchen, und sie sind angenehm anzusehen.

Ergebnis

Zen AI hat die ursprünglichen Ziele nicht nur erreicht, sondern übertroffen. Nutzer können effizienter mit KIs kommunizieren, ohne ständig alles erklären zu müssen. Dank der Integration von Kalender und E-Mail ist es möglich, wichtige Informationen ohne Ablenkung zu erfassen und effizienter zu arbeiten. Zen AI merkt sich Informationen, sodass die KI besser versteht, worüber du sprichst.

Ein Erfolg von Zen AI liegt in der Effizienzsteigerung bei der Bereitstellung von Informationen. Der Einsatz von Triggerwörtern und Notizen stellt den Kontext für die KI präzise bereit, was die Relevanz der Informationen erhöht. Das reduziert den „Needle-in-a-Haystack“-Effekt und erleichtert die Interaktion mit der KI. Zudem ist Zen AI auf Dauer energieeffizienter, da die KI nicht ständig alles neu evaluieren muss, im Gegensatz zu traditionellen Systemen wie dem ChatGPT-Memory-System. Die Daten werden dabei sicher gespeichert.

Die Integration von E-Mails und Kalendern hat die Funktionalität und Benutzerfreundlichkeit verbessert. Nutzer können E-Mails effizient verwalten und wichtige Termine im Blick behalten, ohne von überflüssigen Informationen überwältigt zu werden. E-Mails werden kategorisiert, zusammengefasst und in ihrer Wichtigkeit bewertet, was die Priorisierung von Aufgaben vereinfacht.

Insgesamt stellt Zen AI einen bedeutenden Fortschritt in der Nutzung von KI im Alltag dar. Es verbessert die Effizienz der Informationsverarbeitung und bietet innovative Ansätze, um die Interaktion mit digitalen Assistenten zu erleichtern. Die Kombination aus technischer Innovation und praktischer Anwendbarkeit macht Zen AI zu einem wertvollen Werkzeug für alle, die ihre digitale Interaktion optimieren möchten.

Ausblick

Aktuell (Stand 14.01.26) gibt es noch Probleme bei der Verbindung von SMTP-Mailfächern. Ich plane, dies innerhalb der nächsten 2 bis 3 Wochen zu beheben.

Wie bereits erwähnt, arbeite ich an der Integration von OpenRouter, um den Nutzern die Möglichkeit zu bieten, nahezu alle KI-Modelle nutzen zu können.

Ein weiterer Punkt auf meiner To-Do-Liste ist die Neuentwicklung der Desktop-App, die derzeit in Flutter geschrieben ist, in React. Ich bin mir nicht sicher,

warum ich damals die Entscheidung für Flutter getroffen habe, da es im Vergleich zu React keinen Vorteil bietet.

Anmerkungen

Vibecoden

Nach der definition von Vibecoden reviewed man den generierten Code nicht. Ich tue dies. Ich verstehe jede Zeile in meiner Codebase und weiß warum sie dort ist. Ich erstelle Blueprints und entscheide welches Framework ich nutzte. Deshalb ist mein Code **nicht** gevibecodet, viele Menschen in diesem Bereich scheinen dies nicht zu verstehen.

“ Vibe coding describes a [chatbot](#)-based approach to creating [software](#) where the developer describes a project or task to a [large language model](#) (LLM), which generates [source code](#) based on the [prompt](#). **The developer does not review or edit the code**, but solely uses tools and execution results to evaluate it and asks the LLM for improvements. Unlike traditional AI-assisted coding or [pair programming](#), **the human developer avoids examination of the code, accepts AI-suggested completions without human review**, and focuses more on iterative experimentation than on code correctness or **structure**.

[Wikipedia](#)

”

KI Nutzung beim Schreiben diese Dokuments

Die Struktur, die Grundidee, die Inhalte und die Referenzen wurden manuell erstellt und lediglich durch KI optimiert, um beispielsweise Fehler zu korrigieren.