KI in der Konsole, App im Web

Live-Coding von der Idee bis zur Cloud

Von der Idee zur Online-Anwendung in 30 Minuten

Java Forum Stuttgart 2025

42

42

> *"Deep Thought berechnete 7,5 Millionen Jahre lang die Antwort auf die Frage nach dem Leben, dem Universum und dem ganzen Rest..."*

--

Das Problem: Schlechtes Prompting führt zu schlechten Antworten

Wie in der Softwareentwicklung: Falsche Anforderungen → Ungewolltes Ergebnis

Wer kennt "Per Anhalter durch die Galaxis"?

Present AI - Mitmachen!

📲 QR-Code scannen und mitmachen:

```
[QR-Code für https://github.com/DG1001/ConfChat]
```

- **Feedback während des Vortrags** geben
- **Fragen stellen**
- **Info-Seite wird generiert**
- **Am Ende schauen wir gemeinsam drauf**

Anschließend beantworte ich offene Fragen per Tool

Was wollen wir heute erreichen?

🎯 Ziel-Projekt: "SnapPic"

Eine **Progressive Web App** für Konferenz-Teilnehmer:

- **Upload:** Teilnehmer können Bilder + Kommentar senden
- **Backend:** Python + Flask empfängt Requests
- **Gallery:** Zeigt die 10 neuesten Bilder
- **Timing:** Jedes Bild ist 5 Sek sichtbar → 15 Sek Fade-out → Löschung
- **Flow:** Neues Bild → Nächster freier Slot

Live-Coding in ~30 Minuten!

Agenda (40 min + 5 min Q&A)

1. **LLM-Landschaft** - Hosted vs. Local (5 min)
2. **Tool-Showcase** - Aider, Claude Code, Gemini CLI (10 min)
3. **Live-Coding Demo** - SnapPic App (20 min)
4. **Deployment-Optionen** - Von lokal bis Cloud (5 min)
5. **Q&A + Diskussion** (5 min)

LLM-Landschaft: Hosted vs. Local

🌐 Hosted LLMs

- **Claude** (Anthropic)
- **GPT-4** (OpenAI)
- **Gemini** (Google)
- **Grok** (xAI)
- **DeepSeek** (DeepSeek AI)

💻 Local/Self-Hosted

- **DeepSeek-Coder**
- **Qwen2.5-Coder**
- **Gemma**
- **QwQ** (Alibaba)
- **Mistral-Dev**

Hosting-Optionen für LLMs

🔗 API-Services

- **OpenRouter** / **Together.ai**
- **Vast.ai** - GPU-Miete ab 0,10€/Stunde
- **Lokale Inferenz** - M3/M4 Mac, RTX 3090/4090
- *Mindestens 16GB GPU-RAM, besser mehr*
- *Kosten: ~5.000€+*

💡 Empfehlung

Hybrid-Ansatz: Lokale LLMs für schnelle Queries, Hosted für komplexe Refactoring

Tool Deep-Dive

🛠️ Unsere Werkzeuge heute:

1. **Aider** - Git-aware Pair Programmer
2. **Claude Code** - Aktuell bestes AI-Coding-Tool
3. **Gemini CLI** - Open Source, kostenloses Kontingent
4. **OpenCode** - Vielversprechende Alternative

📋 Backup-Tools:

- **Manus.im** - Vollständige Coding-Agents
- **AI Builder** (Google)

Aider - Git-Aware Pair Programming

✨ Features

- **Git-Integration:** Jede Änderung = Commit
- **Multi-LLM Support:** OpenAI, Anthropic, OpenRouter
- **Agile Workflows:** Iterative Entwicklung

🚀 Installation & Usage

```
pip install aider-chat
aider --openai-api-key YOUR_KEY
```

💪 Stärken

- Hervorragend für **Refactoring**
- **Multi-File Edits**
- **Commit-History** bleibt sauber

Claude Code - Das Profi-Tool

🎯 Warum Claude Code?

- **Aktuell bestes AI-Coding-Tool**
- **Nur Claude-Modelle** (API-Key oder Abo)
- **20€/Monat Abo** für intensive Nutzung
- **Profi-Einsatz:** 100-200€/Monat

🔧 Features

- **MCP-Integration** - Unbegrenzte Automatisierung
- **Sehr gute Code-Qualität**
- **Große Kontextfenster**

💡 Praxis-Tipp

Kombination mit Gemini CLI für große Projekte

Gemini CLI - Open Source Alternative

🆓 Vorteile

- **Open Source** von Google
- **Kostenloses Kontingent:** 1000 Calls/Tag
- **Riesiger Kontext:** Bis zu 1M Tokens
- **Multimodal:** Text, Code, Bilder

⚠️ Einschränkungen

- **Nur Gemini-Models**
- **Quality-Drops:** Pro → Flash (Syntax-Fehler)
- **Kostenpflichtig** für intensive Nutzung

🛠️ Installation

```
curl -fsSL https://gemini.google.com/cli/install.sh | sh
```

OpenCode - Der Newcomer

🌟 Features

- **75+ LLM-Provider** out-of-the-box
- **LSP-Integration** - Versteht Projektkontext
- **Agent-Verhalten** - Multi-File Operations
- **Terminal UI** - Professionelle Oberfläche

🔄 Status

- **Aktive Entwicklung** (2 Updates während Vorbereitung!)
- **OpenRouter-Integration** teilweise buggy
- **Große Auswahl** an Modellen

Manus.im - Der Coding-Agent

🤖 Wasserfallmodell für KI

- **Ausführlicher Prompt** = Requirements
- **30+ Minuten** autonome Entwicklung
- **Vollständige Projekte** möglich

📋 Erfolgreiche Projekte

- **JFS Talk-Auswahlprogramm** initial erstellt
- **Dokumentation & Bilder** als Input
- **Architektur-Vorgaben** wichtig

💡 Tipp

Prompt von anderer KI optimieren lassen

Live-Coding: SnapPic App

🎯 Technologie-Stack

- **Backend:** Python + Flask
- **Frontend:** HTML/CSS/JavaScript (PWA)
- **Deployment:** Git → Server
- **Domains:** Vorkonfiguriert auf sensem.de

🔧 Parallel-Entwicklung

- **aider.sensem.de** - Aider + DeepSeek
- **claude.sensem.de** - Claude Code
- **gemini.sensem.de** - Gemini CLI
- **opencode.sensem.de** - OpenCode

Live-Demo: Prompt Engineering

📝 Projekt-Prompt Template

```
Rolle: Senior Python-Entwickler
Aufgabe: Erstelle eine Flask-App namens "SnapPic"

Funktionalität:
- PWA für Bild-Upload mit Kommentar
- Gallery zeigt 10 neueste Bilder
- Timing: 5s sichtbar → 15s fade → delete
- Backend: Python/Flask, Frontend: HTML/CSS/JS

Constraints:
- Keine zusätzlichen Dependencies
- Deployment-ready
- Responsive Design
```

Live-Coding - Teil 1: Setup

🚀 Projekt-Initialisierung

```
# Aider Workflow
cd /tmp/snappic-aider
git init
aider --openai-api-base https://openrouter.ai/api/v1
```

📋 Erster Prompt

"Erstelle eine Flask-App namens SnapPic..."

✅ Erwartetes Ergebnis

- app.py mit Flask-Setup
- templates/ Ordner
- static/ für CSS/JS
- uploads/ für Bilder

Live-Coding - Teil 2: Core Features

🖼️ Upload-Funktionalität

```
@app.route('/upload', methods=['POST'])
def upload_image():
    # File handling
    # Timestamp naming
    # Gallery update
```

📱 PWA-Features

- Manifest.json
- Service Worker
- Mobile-optimiert

🎨 Gallery mit Timing

- JavaScript Timer
- Fade-Effekte
- Auto-Refresh

Live-Coding - Teil 3: Parallel-Vergleich

🔄 Gleichzeitige Entwicklung

1. **Aider** mit DeepSeek-Coder
2. **Claude Code** mit Claude-3.5
3. **Gemini CLI** mit Gemini-Pro
4. **OpenCode** (falls Zeit)

📊 Live-Bewertung

- ▸ **Code-Qualität**
- ▸ **Geschwindigkeit**
- ▸ **Fehlerrate**
- ▸ **Benutzerfreundlichkeit**

Deployment-Optionen

🌐 Frontend-Hosting

- **GitHub Pages** - Kostenlos, HTTPS
- **Netlify/Vercel** - Serverless Functions
- **Cloudflare Pages** - Global CDN

🖥️ Backend-Hosting

- **Google Cloud VM** - Ab 5€/Monat
- **Hetzner Cloud** - Deutsche Server
- **Railway/Fly.io** - Auto-Deployment

🔧 Unsere Demo-Infrastruktur

- **Contabo Server** mit vorkonfiguriertem Nginx
- **Subdomains** bereits geroutet
- **SSL-Zertifikate** von Let's Encrypt

Domain & SSL Setup

🌎 Kostenlose Domains

- **DuckDNS** - Kostenlose Subdomains
- **Freenom** - Kostenlose TLDs
- **GitHub Pages** - .github.io

🔒 SSL-Zertifikate

- **Let's Encrypt** - Kostenlos, automatisch
- **Cloudflare** - Proxy + SSL
- **Certbot** - Automatische Erneuerung

⚙️ Nginx-Konfiguration

```
server {
    listen 443 ssl;
    server_name aider.sensem.de;
    location / {
        proxy_pass http://localhost:5000;
    }
}
```

Kosten-Übersicht

💰 LLM-Kosten (pro Projekt)

- **Local Models:** 0€ (nach Initial-Setup)
- **OpenRouter:** 0,10€ - 0,50€
- **Claude API:** 0,50€ - 2,00€
- **GPT-4:** 1,00€ - 3,00€

🖥️ Hosting-Kosten (monatlich)

- **Static Hosting:** 0€
- **Shared Hosting:** 2-5€
- **VPS:** 5-10€
- **Cloud VM:** 10-20€

📊 Gesamt-TCO

Typischer MVP: 1-5€/Monat (statt 500-2000€ traditionell)

Pitfalls & Best Practices

⚠️ Häufige Fehler

- **Unspezifische Prompts** → Frankenstein-Code
- **Fehlende Tests** → Instabile Anwendungen
- **Keine Versionierung** → Chaos bei Fehlern
- **Blindes Vertrauen** → Sicherheitslücken

✅ Best Practices

- **Klare Prompts:** Rolle → Aufgabe → Constraints
- **Iterative Entwicklung:** Kleine Schritte
- **Code-Review:** AI ist Junior-Developer
- **Security-Scans:** Snyk, Trivy integration

Fallback-Strategien

🎬 Video-Demo

Falls Live-Coding scheitert

🔄 Vorbereitung

- **Screen-Recording** aller Tools
- **Fertige Projekte** als Backup
- **Lokale Entwicklung** als Plan B

💪 Robustheit

- **Offline-Präsentation** möglich
- **Multiple Tool-Optionen**
- **Vorkonfigurierte Server**

Ausblick: Zukunft des AI-Coding

🤖 Trends 2025

- **Agentic Workflows** - Multi-Step Automation
- **Größere Kontextfenster** - Whole-Repo Understanding
- **On-Device Models** - Offline-Coding
- **Better Integration** - IDE-Native AI

🔮 Prediction

"Heutige 30-Minuten-MVP wird zu 10-Minuten-MVP"

📈 Skill-Entwicklung

- **Prompt Engineering** wird Kernkompetenz
- **AI-Human Collaboration** patterns
- **Quality Assurance** wichtiger denn je

Zusammenfassung

🎯 Was wir gelernt haben

1. **Tool-Landschaft** - Richtige Auswahl für Use-Case
2. **Prompt Engineering** - Basis für gute Ergebnisse
3. **Deployment-Pipeline** - Von Code zu Production
4. **Kostenoptimierung** - Günstige Lösungen für MVPs

💡 Key Takeaways

- **AI beschleunigt** Entwicklung dramatisch
- **Hybrid-Ansatz** ist optimal
- **Prompt-Qualität** entscheidet über Erfolg
- **Continuous Learning** ist essentiell

Call to Action

🚀 Deine nächsten Schritte

1. **Installiere** ein Tool (Aider, Claude Code, Gemini CLI)
2. **Erstelle** dein erstes AI-Projekt
3. **Experimentiere** mit lokalen LLMs
4. **Teile** deine Erfahrungen

🤝 Community

- **GitHub:** Stars & Contributions
- **Discord/Slack:** Erfahrungsaustausch
- **Meetups:** Lokale AI-Coding Groups

Kontakt

Frederik Wystup - frederik.wystup@meiluft.de - https://github.com/dg1001

Q&A + Diskussion

❓ Fragen willkommen!

- Tool-Empfehlungen
- Spezifische Use-Cases
- Deployment-Strategien
- Kosten-Optimierung

🔗 Ressourcen

- GitHub: Slides & Code
- Present AI: Zusammenfassung
- Weiterführende Links

📞 Kontakt

Frederik Wystup - frederik.wystup@meiluft.de

Danke!

🙏 Java Forum Stuttgart 2025

Von der Idee zur App in 30 Minuten *Mit KI-Tools ist es möglich!*

👨‍💻 Frederik Wystup

- **E-Mail:** frederik.wystup@meiluft.de
- **GitHub:** https://github.com/dg1001

- *Fast durchgängig KI-Coding-Projekte*

🚀 Viel Erfolg bei euren Projekten!

Fragen? → Present AI oder direkter Kontakt

KI in der Konsole, App im Web – Live-Coding von der Idee bis zur Cloud

Von der Idee zur Online-Anwendung in 30 Minuten: Günstige Wege mit modernen KI-Tools

Live-Informationen

KI in der Konsole, App im Web – Live-Coding von der Idee bis zur Cloud

Beschreibung

Von der Idee zur Online-Anwendung in 30 Minuten: Günstige Wege mit modernen KI-Tools

Dieser Vortrag zeigt, wie mit modernen KI-Tools aus einer Idee in nur 30 Minuten eine lauffähige Web-App entsteht – und das ohne nennenswertes Budget. Ziel ist es, zu inspirieren und zu demonstrieren, wie der Weg von einer vagen Idee bis zur fertigen Cloud-Anwendung schnell, unkompliziert und nahezu kostenfrei machbar ist.

Kontext/Hintergrund

Im Live-Coding-Format demonstriert der Referent, wie mithilfe aktueller KI-gestützter Werkzeuge eine Anwendung in kurzer Zeit entwickelt und online bereitgestellt werden kann. Die Präsentation richtet sich an technikaffine Teilnehmende mit IT-Grundkenntnissen und bietet praktische, praxiserprobte Einblicke.

Hauptinhalt

Von der Idee zur Cloud-Anwendung: Der Ablauf

1. **Ideenfindung und Zieldefinition**
2. Konkrete Beispiele: Bilder-Tauschplattform, Bewertungs-App für Konferenz-Talks
3. **KI-gestützte Code-Generierung**
4. Einsatz moderner KI-Unterstützung (z. B. Aider)
5. Nutzung günstiger/free KI-Modelle über Plattformen wie OpenRouter oder Together
6. **Schnelle Entwicklung direkt im Live-Coding**
7. In Echtzeit Erstellung einer funktionsfähigen App
8. **Deployment via Cloud-Plattformen**
9. Veröffentlichung der Anwendung über kostenfreie Dienste:
    - Vercel
    - Hugging Face
    - Google Cloud VM
    - Replit
10. Mit kostenfreien Domains und HTTPS-Zertifikaten

Anforderungen & Zielgruppe

- Technikaffine Teilnehmende mit IT-Basiswissen
- Praktisches Live-Coding statt theoretischer Betrachtungen
- Inspiration durch schnelles, unkompliziertes Realisieren eigener Ideen

---

Hilfreiche Links & Tools

Benchmarks & Vergleiche von LLMs

- Aider Leaderboards
- Dubesor Benchtable

- OpenRouter AI Programming Rankings (Week)

Zugriff auf LLM-Modelle (API Keys)

- OpenRouter
- Groq Console
- Together AI

GPU-Server (zum Mieten)

- Vast.AI

Eingesetzte Tools

- opencode (sst)
- Claude Code
- Codex (OpenAI)
- Gemini CLI
- Aider

Weitere empfehlenswerte, eventuell kostenpflichtige Tools

- Manus(im) – mit kostenloser Probephase

Repositories und Beispiele

- Vortragender auf GitHub, umfangreich mit KI entwickelt
- JFS Talk Bewertungs-App via DeepWiki

Nützliche Infrastruktur-Services

- DuckDNS – kostenlose Subdomains für Tests

Weitere interessante Links

- Google AI Studio
- Hugging Face – Zugriff auf tausende LLMs
- Awesome MCP Servers
- KI-Chat-Dienste:
  - DeepSeek Chat
  - ChatGPT
  - Mistral Chat
  - Qwen Chat
  - Claude AI
  - Perplexity AI
  - Grok
  - Duck AI

Zusätzliche Informationen vom Präsentierenden

Hinzugefügt am 10.07.2025, 20:14

Antworten auf häufig gestellte Fragen:

- **Bist du eine KI?**
  Nein, aber in manchen Momenten kommt es einem fast so vor …
- **Werden die Folien bereitgestellt?**
  Ja, werden demnächst in der JFS-App verfügbar sein.
- **Bestes KI-Modell aktuell (subjektiv)?**
  Für Coding: Claude (vor allem Claude Code) steht derzeit sehr hoch im Kurs, gefolgt von Google Gemini Pro 2.5. OpenAI-Modelle sind ebenfalls stark – bei Preis/Leistung dominiert aktuell Deepseek. Benchmarks (siehe oben) können bei der Auswahl helfen; die Entwicklung ist hier sehr dynamisch.
- **Wie lange dauert es, ein gutes Prompt zu erstellen?**
  Das ist schwer pauschal zu beantworten. Ausprobieren lohnt sich: Das Prompt so wie für ein neues Teammitglied formulieren, explizite Nachfragen einbauen ("Frage mich, wenn keine 95% Sicherheit!"). Prompt Engineering lässt sich zum Teil auch durch KI selbst unterstützen.

- **Unit-Test-Tracking per Prompt?**
  Ja, Anforderungen können im Prompt nummeriert und den Unit Tests explizit zugeordnet werden. Beispiele im Prompt helfen einem LLM zusätzlich.

- **Security der KI-generierten Sourcecodes?**
  Sicherheit hängt primär von guter Promptgestaltung und Code-Review ab, nicht vom Modellpreis. Auch günstige Modelle liefern sicheren Code, wenn explizit nach Security Practices gefragt wird. Mehrfache Review-Iterationen können jedoch zu leicht erhöhten Kosten führen.

- **Empfohlenes Setup für eigene Entwicklung (offline/Private Cloud)?**
  Empfehlung: Ollama (für LLM-Inferenz) + Open WebUI (als Oberfläche). Dieses Setup ist schnell startklar und wird unter anderem von vast.ai angeboten.

- **Lokale Modelle mit 16 GB GPU: Wie schlagen sich diese?**
  16 GB ist für viele Top-Modelle knapp. Modelle wie Qwen2.5-Coder 14B oder Deepseek (chat, r1) laufen gut und sind stark, aber bei komplexen Aufgaben bleibt Claude Code im Vorteil – der Abstand schrumpft jedoch langsam.

- **Funktioniert das nur für kleine Apps? Und bei komplexen Anwendungen?**
  Auch komplexere Anwendungen sind möglich, wenn modular entwickelt und progressiv mit guten Architektur-Prompts gearbeitet wird. Größere Projekte erfordern bessere Planung und Promptgestaltung.

- **Wie wird sich die Nutzung von KI weiterentwickeln?**
  Schätzung: KI wird Standard wie IDEs heute. In naher Zukunft werden (fast) alle Entwickler KI-gestützt arbeiten; die Grenzen zwischen klassischem "Coden" und "Prompten" werden zunehmend verschwimmen.

- **Link zu Beispiel-Repository?**
  Siehe Abschnitt "Repositories und Beispiele" (s.o.).

Ende der Informationsseite.

Feedback

Fragen

- Bist du eine KI?
- Werden die Folien zur Verfügung gestellt?
- Was ist momentan Deiner Meinung nach das Beste Modell?
- Wie lange brauche ich um ein gutes prompt zu erschaffen?
- Kann ich requirements (Nummern) vergeben die per unit tests getrackt werden können?
- Gibt es Unterschiede bei der Security des generierten sources Codes? Muss ich für guten (Secure) Code mehr Geld zahlen?
- Welches Setup würdest du für die Entwicklung auf dem eigenen Rechner bzw. in der Private Cloud empfehlen?
- Können lokale Modelle mit einer GPU mit 16 GB VRAM an Claude Code ansatzweise heranreichen?
- Das geht aber nur für kleine Apps oder? Wie sieht es aus bei komplexen Enterprise applications?
- Wie sehen Sie die Zukunft davon aus?
- Kannst du den Link zu dem Repository mit den Beispiel zu Verfügung stellen?

Positive Kommentare und Meinungen

- Sehr spannend
- Vielen Dank!
- Geiler Scheiss!!!
- Super Vortrag 👍

⚠️ Ungeprüfte Links

- PEP Digital - Webpräsenz mit unverifiziertem Inhalt aus neuem Feedback

Haben Sie Fragen oder möchten weitere Informationen?

Ihr Name:

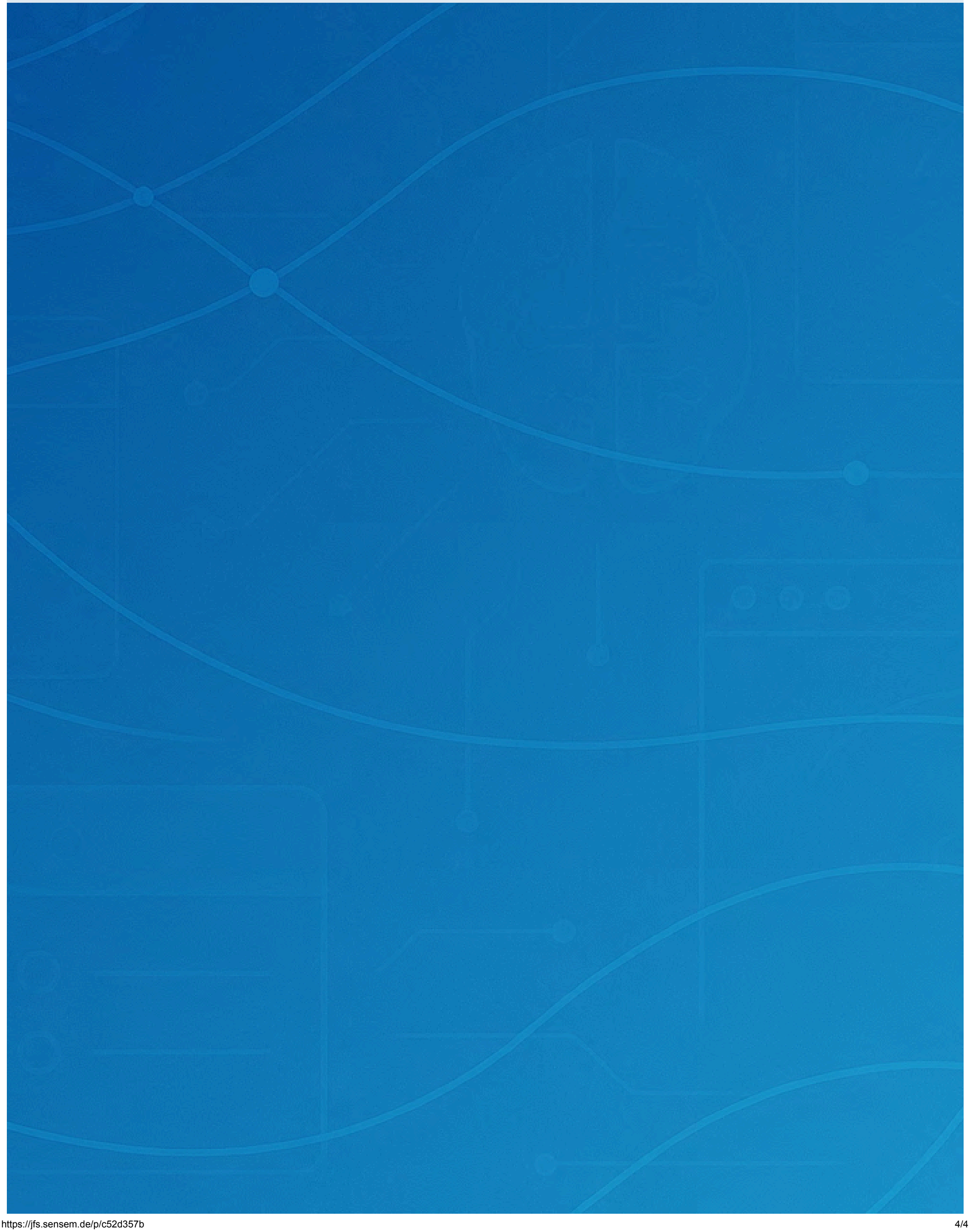
Geben Sie hier Ihren Namen ein...

Ihre Nachricht:

Geben Sie hier Ihre Frage oder zusätzliche Informationen ein...

Maximum 500 Zeichen 0/500

Absenden



Überarbeiteter Prompt:

SnapPic - Live-Coding Prompt

Basis-Prompt für alle Tools

Rolle

Du bist ein erfahrener Python-Entwickler mit Expertise in Flask, JavaScript und Progressive Web Apps.

Aufgabe

Erstelle eine Flask-Web-App namens "SnapPic" mit folgender Funktionalität:

Frontend (PWA)

- **Upload-Interface:** Benutzer können Bilder mit der Kamera des Smartphones aufnehmen oder aus der Galerie wählen und hochladen
- **Kommentar-Feld:** Kurzer Text zum Bild (max. 100 Zeichen)
- **Responsive Design:** Mobile-first, funktioniert optimal auf Smartphones
- **PWA-Features:** Manifest, Service Worker für Offline-Funktionalität, App-Icon
- **Design:** Modernes, professionelles Design mit Material Design Prinzipien

Backend (Flask)

- **Upload-Endpoint:** `/upload` (POST) für Bild + Kommentar
- **Gallery-Endpoint:** `/gallery` (GET) für Gallery-Ansicht
- **API-Endpoint:** `/api/images` (GET) für JSON-Daten der aktuellen Bilder
- **Static-Serving:** Bilder unter `/uploads/` verfügbar
- **Home-Route:** `/` für Upload-Interface

Kern-Logik

- **Timing:** Jedes Bild ist 5 Sekunden sichtbar, dann 10 Sekunden Fade-out-Animation, dann automatische Löschung
- **Kapazität:** Maximal 10 Bilder gleichzeitig in der Gallery
- **FIFO-Prinzip:** Neue Bilder verdrängen die ältesten Bilder
- **Auto-Refresh:** Gallery aktualisiert sich alle 2 Sekunden automatisch
- **Datei-Cleanup:** Automatische Löschung der Bilddateien nach Ablauf

Technische Anforderungen

- **Dependencies:** Nur Flask (keine zusätzlichen Python-Pakete)
- **Datenspeicherung:** Lokale JSON-Datei für Metadaten, Bilder im Dateisystem
- **Deployment:** Kann direkt mit `python app.py` gestartet werden
- **Port:** Flask-App läuft auf Port 5000
- **Dateiformate:** Unterstützt JPG, PNG, WEBP

- **Dateigröße:** Maximal 5MB pro Bild

Constraints

- Verwende nur Standard-Python-Bibliotheken + Flask
- Keine externen CDNs, APIs oder Frameworks
- Vanilla HTML/CSS/JavaScript (kein Frontend-Framework)
- Kompatibel mit Python 3.8+
- Bilder werden im `uploads/` Ordner gespeichert
- Metadaten in `data.json` im Hauptverzeichnis

Erwartete Dateistruktur

```
snappic/
├── app.py                 # Flask-App mit allen Routen
├── data.json             # Metadaten der Bilder
├── templates/
│   ├── index.html        # Upload-Interface
│   └── gallery.html      # Gallery-Ansicht
├── static/
│   ├── style.css         # Styling für alle Seiten
│   ├── script.js         # JavaScript-Funktionalität
│   ├── manifest.json     # PWA-Manifest
│   └── icon-192.png      # PWA-Icon
├── uploads/              # Bild-Uploads (wird automatisch erstellt)
└── requirements.txt      # Dependencies (nur Flask)
```

Implementierungsdetails

- **Timestamps:** Verwende `datetime.now().isoformat()` für eindeutige Dateinamen
- **Validierung:** Dateityp, Dateigröße und Kommentarlänge prüfen
- **Responsive CSS:** CSS Grid/Flexbox für alle Bildschirmgrößen
- **JavaScript:** Timer-Management und Auto-Refresh-Logik
- **Fehlerbehandlung:** Aussagekräftige Fehlermeldungen für Upload-Probleme
- **Threading:** Background-Timer für automatische Bild-Löschung

Zusätzliche Anforderungen

- Implementiere grundlegende Sicherheitsmaßnahmen gegen schädliche Uploads
- Erstelle einen funktionsfähigen Service Worker für PWA-Features
- Stelle sicher, dass die App auch bei langsameren Verbindungen funktioniert
- Füge Loading-Indikatoren für Upload-Vorgänge hinzu

Wichtige Änderungen:

1. **Klarere Spezifikationen:** Dateiformate, Dateigröße und technische Details präzisiert
2. **Konsistenz:** "Apple Material Design" → "Material Design Prinzipien"
3. **Datenspeicherung:** Explizite Erwähnung der JSON-Datei für Metadaten

4. **Sicherheit:** Validierung und Sicherheitsmaßnahmen ergänzt
5. **Threading:** Explizite Erwähnung für Background-Timer
6. **PWA-Details:** Service Worker und Icon-Anforderungen konkretisiert
7. **Struktur:** Klarere Trennung zwischen Frontend und Backend-Anforderungen