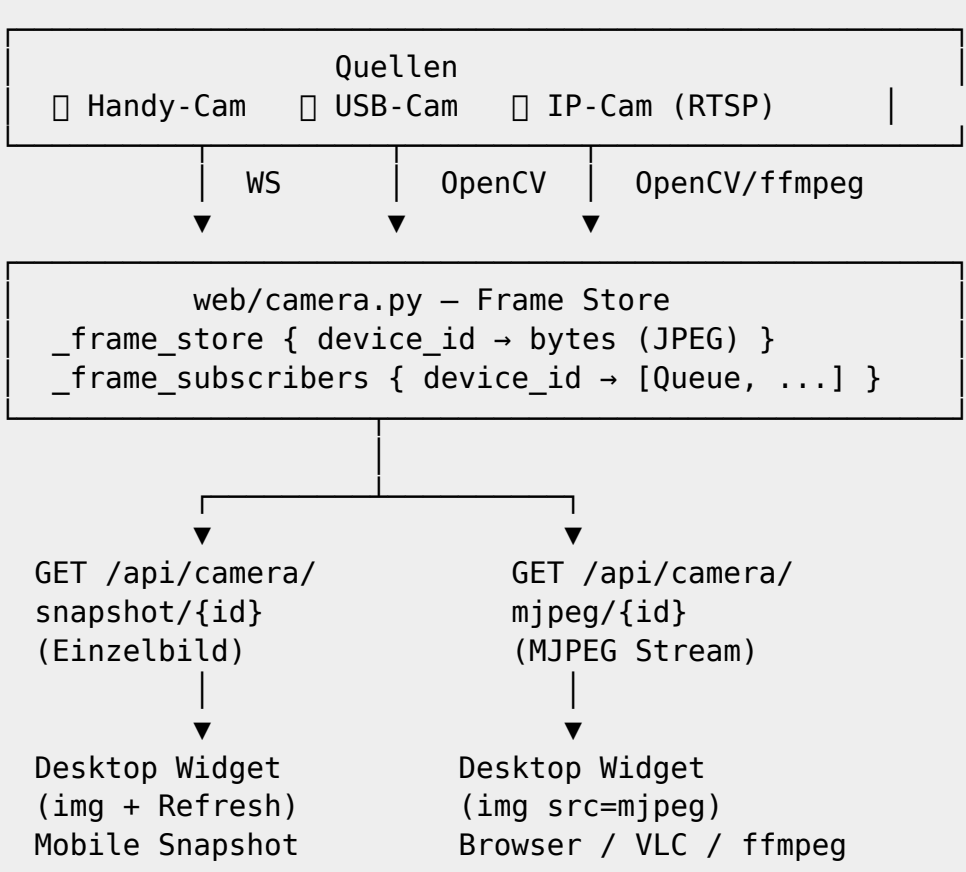
ionpy – Kamera & Video Stream Roadmap

USB-Cams, Netz-Cams, Mobile Cam & Handy-Sensoren

Erstellt: 2026-02-27
Status: In Planung
Abhängigkeiten: Keine – unabhängig von Wizard und PWA MVP

Architektur-Übersicht



Design-Prinzip: Alle Quellen landen im selben `_frame_store`. Alle Consumers lesen aus demselben MJPEG-Endpoint. Quellen und Consumers kennen sich nicht.

Neue Abhängigkeiten

```
# requirements.txt Ergänzungen:
opencv-python-headless    # USB + RTSP Cams (headless = kein GUI)
```

```
                                # Oder: opencv-python wenn GUI gewünscht
```

Hinweis: Kein `ffmpeg` Binary nötig – OpenCV bringt eigene Codec-Unterstützung mit. MJPEG-Streaming ist pure Python.

PHASE 0: Backend Fundament

(~1.5 Tage)

0.1 Frame Store & MJPEG Router

Datei: `web/camera.py` (NEU)

Hinweis für KI: Der `_frame_store` und `_frame_subscribers` sind globale Dicts – das ist bewusst simpel gehalten. Bei mehreren Worker-Prozessen (z.B. Gunicorn) müsste man Redis verwenden, aber für Single-Process FastAPI/Uvicorn ist das absolut ausreichend.

```python
"""
web/camera.py
Zentrales Kamera-Backend für ionpy.
Empfängt Frames von beliebigen Quellen und stellt sie
als MJPEG-Stream und Snapshot bereit.
"""
import asyncio
import time
import logging
from pathlib import Path
from typing import Optional

from fastapi import APIRouter, WebSocket, WebSocketDisconnect, UploadFile,
Form
from fastapi.responses import StreamingResponse, JSONResponse

logger = logging.getLogger("Web.Camera")
router = APIRouter(prefix="/api/camera", tags=["Camera"])

# ═══════════════════════════════════════════════════════════════════════
# GLOBALER FRAME STORE
# ═══════════════════════════════════════════════════════════════════════
# Letzter JPEG-Frame pro device_id
_frame_store: dict[str, bytes] = {}

# Unix-Timestamp des letzten Frames pro device_id
_frame_timestamps: dict[str, float] = {}
```

```python
# Aktive MJPEG-Consumer pro device_id (Queue bekommt neue Frames)
_frame_subscribers: dict[str, list[asyncio.Queue]] = {}

# Kamera-Metadaten (Name, Beschreibung, etc.)
_camera_meta: dict[str, dict] = {}


def _notify_subscribers(device_id: str, frame: bytes):
    """Neuen Frame an alle wartenden MJPEG-Consumer senden."""
    for q in _frame_subscribers.get(device_id, []):
        try:
            q.put_nowait(frame)
        except asyncio.QueueFull:
            pass  # Langsamer Consumer – Frame überspringen


def register_camera(device_id: str, name: str = None,
                    source_type: str = "unknown"):
    """Kamera im System anmelden (optional, für /list Endpoint)."""
    _camera_meta[device_id] = {
        "device_id":   device_id,
        "name":        name or device_id,
        "source_type": source_type,  # "mobile", "usb", "rtsp", "http"
        "registered":  time.time()
    }


# ============================================================================
# FRAME EMPFANG – Handy via POST
# ============================================================================
@router.post("/frame")
async def receive_frame(
    device_id: str = Form(...),
    image:     UploadFile = None
):
    """
    Empfängt einen JPEG-Frame von der Mobile PWA.
    Die PWA sendet alle N Sekunden ein Standbild.
    """
    if not image:
        return JSONResponse(status_code=400,
                            content={"error": "Kein Bild erhalten"})

    data = await image.read()

    # Größe sanity check (max 5 MB)
    if len(data) > 5 * 1024 * 1024:
        return JSONResponse(status_code=413,
                            content={"error": "Frame zu groß (max 5MB)"})

    _frame_store[device_id]      = data
```

```
    _frame_timestamps[device_id] = time.time()
    _notify_subscribers(device_id, data)

    if device_id not in _camera_meta:
        register_camera(device_id, source_type="mobile")

    return {"status": "ok", "size_bytes": len(data)}


# ============================================================================
# FRAME EMPFANG — Handy via WebSocket (Binary Stream)
# ============================================================================
@router.websocket("/stream/{device_id}")
async def camera_websocket_stream(
    websocket: WebSocket,
    device_id: str
):
    """
    Empfängt kontinuierliche JPEG-Frames via WebSocket.
    Für Mobile-Cam Echtzeit-Streaming (10 FPS).
    Binary-Protokoll: Jede Nachricht = ein vollständiger JPEG-Frame.
    """
    await websocket.accept()
    logger.info(f"Kamera-Stream verbunden: {device_id}")

    if device_id not in _camera_meta:
        register_camera(device_id, source_type="mobile_ws")

    try:
        while True:
            data = await websocket.receive_bytes()

            # Größen-Check
            if len(data) > 5 * 1024 * 1024:
                logger.warning(f"Frame zu groß von {device_id}: {len(data)}
bytes")
                continue

            _frame_store[device_id]      = data
            _frame_timestamps[device_id] = time.time()
            _notify_subscribers(device_id, data)

    except WebSocketDisconnect:
        logger.info(f"Kamera-Stream getrennt: {device_id}")
        # Letzten Frame für ~30s behalten, dann als inaktiv markieren
    except Exception as e:
        logger.error(f"Kamera WS Fehler ({device_id}): {e}")


# ============================================================================
# CONSUMER — Einzelbild
```

```python
# ============================================================================
@router.get("/snapshot/{device_id}")
async def get_snapshot(device_id: str):
    """
    Gibt den aktuell gespeicherten Frame als JPEG zurück.
    Ideal für: gelegentliche Aktualisierung, Thumbnails.
    """
    frame = _frame_store.get(device_id)
    if not frame:
        return JSONResponse(status_code=404,
                            content={"error": "Kein Frame verfügbar",
                                     "device_id": device_id})

    ts = _frame_timestamps.get(device_id, 0)
    return StreamingResponse(
        iter([frame]),
        media_type="image/jpeg",
        headers={
            "Cache-Control":    "no-cache, no-store",
            "X-Frame-Age-Ms":   str(int((time.time() - ts) * 1000)),
            "X-Frame-Timestamp": str(ts)
        }
    )


# ============================================================================
# CONSUMER — MJPEG Live Stream
# ============================================================================
@router.get("/mjpeg/{device_id}")
async def mjpeg_stream(device_id: str):
    """
    MJPEG-Stream für Browser, VLC, ffmpeg, OpenCV.
    Einfach als <img src="/api/camera/mjpeg/device_id"> verwenden —
    kein JavaScript nötig!

    Kompatibel mit:
      - Browser: <img>, <video> (limitiert)
      - Python: cv2.VideoCapture("http://host/api/camera/mjpeg/id")
      - ffmpeg: ffmpeg -i http://host/api/camera/mjpeg/id ...
      - VLC: Netzwerk-Stream öffnen
    """
    # Queue für diesen Consumer registrieren
    q: asyncio.Queue = asyncio.Queue(maxsize=10)

    if device_id not in _frame_subscribers:
        _frame_subscribers[device_id] = []
    _frame_subscribers[device_id].append(q)

    # Sofort den letzten bekannten Frame senden (kein Warten auf nächsten)
    if device_id in _frame_store:
        await q.put(_frame_store[device_id])
```

```
    async def generate():
        try:
            while True:
                try:
                    # Auf nächsten Frame warten (Timeout = Keepalive)
                    frame = await asyncio.wait_for(q.get(), timeout=15.0)
                    yield (
                        b'--ionpy_frame\r\n'
                        b'Content-Type: image/jpeg\r\n'
                        b'Content-Length: ' + str(len(frame)).encode() +
b'\r\n'
                        b'\r\n'
                        + frame +
                        b'\r\n'
                    )
                except asyncio.TimeoutError:
                    # Keepalive: leeren Kommentar senden
                    yield b'--ionpy_frame\r\nContent-Type:
text/plain\r\n\r\n\r\n'

        except asyncio.CancelledError:
            pass
        except Exception as e:
            logger.debug(f"MJPEG Stream ({device_id}) beendet: {e}")
        finally:
            # Consumer abmelden – wichtig gegen Memory Leak!
            if device_id in _frame_subscribers:
                try:
                    _frame_subscribers[device_id].remove(q)
                except ValueError:
                    pass

    return StreamingResponse(
        generate(),
        media_type="multipart/x-mixed-replace; boundary=ionpy_frame",
        headers={"Cache-Control": "no-cache"}
    )


# ============================================================================
# VERWALTUNG
# ============================================================================
@router.get("/list")
async def list_cameras():
    """Alle bekannten Kamera-Quellen mit Status."""
    now = time.time()
    result = []

    all_ids = set(_frame_store.keys()) | set(_camera_meta.keys())
```

```
    for device_id in all_ids:
        ts      = _frame_timestamps.get(device_id, 0)
        age_sec = round(now - ts, 1) if ts > 0 else None
        active  = age_sec is not None and age_sec < 10

        meta = _camera_meta.get(device_id, {})
        result.append({
            "device_id":       device_id,
            "name":            meta.get("name", device_id),
            "source_type":     meta.get("source_type", "unknown"),
            "active":          active,
            "last_frame_age":  age_sec,
            "frame_size":      len(_frame_store.get(device_id, b"")),
            "subscribers":     len(_frame_subscribers.get(device_id, [])),
            "snapshot_url":    f"/api/camera/snapshot/{device_id}",
            "mjpeg_url":       f"/api/camera/mjpeg/{device_id}",
            "stream_ws_url":   f"ws://[host]/api/camera/stream/{device_id}"
        })

    return sorted(result, key=lambda x: x["device_id"])


@router.delete("/camera/{device_id}")
async def remove_camera(device_id: str):
    """Kamera aus dem System entfernen (löscht Frame und Meta)."""
    _frame_store.pop(device_id, None)
    _frame_timestamps.pop(device_id, None)
    _frame_subscribers.pop(device_id, None)
    _camera_meta.pop(device_id, None)
    return {"status": "ok", "device_id": device_id}
```

0.2 Router in server.py einbinden

Datei: `web/server.py`

```
# In create_app() nach den bestehenden Routern:
from web.camera import router as camera_router
app.include_router(camera_router)
```

PHASE 1: Server-seitige Kameras (USB & RTSP)

(~2 Tage)

1.1 ServerCamera Klasse

Datei: `devices/drivers/camera/server_camera.py` (NEU)

Hinweis für KI:

- OpenCV ist **nicht async** – läuft in einem Thread
- `threading.Thread(daemon=True)` damit der Thread beim

Prozessende automatisch stirbt

- Kein asyncio in der Capture-Loop – nur in `start()` / `stop()`
- Frame-Rate über `time.sleep()` steuern
- `cv2.VideoCapture(0)` für erste USB-Cam
- `cv2.VideoCapture("rtsp:…")` *für IP-Cams <code python> """ devices/drivers/camera/server_camera.py Kamera-Treiber für USB- und RTSP-Kameras am Server. Verwendet OpenCV für Frame-Capture, schreibt in web/camera.py Frame Store. """ import cv2 import time import threading import logging from typing import Union from web.camera import (_frame_store, _frame_timestamps, _notify_subscribers, register_camera) logger = logging.getLogger("Camera.Server") class ServerCamera: """ Liest Frames von einer USB-Kamera oder RTSP-Quelle. Schreibt JPEG-Frames in den zentralen Frame Store. """ def init(self, device_id: str, source: Union[int, str], # 0, 1, 2 oder "rtsp:…" fps: int = 10, width: int = 1280, height: int = 720, jpeg_quality: int = 75, # 0-100, niedriger = kleiner name: str = None): self.device_id = device_id self.source = source self.fps = fps self.width = width self.height = height self.jpeg_quality = jpeg_quality self.name = name or f"Kamera {device_id}" self._running = False self._thread = None self._cap = None self.error_msg = None self.frame_count = 0 register_camera(device_id, name=self.name, source_type="usb" if isinstance(source, int) else "rtsp") def start(self) → bool: """Startet den Capture-Thread. Gibt True zurück wenn Kamera gefunden.""" if self._running: return True # Vorab prüfen ob Kamera vorhanden cap = cv2.VideoCapture(self.source) if not cap.isOpened(): self.error_msg = f"Kamera '{self.source}' konnte nicht geöffnet werden" logger.error(self.error_msg) cap.release() return False cap.release() self._running = True self._thread = threading.Thread(target=self._capture_loop, daemon=True, name=f"cam_{self.device_id}") self._thread.start() logger.info(f"Kamera gestartet: {self.device_id} " f"({self.source}, {self.fps}fps, " f"{self.width}x{self.height})") return True def stop(self): """Stoppt den Capture-Thread sauber.""" self._running = False if self._thread: self._thread.join(timeout=3.0) logger.info(f"Kamera gestoppt: {self.device_id}") def _capture_loop(self): """Haupt-Capture-Loop (läuft im eigenen Thread).""" self._cap = cv2.VideoCapture(self.source) # Auflösung setzen self._cap.set(cv2.CAP_PROP_FRAME_WIDTH, self.width) self._cap.set(cv2.CAP_PROP_FRAME_HEIGHT, self.height) # Buffer minimieren → weniger Latenz self._cap.set(cv2.CAP_PROP_BUFFERSIZE, 1) interval = 1.0 / self.fps last_frame = 0 error_count = 0 MAX_ERRORS = 10 encode_params = [cv2.IMWRITE_JPEG_QUALITY, self.jpeg_quality] while*

*self._running: now = time.time() # Frame-Rate einhalten elapsed = now - last_frame if elapsed < interval: time.sleep(interval - elapsed) continue ret, frame = self._cap.read() if not ret: error_count += 1 logger.warning(f“Frame-Read Fehler ({self.device_id}): ” f“{error_count}/{MAX_ERRORS}”) if error_count >= MAX_ERRORS: logger.error(f“Kamera {self.device_id} hat aufgehört ” f“zu senden. Versuche Reconnect…”) self._cap.release() time.sleep(2.0) self._cap = cv2.VideoCapture(self.source) error_count = 0 continue error_count = 0 last_frame = time.time() # JPEG encodieren success, jpeg = cv2.imencode('.jpg', frame, encode_params) if not success: continue jpeg_bytes = jpeg.tobytes() # In Frame Store schreiben _frame_store[self.device_id] = jpeg_bytes _frame_timestamps[self.device_id] = last_frame _notify_subscribers(self.device_id, jpeg_bytes) self.frame_count += 1 if self._cap: self._cap.release() @property def is_running(self) → bool: return self._running and (self._thread is not None and self._thread.is_alive()) def get_status(self) → dict: return { “device_id”: self.device_id, “name”: self.name, “source”: str(self.source), “running”: self.is_running, “fps”: self.fps, “resolution”: f“{self.width}x{self.height}”, “frame_count”: self.frame_count, “error”: self.error_msg } </code> ===== 1.2 Kamera-Manager ===== **Datei:** core/camera_manager.py (NEU)*
***Aufwand:** 0.5 Tage **Hinweis für KI:** Wird von SystemEngine verwaltet, ähnlich wie DeviceManager. Liest Kamera-Konfiguration aus config.yaml. <code python> “”“ core/camera_manager.py Verwaltet alle Server-seitigen Kameras. Liest Konfiguration aus config.yaml Sektion 'cameras'. ”“” import logging from typing import Dict from devices.drivers.camera.server_camera import ServerCamera logger = logging.getLogger(“CameraManager”) class CameraManager: def init(self): self.cameras: Dict[str, ServerCamera] = {} def load_config(self, config: dict): “”“ Liest Kameras aus config.yaml. Beispiel config.yaml: cameras: - id: “usb_cam_0” source: 0 fps: 10 width: 1280 height: 720 name: “Laborübersicht” - id: “ip_cam_garage” source: “rtsp:192.168.1.100:554/stream” fps: 5 name: “Eingangskamera” ”“” camera_list = config.get(“cameras”, []) if not camera_list: logger.info(“Keine Kameras in config.yaml konfiguriert.”) return for cam_conf in camera_list: cam_id = cam_conf.get(“id”) if not cam_id: logger.error(“Kamera-Eintrag ohne 'id' – übersprungen”) continue source = cam_conf.get(“source”, 0) # String “0”, “1” → int konvertieren if isinstance(source, str) and source.isdigit(): source = int(source) cam = ServerCamera(device_id = cam_id, source = source, fps = cam_conf.get(“fps”, 10), width = cam_conf.get(“width”, 1280), height = cam_conf.get(“height”, 720), jpeg_quality = cam_conf.get(“quality”, 75), name = cam_conf.get(“name”, cam_id)) if cam.start(): self.cameras[cam_id] = cam logger.info(f“Kamera '{cam_id}' gestartet”) else: logger.error(f“Kamera '{cam_id}' konnte nicht gestartet werden”) def stop_all(self): for cam in self.cameras.values(): cam.stop() self.cameras.clear() def get_status(self) → list: return [cam.get_status() for cam in self.cameras.values()] </code> ===== 1.3 Engine: CameraManager einbinden ===== **Datei:** core/engine.py <code python> # In SystemEngine.init(): from core.camera_manager import*

*CameraManager self.camera_manager = CameraManager() # In start(): self.camera_manager.load_config(self._load_raw_config()) # Hilfsmethode: def _load_raw_config(self) → dict: import yaml with open(self.config_path, 'r') as f: return yaml.safe_load(f) or {} # In stop(): self.camera_manager.stop_all() </code> ===== 1.4 REST Endpoint: Kamera-Status ===== **Datei:** web/camera.py – ergänzen <code python> @router.get("/server/status") async def get_server_cameras(request: Request): """Status aller Server-seitig konfigurierten Kameras.""" engine = request.app.state.engine if not engine or not hasattr(engine, 'camera_manager'): return [] return engine.camera_manager.get_status() </code> ===== 1.5 USB-Kamera Discovery ===== **Datei:** web/inventory/cameras.py (NEU)*
Aufwand: *0.5 Tage <code python> from fastapi import APIRouter import cv2 router = APIRouter() @router.get("/cameras/usb") def list_usb_cameras(): """ Scannt USB-Kamera-Indices 0-9. Gibt alle tatsächlich öffnenbaren Kameras zurück. Etwas langsam (~1s pro Index) aber unumgänglich ohne plattformspezifische APIs. """ found = [] for index in range(10): cap = cv2.VideoCapture(index) if cap.isOpened(): # Eigenschaften auslesen w = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH)) h = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT)) fps = int(cap.get(cv2.CAP_PROP_FPS)) cap.release() found.append({ "index": index, "source": index, "resolution": f"{w}x{h}", "fps": fps, "suggested_id": f"usb_cam_{index}" }) else: cap.release() return found </code> -- ====== PHASE 2: Mobile Kamera ===== (~1.5 Tage) ===== 2.1 PWA: Kamera-Komponente ===== **Datei:** static/mobile/views/camera_sender.js (NEU) **Hinweis für KI:** * Zwei Modi: snapshot (POST alle N Sekunden) und stream (WebSocket Binary) * Standard: snapshot Modus, einfacher und batteriesparender * Rückkamera bevorzugen: facingMode: 'environment' * Canvas zum JPEG-Encodieren im Browser verwenden * Qualitäts-Slider damit User Bandbreite vs. Qualität abwägen kann <code javascript> static/mobile/views/camera_sender.js export default { setup() { const { ref, onUnmounted } = Vue; const isStreaming = ref(false); const mode = ref('snapshot'); 'snapshot' | 'stream' const intervalSec = ref(2); const quality = ref(0.7); JPEG Qualität 0-1 const deviceId = ref('mobile_' + Math.random().toString(36).slice(2, 8)); const statusText = ref('Bereit'); const frameCount = ref(0); const lastFrameSize = ref(0); let videoEl = null; let canvasEl = null; let stream = null; let timer = null; let ws = null; const startCamera = async () ⇒ { try { stream = await navigator.mediaDevices.getUserMedia({ video: { facingMode: 'environment', width: { ideal: 1280 }, height: { ideal: 720 } } }); videoEl = document.createElement('video'); videoEl.srcObject = stream; videoEl.muted = true; await videoEl.play(); canvasEl = document.createElement('canvas'); canvasEl.width = 1280; canvasEl.height = 720; statusText.value = 'Kamera aktiv'; isStreaming.value = true; if (mode.value === 'snapshot') { startSnapshotMode(); } else { await startStreamMode(); } } catch (e) { statusText.value = 'Fehler: ' + e.message; } }; const startSnapshotMode = () ⇒ { const ctx = canvasEl.getContext('2d'); timer = setInterval(async () ⇒ { ctx.drawImage(videoEl, 0, 0); const blob = await new Promise(resolve ⇒ canvasEl.toBlob(resolve, 'image/jpeg', quality.value)); lastFrameSize.value = blob.size; const fd = new FormData();*

*fd.append('device_id', deviceId.value); fd.append('image', blob, 'frame.jpg'); try { await fetch('/api/camera/frame', { method: 'POST', body: fd }); frameCount.value++; } catch (e) { statusText.value = 'Upload Fehler: ' + e.message; } }, intervalSec.value * 1000); }; const startStreamMode = async () ⇒ { const protocol = location.protocol === 'https:' ? 'wss:' : 'ws:'; ws = new WebSocket(`${protocol}${location.host}/api/camera/stream/${deviceId.value}`); ws.binaryType = 'arraybuffer'; ws.onopen = () ⇒ { statusText.value = 'Stream aktiv'; }; ws.onerror = () ⇒ { statusText.value = 'WS Fehler'; }; const ctx = canvasEl.getContext('2d'); 10 FPS timer = setInterval(async () ⇒ { if (ws.readyState !== WebSocket.OPEN) return; ctx.drawImage(videoEl, 0, 0, 640, 480); 640×480 für Stream – weniger Bandbreite const blob = await new Promise(resolve ⇒ canvasEl.toBlob(resolve, 'image/jpeg', quality.value)); lastFrameSize.value = blob.size; const buffer = await blob.arrayBuffer(); ws.send(buffer); frameCount.value++; }, 100); }; const stopCamera = () ⇒ { if (timer) { clearInterval(timer); timer = null; } if (ws) { ws.close(); ws = null; } if (stream) { stream.getTracks().forEach(t ⇒ t.stop()); } isStreaming.value = false; statusText.value = 'Gestoppt'; }; onUnmounted(stopCamera); const formatBytes = (b) ⇒ { if (b < 1024) return b + ' B'; return (b / 1024).toFixed(1) + ' KB'; }; return { isStreaming, mode, intervalSec, quality, deviceId, statusText, frameCount, lastFrameSize, startCamera, stopCamera, formatBytes }; }, template: `*

```
      <div style="background: #27272a; border-radius: 10px; padding:
15px;">
          <div style="font-size: 12px; color: #a1a1aa; margin-bottom:
10px;">
              Geräte-ID (am Desktop sichtbar als):
          </div>
          <input v-model="deviceId"
              :disabled="isStreaming"
              style="width: 100%; background: #18181b; border: 1px
solid #3f3f46;
                     color: #fff; padding: 8px 12px; border-radius:
6px;
                     font-family: var(--font-mono); font-size: 13px;">
      </div>
```

```
      <div style="background: #27272a; border-radius: 10px; padding:
15px;
                  display: flex; flex-direction: column; gap: 12px;">
          <div>
              <div style="font-size: 11px; color: #71717a; margin-
bottom: 6px;">
                  Modus
              </div>
              <div style="display: flex; gap: 8px;">
                  <button v-for="m in ['snapshot', 'stream']" :key="m"
                      @click="!isStreaming && (mode = m)"
```

```
                    :style="{
                        flex: 1, padding: '8px',
                        borderRadius: '6px', border: 'none',
                        background: mode === m ? '#0ea5e9' :
'#3f3f46',
                        color: 'white', fontSize: '12px',
                        opacity: isStreaming ? 0.6 : 1
                    }">
                    {{ m === 'snapshot' ? '□ Snapshot' : '□ Stream'
}}
                </button>
            </div>
        </div>
```

```
        <div v-if="mode === 'snapshot'">
            <div style="font-size: 11px; color: #71717a; margin-
bottom: 4px;">
                Intervall: {{ intervalSec }}s
            </div>
            <input type="range" v-model.number="intervalSec"
                min="1" max="30" :disabled="isStreaming"
                style="width: 100%; accent-color: #0ea5e9;">
        </div>
```

```
        <div>
            <div style="font-size: 11px; color: #71717a; margin-
bottom: 4px;">
                Qualität: {{ Math.round(quality * 100) }}%
            </div>
            <input type="range" v-model.number="quality"
                min="0.1" max="1" step="0.1" :disabled="isStreaming"
                style="width: 100%; accent-color: #0ea5e9;">
        </div>
    </div>
```

```
    <!-- Status -->
    <div style="background: #18181b; border-radius: 8px; padding:
12px;
                display: flex; justify-content: space-between;
                font-family: var(--font-mono); font-size: 12px;">
        <span style="color: #a1a1aa;">{{ statusText }}</span>
        <span style="color: #0ea5e9;">
            {{ frameCount }} Frames
            <span v-if="lastFrameSize > 0" style="color: #71717a;">
                ({{ formatBytes(lastFrameSize) }})
            </span>
        </span>
    </div>
```

```
    <!-- Start / Stop -->
    <button @click="isStreaming ? stopCamera() : startCamera()"
```

```
          :style="{
              padding: '14px',
              borderRadius: '10px',
              border: 'none',
              background: isStreaming ? '#7f1d1d' : '#10b981',
              color: 'white',
              fontSize: '15px',
              fontWeight: 'bold'
          }">
          {{ isStreaming ? '□ Kamera stoppen' : '□ Kamera starten' }}
      </button>
  </div>
  `
```

}; </code>

PHASE 3: Desktop Camera Widget

(~1 Tag)

3.1 camera_view.js – Widget

Datei: `static/views/camera_view.js` (NEU)

Hinweis für KI:

- Einfachste mögliche Implementierung: `<img>` mit MJPEG-URL
- Kein kompliziertes JS – Browser macht den Stream selbst
- Snapshot-Modus als Fallback (`<img>` + `setInterval`)
- `exportState()` für Workspace-Persistenz

```
import { globalStore } from '/static/js/store.js';

export default {
    props: ['deviceId', 'widgetState'],

    setup(props) {
        const { ref, computed, onMounted, onUnmounted } = Vue;

        const mode        = ref(props.widgetState?.mode || 'mjpeg');
        const isActive    = ref(false);
        const snapshotSrc = ref('');
        const frameAge    = ref(null);
        let snapshotTimer = null;

        // Direkte MJPEG-URL — Browser streamt selbst
        const mjpegUrl = computed(() =>
            `/api/camera/mjpeg/${props.deviceId}?t=${Date.now()}`
```

```
        );

        const snapshotUrl = computed(() =>
            `/api/camera/snapshot/${props.deviceId}`
        );

        // Snapshot-Modus: alle 2s neu laden
        const startSnapshot = () => {
            const refresh = () => {
                snapshotSrc.value = snapshotUrl.value + '?t=' +
Date.now();
            };
            refresh();
            snapshotTimer = setInterval(refresh, 2000);
        };

        onMounted(() => {
            if (mode.value === 'snapshot') startSnapshot();
        });

        onUnmounted(() => {
            if (snapshotTimer) clearInterval(snapshotTimer);
        });

        const exportState = () => ({ mode: mode.value });

        return {
            mode, isActive, snapshotSrc, mjpegUrl, snapshotUrl,
frameAge,
            exportState
        };
    },

    template: `
    <div style="display: flex; flex-direction: column;
                height: 100%; background: #000; position: relative;">

        <!-- Toolbar -->
        <div style="display: flex; justify-content: space-between;
                    align-items: center; padding: 6px 10px;
                    background: #1c1c1f; border-bottom: 1px solid
#3f3f46;
                    flex-shrink: 0;">
            <span style="font-size: 11px; color: #a1a1aa;">
                 {{ deviceId }}
            </span>
            <div style="display: flex; align-items: center; gap:
10px;">
                <!-- Modus Toggle -->
                <div style="display: flex; background: #27272a;
                            border-radius: 4px; overflow: hidden;
```

```
                            border: 1px solid #3f3f46;">
                        <button v-for="m in ['mjpeg', 'snapshot']" :key="m"
                            @click="mode = m"
                            :style="{
                                padding: '3px 8px', border: 'none',
                                background: mode === m ? '#0ea5e9' :
'transparent',
                                color: mode === m ? '#fff' : '#a1a1aa',
                                fontSize: '10px', cursor: 'pointer'
                            }">
                            {{ m === 'mjpeg' ? '▶ Live' : '□ Snap' }}
                        </button>
                    </div>
                    <!-- Status Dot -->
                    <div :style="{
                        width: '8px', height: '8px', borderRadius: '50%',
                        background: isActive ? '#10b981' : '#ef4444',
                        boxShadow: isActive ? '0 0 6px #10b981' : 'none'
                    }"></div>
                </div>
            </div>

            <!-- MJPEG: Ein einziges <img> Tag genügt! -->
            <img v-if="mode === 'mjpeg'"
                :src="mjpegUrl"
                style="flex: 1; object-fit: contain; width: 100%;
                       min-height: 0;"
                @load="isActive = true"
                @error="isActive = false">

            <!-- Snapshot Modus -->
            <img v-else-if="snapshotSrc"
                :src="snapshotSrc"
                style="flex: 1; object-fit: contain; width: 100%;
                       min-height: 0;"
                @load="isActive = true"
                @error="isActive = false">

            <!-- Kein Stream verfügbar -->
            <div v-else style="flex: 1; display: flex; align-items: center;
                               justify-content: center; color: #52525b;
                               flex-direction: column; gap: 10px;">
                <i class="mdi mdi-camera-off" style="font-size: 48px;"></i>
                <span style="font-size: 12px;">Kein Stream verfügbar</span>
            </div>
        </div>
    `
};
```

3.2 Widget-Wizard: Kamera-Eintrag

Datei: `static/components/widget_wizard.js`

```
// In widgetCatalogue Array ergänzen:
{
    id:    'camera_view',
    type:  'camera_view',
    title: 'Kamera-Stream',
    icon:  'camera',
    desc:  'MJPEG Live-Stream von USB-, IP- oder Handy-Kamera.',
    needs: 'camera'   // Neuer needs-Typ → zeigt Kamera-Liste
}
```

Hinweis für KI: Der `needs: 'camera'` Typ braucht einen neuen Zweig im `confirmWizard()` der `GET /api/camera/list` aufruft und eine Kamera-Auswahl anzeigt statt dem Entity-Picker.

PHASE 4: Handy als Sensor-Device

(~2 Tage – Rückkanal vom Handy ans Backend)

4.1 Backend: Bidirektionaler WebSocket

Datei: `web/realtime.py`

Hinweis für KI: Den bestehenden WebSocket-Endpoint um einen `receiver()` Task erweitern. Der `sender()` Task bleibt identisch. Beide laufen via `asyncio.gather()`.

Vollständige Implementierung und `MobileSensorSample` sind in der PWA-Roadmap Phase 4.6 beschrieben.

Neuer Sample-Typ:

```
# structures/samples/mobile.py (NEU)
from dataclasses import dataclass
from typing import Any
from structures.samples.base import BaseSample

@dataclass(kw_only=True)
class MobileSensorSample(BaseSample):
    type:     str = "mobile_sensor"
    value:    Any
    unit:     str = ""
    accuracy: int = 2
```

4.2 Verfügbare Handy-Sensoren

Sensor	Browser API	iOS	Android	Aufwand
Batterie Level/Status	Battery Status API	❌	✅	1h
Beschleunigung X/Y/Z	DeviceMotion	⚠ HTTPS+Permission	✅	2h
Gyroskop	DeviceMotion	⚠	✅	inkl. oben
GPS Position	Geolocation	✅	✅	2h
GPS Speed/Heading	Geolocation	✅	✅	inkl. oben
Lärmpegel dB	Web Audio API	✅	✅	3h
Netzwerk Latenz	Network Info API	❌	✅	1h
Bildschirm-Orientierung	Screen API	✅	✅	0.5h
Umgebungslicht	DeviceLight (experimentell)	❌	⚠ Chrome only	–

Empfehlung für MVP:

1. Batterie (einfachster Start)
2. Beschleunigung (coolster Wow-Effekt)
3. GPS (nützlichster für Feldmessung)

4.3 MobileSensorDevice – Browser-Treiber

Datei: `static/mobile/sensor_device.js` (NEU)

Vollständige Implementierung in der PWA-Roadmap, Abschnitt “Frage 1 & 2” – `MobileSensorDevice` Klasse.

4.4 UI: Sensor-Aktivierung in der PWA

Datei: `static/mobile/views/sensors.js` (NEU)

Dritter Tab in der Bottom-Navigation:

- Toggle pro Sensor-Typ (Batterie, Bewegung, GPS, Mikrofon)
- Sendeintervall konfigurierbar
- Live-Vorschau der gesendeten Werte
- Geräte-ID editierbar (wie der Handy am Desktop heißt)
- Verbindungsstatus zum Backend

PHASE 5: Erweiterte Kamera-Features

5.1 HTTP MJPEG Quellen (IP-Cams ohne RTSP)

Aufwand: 0.5 Tage

Viele günstige IP-Kameras senden bereits MJPEG über HTTP. Diese kann man direkt als Quelle lesen:

```
# devices/drivers/camera/http_mjpeg_camera.py (NEU)
import urllib.request
import threading
import time

class HttpMjpegCamera:
    """
    Liest MJPEG-Stream von einer HTTP-URL.
    Beispiel: http://192.168.1.50/video.mjpeg
    Kein OpenCV nötig!
    """
    def __init__(self, device_id, url, ...):
        ...

    def _parse_mjpeg_stream(self, response):
        """MJPEG-Boundary-Parser: extrahiert einzelne JPEG-Frames."""
        boundary = None
        buffer   = b""
        # Boundary aus Content-Type Header extrahieren
        # Dann in einem Loop Frames extrahieren
        ...
```

5.2 Bewegungserkennung (Motion Detection)

Aufwand: 1 Tag
Usecase: Alarm wenn Kamera Bewegung erkennt

```
# In ServerCamera._capture_loop() optional:
import cv2

class MotionDetector:
    def __init__(self, threshold=1000):
        self.prev_frame = None
        self.threshold  = threshold

    def check(self, frame) -> bool:
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        gray = cv2.GaussianBlur(gray, (21, 21), 0)

        if self.prev_frame is None:
            self.prev_frame = gray
            return False

        delta = cv2.absdiff(self.prev_frame, gray)
        self.prev_frame = gray

        thresh = cv2.threshold(delta, 25, 255, cv2.THRESH_BINARY)[1]
        changed_pixels = cv2.countNonZero(thresh)

        return changed_pixels > self.threshold
```

Bei erkannter Bewegung: SystemEvent publishen → Desktop-Alarm.

5.3 Bild-Speicherung & Zeitraffer

Aufwand: 1 Tag

```
# In web/camera.py:
@router.post("/camera/{device_id}/record/start")
async def start_recording(device_id: str, interval_sec: float = 1.0):
    """Startet Zeitraffer-Aufnahme: speichert alle N Sekunden ein
Bild."""
    ...

@router.post("/camera/{device_id}/record/stop")
async def stop_recording(device_id: str):
    """Stoppt Aufnahme und gibt Bildanzahl zurück."""
    ...

@router.get("/camera/{device_id}/timelapse")
async def create_timelapse(device_id: str, fps: int = 10):
    """Erstellt MP4-Zeitraffer aus gespeicherten Bildern (benötigt
ffmpeg)."""
    ...
```

5.4 WebRTC (echter Low-Latency Video-Stream)

Aufwand: 3-5 Tage
Wann: Wenn MJPEG Latenz (1-3s) zu hoch ist

WebRTC ermöglicht <200ms Latenz, ist aber deutlich komplexer. Empfohlene Library: `aiortc` (Python WebRTC)

```
pip install aiortc
```

Sinnvoll wenn:

- Kamera-gestützte Live-Steuerung (Roboter, CNC)
- Mehrere gleichzeitige Zuschauer (MJPEG skaliert schlecht)
- Mobile Handy-Cam mit sehr niedriger Latenz

API-Übersicht

Endpoint	Methode	Beschreibung
`/api/camera/frame`	POST	Frame von Mobile (Formdata)
`/api/camera/stream/{id}`	WS	Frame-Stream von Mobile (Binary)
`/api/camera/snapshot/{id}`	GET	Letztes Standbild
`/api/camera/mjpeg/{id}`	GET	MJPEG Live-Stream

Endpoint	Methode	Beschreibung
/api/camera/list	GET	Alle Kamera-Quellen
/api/camera/server/status	GET	Server-Kamera Status
/api/inventory/cameras/usb	GET	USB-Kamera Discovery
/api/camera/{id}	DELETE	Kamera entfernen

Gesamtübersicht Zeitplan Kamera

Phase	Inhalt	Tage	Ergebnis
0	Frame Store + MJPEG Backend	1.5	Fundament steht
1	USB + RTSP Server-Cams	2.0	Kamera am Server läuft
2	Mobile Kamera Sender	1.5	Handy sendet ins Backend
3	Desktop Camera Widget	1.0	Kamera im Dashboard
MVP		**~6 Tage**	**Vollständig nutzbar**
4	Handy als Sensor-Device	2.0	Handy-Sensoren im Store
5.1	HTTP MJPEG IP-Cams	0.5	Günstige IP-Cams
5.2	Bewegungserkennung	1.0	Alarm bei Bewegung
5.3	Zeitraffer-Aufnahme	1.0	Langzeit-Dokumentation
5.4	WebRTC	4.0	Ultra-Low-Latency

---

config.yaml Erweiterung

```yaml
# config.yaml — Kamera-Sektion
cameras:
  - id: "usb_cam_0"
    source: 0              # USB-Index
    fps: 10
    width: 1280
    height: 720
    quality: 75            # JPEG Qualität 0-100
    name: "Laborübersicht"
    enabled: true

  - id: "ip_cam_eingang"
    source: "rtsp://192.168.1.100:554/stream1"
    fps: 5
    width: 1920
    height: 1080
    quality: 80
    name: "Eingangskamera"
    enabled: true

  - id: "ip_cam_http"
    source: "http://192.168.1.101/video.mjpeg"
```

```
    fps: 10
    name: "Decken-Cam"
    enabled: false
```

Ende Kamera-Roadmap
Code-Beispiele sind Implementierungshinweise – kein fertiger Produktionscode
MJPEG-Streaming ist die empfohlene Einstiegstechnologie – WebRTC als optionale Erweiterung wenn Latenz kritisch wird