Leistungskontrolle

Name: _____ Datum: _____

Rapid Prototyping & AI-Integration

Hallo zusammen,

wir nutzen den heutigen Homeoffice-Tag für einen praxisnahen Leistungsnachweis. Wir simulieren eine „Rapid Prototyping"-Situation: „*Ein Kunde benötigt bis heute Mittag einen funktionierenden Proof-of-Concept (MVP) für eine spezifische Anforderung*".

Ihr habt 4 Stunden Zeit. Das Ziel ist *nicht* die perfekte *„Gold-Standard"*-Software, sondern eine lauffähige, stabile Anwendung, die das Kernproblem löst.

Der Einsatz von KI-Tools (ChatGPT, Copilot, Claude etc.) ist ausdrücklich *erlaubt* und erwünscht. Ich bewerten nicht nur das Coden, sondern eure Kompetenz, KI-Ergebnisse zu verifizieren, zu integrieren und zu dokumentieren.

Zeitplan (Deadline: 23:59 Uhr)

08:10 – 09:20: Themenvergabe, Repository-Setup, Architektur-Entscheidung.

09:20 – 10:00: Development Sprint (Fokus auf Kernfunktionalität).

10:00 – 10:45: Refactoring, Bugfixing & Dokumentation.

10:45 – 11:30: Finaler Commit, Push & Abgabe.

Die Anforderungen

Technologie: Freie Wahl (Java, C#, Python, JS/TS, Dart/Flutter, PHP...). Wählt das, womit ihr am schnellsten seid.

Plattform: Web, Mobile, Desktop oder CLI – passend zum Thema.

Qualität: Die App darf Ecken und Kanten haben, darf aber bei korrekter Eingabe nicht abstürzen (Happy Path).

Abgabe: Link zum Git-Repository (GitHub/GitLab) oder ZIP-Archiv.

Die Dokumentation (WICHTIG!)

Statt einer klassischen Projektdokumentation erstellt ihr eine ausführliche **README.md** im Root-Verzeichnis.
Diese muss folgende Punkte enthalten:

- **Kurzbeschreibung**: Was macht die App? Wie starte/installiere ich sie?
- **KI-Reflexions-Protokoll** (30% der Note):
  - Für welche Teile habt ihr KI genutzt (Logik, UI, Regex, Boilerplate)?
  - Beispiel: „Prompt für die RegEx war X, das Ergebnis war fehlerhaft, ich musste Y manuell korrigieren."
  - Wo hat die KI halluziniert oder schlechten Code geliefert?

1. Der „Code-Review“-Grundsatz

Wenn ihr KI Code generieren lasst, seid ihr nicht mehr der Schreiber, sondern der Reviewer. Das ist eine Senior-Dev-Tätigkeit.

Kein Copy & Paste ohne Verständnis: Ihr müsst jede Zeile, die die KI ausspuckt, erklären können. Wenn ich im Unterricht (oder die Prüfer im Fachgespräch) auf Zeile 42 zeige und frage: „Warum wurde hier async/await statt eines Promises verwendet?“ und eure Antwort ist „Hat die KI so gemacht“, dann habt ihr ein Problem.

Sicherheit & Datenschutz: Ihr postet keine API-Keys, keine Kundendaten und keine internen Firmengeheimnisse in einen öffentlichen LLM-Prompt. Das ist ein Kündigungsgrund, kein „Ups“.

Veraltete Bibliotheken: KIs halluzinieren oft veraltete Packages (gerade bei Flutter oder React Native). Prüft immer die offizielle Dokumentation.

2. Plattformübergreifende Architektur

Ihr entwickelt für verschiedene Plattformen (Web, Android, iOS, Desktop). Die KI ist gut darin, kleine Funktionen zu schreiben, aber sie ist meistens schlecht darin, eine saubere Architektur aufzusetzen.

Wir müssen uns darauf konzentrieren, wie wir die Schichten trennen:

- UI Layer: Wie sieht es aus? (Hier hilft KI gut).
- Business Logic: Was passiert hier? (Hier macht KI oft logische Fehler).
- Data Layer: Woher kommen die Daten?

Wenn ihr die Architektur nicht vorgebt, baut euch die KI einen „Spaghetti-Code-Monolithen“, der unwartbar ist. Eure Aufgabe ist es, das Design Pattern (z.B. MVVM, BLoC, Clean Architecture) vorzugeben und die KI nur die Boilerplate-Arbeit machen zu lassen.

1. MVP (Minimum Viable Product)

Was ist das? Das „kleinstmögliche funktionierende Produkt“.

Für heute bedeutet das: Baut keine perfekte App mit goldenen Rändern. Baut eine App, die funktioniert. Design ist zweitrangig. Wenn der Kunde einen Taschenrechner will, muss „2+2=4“ funktionieren – die Farbe der Tasten ist egal.

2. API (Application Programming Interface)

Was ist das? Ein „Kellner“, der Daten aus der Küche (Server) holt.

Beispiel: Eure App kennt das Wetter nicht. Sie fragt eine Wetter-API (Server im Internet). Die API antwortet mit „20 Grad“.

Wichtig: Ihr braucht keine eigene Datenbank, ihr nutzt Daten aus dem Internet.

3. JSON (JavaScript Object Notation)

Was ist das? Ein Format, wie Daten ausgetauscht werden. Es sieht aus wie Text, ist aber strukturiert.

Beispiel: {„name“: „Max“, „alter“: 17}. Eure App muss diesen Text lesen und verstehen können (das nennt man „Parsen“).

4. Repo / Commit / Push (Git)

Repo (Repository): Der Ordner im Internet (GitHub/GitLab), wo euer Code liegt.

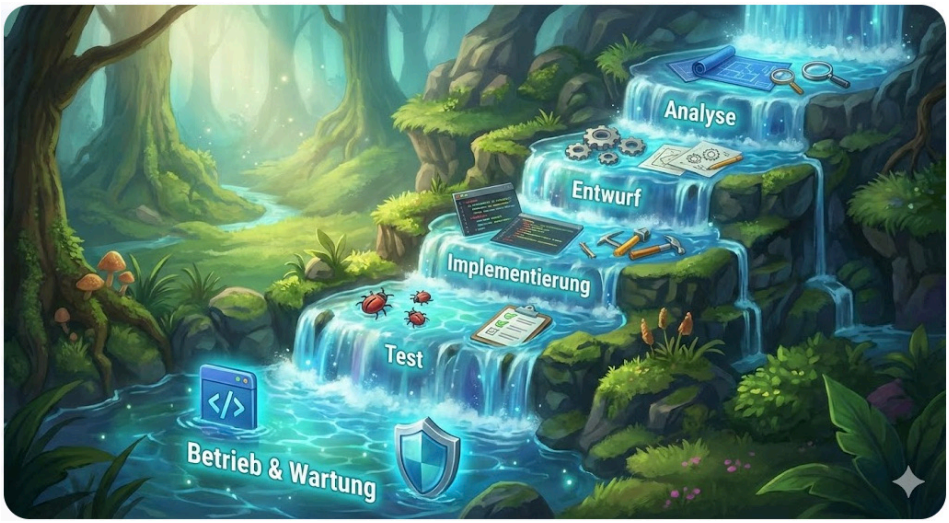
Commit: Das „Speichern“ eines Zwischenstandes.

Push: Das Hochladen auf den Server.

5. Refactoring

Was ist das? Den Code aufräumen.

Warum? KI-Tools schreiben oft Code, der funktioniert, aber chaotisch aussieht. Eure Aufgabe ist es, Variablen sinnvoll zu benennen und unnötiges Zeug zu löschen.



Das Wasserfall-Modell (Der Klassiker)

Das Prinzip: Stell dir vor, du baust ein Haus. Du kannst das Dach nicht decken, bevor das Fundament gegossen ist. Beim Wasserfall-Modell folgt alles einer strengen Reihenfolge. Man darf nicht in die vorherige Phase zurückspringen (außer man reißt alles ab).

Die 5 Phasen:

Analyse: Was soll gebaut werden? (Lastenheft/Pflichtenheft).

Entwurf: Wie soll es technisch aussehen? (Datenbank-Modell, Diagramme).

Implementierung: Jetzt wird programmiert (und zwar erst jetzt!).

Test: Fehlersuche.

Betrieb/Wartung: Die Software wird ausgeliefert.

Vorteile:

Klare Struktur, einfach zu planen.

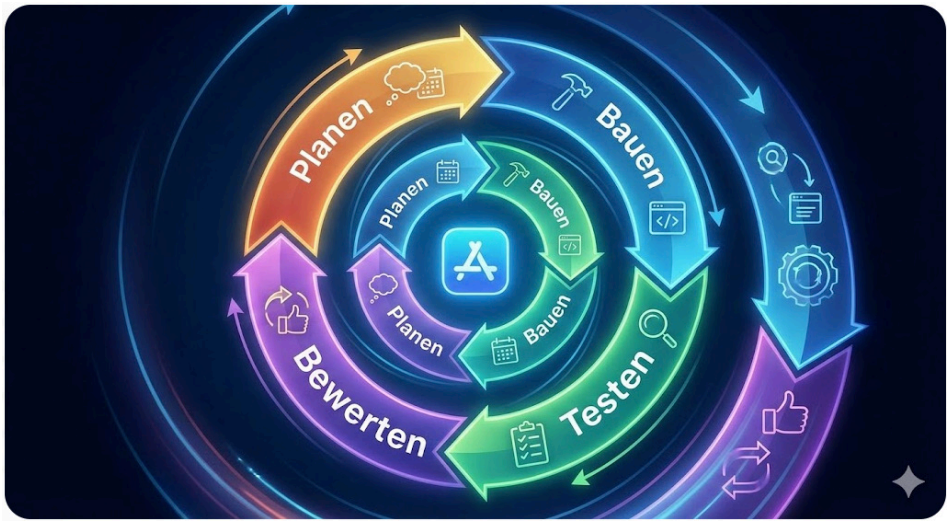
Man weiß am Anfang genau, was es kostet und wann es fertig ist.

Nachteile:

Sehr starr. Wenn der Kunde in Phase 3 merkt „Ich will doch einen blauen Button“, ist es zu spät oder sehr teuer.

Man sieht erst ganz am Ende das fertige Produkt.

Wann nutzt man es? Bei kritischen Systemen (Medizin, Flugzeuge), wo Fehler tödlich sind und man nicht „einfach mal probieren“ darf.



Das Iterative Modell / Rapid Prototyping (Unser Weg heute)

Das Prinzip: Statt das ganze Haus auf einmal zu planen, bauen wir erst eine Hütte, dann ein Haus, dann eine Villa. Wir arbeiten in „Schleifen“ (Iterationen). Das Ziel ist es, schnell etwas Sichtbares zu haben und es dann zu verbessern.

Der Ablauf (Der Kreislauf):

Planen: Was ist das wichtigste Feature für jetzt? (z.B. Nur das Rechen-Modul, noch kein Design).

Entwerfen & Bauen: Schnell programmieren (auch mit KI-Hilfe).

Testen & Bewerten: Läuft es? Gefällt es?

Wiederholen: Okay, es läuft. Jetzt machen wir im nächsten Durchlauf das Design hübsch.

Vorteile:

Man hat sehr schnell ein funktionierendes Programm (MVP).

Man kann Fehler früh erkennen und korrigieren.

Perfekt für die Zusammenarbeit mit KI (KI liefert Code -> Testen -> Verbessern).

Nachteile:

Man verliert leicht den Überblick („Spaghetti-Code“), wenn man nicht sauber dokumentiert.

Es ist schwer vorherzusagen, wann man „wirklich fertig“ ist.

Leistungskontrolle

Name: _____ Datum: _____

Bewertungskriterien

Funktionalität (40%): Tut es das, was gefordert war?

Code-Qualität (30%): Sauberer Code, sinnvolle Variablen, keine Secrets im Code, aufgeräumte Struktur (trotz KI-Einsatz!).

Dokumentation & KI-Reflexion (30%): Ist nachvollziehbar, was Eigenleistung und was KI-Leistung war?

Themen:

Projekt 1: Der Währungs-Umrechner

Ziel: Umrechnung eines Betrags (z.B. EUR) in USD/JPY.
Pflicht: Muss eine echte, kostenlose API live abfragen (keine Hardcoded-Kurse!).
Tech-Fokus: API-Integration, JSON-Parsing

Projekt 2: Markdown-Notizblock

Ziel: Eingabefeld für Markdown-Text -> Echtzeit-Vorschau als HTML.
Pflicht: Lokales Speichern (Browser LocalStorage oder Datei), damit der Text nach Refresh noch da ist.
Tech-Fokus: String-Manipulation, UI-Sate

Projekt 3: Passwort-Tresor (Generator)

Ziel: Generierung sicherer Passwörter (Länge/Zeichen wählbar).
Pflicht: Button zum Kopieren in die Zwischenablage + Visuelle Anzeige der Passwortstärke (Logik!).
Tech-Fokus: Algorithmus, Clipboard-Access

Projekt 4: Pomodoro-Timer

Ziel: Countdown-Timer (25min Arbeit / 5min Pause).
Pflicht: Akustisches Signal oder Desktop-Notification am Ende. Visueller Fortschrittsbalken.
Tech-Fokus: Async/Threading, State-Management

Projekt 5: IP-Subnetz-Rechner

Ziel: Eingabe IP + Subnetzmaske (CIDR).
Pflicht: Berechnung und Ausgabe von Netz-ID, Broadcast-IP und Anzahl nutzbarer Hosts.
Tech-Fokus: Bit-Operationen, Logik

Projekt 6: JSON-to-CSV Converter

Ziel: Tool konvertiert JSON-Array (Text oder Datei) in CSV-Format zum Download.
Pflicht: Muss Fehler abfangen, wenn Objekte im JSON unterschiedliche Felder haben.
Tech-Fokus: File-I/O, Data Parsing

Projekt 7: Das "Daily Standup" Log

Ziel: Erfassung von: „Gestern getan“, „Heute geplant“, „Blocker“.
Pflicht: Speichern der Einträge mit Zeitstempel (lokale DB/Datei) + Historien-Ansicht.
Tech-Fokus: CRUD-Operationen (Create, Read)

PROJEKT 8: SCHNELL-QUIZ (TRIVIA)

Ziel: Quiz-App mit Frage und 4 Antworten.
Pflicht: Fragen müssen aus externer Quelle geladen werden (JSON-Datei oder API). Punktezähler am Ende.
Tech-Fokus: UI-Logik, Datenstruktur

PROJEKT 9: DER QR-CODE GENERATOR

Ziel: Eine App, die einen eingegebenen Text/URL in einen QR-Code umwandelt und anzeigt.
Pflicht: Der generierte QR-Code muss als Bilddatei (PNG/JPG) lokal gespeichert werden können.
Tech-Fokus: Einbindung externer Bibliotheken (Libraries), Grafik-Output.

PROJEKT 10: LOG-FILE ANALYZER (MINI-TOOL)

Ziel: Ein Tool, das eine Textdatei (simuliertes Server-Log) einliest.
Pflicht: Zählen und Ausgeben, wie oft das Wort „ERROR“ und „WARNING“ vorkommt. Einfache Statistik-Anzeige (z.B. „12 Fehler gefunden“).
Tech-Fokus: File I/O, String-Parsing oder RegEx.

PROJEKT 11: DER "THUMBNAILER" (BILD-RESIZER)

Ziel: Ein oder mehrere Bilder auswählen und auf eine feste Breite (z.B. 200px) skalieren.
Pflicht: Das Seitenverhältnis (Aspect Ratio) darf nicht verzerrt werden. Das Ergebnis muss als neue Datei gespeichert werden.
Tech-Fokus: Medien-Verarbeitung, Dateipfade, Batch-Processing (Schleifen).

PROJEKT 12: CODE-SNIPPET MANAGER

Ziel: Verwaltung für häufig genutzte Code-Schnipsel (Titel, Sprache, Code).
Pflicht: Suchfunktion (Filter), um ein Snippet anhand des Titels schnell zu finden.
Tech-Fokus: Listen-Filterung, CRUD (Create, Read), UI-Design.

PROJEKT 13: ARBEITSZEIT-RECHNER

Ziel: Eingabe von „Startzeit“, „Endzeit“ und „Pausendauer“ (in Minuten).
Pflicht: Berechnung der Netto-Arbeitszeit. Warnmeldung, wenn die Zeit > 10 Stunden ist (Verstoß Arbeitszeitgesetz).
Tech-Fokus: Datum/Zeit-Berechnungen (Date/Time Objects), Validierungslogik.

PROJEKT 14: DER TRINKGELD-RECHNER (TIP SPLITTER)

Ziel: Eine App für Restaurantbesuche. Man gibt den Rechnungsbetrag ein, wählt das gewünschte Trinkgeld (z.B. 10%, 15%, 20%) und die Anzahl der Personen.
Pflicht: Berechnung des Gesamtbetrags und – ganz wichtig – wie viel jede einzelne Person zahlen muss.
Tech-Fokus: Rechnen mit Kommazahlen (Floating Point), Eingabe-Validierung.

ZUORDNUNG

PROJEKT	BEARBEITER
PROJEKT 1: DER WÄHRUNGS-UMRECHNER	
PROJEKT 2: MARKDOWN-NOTIZBLOCK	
PROJEKT 3: PASSWORT-TRESOR (GENERATOR)	
PROJEKT 4: POMODORO-TIMER	
PROJEKT 5: IP-SUBNETZ-RECHNER	
PROJEKT 6: JSON-TO-CSV CONVERTER	
PROJEKT 7: DAS „DAILY STANDUP“ LOG	
PROJEKT 8: SCHNELL-QUIZ (TRIVIA)	
PROJEKT 9: DER QR-CODE GENERATOR	
PROJEKT 10: LOG-FILE ANALYZER (MINI-TOOL)	
PROJEKT 11: DER „THUMBNAILER“ (BILD-RESIZER)	
PROJEKT 12: CODE-SNIPPET MANAGER	
PROJEKT 13: ARBEITSZEIT-RECHNER	
PROJEKT 14: DER TRINKGELD-RECHNER (TIP SPLITTER)	

Kleines IT-Glossar für das heutige Projekt

1. Projektmanagement & Arbeitsweise

MVP (Minimum Viable Product):

Das „kleinstmögliche, funktionierende Produkt“.

Kein Schnickschnack, nur die Hauptfunktion muss laufen (z.B. Auto fährt, hat aber noch keine Klimaanlage).

Rapid Prototyping:

Schnelles Entwickeln eines Test-Modells.

Ziel: Schnell Ergebnisse sehen, um zu prüfen, ob die Idee funktioniert.

Refactoring:

Code „aufräumen“ und verbessern, ohne die Funktion zu ändern.

Beispiel: Variablennamen verständlicher machen, doppelten Code löschen.

Happy Path:

Der „ideale Weg“ durch eine App, bei dem der Nutzer keine Fehler macht.

Heute wichtig: Wir programmieren nur für den Happy Path (User gibt alles richtig ein).

Edge Case:

Ein Grenzfall oder unwahrscheinlicher Fehlerfall (z.B. User gibt als Alter „-5“ ein).

Dürfen wir heute vernachlässigen.

Proof-of-Concept (PoC):

Der Beweis, dass eine technische Idee prinzipiell umsetzbar ist.

2. Daten & Technik

API (Application Programming Interface):

Eine Schnittstelle, um Daten von einem fremden Server zu holen (z.B. Wetterdaten, Währungskurse).

Wie ein Kellner: Du bestellst (Request), der Kellner bringt das Essen (Response).

JSON (JavaScript Object Notation):

Ein Textformat, um Daten zu speichern und auszutauschen.

Sieht aus wie: {„name“: „Lisa“, „alter“: 19}.

Parsing / Parsen:

Das Zerlegen und „Verstehen“ von Daten.

Wenn dein Programm einen JSON-Text einliest und in Variablen umwandelt, nennt man das Parsen.

CSV (Comma Separated Values):

Einfaches Dateiformat für Tabellen.

Spalten werden durch Kommas getrennt (z.B. Name,Alter,Wohnort).

RegEx (Regular Expression):

Ein Suchmuster für Text.

Wird genutzt, um zu prüfen, ob eine E-Mail-Adresse gültig ist oder um bestimmte Wörter in einem Text zu finden.

Async / Asynchron:

Dinge, die nicht sofort passieren (z.B. Datei laden, API abfragen).

Das Programm darf nicht „einfrieren“, während es wartet, sondern muss im Hintergrund weiterarbeiten.

3. Tools & Infrastruktur

Repository (Repo):

Der „Lagerplatz“ für deinen Code (z.B. auf GitHub oder GitLab).

Dort liegt die aktuelle Version deines Projekts.

Commit:

Das Speichern eines Zwischenstandes in der Versionsverwaltung (Git).

Wie ein „Speicherpunkt“ in einem Videospiel.

Push:

Das Hochladen deiner lokalen Commits auf den Server (ins Repository).

Local Storage:

Ein kleiner Speicher im Browser des Nutzers.

Daten bleiben erhalten, auch wenn die Seite neu geladen wird (ähnlich wie Cookies, aber moderner).

Framework vs. Library:

Library (Bibliothek): Eine Sammlung von Funktionen, die du nutzt (z.B. für PDF-Erstellung). Du hast die Kontrolle.

Framework: Ein Grundgerüst, das dir vorgibt, wie du programmieren musst (z.B. React, Flutter). Das Framework hat die Kontrolle.

4. KI-Spezifisches

Prompt:

Der Befehl oder die Frage, die du in die KI (ChatGPT etc.) eintippst.

Halluzination:

Wenn die KI selbstbewusst eine Antwort gibt, die komplett falsch oder erfunden ist (z.B. erfundene Befehle, die es gar nicht gibt).

Deshalb: Immer testen!