PROJEKTTAG · PROGRAMMIEREN MIT KI

Vibe-Coding Workshop

Heute baut ihr eure eigene Web-App – mit Ideen, klaren Worten und einer KI als Programmier-Partnerin.

⏱️ 08:00 – 12:15 Uhr | 🧠 Keine Vorkenntnisse nötig | 🌐 Nur ein Browser | 🚀 Am Ende: eure App, online

VibeFabrik · vibefabrik.de · Projekttag für Schulklassen

02 KURZ ZU MIR

Wer euch heute begleitet

DR. PHILIP SCHNEIDER

Produkt, Strategie, IT.

Promotion im Innovationsmanagement, Erfahrung in Beratung, Healthcare-Startup und Automotive-Software.

- ✓ **Hauptberuflich** IT-Portfolio-Manager bei **CARIAD** – der Software-Tochter von **Volkswagen**
- ✓ **Gründer der VibeFabrik** bei Bamberg: individuelle Apps, Webtools und KI-Automatisierungen – in Tagen statt Monaten
- ✓ **Heute:** zeige ich euch genau das, was ich beruflich täglich mache

03 KURZ ZU EUCH

Bevor es losgeht: ein paar Handzeichen

Ich will wissen, wo ihr steht – das hilft mir, das Tempo richtig zu setzen:

🙋

Wer hat schon mal programmiert?

Egal ob Scratch, Python-Unterricht oder nur mal reingeschnuppert – Handzeichen!

🤖

Wer hat schon mit einer KI gebaut?

ChatGPT, Claude oder Ähnliches schon mal Code schreiben lassen?

✨

Und: Was wollt ihr heute bauen?

Ein Spiel? Ein Quiz? Ruft's einfach rein – ich sammle Ideen für später.

Keine Sorge: Egal ob ihr schon programmiert habt oder heute zum ersten Mal einen Code-Editor seht – ihr startet alle mit dem Gleichen: einer Idee und einer KI, die zuhört.

04 ZIEL DES VORMITTAGS

Was lernt ihr heute?

- ✓ Wie Programmieren grundsätzlich funktioniert – die **Grundbausteine**, die es in jeder Programmiersprache gibt
- ✓ Wie eine Webseite aufgebaut ist – **HTML** lesen und verstehen
- ✓ Wie man ihr ein Aussehen gibt – **CSS** in seinen Grundzügen
- ✓ Wie sie lebendig wird – **JavaScript**: Klicks, Punkte, Zufall
- ✓ Wo Daten bleiben – **Datenbank** und ein erster Blick auf **SQL**
- ✓ Wie man einer KI gute Anweisungen gibt: kleine Schritte, klare Wünsche
- ✓ Wie ihr Fehler findet, Feedback gebt und eure App am Ende veröffentlicht

Wichtig: Die KI macht auch Fehler. Wer versteht, was der Code ungefähr tut, merkt das sofort – deshalb starten wir mit den Grundlagen.

05 DER PLAN FÜR HEUTE

Ablauf

08:00	Ankommen & Technik-Check · Browser auf
08:05	**Was ist Vibe Coding?** Was lernt ihr heute? Echte Beispiele
08:25	**HTML** – die Bausteine jeder Seite, inkl. Beispiel-Prompt · dann ⚡ Schritt 1: unsere Quiz-App entsteht
08:45	**CSS** – Farben, Boxen, Knöpfe · dann ⚡ Schritt 2: das Quiz wird schön
08:55	**JavaScript** – das Verhalten der Seite · dann ⚡ Schritt 3: das Quiz wird spielbar
09:10	**SQL** – Daten dauerhaft speichern · dann ⚡ Schritt 4: Bestenliste für die Klasse
09:20	**Vibecoding Essentials** · die Methode, gute Prompts, wo die KI Fehler macht
09:30	**Pause (bis 09:45)** 🥨
09:45	**Gruppenphase:** Konzept, dann Sprints im Studio
11:15	**Pause (bis 11:30)** 🥨
11:30	**Feedback-Runde**, dann finalisieren und veröffentlichen
11:55	**MVP-Präsentationen:** 3 Minuten pro Gruppe
12:10	**Abschluss:** Eure Apps bleiben online – ihr nehmt den Link mit

06 LOS GEHT'S

Was ist Vibe Coding?

Du beschreibst in normaler Sprache, was du bauen willst – **eine KI schreibt den Code.** Du bist die Chefin oder der Chef, die KI ist dein superschnelles Programmier-Team.

🗣️

Du sagst, was du willst

„Baue mir ein Snake-Spiel mit Punkten“ – ganz normale Sätze reichen. So eine Anweisung an die KI nennt man **Prompt**.

🤖

Die KI schreibt den Code

In Sekunden entsteht echter Programm-Code. **KI** heißt „Künstliche Intelligenz“ – ein Programm, das Sprache versteht und selbst Texte und Code schreibt.

🔁

Ihr verbessert gemeinsam

Testen, Wünsche nachschieben, wieder testen. Genau so arbeiten echte Entwickler-Teams – nur dass euer Teammitglied nie müde wird.

Ganz frisch: Den Begriff gibt es erst seit Februar 2025. Ihr lernt heute also etwas, das jünger ist als eure letzte Klassenfahrt.

07 DIE KURZE GESCHICHTE

Woher das alles kommt

PROGRAMMIEREN, GANZ ALLGEMEIN

- **1843** **Ada Lovelace** schreibt den ersten Programm-Algorithmus – für eine mechanische Rechenmaschine.
- **1969** Das **ARPANET** geht ans Netz: die Geburtsstunde des **Internets**.
- **1974** **SQL** entsteht bei IBM – die Sprache, mit der man Datenbanken abfragt.
- **1991** Tim Berners-Lee veröffentlicht **HTML** und das **World Wide Web**.
- **1995** **JavaScript** macht Webseiten interaktiv.
- **1996** **CSS** trennt Inhalt und Design – Webseiten werden hübsch.

UND DANN: VIBE CODING

- **2021** **GitHub Copilot** erscheint: Die KI vervollständigt beim Tippen einzelne Zeilen.
- **NOV 2022** **ChatGPT** kommt heraus – jeder kann mit einer KI reden, die auch Code schreibt.
- **2023/24** Die KI schreibt nicht mehr Zeilen, sondern **ganze Dateien**.
- **2. FEB 2025** **Andrej Karpathy** erfindet auf X das Wort **„Vibe Coding“**.
- **2025** **fly.pieter.com** geht viral. Collins kürt „vibe coding“ zum **Wort des Jahres**.
- **HEUTE** **Agenten** planen, bauen und testen selbstständig – genau das gleich in unserem Studio.

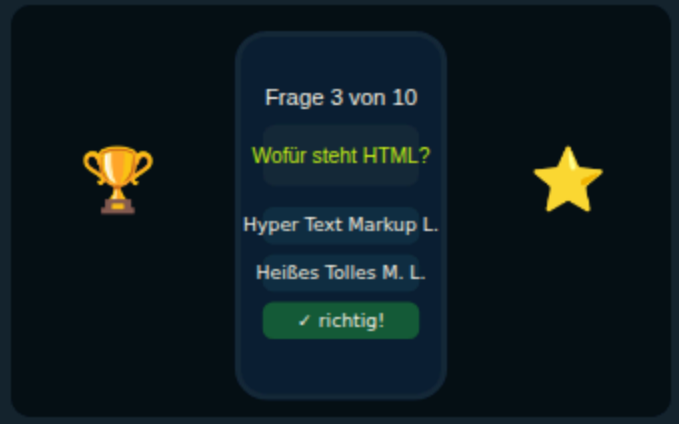
08 ZUM STAUNEN

Was mit Vibe Coding entsteht

Drei Beispiele, die genau so mit KI gebaut wurden – vom ersten Prompt bis zur fertigen App:

🎮 Snake mit Highscore-Liste

Ein komplettes Spiel mit Steuerung, Punkten und einer Rangliste für die ganze Klasse.



🏆 Quiz mit Rangliste

Fragen, Antworten, Auswertung – die besten Ergebnisse landen in der Datenbank.

🌐 Echte Webseiten

Auch „richtige“ Seiten – für den Verein, die Band oder das Hobby. Diese Folien hier sind übrigens selbst vibe-gecodet.

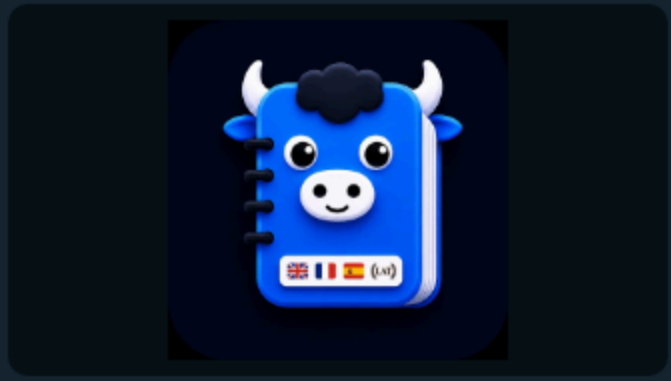
09 UND IM ECHTEN LEBEN?

Beispiele aus der Praxis

Nicht nur Spielereien – das hier gibt es wirklich, ihr könnt es selbst aufrufen:

✈️ fly.pieter.com

Ein Flugsimulator im Browser, den man zu mehreren fliegt. Eine einzige Person baute die erste Version im Februar 2025 in rund **drei Stunden** mit KI – nach 17 Tagen lief er auf 1 Mio. $ Jahresumsatz.



📱 VokaBüffel

Meine eigene Vokabel-App für iPhone und Android: Karteikarten, Lernstand, tägliche Erinnerung. Beschreiben, bauen lassen, testen.

🏢 Apps im Konzern

Bei CARIAD (Volkswagen) laufen Anwendungen für Portfolio- und Finanzsteuerung – gebaut mit **GitHub Copilot**, betrieben auf **Microsoft Azure**. Dieselben Bausteine wie heute, nur größer.

Die Botschaft: Die KI schreibt den Code. Der Mensch versteht das Problem, beschreibt es klar, testet – und entscheidet.

10 DER ÜBERBLICK

Die vier Zutaten einer Web-App

HTML

Der Aufbau. Was steht auf der Seite: Überschrift, Text, Knopf?

```
<button>Antwort A</button>
```

CSS

Das Aussehen. Farben, Schriften, Abstände, runde Ecken.

```
button { background: #c8ef19; }
```

JavaScript

Das Verhalten. Was passiert beim Klicken, Zählen, Spielen?

```
knopf.addEventListener("click", …)
```

SQL

Das Gedächtnis. Was bleibt gespeichert, auch nach dem Schließen?

```
SELECT name, punkte FROM …
```

Merksatz: **HTML sind die Knochen, CSS ist die Kleidung, JavaScript sind die Muskeln – und SQL ist das Gedächtnis.** In genau dieser Reihenfolge gehen wir sie heute durch.

Unser Beispiel heute: eine Lern-Quiz-App

Nach jedem Grundlagen-Teil bauen wir gemeinsam ein Stück weiter – **mit einem einzigen Prompt pro Schritt.** Am Ende steht eine fertige App, die die ganze Klasse spielen kann.

1️⃣

Nach HTML

Das Quiz **existiert**: Frage, vier Antwort-Knöpfe, Punktestand. Grau und schmucklos – aber da.

2️⃣

Nach CSS

Das Quiz **sieht gut aus**: dunkle Karte, große runde Knöpfe, die auf die Maus reagieren.

3️⃣

Nach JavaScript

Das Quiz **funktioniert**: richtig wird grün, falsch rot, Punkte zählen, nächste Frage.

4️⃣

Nach SQL

Das Quiz **erinnert sich**: Namen eingeben, Bestenliste für die ganze Klasse.

Das ist die wichtigste Lektion des Tages: Niemand baut eine App in einem Rutsch. Man baut sie in kleinen Schritten – und nach jedem Schritt schaut man nach, ob sie noch funktioniert.

1 HTML per Prompt

Der Auftrag: die Quiz-App entsteht

PROMPT 1 VON 4

```
Baue mir eine Lern-Quiz-App über das Sonnensystem als eine einzige HTML-Datei. Wichtig: nur HTML, noch kein CSS und kein JavaScript. Oben eine Überschrift „Weltraum-Quiz“, darunter eine Frage, vier Antworten als Knöpfe und ein Absatz für den Punktestand. Erkläre mir danach in zwei Sätzen, welche Tags du benutzt hast.
```

Vier Zutaten für jeden guten Prompt: *Was* soll es sein · *was* soll es können · *womit* gebaut · *wie* soll es aussehen. Und immer nur ein Schritt auf einmal – „Mach mir eine coole App“ führt zu nichts.

Und jetzt: Diesen Prompt schicken wir gleich ab – der Agent baut im Hintergrund weiter, während wir uns die HTML-Grundlagen anschauen. Am Ende des Kapitels checken wir, was daraus geworden ist.

HTML – die Sprache der Webseiten

HTML (HyperText Markup Language) ist die Bausprache jeder Webseite. Sie besteht aus **Tags** – Bausteinen in spitzen Klammern wie <h1>. Sie sagen dem **Browser** (dem Programm, mit dem ihr Webseiten anschaut), *was* etwas ist.

```
<h1>Das ist eine Überschrift</h1>
  ▲           ▲              ▲
Start-Tag   Inhalt      Schluss-Tag
```

MERKT EUCH

Jeder Tag wird geöffnet und (fast jeder) wieder geschlossen – mit </…>, wie eine Klammer.

Tags dürfen ineinander stecken, aber nie überkreuz.

Überschrift und Absatz

```
<h1>Willkommen auf meiner Seite!</h1>
<p>Ich bin Alex und das ist meine
allererste eigene Webseite.</p>
```

SO ZEIGT ES DER BROWSER

Willkommen auf meiner Seite!

Ich bin Alex und das ist meine allererste eigene Webseite.

`<h1>` = größte Überschrift (h2, h3 … werden kleiner) · `<p>` = Absatz (englisch „paragraph“).

Liste und Link

```html
<h1>Meine Hobbys</h1>
<ul>
  <li>Fußball</li>
  <li>Zeichnen</li>
  <li>Gaming</li>
</ul>
<a href="https://www.wikipedia.de">
  Mein Lieblings-Lexikon
</a>
```

SO ZEIGT ES DER BROWSER

Meine Hobbys

- Fußball
- Zeichnen
- Gaming

Mein Lieblings-Lexikon

`<ul>` = Liste, `<li>` = ein Eintrag · `<a>` = Link. `href` ist ein **Attribut**: eine Zusatz-Info im Tag, hier die Zieladresse.

Bild einbauen

```
<img src="katze.jpg"
     alt="Meine Katze Minka">
```

SO ZEIGT ES DER BROWSER

katze.jpg

`src` sagt, *welche* Bilddatei gezeigt wird, `alt` beschreibt das Bild in Worten – wichtig für blinde Menschen und wenn das Bild mal fehlt. Dieser Tag braucht keinen Schluss-Tag.

Ein Knopf – noch ohne Funktion

```
<button>Klick mich!</button>
```

SO ZEIGT ES DER BROWSER

Ein `<button>` ist erst mal nur ein Knopf – grau und ohne Wirkung. Für das *Aussehen* braucht es gleich **CSS**, für die *Funktion* danach **JavaScript**.

Tabelle – schon mal für euren Highscore

```
<table>
  <tr> <th>Name</th> <th>Punkte</th> </tr>
  <tr> <td>Mia</td>  <td>2100</td> </tr>
  <tr> <td>Ben</td>  <td>1900</td> </tr>
</table>
```

SO ZEIGT ES DER BROWSER

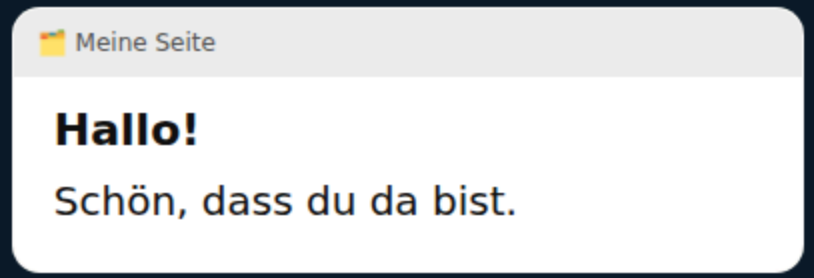
Name	Punkte
Mia	2100
Ben	1900

`<table>` = Tabelle, `<tr>` = Zeile, `<th>` = Überschrift-Zelle, `<td>` = normale Zelle. Merkt sie euch – nachher speichert ihr echte Highscores hinein!

Das Gerüst einer ganzen Seite

```html
<!doctype html>                ← „Das ist eine Webseite“
<html lang="de">
  <head>                       ← Infos für den Browser
    <meta charset="utf-8">        (Umlaute!)
    <title>Meine Seite</title>  ← Text im Tab
  </head>
  <body>                       ← alles Sichtbare
    <h1>Hallo!</h1>
    <p>Schön, dass du da bist.</p>
  </body>
</html>
```

IM BROWSER

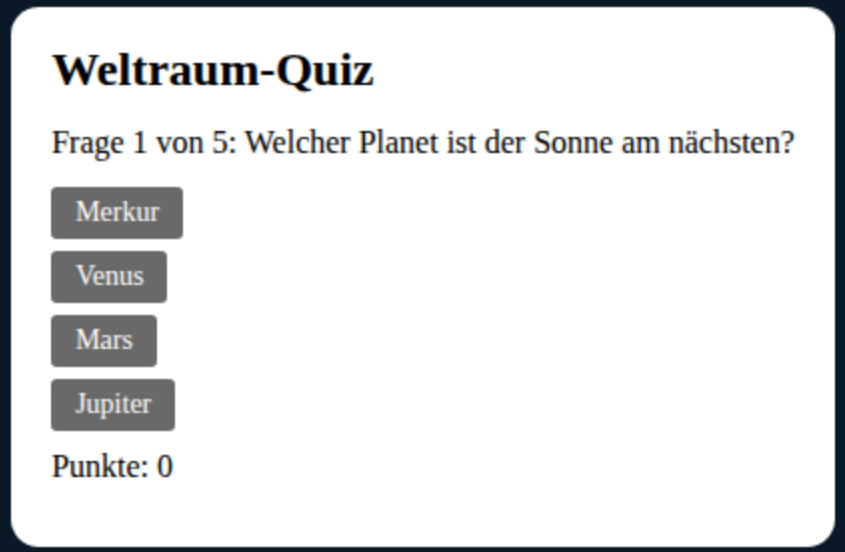


Genau so eine Datei speichert ihr gleich als `meine-seite.html`.

1 HTML per Prompt

Die Quiz-App ist entstanden

WAS DIE KI ZURÜCKGEGEBEN HAT



Genau die Tags von eben: `<h1>`, `<p>`, `<button>`. Grau, schmucklos, klickt nicht – **aber es ist eine echte Webseite.**

2 CSS per Prompt

Der Auftrag: das Quiz bekommt ein Gesicht

PROMPT 2 VON 4

Gestalte das Quiz jetzt mit CSS, **ohne den Inhalt zu ändern**: dunkelblauer Hintergrund, helle Schrift, alles mittig in einer Karte mit Schatten. Die vier Antworten als große runde Knöpfe untereinander, die beim Überfahren die Farbe wechseln.

Der wichtigste Trick steht mitten im Satz: „ohne den Inhalt zu ändern“. Sagt immer dazu, was *bleiben* soll – sonst baut die KI gern alles um.

Und jetzt: Prompt abschicken, Agent arbeiten lassen – und während er baut, schauen wir uns CSS in Ruhe an.

CSS – so wird die Seite schön

CSS (Cascading Style Sheets) bestimmt, *wie* die Sachen aussehen. Eine CSS-Regel besteht immer aus drei Teilen:

```
h1 { color: red; }
 ▲       ▲       ▲
Wer?  Was daran?  Wie?

Wer  = Selektor      (h1, p, .karte)
Was  = Eigenschaft   (color, font-size)
Wie  = Wert          (red, 20px)
```

WO STEHT DAS CSS?

Im `<head>` in einem `<style>`-Block …

… oder in einer eigenen Datei `styles.css`.
Beides ist üblich – die KI macht das für euch.

Farben und Schrift

```css
body { background: #fdf3d8; }

h1 {
  color: #e5484d;
  font-family: Verdana, sans-serif;
}

p {
  color: #4b4b4b;
  font-size: 16px;
}
```

OHNE CSS → MIT CSS

Meine Seite

Willkommen bei mir!

Meine Seite

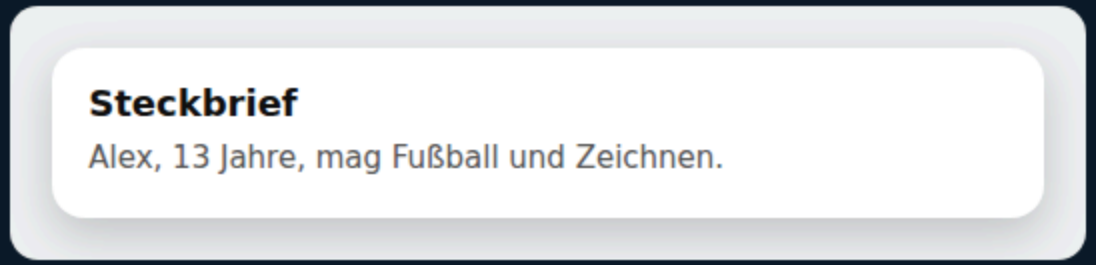
Willkommen bei mir!

Farben schreibt man als **Hex-Code**: `#e5484d` – je zwei Zeichen für Rot, Grün und Blau. Es geht auch einfach `red` oder `gold`.

Jedes Element ist eine Box

```css
/* class="karte" im HTML setzen,
   hier mit einem Punkt ansprechen */
.karte {
  background: white;
  padding: 18px;        ← Luft innen
  margin: 12px;         ← Luft außen
  border-radius: 16px;  ← runde Ecken
  box-shadow: 0 8px 22px #0003;
}
```

SO ZEIGT ES DER BROWSER



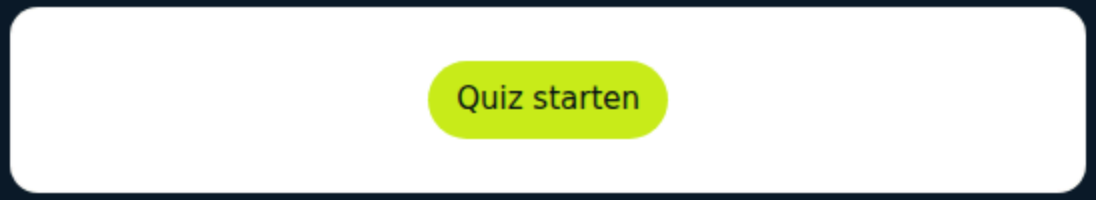
`padding` = Polster innen · `margin` = Abstand nach außen.

Aus dem grauen Knopf wird ein schöner

```css
button {
  background: #c8ef19;
  color: #061118;
  border: none;
  padding: 12px 24px;
  border-radius: 999px;    ← ganz rund
  font-weight: 800;
  cursor: pointer;
}

button:hover {             ← Maus drüber
  background: #061118;
  color: #c8ef19;
}
```

FAHRT MIT DER MAUS DRÜBER!

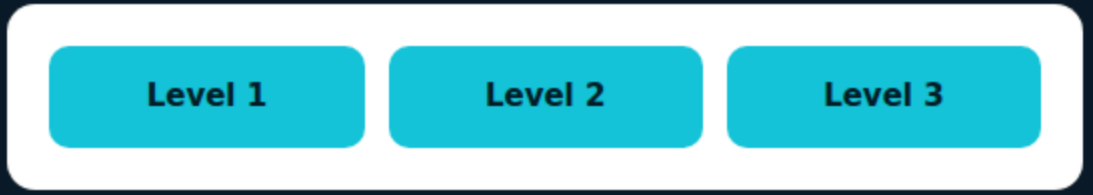


`:hover` heißt: „nur solange die Maus darauf zeigt“. Solche Zusätze nennt man Pseudo-Klassen.

Dinge nebeneinander anordnen

```css
.reihe {
  display: flex;           ← nebeneinander
  gap: 12px;               ← Abstand dazwischen
  justify-content: center; ← mittig
}
```

SO ZEIGT ES DER BROWSER



Ohne `display: flex` stünden die drei Kästen untereinander. Flexbox ist das wichtigste Layout-Werkzeug im Web.

Genau solche Regeln meint ihr, wenn ihr der KI sagt: „Ordne die Knöpfe nebeneinander an und zentriere sie.“

2 CSS per Prompt

Dasselbe Quiz, jetzt mit Gesicht

WAS DIE KI ZURÜCKGEGEBEN HAT

Weltraum-Quiz

Frage 1 von 5: Welcher Planet ist der Sonne am nächsten?

- Merkur
- Venus
- Mars
- Jupiter

Punkte: **0**

Kein Wort im Text hat sich geändert – nur das CSS kam dazu. Fahrt mit der Maus über einen Knopf.

3 JavaScript per Prompt

Der Auftrag: das Quiz wird spielbar

PROMPT 3 VON 4

```
Mach das Quiz mit JavaScript spielbar, ohne Aussehen und Text zu verändern: Beim Klick auf die richtige Antwort wird der Knopf grün und es gibt einen Punkt, bei einer falschen rot. Danach kommt automatisch die nächste Frage. Leg fünf Fragen in einer Liste an und zeige am Ende das Ergebnis.
```

Und jetzt: Prompt abschicken – während der Agent das Verhalten baut, schauen wir uns an, wie JavaScript überhaupt funktioniert.

JavaScript – jetzt passiert etwas

JavaScript ist die Programmiersprache im Browser. Sie kann sich Dinge merken, rechnen, würfeln – und vor allem: auf euch reagieren.

🔍

Element suchen

`document.querySelector("#knopf")` – „hol mir das Element mit der id *knopf*“.

👂

Zuhören

`addEventListener("click", …)` – „wenn geklickt wird, mache Folgendes“.

✏️

Ändern

`element.textContent = "Hallo"` – „schreibe da jetzt etwas anderes hin“.

Das JavaScript steht im `<script>`-Tag oder in einer Datei `app.js`. Mehr Vokabeln braucht ihr für die nächsten vier Folien nicht.

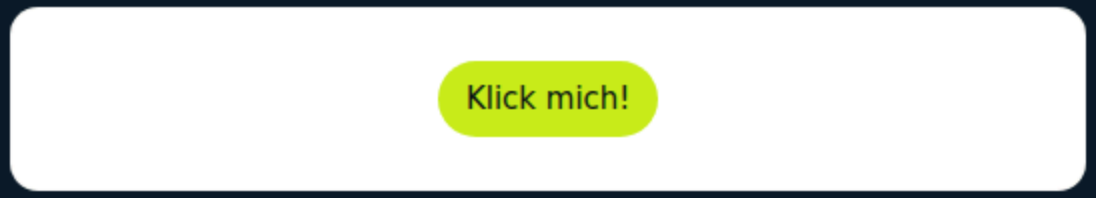
Der Knopf von vorhin – jetzt lebendig

```html
<!-- HTML -->
<button id="knopf">Klick mich!</button>

// JavaScript
const knopf = document.querySelector("#knopf");

knopf.addEventListener("click", () => {
  knopf.textContent = "Autsch! 😄";
});
```

LIVE – PROBIERT ES AUS



`const` = ein fester Name für etwas. Die Zeile `() => { … }` ist der Auftrag, der beim Klick ausgeführt wird.

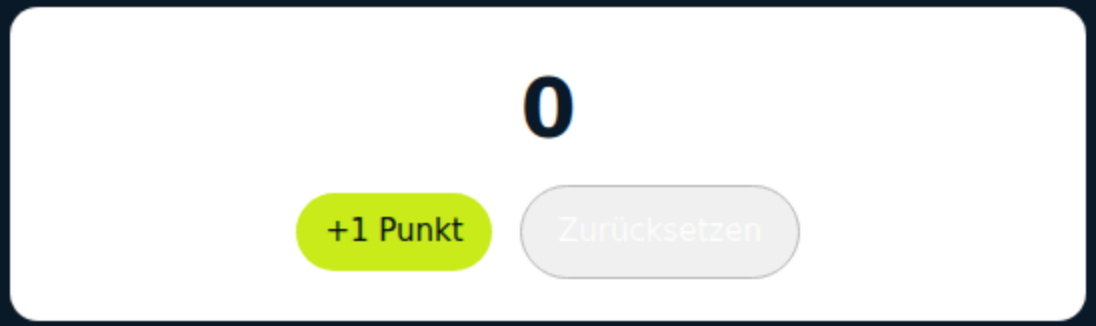
Sich etwas merken: der Punktezähler

```
let punkte = 0;                ← merkt sich eine Zahl

plus.addEventListener("click", () => {
  punkte = punkte + 1;
  anzeige.textContent = punkte;
});

reset.addEventListener("click", () => {
  punkte = 0;
  anzeige.textContent = punkte;
});
```

LIVE – ZÄHLT MIT



`let` ist eine **Variable**: ein Kästchen mit Namen, dessen Inhalt sich ändern darf.

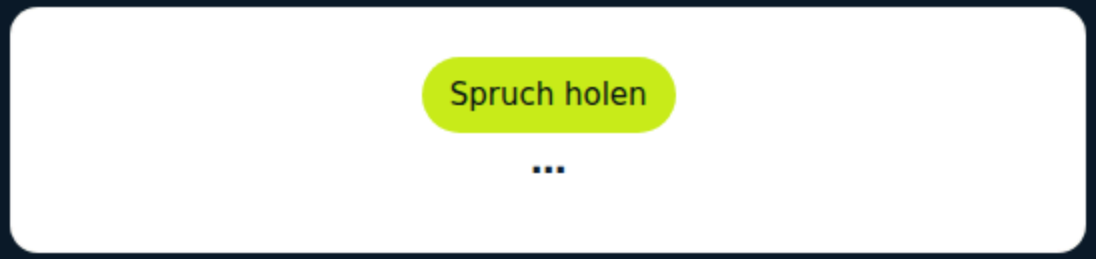
Listen und Zufall

```
const sprueche = [
  "Du schaffst das! 💪",
  "Stark gemacht! ⭐",
  "Weiter so! 🚀",
  "Fast wie ein Profi! 😎"
];

const zufall = Math.floor(
  Math.random() * sprueche.length
);

anzeige.textContent = sprueche[zufall];
```

LIVE – DRÜCKT MEHRMALS



Eckige Klammern `[ … ]` sind eine **Liste**. `Math.random()` würfelt eine Zufallszahl.

Eingaben lesen und entscheiden

```
const name = feld.value;   ← was getippt wurde

if (name === "") {
  gruss.textContent = "Bitte Namen eingeben.";
} else {
  gruss.textContent = "Hallo " + name + "! 👋";
}
```

LIVE – TIPPT EUREN NAMEN

`if … else` = „wenn … sonst“. Damit trifft ein Programm Entscheidungen – die Grundlage jedes Quiz.

Die vier Bausteine jedes Programms

Egal ob Quiz, Snake oder Bank-Software – alles besteht aus diesen vier Dingen:

📦

Variable

Ein Kästchen mit Namen. Merkt sich einen Wert, der sich ändern darf.

```
let punkte = 0;
punkte = punkte + 1;
```

🔀

Bedingung

„Wenn … dann … sonst.“ Damit trifft das Programm Entscheidungen.

```
if (punkte > 8) {
  lob();
} else {
  nochmal();
}
```

🔁

Schleife

Wiederholt etwas, ohne dass man es abtippt – z. B. für alle 10 Fragen.

```
for (let i = 0; i < 10; i++) {
  zeigeFrage(i);
}
```

🧰

Funktion

Ein Arbeitsschritt mit Namen. Einmal schreiben, beliebig oft aufrufen.

```
function lob() {
  alert("Super!");
}
lob();
```

Warum ihr das kennen solltet: Ihr müsst es nicht tippen können – aber wenn ihr es im KI-Code *wiedererkennt*, versteht ihr, was eure App tut. Und ihr könnt viel genauer sagen, was sie stattdessen tun soll.

Alles zusammen – in einer einzigen Datei

```html
<!doctype html>
<html lang="de">
<head>
  <meta charset="utf-8">
  <title>Klick-Zähler</title>
  <style>                          ← CSS
    body   { text-align: center; }
    button { background: #c8ef19; border: none; }
  </style>
</head>
<body>                             ← HTML
  <h1>Mein Zähler</h1>
  <p id="anzeige">0</p>
  <button id="plus">+1</button>
  <script>                         ← JavaScript
    let punkte = 0;
    plus.onclick = () =>
      anzeige.textContent = ++punkte;
  </script>
</body>
</html>
```

DAS IST EINE KOMPLETTE APP

Speichern als `index.html`, Doppelklick – fertig.

Kein Server, keine Installation, kein Konto. Genau so startet ihr gleich.

Am Vormittag reicht **eine** Datei. Im Studio verteilt der KI-Agent den Code später sauber auf `index.html`, `styles.css` und `app.js`.

3 JavaScript per Prompt

Jetzt kann man es wirklich spielen

LIVE – SPIELT EINE RUNDE

Weltraum-Quiz

Frage 1 von 3: Wofür ist HTML zuständig?

- Den Aufbau der Seite
- Die Farben
- Die Datenbank

Punkte: **0**

Jetzt sofort testen: Klickt eine falsche Antwort. Zählt sie fälschlich einen Punkt? Kommt die nächste Frage? Genau solche Fehler baut die KI regelmäßig ein – und genau deshalb probiert ihr jede Version selbst aus.

4 Datenbank per Prompt

Der Auftrag: die Bestenliste für die ganze Klasse

PROMPT 4 VON 4 · DER LETZTE SCHRITT

```
Erweitere das Quiz um eine Bestenliste: Vor dem Start gibt man seinen Namen ein. Speichere Name und Punktzahl am Ende dauerhaft in der Datenbank. Zeige darunter die Top 10 mit Platz, Name und Punkten, die sich nach jeder Runde automatisch aktualisiert.
```

Und jetzt: Prompt abschicken – während der Agent die Datenbank anbindet, schauen wir uns SQL und den Aufbau einer Web-App an.

Wo bleiben die Punkte? SQL & Datenbank

```sql
-- Alle Highscores holen, beste zuerst
SELECT name, punkte
FROM highscores
ORDER BY punkte DESC
LIMIT 10;

-- Einen neuen Eintrag speichern
INSERT INTO highscores (name, punkte)
VALUES ('Mia', 2100);
```

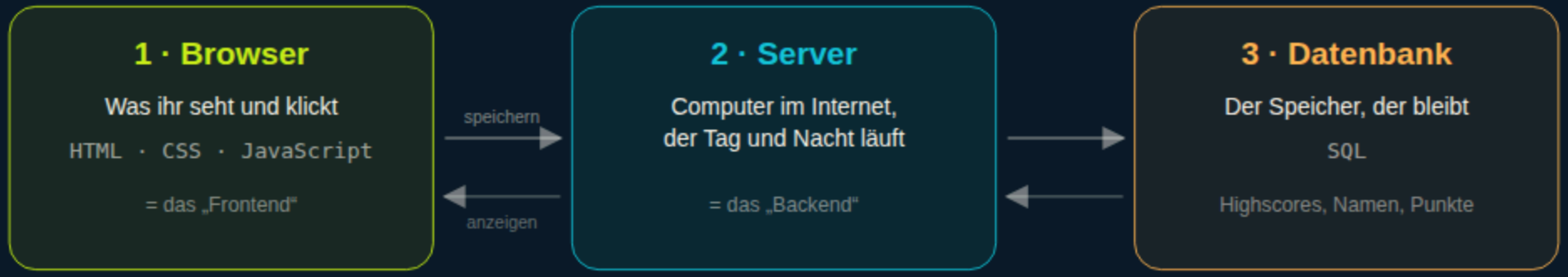
DAS ERGEBNIS

#	Name	Punkte
1	Mia	2100
2	Ben	1900
3	Lea	1500

SQL ist die Sprache, mit der man eine Datenbank fragt. Sie liest sich fast wie Englisch.

Ohne Datenbank ist der Highscore nach dem Neuladen weg und jeder sieht nur seinen eigenen.
Mit Datenbank spielt die ganze Klasse um dieselbe Bestenliste.

Wie eine Web-App aufgebaut ist



Die bisherigen Beispiele sind reines **Frontend** – sie laufen komplett im Browser. Im Studio kommen Server und Datenbank dazu; unter „Code → Vibe-Backend“ könnt ihr ihnen bei der Arbeit zusehen.

4 Datenbank per Prompt

Die Bestenliste für die ganze Klasse

WAS DIE KI ZURÜCKGEGEBEN HAT

Weltraum-Quiz · Bestenliste

#	NAME	PUNKTE
1	Mia	5
2	Ben	4
3	Du 👋	4
4	Lea	3
5	Jonas	2

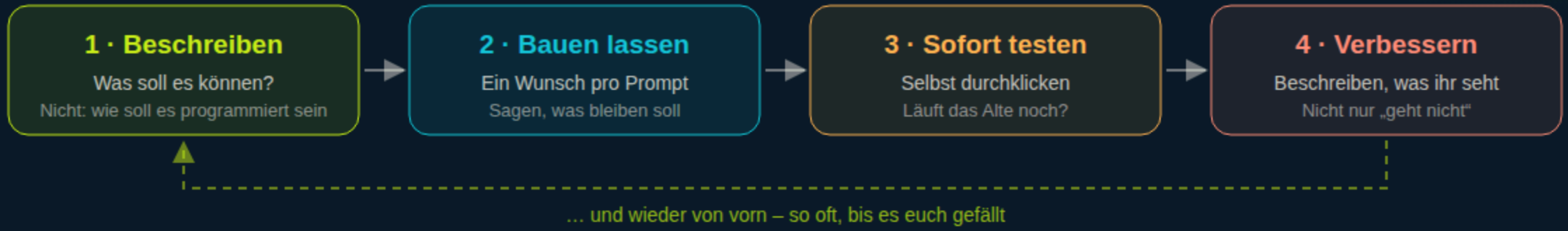
Gespeichert auf dem Server – auch morgen noch da.

Der Unterschied zu allem davor: Diese Liste sehen **alle**, auf jedem Gerät. Erinnert ihr euch an `SELECT name, punkte FROM highscores ORDER BY punkte DESC`? Genau das schreibt die KI jetzt – ihr musstet es nur beschreiben können. **Vier Prompts, eine komplette App.**

41 DIE METHODE

Wie man mit Vibe Coding eine App baut

Ihr habt es gerade viermal gesehen. Dahinter steckt immer dieselbe Schleife – und die dreht ihr gleich selbst:



DREI REGELN, DIE DEN UNTERSCHIED MACHEN

🎯 Eine Kernfunktion
Startet mit der *einen* Sache, nicht mit der ganzen App.

💾 Stand sichern
Erst sichern, was läuft – dann Großes umbauen.

🔒 Keine echten Daten
Niemals echte Namen, Adressen oder Passwörter in die App.

42 BEVOR IHR LOSLEGT

Vier Zutaten für einen guten Prompt

Ihr habt es heute Vormittag schon viermal gesehen – das ist die Formel dahinter:

🎯

Was soll es sein?

Eine Quiz-App, ein Spiel, eine Webseite – nennt das Ding beim Namen.

⚙️

Was soll es können?

Die eine Kernfunktion, die gerade dran ist – nicht die ganze App auf einmal.

🧱

Womit gebaut?

Nur HTML? Auch CSS? Schon JavaScript? Sagt es der KI explizit.

🎨

Wie soll's aussehen?

Farben, Stimmung, Vorbilder – je konkreter, desto besser das Ergebnis.

SO NICHT

```
Mach mir eine coole App.
```

BESSER

```
Baue mir eine Quiz-App über Tiere als eine
HTML-Datei. Oben eine Überschrift, darunter
eine Frage mit vier Antwort-Knöpfen.
```

Die KI macht Fehler – ihr entscheidet

🧪

Immer testen

Klickt jede neue Funktion selbst durch. Was nicht getestet ist, funktioniert erfahrungsgemäß nicht.

🗑️

Sie überschreibt gern

Manchmal verschwindet etwas, das vorher lief. Sagt darum dazu: „Behalte alles Bisherige.“

🕵️

Sie klingt immer sicher

Auch falsche Antworten wirken überzeugend. Ihr seid die Prüfinstanz – nicht umgekehrt.

Beim Fehlersuchen hilft: beschreibt, *was ihr seht* („der Knopf reagiert nicht mehr, seit der Highscore dazu kam“), statt nur „geht nicht“. Und: keine echten Namen, Adressen oder Passwörter in die App.

THEORIE FERTIG

Ab jetzt baut ihr

Sucht euch eine Gruppe, überlegt euch eine Idee – und dann geht's ins VibeFabrik Studio.

👥 In Gruppen | 💡 Eigene Idee | 🛠️ Im Studio | ⏱️ Bis 11:15 Uhr

45 AB 09:45 UHR

Gruppenphase: Was könnt ihr bauen?

Sucht euch etwas aus – oder erfindet etwas ganz Eigenes:

🎮

Spiele

Snake, Reaktionstest, Memory, Weltraum-Ausweichen – mit Punkten, Leveln und Highscore.

🏆

Quiz & Lernen

Quiz über euer Lieblingsthema, Vokabel-Trainer, Mathe-Blitzrechnen – mit Rangliste.

🗳️

Klassen-Tools

Abstimmung für den Wandertag, Klassendienst-Rad, Playlist-Wunschliste – mit gespeicherten Stimmen.

🎨

Kreatives

Zeichenprogramm, Steckbrief-Generator, Witz-Maschine, digitale Glückwunschkarte.

Erst überlegen, dann tippen: Wer ist die Zielgruppe? Was ist die *eine* Kernfunktion? Damit startet ihr – alles andere kommt später.

So läuft die Gruppenphase

Nicht sofort lostippen. Profis arbeiten in dieser Reihenfolge – ihr heute auch:

1. **Konzept – 10 Min, auf Papier.** Wer benutzt die App? Welches *eine* Problem löst sie? Erst wenn ihr das in zwei Sätzen sagen könnt, geht's an den Rechner. noch kein Prompt
2. **Bauen & testen – 65 Min, in Sprints.** Ein Prompt, ein Schritt, sofort ausprobieren. Erst dann der nächste Wunsch. Pause 11:15 – 11:30
3. **Feedback – 15 Min.** Die Nachbargruppe spielt eure App, *ohne dass ihr etwas erklärt.* Ihr schaut zu und schreibt mit, wo sie hängenbleibt.
4. **Finalisieren – 10 Min.** Genau *eine* Verbesserung umsetzen, dann veröffentlichen und den Link sichern.
5. **MVP-Präsentation – 3 Minuten pro Gruppe.** Zeigen, was läuft – nicht, was ihr noch vorhattet.

MVP = „Minimum Viable Product“: die kleinste Version, die schon *wirklich* läuft. Lieber ein Quiz mit drei Fragen, das funktioniert, als zwanzig, die keiner spielen kann.

Jetzt startet das VibeFabrik Studio

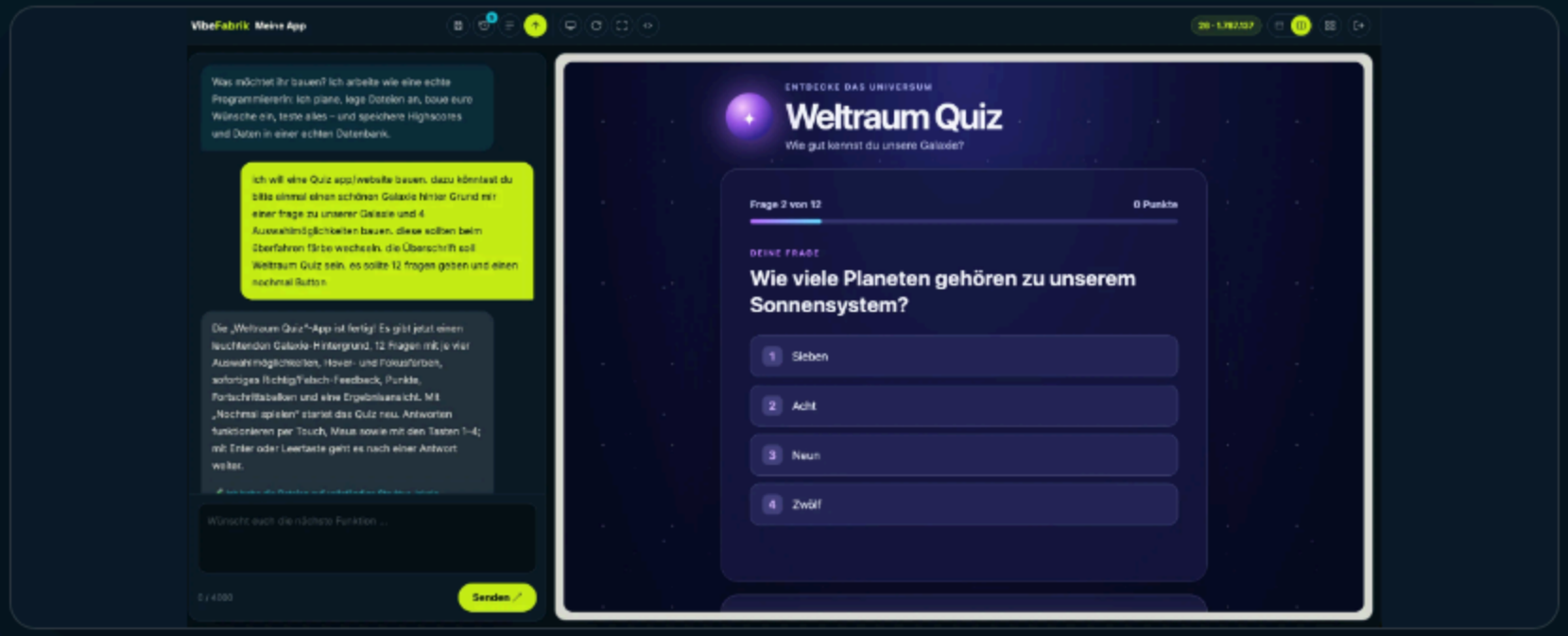
Der KI-Agent im Studio *plant*, legt *mehrere Dateien* an, *testet* seinen Code selbst und nutzt eine *echte Datenbank*.

1. Geht im Browser auf **vibefabrik.de/workshop**.
2. Gebt den **Gruppencode** ein, den ihr von mir bekommt. Kein Konto, keine E-Mail nötig.
3. Beschreibt im Chat eure Idee – am besten mit dem ersten kleinen Schritt beginnen.
4. Testet jede Version sofort in der Live-Vorschau (auch als Smartphone-Ansicht!). Unter „Code“ seht und ändert ihr alle Dateien.
5. Am Ende: **„Veröffentlichen“** klicken – euer App-Link erscheint unten in der Leiste.

Achtung, Budget: Jede Gruppe hat eine begrenzte Zahl an KI-Anfragen (**Tokens** sind die Wort-Bausteine, in denen KI-Arbeit gemessen wird). Überlegt eure Prompts gemeinsam, dann kommt ihr locker hin.

Live-Demo: Wir bauen das Quiz noch einmal – im Studio

Bevor ihr selbst loslegt: unser Weltraum-Quiz von heute Vormittag entsteht noch einmal komplett im VibeFabrik Studio – als echter Agent, mit mehreren Dateien, Selbsttests und einer echten Datenbank.



So sieht es aus, wenn der Agent fertig ist – schaut jetzt live zu, wie er dorthin kommt, bevor eure eigene Gruppenphase startet.

Feedback: Zwei Sterne und ein Wunsch

Zum Schluss testet ihr die Apps der anderen Gruppen – so wie echte Nutzerinnen und Nutzer:

1. Jede Gruppe öffnet über die **Galerie** die App der Nachbargruppe und spielt sie 5 Minuten wirklich durch.
2. Danach sagt ihr **zwei Sterne** ⭐⭐ – zwei Dinge, die richtig gut sind. Konkret: „Die Touch-Steuerung reagiert super“ statt nur „cool“.
3. … und **einen Wunsch** 🌠 – eine Idee, was die App besser machen würde. Immer über die App sprechen, nie über Personen.
4. Wer mag, setzt den Wunsch direkt als letzten Prompt um – oft dauert das nur eine Minute!

50 11:55 · DER GROSSE MOMENT

Eure MVP-Präsentation: 3 Minuten

Drei Minuten sind mehr, als ihr denkt – nutzt sie für eine echte Vorführung statt für Folien:

1. **„Unsere App heißt … und hilft …“** – Name und der eine Zweck, in einem Satz. **20 Sek**
2. **Live vorführen** – eine Runde wirklich spielen, nicht über Bildschirmfotos reden. Was hakt, darf haken. **90 Sek**
3. **„Am stolzesten sind wir auf …“** – die Funktion, die am besten geworden ist. **20 Sek**
4. **„Am meisten gelernt haben wir …“** – gern aus einem Fehler, den die KI gebaut hat. **30 Sek**
5. **Eine Frage aus der Klasse** – wer will wissen, wie ihr etwas hinbekommen habt? **20 Sek**

Keine Angst vor Bugs. Auch in echten Firmen wird ein MVP mit Macken gezeigt – genau dafür ist er da. Was heute läuft, ist mehr, als die meisten Erwachsenen je selbst gebaut haben.

GUT GEMACHT

Zum Abschluss

Was aus eurer App wird, wie ihr sie mitnehmt – und was noch alles geht.

Eure Apps nehmt ihr mit

Alles, was ihr heute baut, gehört euch – und bleibt **vier Wochen lang** online:

🔗

Fester Link · 4 Wochen

Mit „Veröffentlichen“ bekommt eure App eine eigene Internet-Adresse. Sie funktioniert zu Hause, auf dem Handy und bei Oma – bis vier Wochen nach dem Workshop.

🖼️

Klassen-Galerie

Alle freigegebenen Apps erscheinen gesammelt in der Workshop-Galerie. Ein Link für die ganze Klasse – auch als QR-Code fürs Klassenzimmer.

💾

Der Code bleibt für immer

Unter „Code“ könnt ihr alle Dateien ansehen und kopieren. **Macht das vor Ablauf der vier Wochen!** Als `.html`-Datei gespeichert, gehört eure App euch dauerhaft.

Warum nur vier Wochen? Danach räume ich die Workshop-Server wieder auf – das gehört zum sauberen Umgang mit euren Daten dazu. *An die Lehrkraft:* Sie bekommen die Galerie- und App-Links direkt nach dem Workshop, zum Weitergeben und Sichern.

Womit die Profis heute bauen

Vier Stufen – von einfach bis mächtig

1. **Im Chat** – ChatGPT, Claude, Gemini im Browser. Ihr kopiert den Code selbst in eine Datei. heute Vormittag
2. **Im Baukasten** – Lovable, Bolt, Replit: Chat, Vorschau und Veröffentlichen in einem. unser Studio
3. **Im Editor** – GitHub Copilot oder Cursor schreiben mitten im Programm mit.
4. **Als Agent** – **Claude Code**, **OpenAI Codex**, **Google Gemini**: bekommen ein Ziel, arbeiten selbstständig über viele Dateien und testen sich selbst.

So geht man typischerweise vor

🧾 Erst beschreiben, dann bauen

Ziel und Funktionen aufschreiben, bevor der erste Prompt rausgeht. Spart die meisten Irrwege.

🪜 Prompt-Kette in kleinen Schritten

Genau das, was ihr heute viermal gesehen habt: Struktur und Optik → Verhalten → Daten (HTML, CSS, JS, SQL).

🔄 Agent mit Test-Schleife

Der Agent baut, testet selbst, findet den Fehler und bessert nach – der Mensch prüft das Ergebnis.

Was mit Vibe Coding sonst noch geht

Webseiten sind nur der Anfang. Nach demselben Prinzip – beschreiben, bauen lassen, testen – entstehen auch:

Handy-Apps

Echte iOS- und Android-Apps, die im App Store landen. Mein VokaBüffel ist genau so entstanden.

Spiele mit Unity

2D- und 3D-Spiele mit einer richtigen Spiele-Engine – KI schreibt die Steuerung und die Spiellogik.

Minecraft-Mods

Eigene Blöcke, Items und Regeln für euren Server. Beschreiben, was der Block können soll – fertig.

3D & Blender

Blender versteht Python-Skripte. Damit lassen sich Modelle, Animationen und ganze Szenen erzeugen.

Und weiter: **Discord-Bots** · **Excel-Auswertungen** · **kleine Roboter mit Arduino** · **Musik- und Video-Werkzeuge**. Die Frage ist nie „geht das?“, sondern: *Könnt ihr genau genug beschreiben, was ihr wollt?*

Zu Hause weiterbauen

Einfach anfangen

ChatGPT oder Claude im Browser reichen für eine `index.html` völlig aus – kostenlos und ohne Installation.

Richtige Werkzeuge

Wer mehr will: Claude Code oder die Codex-App auf dem Rechner. Kostenpflichtige Abos vorher mit den Eltern anschauen.

Ins Internet stellen

Zum Veröffentlichen gibt es Vercel, für Datenbank und Login Supabase – beides mit kostenlosem Einstieg.

Weiterlernen

Tutorials schauen, z. B. auf youtube.com/@JonasKeil – einfach ausprobieren, und wenn's hakt, die KI selbst fragen, wie es weitergeht.

Startregel: ein eigenes Problem, eine Kernfunktion, testen, verbessern. Nicht die perfekte App zählt, sondern dass aus eurer Idee etwas Echtes wird.

Wörter-Hilfe

Vibe Coding	Programmieren, indem man der KI in normaler Sprache beschreibt, was man haben will.
KI	Künstliche Intelligenz – ein Programm, das Sprache versteht und Texte oder Code erzeugt.
Prompt	Eure Anweisung an die KI. Gute Prompts sind konkret und wünschen einen Schritt auf einmal.
Browser	Das Programm, mit dem man Webseiten anschaut (Chrome, Safari, Firefox, Edge).
HTML	Die Bausprache der Webseiten: legt fest, *was* auf der Seite ist.
Tag	Ein HTML-Baustein in spitzen Klammern, z. B. `<h1>`.
Attribut	Zusatz-Info in einem Tag, z. B. `href="…"` oder `src="…"`.
Selektor	Der CSS-Teil, der sagt, *wer* gestaltet wird: `h1`, `p`, `.karte`.
Frontend	Der sichtbare Teil im Browser: HTML, CSS und JavaScript zusammen.
MVP	Die kleinste Version einer App, die schon wirklich funktioniert und die man zeigen kann.
CSS	Die Gestaltungs-Sprache: Farben, Schriften, Abstände – wie die Seite *aussieht*.
JavaScript	Die Programmiersprache im Browser: Klicks, Spiele, Rechnen – was die Seite *tut*.
Variable	Ein Kästchen mit Namen, in dem sich das Programm etwas merkt (z. B. die Punkte).
Server	Ein Computer im Internet, der ständig läuft und Webseiten oder Daten ausliefert.
Backend	Der unsichtbare Teil auf dem Server – z. B. das Speichern eurer Highscores.
Datenbank	Der geordnete Speicher auf dem Server. Was hier liegt, bleibt – für alle gemeinsam.
SQL	Die Sprache, mit der man eine Datenbank fragt: `SELECT … FROM …`.
Token	Die „Wort-Bausteine“, in denen KI-Arbeit gemessen wird. Euer Gruppen-Budget.
Agent	Eine KI, die selbstständig in Schritten arbeitet: planen → bauen → testen → berichten.
Sprint	Ein kurzer Bau-Abschnitt: ein Prompt, ein Schritt, danach sofort testen.

Fazit: Das nehmt ihr heute mit

🧱

Ihr wisst, woraus eine App besteht

HTML baut auf, CSS gestaltet, JavaScript handelt, SQL erinnert sich. Ihr erkennt diese vier im Code wieder – das reicht, um mitzureden.

🗣️

Beschreiben ist die neue Fähigkeit

Nicht das Tippen entscheidet, sondern wie genau ihr sagt, was ihr wollt: ein Schritt pro Prompt, konkret, und immer dazu, was bleiben soll.

🧪

Testen bleibt eure Aufgabe

Die KI klingt immer überzeugt – auch wenn sie falsch liegt. Wer selbst durchklickt, findet die Fehler, die sie übersieht.

🚀

Ihr habt heute etwas Echtes gebaut

Keine Übungsaufgabe, sondern eine App mit eigener Adresse, die andere benutzen können. Vor drei Jahren war das an einem Vormittag unmöglich.

Der wichtigste Satz zum Mitnehmen: Die Idee bleibt eure. Die KI ist schnell – aber sie weiß nie, *was* gebaut werden soll. Das entscheidet ihr.

GENUG GEREDET

Los geht's!

Eine Idee reicht. Der erste Prompt ist nur einen Satz entfernt – und in ein paar Stunden gibt es eure App, die es vorher nicht gab.

💡 **Klein anfangen** | 🧪 **Ständig testen** | 🗣️ **Konkret prompten** | 🎉 **Am Ende: eure App, live im Netz**

VibeFabrik baut individuelle Apps, Webtools und KI-Automatisierungen – in Tagen statt Monaten.

VibeFabrik · vibefabrik.de · Vibe-Coding Workshop für Schulklassen

Diese Folien wurden – natürlich – mit Vibe Coding gebaut. 💚