

Mini-Projekt – Ereignisprotokoll & Analyse (Abschluss Modul 3)

Szenario

Wir entwickeln ein Ereignisprotokoll-Tool für ein Web-Dashboard. Es dokumentiert Benutzeraktionen, speichert Texteingaben und Zeitstempel, stellt diese dar und ermöglicht statistische Auswertungen sowie JSON-Export. Beispiele für solche Ereignisse:

- „Meeting gestartet“
- „Produktivsystem wurde aktualisiert“
- „Erinnerung: Kundenrückruf um 15 Uhr“
- „Neue Aufgabe hinzugefügt“

Es handelt sich um **Benutzereingaben**, die gespeichert, angezeigt und analysiert werden – ähnlich wie bei einer Notiz- oder Aufgabenliste.

Projektüberblick

Funktionen, die umgesetzt werden:

1. **Formular** für Benutzereingabe eines Ereignisses (Textfeld + Button)
 2. Automatische **Speicherung** des Ereignisses mit ID, Zeitstempel und Textinhalt
 3. **Anzeige** der Ereignisse als Karten-Elemente (ohne Tabelle)
 4. **Anzeige von Statistik** (z. B. Anzahl und durchschnittliche Länge)
 5. **Export** der Datenstruktur als JSON im Entwickler-Tool (Konsole)
-

Schritt-für-Schritt-Anleitung

Schritt 1: HTML-Struktur erstellen

- Baue ein `<form>` mit einem Texteingabefeld (z. B. `id="eventInput"`) und einem Button zum Absenden.
- Lege einen `<div>`-Bereich mit der ID `eventList` an, in dem die Ereignisse später dargestellt werden.
- Ergänze eine Stelle für Statistik (`<p id="stats">`) und einen Button (`id="exportBtn"`) zum Exportieren der Daten.

💡 *Hinweis: Nutze `<div class="event-list">` statt Tabellen – moderner und flexibler.*

Schritt 2: Ereignisobjekte modellieren

- Überlege dir eine Datenstruktur mit folgenden Feldern: eindeutige ID (Zahl), Textinhalt, Zeitstempel.
- Verwende zum Generieren der ID `Math.random()` oder `Date.now()`.
- Der Zeitstempel sollte mit `new Date()` erzeugt werden.

💡 *Tipp: Erstelle eine eigene Funktion `createEvent(text)`, die ein neues Ereignisobjekt zurückgibt.*

Schritt 3: Eingabe verarbeiten und speichern

- Lege ein globales Array `logData` an, in dem alle Ereignisse gesammelt werden.
- Erzeuge bei jedem Absenden des Formulars ein neues Ereignis und füge es zum Array hinzu.
- Leere anschließend das Eingabefeld und aktualisiere die Anzeige.

💡 *Verwende `addEventListener("submit", ...)` auf dem Formular und `preventDefault()`.*

Schritt 4: Ereignisliste anzeigen

- Schreibe eine Funktion, die das Array `logData` durchläuft und alle Ereignisse in `<div>`-Elemente umwandelt.
- Diese Einträge sollen im Container `#eventList` angezeigt werden.

💡 *Tipp: Nutze `document.createElement("div")` und CSS-Klassen für optische Gestaltung.*


Schritt 5: Statistik berechnen

- Zähle alle gespeicherten Ereignisse mit `logData.length`.
- Berechne die durchschnittliche Länge der Texte (Summe aller Längen durch Anzahl).
- Gib diese Info in einem `<p>`-Element auf der Seite aus.

 Hinweis: Nutze `Array.prototype.reduce()` für die Summierung der Textlängen.

Schritt 6: Export als JSON

- Erzeuge mit `JSON.stringify(logData)` eine String-Version deiner Datenstruktur.
- Gib diesen JSON-String in der Entwicklerkonsole aus, sobald der Nutzer auf "Exportieren" klickt.

 Optional: Zeige eine Meldung oder kopiere den JSON-String automatisch in die Zwischenablage.

HTML-Grundstruktur

```
<main>
  <section id="eventLogger">
    <form id="logForm">
      <input type="text" id="eventInput" placeholder="Ereignis eingeben" required />
      <button type="submit">Speichern</button>
    </form>

    <div id="eventList" class="event-list"></div>

    <section class="controls">
      <button id="exportBtn">Daten exportieren (JSON)</button>
      <p id="stats"></p>
    </section>
  </section>
</main>
```

JavaScript-Funktionen (Gerüst)

```
// Ereignisobjekt erzeugen
function createEvent(text) {
  // TODO: gib ein Objekt mit id, text, timestamp zurück
}

// Neues Ereignis hinzufügen
function addEvent(e) {
  // TODO: Eingabetext auslesen, validieren, speichern, rendern, Statistik aktualisieren
}

// Ereignisse anzeigen
function renderEventList(data) {
  // TODO: bestehende Anzeige leeren, alle Events als HTML-Karten einfügen
}

// Statistik aktualisieren
function updateStats() {
  // TODO: Anzahl und durchschnittliche Länge berechnen und anzeigen
}

// JSON exportieren
function exportData() {
  // TODO: logData in JSON konvertieren und in Konsole ausgeben
}

// Event Listener verbinden
function setupListeners() {
  // TODO: submit, export-Button mit Funktionen verbinden
}
```

```
}

// Initialisierung beim Laden
function init() {
  setupListeners();
  // ggf. Beispiel-Daten laden oder Anzeige vorbereiten
}

document.addEventListener("DOMContentLoaded", init);
```

Fazit

Diese Version des Mini-Projekts verzichtet bewusst auf Tabellen und RegExp, um sich ganz auf DOM-Interaktion, Datenhaltung und String-/Zahlenlogik zu konzentrieren. Die visuelle Ausgabe über Karten-Elemente wirkt moderner und ist realitätsnaher für Webanwendungen.

Im nächsten Modul lernst du, diese Komponenten weiter zu modularisieren, dynamisch zu stylen und z. B. mit Servern zu verbinden oder dauerhaft zu speichern.