Pflichtenheft

KB ToDo – Single Page Application

Modul 294

Verfasser	Kasum Bajrami
Schule	Informatikmittelschule (IMS), Modul 294
Auftraggeberin	Frau Duc (Lehrperson)
Abgabe Pflichtenheft	24.05.2026

1. Projektübersicht

1.1 Beschreibung des Projekts

In meinem Modul 294 muss ich eine eigene Single Page Application bauen. Ich habe mich für eine ToDo-App entschieden, weil ich selbst oft den Überblick über meine Aufgaben verliere – vor allem zwischen Schule, Lehrbetrieb und meinen privaten Sachen. Die App heisst KB ToDo (KB sind meine Initialen, Kasum Bajrami).

Die App soll Aufgaben nach der Eisenhower-Matrix sortieren. Das ist eine bekannte Methode, bei der man jede Aufgabe nach Wichtig und Dringend einteilt. So sieht man auf einen Blick, was man sofort machen muss und was man delegieren oder weglassen kann.

Die App läuft komplett im Browser. Man muss nichts installieren und auch keinen Server haben. Ich möchte einfach die index.html doppelklicken und es geht los. Die Daten werden im localStorage gespeichert, also direkt im Browser des Benutzers.

1.2 Projektziele

Diese Ziele will ich am Ende erreichen:

- Die App ist eine echte Single Page Application – die Seite lädt nie neu, alle Aktionen passieren live.
- Ich kann TODOs erstellen, anzeigen, suchen, ändern und löschen (also CRUD).
- Alle elf Pflichtfelder sind vorhanden und funktionieren (Titel, Task/Event, Beschreibung, Autor, Kategorie, Wichtig, Dringend, Priorität, Start, Ende, Fortschritt).
- Die Priorität wird automatisch aus Wichtig und Dringend berechnet (Eisenhower).
- Alle Eingaben werden mit JavaScript validiert, mit eigenen deutschen Fehlermeldungen.
- Die Daten bleiben nach dem Schliessen des Browsers erhalten.
- HTML und CSS bestehen die W3C-Validierung ohne Fehler.
- Der Code ist sauber kommentiert, damit die Lehrperson alles nachvollziehen kann.

1.3 Endprodukt

Am Schluss gebe ich eine einzige Datei ab: index.html. Da sind HTML, CSS und JavaScript drin. Man kann sie auf einen USB-Stick ziehen, doppelklicken und sie läuft im Browser. Zusätzlich gibt es das Journal, in dem ich erkläre wie ich vorgegangen bin, welche Probleme aufgetreten sind und was ich gelernt habe.

Wichtig: Die App ist nicht für ein grosses Publikum gedacht. Sie ist ein Schulprojekt, das zeigen soll, dass ich die Konzepte aus dem Modul 294 verstanden habe.

2. Technische Anforderungen

2.1 Softwarearchitektur

Die App ist eine reine Frontend-Anwendung. Anderes ist nicht erlaubt. Ich verwende:

- HTML5 für die Struktur.
- CSS3 für das Design (mit Custom Properties für Light- und Dark-Mode).
- Vanilla JavaScript (ohne Frameworks wie React).

Alles liegt in einer einzigen Datei (index.html). HTML steht im Body, CSS in einem style-Block im head, JavaScript in einem script-Block am Ende des Body. So bleibt das Projekt portabel und läuft ohne Build-Tool.

Die App hat zwei Ansichten, zwischen denen man per Tab umschaltet: die Liste mit Suchfeld und das Formular zum Erstellen oder Bearbeiten. Beide Ansichten teilen sich dasselbe Formular, das je nach Modus seinen Titel ändert (Neues TODO oder TODO bearbeiten).

2.2 Datenmodelle

Jedes TODO ist ein JavaScript-Objekt mit den folgenden Feldern:

Feld	Typ	Beschreibung
id	String	Eindeutige ID (Date.now als String)
title	String	Pflicht, maximal 255 Zeichen
description	String	Optional, maximal 1000 Zeichen
author	String	Pflicht, maximal 20 Zeichen
category	String	Pflicht, aus Dropdown (Arbeit, Sport, Reisen, Gesundheit, Finanzen)
isEvent	Boolean	true = Event, false = Task
isImportant	Boolean	Eisenhower-Achse Wichtig
isUrgent	Boolean	Eisenhower-Achse Dringend
startDate	String (ISO)	Pflicht, Format YYYY-MM-DD
endDate	String (ISO)	Pflicht, muss >= startDate sein
progress	Number	0 bis 100 (Prozent erledigt)
createdAt	String (ISO)	Zeitstempel der Erstellung

Die Priorität (1 bis 4) wird nicht gespeichert. Sie wird bei jedem Rendern aus isImportant und isUrgent neu berechnet. So gibt es keine Inkonsistenz.

Alle TODOs werden zusammen als Array im localStorage unter dem Schlüssel “kbTodo_data_v1“ gespeichert. Das “_v1” am Ende hilft mir, später eventuell auf eine neue Version umzustellen, ohne die alten Daten kaputt zu machen.

2.3 Schnittstellen

Die App hat keine externen Schnittstellen wie REST-APIs oder Datenbanken. Es ist alles lokal im Browser. Die einzigen Schnittstellen, die ich nutze, sind Browser-Schnittstellen:

- localStorage – zum Speichern der TODOs und der Theme-Einstellung.
- matchMedia (prefers-color-scheme) – um beim ersten Start zu sehen, ob der Benutzer Dark Mode bevorzugt.
- DOM-API – zum Anzeigen und Manipulieren der Inhalte.

Falls ich später mal eine Sync-Funktion mit Cloud einbauen will, könnte ich die Storage-Funktionen (getTodos, saveTodos) einfach durch fetch-Aufrufe ersetzen. Das ist aber kein Teil von diesem Projekt.

2.4 Sicherheitsanforderungen

Da die App nur lokal im Browser läuft und keine Server-Verbindung hat, sind die Sicherheitsanforderungen überschaubar. Trotzdem habe ich folgendes umgesetzt:

- XSS-Schutz: Alle Texte, die der Benutzer eingibt (Titel, Beschreibung, Autor, Kategorie) werden mit einer escapeHtml-Funktion gefiltert, bevor sie ins DOM geschrieben werden. Wenn jemand also <script>alert(1)</script> als Titel eingibt, wird das als Text angezeigt und nicht als Code ausgeführt.
- Eingabe-Validierung: Pflichtfelder werden mit required und mit JavaScript geprüft. So gehen keine kaputten Daten in den Speicher.
- Begrenzte Eingabelängen: maxlength am Input verhindert, dass der Benutzer den Storage mit endlos langen Texten flutet.
- Authentifizierung: nicht nötig, weil keine Multi-User-App. Die Daten gehören dem Browser-Profil und sind nur für diesen User sichtbar.
- Datenschutz: Es werden keine Daten ans Internet gesendet. Alles bleibt lokal.

3. Zeitplan

3.1 Meilensteine

Die Lehrperson hat einen festen Zeitplan vorgegeben. Ich habe meine eigenen Etappen darauf abgestimmt:

Datum	Meilenstein	Was muss fertig sein
13.05.2026	Projektstart	Thema gewählt, Auftrag verstanden, Repository angelegt
24.05.2026	Pflichtenheft	Dieses Dokument ist abgegeben und genehmigt
29.05.2026	Analyse + Masken	Wireframes der drei Ansichten (Liste, Formular, Lösch-Modal) und detaillierte Planung sind im Journal
05.06.2026	Grundgerüst	HTML-Struktur und CSS-Layout stehen, noch ohne Funktionen
12.06.2026	CRUD fertig	Erstellen, Listen, Ändern, Löschen, Suchen funktionieren und werden gespeichert
17.06.2026	Validierung + Tests	HTML und CSS validiert (W3C), alle Felder JS-validiert, manuelle Tests dokumentiert
19.06.2026	Final-Abgabe	index.html und Journal abgegeben

3.2 Liefertermine

Es gibt drei offizielle Liefertermine bei der Lehrperson:

- **24.05.2026 –** Pflichtenheft (dieses Dokument).
- **29.05.2026 –** Analyse, Planung und Masken.
- **19.06.2026 –** Endabgabe: index.html und Journal.

3.3 Abhängigkeiten

Die Aufgaben hängen voneinander ab. Wenn ein Schritt nicht fertig ist, kann der nächste nicht starten:

- Das Pflichtenheft muss genehmigt sein, bevor ich richtig mit dem Programmieren anfange. Sonst baue ich vielleicht das falsche Produkt.
- Die Masken (Wireframes) sollten vor dem CSS-Layout stehen, sonst stylt man im Blindflug.
- Die HTML-Struktur muss stehen, bevor das JavaScript darauf zugreifen kann.
- CRUD-Funktionen müssen funktionieren, bevor man die Validierung dranschrauben kann.
- Die Validierung muss laufen, bevor ich beim W3C einreiche und das Journal fertig schreibe.

Risiko-Punkt: Wenn ich beim Pflichtenheft zu lange brauche, verschiebt sich alles. Darum habe ich mir vorgenommen, das Pflichtenheft vor dem Wochenende fertigzustellen.

4. Problemanalyse

4.1 Potenzielle Probleme

Beim Planen habe ich überlegt, was alles schiefgehen kann. Hier die Punkte, die mir realistisch erscheinen:

Problem	Wahrsch.	Auswirkung	Was tue ich dagegen
localStorage wird vom Benutzer gelöscht (Browser-Cache leeren)	mittel	hoch	Im Journal dokumentieren, dass die App lokal ist. Später wäre ein Export/Import als JSON eine Erweiterung.
XSS-Angriff durch bösartige Eingabe im Titel	tief	hoch	escapeHtml-Funktion auf alle User-Inputs anwenden, bevor sie ins DOM kommen.
HTML/CSS-Validator findet Fehler kurz vor der Abgabe	hoch	mittel	Schon zwei Wochen vor Abgabe validieren, nicht erst am letzten Tag.
Zeitdruck wegen Schule und Lehrbetrieb	hoch	hoch	Pufferzeiten im Zeitplan einplanen, lieber zwei Stunden pro Abend als ein Crunch am Wochenende.
Browser-Inkompatibilität (alte Versionen)	tief	mittel	In den gängigen Browsern (Chrome, Firefox, Edge) testen. Keine experimentellen APIs verwenden.
Datum-Validierung verpasst (Ende vor Start)	mittel	mittel	Cross-Field-Check in validateEndDate: new Date(endDate) < new Date(startDate).
Doppelter Code, weil Erstellen und Bearbeiten getrennt sind	mittel	tief	Ein einziges Formular für beide Modi benutzen. Das verstecke Feld todoId entscheidet, ob CREATE oder UPDATE.
Code wird unübersichtlich (alles in einer Datei)	hoch	mittel	Den JavaScript-Teil in klar abgegrenzte Sektionen aufteilen (Theme, Storage, Validierung, Render, Events, Init) und alles auf Deutsch kommentieren.

4.2 Lösungsansätze

Die wichtigsten Strategien, mit denen ich Probleme vermeide:

- **Defensiv programmieren:** Beim Laden der Daten aus dem localStorage habe ich einen try/catch eingebaut. Falls die Daten kaputt sind, gebe ich einfach ein leeres Array zurück statt die App abstürzen zu lassen.
- **Früh validieren:** Ich validiere das HTML beim W3C, sobald die Struktur steht, und nicht erst am Schluss. So sind die Fehler kleiner und einfacher zu fixen.
- **Inkrementell entwickeln:** Ich baue eine Funktion nach der anderen. Erst Listen-Anzeige, dann Erstellen, dann Ändern, dann Löschen, dann Suche. Nach jedem Schritt teste ich.
- **Klare Trennung:** HTML für Struktur, CSS für Aussehen, JavaScript für Verhalten. Keine Inline-Styles, keine Inline-Event-Handler.
- **Versionierung:** Ich speichere bei wichtigen Etappen eine Sicherheitskopie der index.html auf einem zweiten Ort, damit ich nichts verliere.
- **Pufferzeit:** Ich plane die Endabgabe innerlich auf den 17.06.2026 statt auf den 19.06.2026. So habe ich zwei Tage Reserve für Probleme.

5. Qualitätssicherung

5.1 Qualitätsanforderungen

Damit das Endprodukt etwas taugt, müssen folgende Punkte stimmen:

- **Funktionsfähigkeit:** Alle CRUD-Operationen laufen ohne Fehler. Suche, Filter und Sortierung nach Priorität funktionieren.
- **Validität:** HTML und CSS bestehen die Prüfung beim W3C-Validator ohne Errors. Maximal kleine Warnings, die ich erklären kann.
- **Eingabevalidierung:** Jedes Feld wird validiert. Falsche Eingaben führen zu einer klaren deutschen Fehlermeldung neben dem Feld, nicht zu einem Absturz.
- **Datenspeicherung:** Nach einem Reload sind die TODOs immer noch da, in derselben Reihenfolge wie vorher.
- **Lesbarkeit des Codes:** Variablen und Funktionen haben sprechende Namen (calculatePriority, validateEndDate). Jede Funktion hat einen Kommentarblock, der erklärt was sie tut und warum.
- **Design:** Die App sieht aufgeräumt aus. Es gibt einen Light- und Dark-Mode. Auf einem Handy ist das Layout immer noch benutzbar.
- **Barrierefreiheit:** Tastaturbedienung ist möglich, jedes Eingabefeld hat ein Label, und das Lösch-Modal fängt den Fokus ein (Focus Trap).

5.2 Massnahmen zur Qualitätssicherung

Konkrete Sachen, die ich tue, um die Qualität zu sichern:

Massnahme	Wie ich das mache
Manuelle Tests	Ich gehe nach jeder Etappe meine Test-Checkliste durch: TODO anlegen, suchen, bearbeiten, löschen, Reload machen und schauen ob die Daten noch da sind.
W3C-Validierung	HTML und CSS jeweils beim validator.w3.org einreichen. Errors fixen. Im Journal Screenshots vom „keine Fehler“-Status festhalten.
Browser-Tests	Ich teste die App in mindestens drei Browsern (Chrome, Firefox, Edge) und in zwei Bildschirmgrössen (Desktop und Handy).
Negativ-Tests	Bewusst falsche Eingaben machen: leeres Titel-Feld, Enddatum vor Startdatum, Fortschritt 150 Prozent. Schauen ob die richtigen Fehlermeldungen kommen.
Code-Review mit mir selbst	Nach jeder grösseren Funktion schaue ich den Code noch einmal durch. Frage: „Verstehe ich das in zwei Wochen noch?“ Wenn nicht, schreibe ich einen besseren Kommentar.
Sicherungskopien	An jedem wichtigen Punkt eine Kopie der index.html mit Datum im Dateinamen ablegen.
Journal mitführen	Ich schreibe das Journal nicht erst am Schluss, sondern nach jeder Phase ein paar Zeilen, damit ich die Probleme und Lösungen nicht vergesse.
Abnahme durch die Lehrperson	Beim nächsten Termin präsentiere ich der Lehrperson kurz den Stand und hole mir Feedback. Verbesserungsvorschläge baue ich noch vor der Endabgabe ein.