Code Reviews

Dozent: Prof. Dr. Michael Eichberg
Kontakt: michael.eichberg@dhbw.de
Version: 0.3

Folien: [HTML] https://delors.github.io/lab-codereviews/folien.de.rst.html
[PDF] https://delors.github.io/lab-codereviews/folien.de.rst.html.pdf
Fehler melden: https://github.com/Delors/delors.github.io/issues

1. Einführung

Was sind Code Reviews

Definition

Bei einem Code Review geben Entwickler Feedback zu (fremdem) Code; typischerweise vor einem Merge.

Code Reviews helfen dabei, Fehler frühzeitig zu finden, die Qualität des Codes zu verbessern und sicherzustellen, dass alle Teammitglieder mit dem Code vertraut sind und verstehen, wie der Code funktioniert.

Zweck von Code Reviews

- Qualitätssicherung
- Frühzeitige Fehlererkennung
- Wissensverbreitung (im Team)

Best Practices

für Reviewer

- konstruktiv und respektvoll bleiben
- (kleine) fokussierte Reviews bevorzugen
- Review mit Kontext (Anforderungen, Ziel) durchführen
- eigene Annahmen hinterfragen
- auf Verständlichkeit und Stil achten (Semantik von Bezeichnern etc.)

für die Autoren des Codes

- Reviews nicht persönlich nehmen
- Nachfragen ggf. als Hinweis verstehen für Dokumentationsmängel

Voraussetzung für Code Reviews

- der Code ist konsistent formatiert (Code Prettifier)
- triviale Fehler wurden bereits behoben (Linting)
- der Code kompiliert
- (Unit-/Integrations-) Tests laufen durch
- Architektur und zentrale Designentscheidungen sind dokumentiert

Achtung!

class: incremental

Es ergibt nur Sinn, Code Reviews für Code durchzuführen, der grundlegenden Qualitätsstandards genügt.

Code Review von CSS

Exemplarische Kriterien

- keine Redundanzen, keine unnötigen Style-Definitionen.
- Konsistente Schreibweise von:
  - CSS-Variablen
  - Klassennamen
  - IDs
- Sind die Namen (CSS-Variablen, Klassennamen und IDs) verständlich und sprechend? (Korrektes Deutsch oder Englisch!)
- Werden CSS-Variablen sinnvoll verwendet? (Z. B. um ein leicht anpassbares Design zu ermöglichen?)
- Werden CSS-Layer verwendet, um die Styles zu organisieren und die Wartbarkeit zu erhöhen (Stichwort: Modularisierung)?
- Wird CSS-Nesting verwendet, um die Struktur der Styles zu verdeutlichen?
- *Einfachheit und Korrektheit von* Selektoren
  (Z. B. keine Verwendung von `!important`.)
- Sind die Selektoren effizient?
- Sind komplizierte Selektoren - falls notwendig - dokumentiert/begründet/nachvollziehbar?
- Konsistente Vorgehensweise in Hinblick auf (z. B.) Responsiveness (z. B. konsistente Verwendung von Größen (em, rem, ...) oder das Farbschema).
- Einfachheit des Layouts?
- Gibt es ein klares Vorgehen (Mobile-first oder Desktop-first)?
- Werden Tools eingesetzt, um bei der Formatierung bzw. dem Linting zu helfen.
- Wird modernes CSS verwendet?
  (Dies gilt immer in Hinblick auf die Zielplattformen/Browserversionen.)

Code Reviews von HTML

Exemplarische Kriterien

- Ist der HTML-Code korrekt?

  > **Achtung!**
  >
  > Folgende Aspekte (Auszug) sollten automatisiert geprüft werden:
  >
  > Sind IDs einmalig, sind alle Elemente korrekt geschlossen, ist die Baumstruktur korrekt.

- Benennung von Klassen und IDs:

  Sind die Namen konsistent vergeben und verständlich und korrekt geschrieben.

- Ist die Struktur einfach nachvollziehbar oder übermäßig komplex?

  (Z. B. gibt es eine hohe Anzahl an verschachtelten DIVs oder gibt es eine sinnvolle Strukturierung der Überschriften?).

- Accessibility?

  Wird primär semantisches HTML genutzt? (D. h. beschränkt sich die Nutzung von `<span>` und `<div>` auf die optische Gestaltung?)

  Werden ARIA-Tags sinnvoll verwendet?

  Haben Bilder ein `alt`-Attribut?

- Keine Verwendung von Inline-Styles?
- Keine Verwendung von inline JavaScript?
- Keine Verwendung von `<br>` für Formatierungszwecke.
- Konsistente Verwendung von Tags?

  (Z. B. ist Code immer in `<pre><code>...</code></pre>`?)

- Werden Eingaben in Eingabefeldern geprüft?
- Werden `meta`-Tags für die Spezifikation des Viewports sowie weiterer Metainformationen verwendet?
- Werden übermäßig große Bilder/Icons vermieden?
- Sind Links inhaltlich nachvollziehbar?

Code Reviews von JavaScript

Exemplarische Kriterien

- Macht der Code, was er vorgibt zu tun?

  (D. h. sind Variablen, Klassen, Methoden, …-Namen korrekt und sinnvoll?)
- Werden Sonderfälle und Fehlerzustände behandelt?
- Ist die Fehlerbehandlung konsistent und für den Endnutzer nachvollziehbar?
- Werden keine Fehler „verschluckt“?
- Wird null/undefined korrekt behandelt?
- Wird modernes JavaScript verwendet (z. B. Klassen, `const` und `let` anstatt von *var*, Destrukturierung, Spread und Rest-Operator etc.)?
- Wird `eval()` nicht verwendet?
- Werden asynchrone Funktionen richtig verwendet?
- Werden Promises korrekt behandelt (keine fehlenden .catch() oder try/catch bei async/await)?
- Gibt es potenzielle Nebenläufigkeitsprobleme bei asynchronen Operationen?
- Werden Event-Listener korrekt registriert und wieder entfernt (Memory Leaks)?
- Ist der Kontrollfluss verständlich?

  (Nachfragen deuten auf Probleme bzgl. der Verständlichkeit hin.)
- Ist der Datenfluss nachvollziehbar?

  (Insbesondere bei der Verwendung von globalem Zustand oder über Modulgrenzen hinweg.)
- Wird die Architektur eingehalten?
- Ist der Code modularisiert?
- Wird auf tief verschachtelte Logik verzichtet?
- Werden Funktionen und Methoden klein und fokussiert gehalten (Single Responsibility)?
- Werden teure Manipulationen (des DOMs) auf das notwendige Minimum beschränkt?
- Werden Eingaben validiert (auf Client- und **Server**-Seite, falls es sich um serverseitigen JavaScript-Code handelt)?
- Werden sensible Daten (Passwörter, API-Keys) nicht im Client-Code hartcodiert?
- Werden keine sensiblen Daten geloggt?
- Ist das Logging von Fehlern (Error) sinnvoll und enthält genug Kontextinformationen?
- Gibt es Testfälle? Falls ja, sind diese ausreichend?

  (Codeabdeckung ist hier *nur* ein erster Indikator.)

- [ ] Wird nur minimaler globaler Zustand verwendet?
- [ ] Ist die Verwendung von Bibliotheken/Dependencies gerechtfertigt und aktuell?
- [ ] Gibt es überflüssige oder ungenutzte Abhängigkeiten?
- [ ] Ist die (Inline-)Dokumentation ausreichend und korrekt?
- [ ] Sind die TODOs und FIXMEs verständlich und umsetzbar?
- [ ] Werden Browser-Kompatibilitätsanforderungen erfüllt?
- [ ] Bei verteilten Anwendungen: ist die Aufteilung der Logik nachvollziehbar und sinnvoll (zum Beispiel auch aus Sicht von Cheating-Möglichkeiten)