# Problemstellung

Als Anfang 2018 die Entwicklung der aktuellen FreeDesign-Anwendung begonnen wurde, hatte das Team nur wenig Erfahrung im Entwickeln von ReactJS-Anwendungen. Des Weiteren wurde die Anwendung unter einem hohen zeitlichen Druck entwickelt.

Dadurch ist eine Reihe von technischen Schulden entstanden. Eine der Hauptschulden, ist eine fehlende Definition der Quelltext-Architektur. Durch die Verwendung von ReactJS und Redux wird zwar bereits eine gewisse Architektur vorgeben, diese bezieht sie jedoch auf die Strukturierung der grafischen Oberfläche. Für die Domain-Logik wurde jedoch keine Architektur festgelegt, was die Pflege und Weiterentwicklung der Anwendung, mitunter erschwert.

Die aktuelle Architektur weist aktuell folgende offensichtliche Schwächen auf:

* Die Architektur ist nicht dokumentiert.
* Quelltext für die grafische Oberfläche und für Domain-Logik sind mitunter viel zu eng gekoppelt, was den Austausch und Aktualisierung von JavaScript-Bibliotheken erschwert.
* Durch die, zuvor genannte, enge Kopplung, ist es für einige Teile des Quelltextes nur schwer Unit-Tests zu erstellen bzw. zu pflegen.
* Einige Teile des Quelltextes weißen Muster von Anti-Pattern.

Da die Anwendung einer permanenten Weiterentwicklung unterliegt, ist es wichtig die Software in eine geeignetere Architektur zur überführen. Weiterhin entwickelt sich die Webtechnologie mit großer Geschwindigkeit weiter, hier ist eine Architektur notwendig, die effiziente Pflege ermöglicht.

# Aktueller Stand der Forschung

# Fragestellung der Arbeit

Die zentrale Frage der Arbeit wird sein, ob eine schrittweise Änderung der Quelltext-Architektur möglich ist.

# Ziel

Das Ziel der Arbeit ist die Ausarbeitung eines Vorgehens zur Überführung von einer Ist-Architektur einer ReactJS-Anwendung in eine Soll-Architektur. Um eine hohe Akzeptanz einer solcher Maßnahme zu erreichen, ist eine Rahmen-Bedingung, dass die Überführung Schrittweise geschieht und die Weiterentwicklungsarbeit der ReactJS-Anwendung begleitet.

# Methoden

* Nachteile der aktuellen Architektur ausarbeiten
* Prototype erstellen

# Vorläufiger Aufbau der Diplomarbeit

* Einführung
  + Firmenvorstellung (ca. 2 Seiten)
  + Grundlagen
    - Projektbeschreibung (ca. 3 Seiten)
    - Technische Projektbeschreibung (ca. 3 Seiten) => DevOps etc. ?
  + Problemstellung (ca. 2 Seiten)
* Methoden (ca. 10 Seiten)
  + Analyse der Methoden
  + Analyse der Werkzeuge
* Ergebnisse
  + Ist-Architektur analysieren (ca. 15 Seiten)
  + Soll-Architektur ausarbeiten
    - Mögliche Architekturen ausarbeiten (ca. 15 Seiten)
    - Ausgearbeitet Architekturen vergleichen und auf eine festlegen (ca. 10 Seiten)
  + Schritte zum Erreichen der Sollarchitektur ausarbeiten (ca. 10 Seiten)
* Diskussion (ca. 10 Seiten)
* Zusammenfassung (ca. 1 Seite)

# Literatur

**Langlebige Software-Architekturen**(C. Lilienthal, 3. Auflage 2020 dpunkt.verlag)

Das Buch arbeitet Ziele und Eigenschafften langlebiger Software-Architekturen aus und stellt heute übliche Architekturstile vor. Weiterhin werde Methoden zum Aufspüren technischer Schulden, die gegen eine solche Architektur verstoßen, vorgestellt, sowie Wege diese zu reduzieren.

**Refactoring, Wie Sie das Design bestehender Software verbessern**(M. Fowler, 2. Auflage 2020 mitp-Verlag)

Martin Fowler stellt in seinem Buch Gründe für eine Refactoring vor, sowie Prinzipien und Vorgehensweisen, dieses durchzuführen.

**Effektives Arbeiten mit Legacy Code**(M. C. Feathers, 1. Auflage 2020 mitp-Verlag)

In diesem Buch beschreibt Michael C. Feather das Arbeiten mit bestehendem Quelltext, sowie wie der Pflege des selbigen.

**Entwurfsmuster**(M. Geirhos, 1. Auflage 2015 Reihnwerk-Verlag)

Das Buch stellt umfangreich Entwurfsmuster, sowie Designprinzipien für die Architektur von Software vor. Weiterhin geht es auch auf sogenannte „Design Smells“ und „Anti-Patterns“ ein, welche einer guten Software-Architektur widersprechen.

# Zeitplan