

# Web-basierte Systeme

## 09: Clientseitige Architekturmuster

---

Wintersemester 2024

Rüdiger Kapitza



Lehrstuhl für Informatik 4  
Systemsoftware



Friedrich-Alexander-Universität  
Technische Fakultät

# Vorläufiger Vorlesungsplan

- 16. Oktober Einführung und Darstellung von Webseiten
- 23. Oktober HTML und CSS
- 30. Oktober Hypertext Transfer Protocol
- 6. November Browser Schnittstellen
- 13. November Kommunikationsschnittstellen im Browser
- 20. November WebAssembly
- 27. November Architektur moderner Browser
- 4. Dezember **Clientseitige Architekturmuster und serverseitige Implementierung von Web-basierten Systemen**
- 11. Dezember Vorbereitung Papieranalyse
- 8. Januar Papieranalyse
- 15. Januar Lastverteilung durch Zwischenspeicher
- 22. Januar Web3
- 29. Januar Aspekte von Web Sicherheit
- 5. Februar Zusammenfassung und Ausblick

# Clientseitige Architekturmuster

---

## Zielsetzung der Lerneinheit

- Kennenlernen der verschiedenen Evolutionsschritte von Webanwendungen und der zugehörigen Systemunterstützung
- Kurzer Überblick und technische Herausforderungen von Single Page Applications (SPAs)

## Ausgangspunkt

- Zunächst nur statische HTML-Seiten und Formulare für Benutzereingaben
- **Common Gateway Interface (CGI)**
  - URLs verweisen auf Programme, die dyna. Webseiten erzeugen
  - Laufzeit der Programme ist auf die Erstellung der Seite begrenzt
  - Wichtige Umgebungsvariablen werden an das Programm übergeben, ebenso alle wichtigen Informationen zur Anfrage
- ⇒ Zustandslose Dienste ermöglichen Skalierbarkeit
  - Jede Anfrage ist unabhängig von der vorangegangenen Anfrage
- Perl war lange Zeit die bevorzugte Sprache für CGI-Programme
- **Nachteile des CGI**
  - Mit jeder Anfrage muss ein Prozess gestartet werden und dieser bspw. im Falle von Perl das entsprechende Skript laden
  - Abhilfe schaffen integrierte Module (z.B. `mod_perl`) oder auch FastCGI deren Laufzeit nicht an einzelne Anfragen gebunden sind

## Erste Frameworks für Webanwendungen

- Laufzeitsystem der verwendeten Programmiersprache ist direkt in einen Webserver integriert
- *Templates* als Mittel der Wahl: Mix aus Code und HTML
- Vielzahl von web-spezifischen Bibliotheken
  - URL Verarbeitung
  - Erzeugung von HTML Code
  - Mechanismen für den Aufbau von Sitzungen
  - Schnittstellen zum Zugriff auf Datenbanken
- Beispiele: PHP, ASP.net und JavaServlets

## Beispiel: JavaServlets

- Java-Klassen mit standardisierter Schnittstelle zur Verarbeitung von Anfragen

```
public class SimpleServlet extends HttpServlet {
    ...

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.getWriter().println("Hello World!");
    }

    @Override
    public void init() throws ServletException {
        System.out.println("Servlet " + this.getServletName() + " has started");
    }

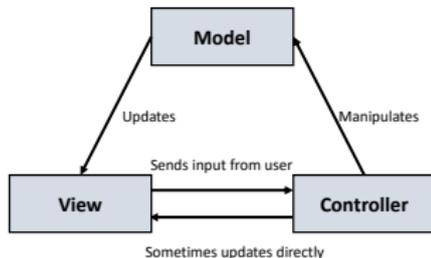
    @Override
    public void destroy() {
        System.out.println("Servlet " + this.getServletName() + " has stopped");
    }
}
```

- Einfach auszuprobieren mit Jetty (<https://www.eclipse.org/jetty/>)

## Zweite Generation von Frameworks für Webanwendungen

- Model-View-Controller (MVC) als Architekturmodell zur Dekomposition von Webanwendungen
- Abbildung von Objekten auf die Strukturen von Datenbanken
  - Einfachere Behandlung von dynamischen Daten
- Beispiele: Ruby on Rails und Django  
(<https://www.djangoproject.com/>)

## Model-View-Controller (MVC) Entwurfsmuster



- **Model:** Verwaltet die Daten der Anwendung
  - JavaScript Objekte – alle wichtigen Daten der Anwendung
- **View:** Darstellung der Webseite (z.B. HTML/CSS)
- **Controller:** Lädt Model und View, kontrolliert/organisiert die Interaktion mit dem Nutzer
  - JavaScript Skripte – Behandlung von DOM-Ereignissen und Interaktion mit dem Webserver
- Als Entwurfsmuster schon lange bekannt – Wurzeln in der Sprache Smalltalk

## Erzeugung einer View

- Webanwendung erzeugt HTML und CSS
- Templates sind der dafür gebräuchliche Ansatz
  - Bspw. zur Ausformulierung von häufig genutzten Teildokumenten
  - Ergänzt durch kleine Codefragmente zur Erzeugung von dynamischen Anteilen
  - Vor Auslieferung wird Code ausgeführt und die Ergebnisse in die Webseite eingefügt
- Vorzüge von Templates (...gegenüber JS Progra. des DOM)
  - HTML-Struktur bleibt erhalten – Templates können oft direkt im Browser betrachtet werden
  - Ermöglicht eine gute Einschätzung, wie sich dynamische Inhalte in die Website einfügen
  - Templates können im Server oder im Browser ausgefüllt werden (!)

## Controller

- Verantwortlichkeiten
  - Verbindet Model und View
    - Dies beinhaltet Kommunikation mit dem Server um Modelle abzuholen und Änderungen abzulegen
  - Kontrolliert welche Templates angezeigt werden
  - Reagiert auf Benutzereingaben
- Mit der dritten Generation von Frameworks werden diese typischerweise im Browser ausgeführt

## Model

- Model stellt alle nicht-statischen Daten bereit, die von Views und Controllern benötigt werden
- Stehen in Bezug und Abhängigkeit zum Datenbankschema der Anwendung
  - Object Relational Mapping – jede Zeile entspricht einem Objekt
- Das Datenbankmodel wird oft durch die Webentwicklung/Views stark beeinflusst
- Mögliche Konflikte zwischen Datenbankschema und Anwendungsmodellen – Anwendungen sind oft dynamischer und erfordern mehr Flexibilität

## Model-View-Controller als allgegenwärtiges Architekturmuster

- Findet Verwendung in vielen Frameworks
- Webanwendungen setzen sich zunehmend aus Komponenten zusammen, welche auch nach dem MVC-Muster strukturiert sind
  - Die Aufteilung in Model, View und Controller ist auf Komponentenebene zu beantworten

## Beispiel: Django

- Viele Frameworks interpretieren MVC auf die eine oder andere Weise. Bei Django spricht man von einem MTV-Framework:
- **M**odel – Datenzugriff und Relation zwischen Daten
- **T**emplate – Präsentationsschicht
- **V**iew – Anwendungslogik
- View ist nicht wirklich View im Sinne von MVC!

## Dritte Generation von Frameworks für Webanwendungen

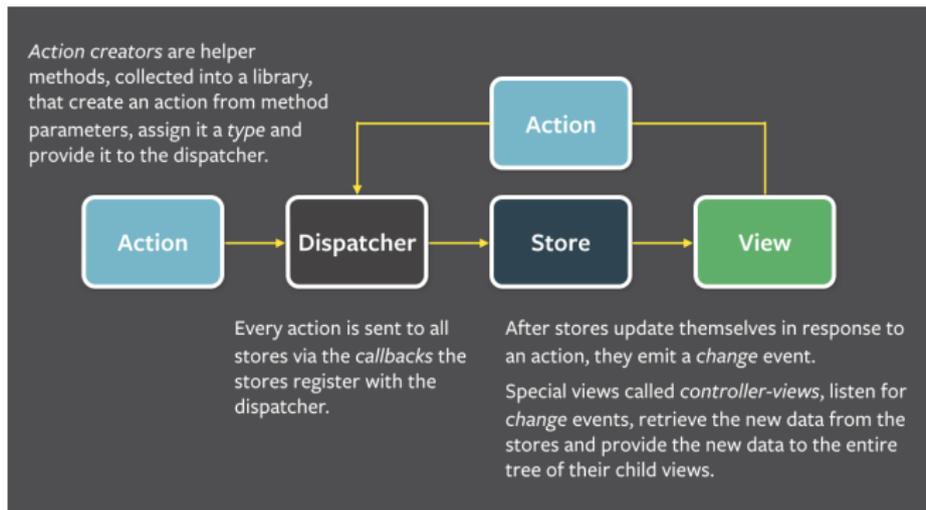
- JS-basierte Frameworks die im Browser ausgeführt werden
  - Mehr Anwendungscharakter als zuvor
  - Interaktive und responsive Anwendungen
- Weniger Anforderungen an die serverseitige Infrastruktur
  - Node . js als Umgebung für JavaScript auf Serverseite
  - NoSQL Datenbank (z.B. MongoDB)
- Weiterentwicklung existierender Konzepte
  - Model-View-Controller
  - Templating
- Beispiele: AngularJS 1 und ReactJS

## Wo geht die Reise aktuell hin?

- Flux Muster für skalierbare Webanwendungen im Browser

## Flux Muster - Erweitert MVC für komplexe Webanwendungen

### ■ Unidirektionales Kommunikationsmodell



## Flux Elemente

- **Actions** kapselt Zustandsänderungen
- **Store** kapselt den Zustand und macht ihn manipulierbar über Actions und aktualisiert die View:
  - Kann durch Komponenten gelesen werden
  - Kann nur durch Actions modifiziert werden
  - Erzeugt Ereignisse bei Zustandsänderungen
- **View** UI,
  - Registriert sich für Veränderungen beim Store
  - Löst Actions basierend auf Nutzereingaben aus
- **Dispatcher** Übermittelt Actions um die Daten des Stores zu verändern

# Single Page Applications

---

## Single Page Applications (SPA)

- SPA entspricht einer Webanwendung, die auf Benutzereingaben mit einer veränderten Darstellung reagiert - ohne eine neue HTML-Seite vom Server zu laden
- Vorteil: Durch das Vermeiden einer Kommunikation mit dem Webserver, wird eine schnellere Reaktion ermöglicht
- Das klingt einfach, aber Benutzer navigierten mit
  - Standardfunktionen des Browsers (vorwärts/rückwärts)
  - Auswahl eines Lesezeichens
  - oder einfach mit dem erneuten Laden einer Seite
- Konsequenz, der aktuelle Bearbeitungsstand geht ohne weitere Maßnahmen verloren
  - URLs (& Cookies) sind die einzigen Informationen, welche erhalten bleiben

## Anforderung an Single Page Applications

- Ausgangspunkt: Server stellt Seiten bereit
  - Jede Seite hat ihre eigene URL und die Anwendung wechselt zwischen ihnen mit Unterstützung des Servers
- Erste JavaScript Anwendungen: Ein Seite mit zugehörigem Skript
  - Problem: Ein Seitenwechsel bedingt den Neustart der Anwendung und es gehen alle Informationen verloren
  - Teillösung: Warnhinweis an den Benutzer!  

```
window.onbeforeunload=function(e){ return 'Gleich ...'; }
```
- **Anforderungen** von Benutzern sind aber eher:
  - Navigation mit der Menüleiste des Browsers
  - Hin- und zurück wechseln zur Anwendung
  - Lesezeichen in die Anwendung
  - Austausch von URLs, welche auf einen bestimmten Anwendungszustand verweisen
  - Neuladen der Seite ohne Datenverlust

## *Deep Linking* und SPAs

- URL einer Webanwendung soll deren Kontext umfassen – so dass der Browser ihn mit Hilfe der URL wiederherstellen kann
  - Vorteilhaft für Lesezeichen und das Teilen von Links
- Wodurch ergibt sich der Kontext? Er wird durch den Anwendungsentwickler festgelegt!
  - Bsp.: Seite enthält Inhaltselemente und ein Fenster zum Editieren
  - Wohin soll der Link verweisen?
  - Macht es einen Unterschied, ob ich den Link speichere oder weiterleite?
  - Wie sieht es mit der Navigation aus?
  - Was passiert beim erneuten Laden der Seite?

## *Deep Linking* für SPAs

Im Prinzip sind zwei Ansätze vorstellbar:

- Zustand in der URL der Anwendung verwalten
  - Lösung zur Navigation und für das erneute Laden einer Seite
  - Benutzer können Link aus der Navigationsleiste nützen
- Es wird ein Funktion zum Erzeugen eines Deep Links bereitgestellt
  - Benutzer können bei Bedarf einen Deep Link erstellen
  - Link in der Navigationsleiste bleibt formschön

In jedem Fall muss die Anwendung in der Lage sein, ihre URLs zu interpretieren und ihren Zustand wiederherzustellen

## Navigation innerhalb einer Anwendung

- `window.location` ermöglicht Zugriff und Manipulation der aktuellen URL
  - Elemente: `protocol`, `hostname`, `pathname`, `search` und `hash`
  - `pathname` indiziert was angezeigt werden soll
  - `search` und `hash` geben zusätzliche Informationen wieder

## Router

- SPAs und zugehörige Frameworks verfügen typischerweise über eine Routingkomponente
  - Diese überprüft auf das Vorkommen einer Route in der URL bzw. typischerweise des `pathname`
  - Gibt es eine Passung, wird eine hierfür registrierte Funktion ausgelöst
  - Entspricht bspw. dem *Observer*-Entwurfsmuster

## Navigation innerhalb einer Anwendung

- Wenn ein Link ausgelöst wird, so übernimmt normalerweise der Browser die Navigation
- Dies kann jedoch durch `event.preventDefault()` unterdrückt werden – und eine Behandlung kann innerhalb der Anwendung erfolgen

## Wie erfolgt die Aufzeichnung eines Verlaufs im Browser?

- Es gibt eine aufgezeichnete Historie, in der vorwärts und rückwärts navigiert werden kann
- Was passiert, wenn man zurückgeht und anders abbiegt? ...oder eine Seite neu lädt?

## Verwendung der history API

- Idee: Anstatt für jede neue URL eine document-Instanz zu erzeugen, wird die aktuelle Instanz wiederverwendet & aktualisiert
- Hierfür stellt der Browser über window die history API bereit
- Kernfunktionen sind
  - go() – Navigation innerhalb des Browserverlaufs
    - z.B., go(-1) eine Schritt zurück
  - pushState() – Erzeugen eines neuen Eintrags im Verlauf ohne Laden einer URL
  - replaceState() – Aktualisieren eines Eintrags (innerhalb der Domäne) ohne erneutes Laden

## Lösung: Verwendung der `history` API

- Parameter von `pushState()` und `replaceState()`
  - `state` - serialisierbares JavaScript-Objekt, das gespeichert wird
    - Problematisch, wenn eine URL direkt zur Navigation verwendet wird, da dann der Zustand nicht vorhanden ist
    - 16 MiB für Firefox
  - `title` - wird noch nicht ausgewertet
  - `path` - URL, absoluter oder relativer Pfad mit der Beschränkung des gleichen Protokolls und Hosts
- Auswirkung von `pushState()` auf den Verlauf: Neuer Eintrag im Verlauf
- Auswirkung von `replaceState()`: Aktueller Eintrag wird verändert.

## Navigation mittels der `history` API

- Man unterbindet das native Verhalten bezüglich der Links der Anwendung
  - Siehe `event.preventDefault()`
- Entsprechende Behandlungsroutinen benutzen dann `pushState()` und `replaceState()`
- Weiterhin muss die entsprechende Routing-Komponente informiert werden
- Es gibt aber Ausnahmen beispielsweise *clicks* mit gedrückten Funktionstasten
- Abfangen der direkten Browsernavigation um die Routerkomponente zu informieren
  - Entsprechende Navigationsereignisse erzeugen ein `popstate`-Ereignis, für welches man sich registrieren kann

## Zusammenfassung

- Normale Navigationsfunktionen müssen ersetzt bzw. umprogrammiert werden
- Notwendige Änderungen sind in aktuellen Frameworks implementiert und man muss sich kaum selber kümmern
- Single Page Applications bringen natürlich noch weitere Herausforderungen mit sich ...