# Webframeworks – Überblick Immer mehr Desktopanwendungen wandern ins Netz. In Form von Web-Applikationen stehen sie Nutzern über den Browser zur Verfügung. Klassische Bespiele sind Webmailer oder Browsergames. Auch im Unternehmenskontext hat sich Software-as-a-Service als Lizenzmodell etabliert. Es gibt **Web-Anwendungen** fürs Customer-Relationship-Management, für Warenwirtschaft, Newsletter, Projektplanung, Mitarbeiter-Zeiterfassung oder Finanzbuchhaltung. Gerade kleine Unternehmen profitieren dabei von bedarfsorientierten Abrechnungsmodellen und einer zentralen Bereitstellung über das Internet. Diese reduziert die Kosten für Wartung und Administration und bietet durch Plattform- und Geräteunabhängigkeit ein Maximum an Flexibilität.

Bei der Entwicklung von Web-Anwendungen greifen Programmierer in der Regel auf sogenannte **Webframeworks** zurück. Aber was ist ein Framework? Und was macht ein Framework speziell für Web-Anwendungen aus?

## Was ist ein Webframework?

Unter einem Framework (deutsch: *Rahmenstruktur*) versteht man ein Programmgerüst, das als Grundlage bei der Software-Entwicklung zum Einsatz kommt. Möchte ein Entwickler eine neue Software schreiben, wäre es ineffizient, jedes Mal bei null zu beginnen. Für zahlreiche Standardfunktionen in der Software-Entwicklung existieren bewährte Lösungsvorschläge in Form von Programm-Codes. In der objektorientierten Programmierung kommen dafür meist Klassen zum Einsatz, die als Baupläne zur Entwicklung von Software-Objekten herangezogen werden. Ein Framework stellt meist eine Sammlung verschiedener kooperierender Klassen dar und bestimmt somit die Designgrundstruktur jeder Software, die auf Basis des Frameworks entwickelt wird. Dient ein Framework als Grundgerüst für Web-Anwendungen, spricht man von einem **Web-Application-Framework** (kurz: Webframework).

Als Grundlage für die Software-Entwicklung sollten Webframeworks einfache, klare Strukturen zur Verfügung stellen, die sich leicht warten lassen und Entwicklern ermöglichen, in kurzer Zeit lauffähige Webanwendungen zu erstellen. Um diesen Anspruch zu erfüllen, berufen sich viele Webframeworks auf drei grundlegende Design-Prinzipien:

* **Don’t repeat yourself (DRY):** Der Leitsatz *Don’t repeat yourself* zielt auf die Vermeidung von Redundanz im Rahmen der Softwareentwicklung ab. Redundante Informationen wie Code-Duplikate sind prinzipiell vermeidbar und wirken sich negativ auf die Wartung der Software aus: Sollen redundante Codeabschnitte angepasst werden, müssen Änderungen an verschiedenen Stellen im Programmcode vorgenommen werden.
* **Keep it short and simple (KISS):** Ein Problem, eine Lösung. Das ist das Grundprinzip des KISS-Paradigmas. Dieses orientiert sich am Prinzip der Parsimonie (*Sparsamkeitsprinzip*): Finden sich für einen Sachverhalt mehrere Lösungsansätze, ist die Lösung zu wählen, die mit den wenigsten Annahmen und Variablen auskommt. Um ein Framework zu verwenden oder ein spezifisches Problem zu beheben, sollte möglichst wenig Code nötig sein.
* **Convention over Configuration:** Je mehr Konfiguration, desto aufwändiger die Handhabung. Konventionen hingegen bietet eine Möglichkeit, die Komplexität zu reduzieren. Webframeworks sollten daher bewährte Vorgehensweisen in Form von Standardeinstellungen vorgeben und weitere Einstellungsmöglichkeiten optional anbieten

Werden diese Design-Prinzipien berücksichtigt, bietet der Einsatz von Webframeworks in der Software-Entwicklung zahlreiche Vorteile. Die Vorgaben eines Frameworks setzen Entwicklern jedoch auch Grenzen. Diese strukturellen Einschränkungen werden mitunter als Nachteil gesehe.

### Vorteile von Webframeworks

Der Einsatz von Webframeworks ist durch eine **Reduktion des Zeit- und Kostenaufwands** bei der Software-Entwicklung motiviert. Grundidee ist die Wiederverwendung von Codes. Diese bezieht sich vor allem auf Grundfunktionalitäten wie Datenbankverbindungen, Webseiten-Templates, Caching-Verfahren oder Sicherheitsfunktionen, die von Frameworks als vorgefertigte Codebausteine zur Verfügung gestellt werden. Damit beschränkt sich der Entwicklungsaufwand auf den spezifischen Programmcode einer neuen Web-Anwendung. Da die meisten Web-Frameworks als freie Software kostenlos angeboten werden, entstehen in der Regel keine Anschaffungskosten.

Darüber hinaus fördern Frameworks die **Generierung sauberer Quellcodes**, da Entwickler für Standardfunktionen auf bewährte Bausteine zurückgreifen können. Die durch Frameworks zur Verfügung gestellten Programmanteile durchlaufen meist zahlreiche Entwicklungszyklen und werden kontinuierlich optimiert. Dabei fungiert die Community als Tester und Mitentwickler. Sicherheitslücken neuer Komponenten werden bei großen Projekten rasch entdeckt und behoben. Ein Austausch von Anwendern und Entwicklern erfolgt in der Regel in Projekt-Foren, die zum Teil durch **Support-Teams** moderiert werden.

### Nachteile von Webframeworks

Im Netz findet sich eine nahezu unüberschaubare Anzahl von Frameworks für die Web-Entwicklung, die sich sowohl in grundlegenden Gestaltungsprinzipien als auch im Funktionsspektrum unterscheiden. Verschiedene Software-Projekte können somit den Einsatz unterschiedlicher Webframeworks erfordern. Entwickler stehen vor dem Problem, das passende Rahmengerüst für das beabsichtigte Vorhaben auswählen zu müssen. Schnell kommt es dabei zu Kompromissen, bei denen das beabsichtigte Softwareprojekt den **Grenzen des Frameworks** angepasst wird. Da Frameworks als Universallösungen entworfen werden, nutzen Entwickler in den seltensten Fällen alle Funktionen der vorgegebenen Struktur. Je nach Umfang des Frameworks enthält eine Anwendung daher mitunter mehr Code als nötig. Man spricht in diesem Fall von **Codeballast**.

Als nachteilig wird zudem die Abhängigkeit gesehen, in die Entwickler zum jeweiligen Webframework, einem speziellen Hersteller oder zur Entwickler-Community geraten. In Einzelfällen ist der Einsatz eines Frameworks mit bestimmten **Lizenz-Bedingungen** verbunden. Zudem kann es zu Problemen kommen, wenn die Weiterentwicklung des Frameworks eingestellt wird. Da Entwickler sich mit dem Aufbau des jeweiligen Programmgerüsts und den Verwendungsmöglichkeiten vertraut machen müssen, ist zudem eine **Einarbeitungszeit** erforderlich, diese relativiert sich jedoch durch die Zeitersparnis aufgrund vorgefertigter Funktionen und Codebausteine. Hier sehen Kritiker die Gefahr, dass Basiswissens verloren geht. Anwender, die ausschließlich auf Grundlage von Frameworks programmieren, setzen sich möglicherweise weniger intensiv mit den verwendeten Programmiersprachen auseinander.

Da der **Quellcode der meisten Webframeworks frei zugänglich** ist, kann sich prinzipiell jeder mit diesem vertraut machen. Werden Anwendungen für den Unternehmenseinsatz auf Basis öffentlich zugänglicher Codebausteine entwickelt, sind diese für Hacker möglicherweise transparenter als Apps, deren Quellcode nicht offen liegt.

## Aufbau eines Webframeworks

Wie Frameworks im Allgemeinen werden auch Web-Application-Frameworks von Programmbibliotheken (*Libraries*) und Web-Development-Tools abgegrenzt. Während Bibliotheken lediglich einzelne Funktionalitäten zur Verfügung stellen, kann ein Framework als ein interaktives Programmgerüst für Standardabläufe betrachtet werden – quasi eine semi-vollständige Anwendung.

### Statische und flexible Komponenten

Die zentralen Grundkomponenten eines Software-Frameworks stellen kooperierende Klassen mit zugeordneten Methoden dar. Dabei handelt es sich um Code-Blöcke, die die Eigenschaften von Software-Objekten und deren Verhaltensweisen beschreiben. Einige dieser Komponenten sind statisch und somit unveränderlich. Andere lassen sich durch den Anwender anpassen – beispielsweise durch das Überschreiben von Methoden. Dieser hat somit die Möglichkeit, das Programmgerüst an eine spezifische Aufgabenstellung anzupassen. Für die flexiblen Komponenten eines Framewoks hat sich die Bezeichnung „**Hot Spots**“ etabliert. Die statischen Anteile werden „**Frozen Spots**“ genannt.

### Inversion of Control

Webframeworks folgen in der Regel dem Konzept der **Inversion of Control (IoC)** (deutsch: *Umkehr der Steuerung*). Dieses lässt sich als eine Anwendung des sogenannten Hollywood-Prinzips auf die objektorientierten Programmierung beschreiben: *„Don’t call us, we call you!“*.

Vereinfacht ausgedrückt bedeutet IoC, dass die Verantwortlichkeit für die Programmausführung nicht bei den einzelnen Komponenten liegt, die auf Basis des Frameworks entwickelt und ausgeführt werden, sondern beim Framework selbst. Dieses übernimmt somit die Funktion eines Hauptprogramms, das den **Kontrollfluss** koordiniert. Weitere Komponenten, die vom Anwender in das Framework implementiert und damit registriert werden, stehen diesem bei Bedarf zur Verfügung. Vergleichen lässt sich dies mit dem klassischen Ausgang eines Hollywood-Castings: *„Wir wissen jetzt, wer du bist, und melden uns, wenn wir dich brauchen!“*

Um diese zentrale Steuerungsfunktion ausführen zu können, geben Webframeworks eine spezielle Schnittstelle vor, die von allen aufbauenden Komponenten implementiert werden muss. Durch dieses Design-Prinzip ist das Framework jederzeit in der Lage, aufbauende Komponenten dazu zu bewegen, [Callback-Methoden](https://www.ionos.at/digitalguide/websites/web-entwicklung/was-ist-eine-callback-function/) zu implementieren. Diese ermöglichen es dem Framework, bei Bedarf Informationen in die Komponenten zu injizieren oder über Methodenaufrufe ein bestimmtes Verhalten der auszulösen. Während Klassen und deren Abhängigkeiten bei klassischen Ansätzen der Softwareentwicklung hart-codiert und damit bereits bei der Kompilierung festgelegt werden, ermöglicht das Konzept Inversion of Control einer Software, Objekte dynamisch während der Laufzeit zu erzeugen.

IoC hat in der Software-Entwicklung den Status eines allgemeinen Gestaltungskonzepts. Zur Anwendung kommt dieses durch Entwurfsmuster wie **Dependency Injection (DI)** und **Dependency Lookup.**

### Trennung von Datenmodell, Präsentation und Programmsteuerung

Die Mehrheit aller Web-Application-Frameworks basiert auf einem Architekturmuster, das **Model-View-Controller (MVC)** genannt wird. Dieses zielt auf eine strikte Trennung von Anwendungslogik und Präsentationsschicht ab und teilt Software bei der Entwicklung in die drei Bereiche **Model** (Datenmodell), **View** (Präsentation) und **Controller** (Programmsteuerung) ein.

* **Datenmodell:** Das Model ist der Teilbereich eines Frameworks, der die darzustellenden Daten sowie die Anwendungslogik und Regeln enthält. Der Datenabruf und Änderungswünsche werden über dafür vorgesehene Methoden abgewickelt.
* **Präsentation**: Die grundlegende Aufgabe des Views ist die Darstellung der vom Model bereitgestellten Daten. Dazu fragt der View den Zustand des Models über Methoden ab und wird von diesem über Änderungen informiert. Eine weitere Aufgabe des Views ist es, Benutzereingaben (Tastaturanschläge, Mausklicks) entgegenzunehmen und an den Controller weiterzuleiten.
* **Programmsteuerung:** Der Controller fungiert als Schnittstelle zwischen Model und View. Dazu verwaltet er einen oder mehrere Views, wertet Benutzereingaben aus und reagiert entsprechend – zum Beispiel indem Daten an das Model weitergegeben oder Änderungen am View veranlasst werden.

Ziel des MVC-Architekturmusters ist ein flexibler Programmentwurf. Die Trennung von Anwendungslogik und Präsentationsschicht soll spätere Änderungen, Erweiterungen sowie die Wiederverwendung einzelner Komponenten erleichtern.

Klassifizierung von Webframeworks

Der Markt für Web-Anwendungen ist vielfällig. Apps, die Nutzern über den Browser zur Verfügung stehen, unterscheiden sich je nach Einsatzbereich und Funktionsspektrum nicht nur in Größe und Erscheinungsbild, sondern auch grundlegend im Softwaredesign. Grund dafür ist die Vielfalt verfügbarer Web-Frameworks, die sich auf unterschiedliche Technologien stützen und **verschiedene Ansätze im Software-Design** verfolgen. Gegenüberstellen lassen sich Single- und Multi-Page-Ansätze, serverzentrische und clientzentrische Ansätze sowie Webframeworks, die aktionsbasiert oder komponentenbasiert arbeiten.

### Single-Page- und Multi-Page-Ansätze

Multi-Page-Anwendungen bestehen aus mehreren HTML-Seiten, die in der Regel durch die Eingabe des jeweiligen URL im Browser geöffnet werden und durch Hyperlinks miteinander verbunden sind. Die Benutzeroberfläche einer Single-Page-Anwendung hingegen besteht lediglich aus einer HTML-Seite, auf der sämtliche Benutzereingaben erfolgen. Diese kann durch Panels, Reiter oder Registrierkarten gegliedert sein. Der URL einer Single-Page-Anwendung ändert sich im Rahmen der Navigation nicht.

### Serverzentrische und clientzentrische Webframeworks

Das Programmiermodell klassischer Web-Anwendung entspricht dem des World Wide Web, dessen Architektur durch das **Hypertext Transfer Protocol (HTTP)** geprägt ist. Wird eine Web-Anwendung von einem Nutzer aufgerufen, sind in jedem Fall ein oder mehrere Server sowie ein Clientprogramm, in der Regel ein Webbrowser, involviert. Je nachdem, wie die Kommunikation zwischen Server und Client gestaltet ist, spricht man von serverzentrierten oder clientzentrierten Anwendungen.

* **Clientzentrisch:** Wird beim Start einer Anwendung die gesamte HTML-Benutzeroberfläche inklusive Anwendungslogik in den Client geladen, spricht man von clientzentrischen Anwendungen. Änderungen an der Oberfläche aufgrund von Benutzereingaben werden in diesem Fall durch clientseitige Programmiersprachen wie JavaScript umgesetzt. Ein solcher Design-Ansatz empfiehlt sich für Apps, bei denen Anwender über einen längeren Zeitraum hinweg in derselben Ansicht arbeiten, da lediglich die auf der Benutzeroberfläche dargestellten Daten vom Server nachgeladen werden. Der clientseitige Ansatz kommt in erster Linie bei der Entwicklung von Single-Page-Apps zum Einsatz und wird von JavaScript-Frameworks wie [*AngularJS*](https://angularjs.org/) oder [*EmberJS*](http://emberjs.com/) verfolgt.
* **Serverzentrisch**: Bei serverzentrischen Anwendungen verbleibt die Anwendungslogik auf dem Server, der die Benutzeroberfläche erstellt und an Clients zur Darstellung ausliefert. Für Änderungen an der Benutzeroberfläche stehen somit serverseitige Programmiersprachen zur Verfügung und die Entwicklung ist von clientseitigen Unsicherheiten weitgehend unabhängig. Dieser Ansatz kommt in der Regel bei Multi-Page-Apps zum Einsatz, bei denen die verschiedenen Seitenansichten bei Bedarf vom Server abgerufen werden. Ein solches Software-Design ist zwar mit längeren Ladezeiten verbunden, reduziert aber den Rechenaufwand auf dem Clientgerät. Bei manchen Apps wird eine Auslagerung der Steuerungslogik aus Sicherheitsgründen vermieden. Eine Umsetzung des serverzentrischen Ansatzes findet sich beispielweise bei den Frameworks *Django*, *Zend* und *Ruby on Rails*.

Ein serverzentrischer Programmieransatz findet sich vor allem bei Frameworks, die **zur Erstellung klassischer Webanwendungen** mit Multi-Page-Struktur und klassischer HTML-Oberfläche entwickelt wurden. Lediglich die Oberfläche wird in der Webanwendung dargestellt und nutzt in der Regel die Browser-Navigation. Web-Apps dieser Art lassen sich unabhängig von verwendeten Betriebssystem oder Webbrowser ausführen. Die Steuerungslogik wird über die HTTP-typische Request-/Response-Kommunikation vom Server verarbeitet.

Clientzentrische Web-Anwendungen werden mitunter auch als Rich Clients oder Rich Internet Applications (RIA) bezeichnet. Anwendungen dieser Art implementieren die Benutzeroberfläche inklusive Anwendungslogik im Client. Diese wird beim Start in der Regel komplett geladen. Anders als bei klassischen Web-Anwendungen lassen sich mit einem clientzentrischen Ansatz auch Eigenschaften wie Drag&Drop-Steuerung, Offlineverfügbarkeit und Festplattenzugriff umsetzen, die eher von Desktop-Anwendungen bekannt sind.

### Aktionsbasierte vs. komponentenbasierte Webframeworks

Auf der Ebene der **Anwendungssteuerung** lassen sich Webframeworks in zwei Klassen einteilen. Während aktionsbasierte (*action-based*) Webframeworks das HTTP-bedingte Request-Response-Muster konsequent abbilden, wird dieses bei komponentenbasierten (*component-based*) Webframeworks abstrahiert.

**Aktionsbasierte Webframeworks:** Bei aktionsbasierten Webframeworks dient der Controller als eine zentrale Instanz, die Clientanfragen entgegennimmt, validiert und eine dazu passende Aktion aufruft. Für jede mögliche Aktion muss vom App-Entwickler im Vorfeld ein Software-Objekt erstellt werden, das die entsprechende Anwendungslogik enthält. Dieses lässt sich in der Regel von abstrakten Klassen ableiten. Wurde die Aktion abgearbeitet, aktualisiert der Controller das Datenmodell und leitet das Ergebnis an den View weiter, der seinerseits die Antwort erstellt und an den Client zurückschickt.

Aktionsbasierte Webframeworks lehnen sich stark an das MVC-Muster an und werden aufgrund der strikten Umsetzung des Request-Response-Schemas auch als request-based bezeichnet. Klassische Vertreter sind:

* **Django**
* **Ruby on Rails**
* **Symfony**
* **Spring MVC**
* **CodeIgniter**
* **CakePHP**

Da die möglichen Aktionen eines aktionsbasierten Webframeworks vom App-Entwickler im Detail selbst definiert werden, spricht man von einem White-Box-Ansatz. Dieser bietet Entwicklern große Freiheiten, erfordert jedoch ein tieferes Verständnis des jeweiligen Webframeworks, da Entwickler für die Erstellung von HTML, CSS und JavaScript selbst verantwortlich sind.

**Komponentenbasierte Webframeworks:** Anders als beim aktionsgesteuerten Ansatz abstrahieren komponentengesteuerte Webframeworks vom HTTP-bedingten Request-Response-Muster, indem die Benutzeroberfläche einer Web-Anwendung als Sammlung von Komponenten betrachtet wird. Für jede dieser Komponenten, die serverseitig mit Software-Objekten verknüpft sind, werden während der Entwicklung der Webanwendung bestimmte Reaktionen definiert. Diese folgen auf Ereignisse, die durch eine Benutzerinteraktion mit der Komponente ausgelöst werden. Man spricht daher auch von ereignisgesteuerten Webframewoks. Klassische Vertreter sind:

* [**Lift**](http://www.liftweb.net/)
* [**Apache Tapestry**](http://tapestry.apache.org/)
* [**JavaServer Faces**](http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html)
* [**Apache Wicket**](http://wicket.apache.org/)

Die Grundidee hinter dem komponentenbasierten Ansatz ist, verwandte Aktionen zu gruppieren. Eine Komponente *AccountController* repräsentiert beispielsweise Aktionen wie *login*, *logout* oder *getAccount*. Ein Software-Objekt kann somit für mehrere Aktionen zuständig sein. Komponentenbasierte Webframeworks bieten in der Regel eine große Auswahl wiederverwendbarer Komponenten, die die Details des zugrundeliegenden Request-Response-Schemas vor dem App-Entwickler verbergen. Man spricht in diesem Zusammenhang von einer Black-Box. Webframeworks dieser Art bieten sich somit für Entwickler an, die sich in erster Linie auf vordefinierte Komponenten stützen möchten. Wer sich mehr Freiheiten in Bezug auf HTTP, HTML, CSS und JavaScripts wünscht, ist bei einem aktionsbasierten Webframework besser aufgehoben.

## Auswahl eines Webframeworks

Da Frameworks einen großen Einfluss auf Funktionen und Gestaltungsmöglichkeiten von Web-Anwendung und den Workflow während der Entwicklung haben, beginnt der Arbeitsprozess in der Regel bereits bei der Auswahl des geeigneten Programmgerüsts. Dabei sollten zum einen das geplante Software-Vorhaben als auch bereits erworbene Vorkenntnisse berücksichtigt werden.

Zentrale Fragen beziehen sich auf die Art der Anwendung und die gewünschte Architektur (serverzentrisch, clientzentrisch). Beide Aspekte haben direkte Auswirkungen auf Bedienung und Nutzerfreundlichkeit einer Web-App haben.

Auch bereits vorhandene Sprachkenntnisse und die Verfügbarkeit der benötigten Infrastruktur stellen einen Ausgangspunkt für die Suche nach dem geeigneten Webframework dar. Insbesondere für weitverbreitete Programmiersprachen wie PHP (z.B. Zend, *Symfony, CakePHP oder CodeIgniter*), Java (z.B. *JavaServer Faces* oder *Apache Wicket*) oder Python (z.B. *Django*) gibt es eine große Auswahl gut dokumentierter Webframeworks. Die Popularität von Ruby in der Web-Entwicklung ist vor allem auf das beliebte Webframework Ruby on Rails zurückzuführen. Clientzentrische Webframeworks stützen sich in der Regel auf die Skriptsprache JavaScript.