infoteam
software

WHITEPAPER

Web-Technologien

Web-Technologien

1 Einleitung

Im beruflichen wie auch im privaten Alltag ist für die meisten Internetnutzer der Browser das Werkzeug, um Zugriff auf die Inhalte des World Wide Web zu erhalten. Was jedoch nur die wenigsten wissen: Er ist nicht beschränkt auf das Anzeigen klassischer Webseiten. Vielmehr kann er auch als Laufzeitumgebung für komplexe Web-Applikationen (Web-Apps) dienen, die vielfach auf Augenhöhe mit klassischen Desktop-Applikationen sind. Je nach Anforderung bieten Web-Apps sogar zahlreiche Vorteile gegenüber den klassischen Anwendungen.

Die Evolution von Web-Technologien und Web-Apps ist sehr schnelllebig und unübersichtlich. Dadurch steigt die Gefahr, trendgetrieben Technologien einzusetzen, die noch nicht vollends ausgereift sind. Die nachfolgenden Kapitel sollen deshalb allgemein die Vor- und Nachteile von Web-Apps aufzeigen und einen Einblick in die Entstehung und Entwicklung der Web-Technologien geben. Denn erst dieses Hintergrundwissen ermöglicht eine fundierte Entscheidungsfindung, welche Technologie die richtige für das angestrebte Ziel ist.

2 Warum Web-Apps?

Web-Apps sind Applikationen, die nicht wie klassische Programme installiert werden, sondern die innerhalb eines Browsers lauffähig sind. Im Gegensatz zu gewöhnlichen Internetseiten imitieren Web-Apps das Verhalten von klassischen Applikationen hinsichtlich Benutzeroberfläche und Bedienung. Dass die Web-Apps auf Web-Technologien basieren, bleibt dem Benutzer verborgen.

Welche Vor- und Nachteile bieten Web-Apps nun gegenüber klassischen Applikationen? Welche Faktoren beeinflussen die Entscheidung, ob eine Software als klassisches Programm oder als Web-App umgesetzt werden soll?

Die Vorteile von Web-Apps sind mannigfaltig:

- Web-Apps müssen nicht installiert werden: Um als Endbenutzer Web-Apps auszuführen, wird lediglich ein Browser (z. B. Internet Explorer, Microsoft Edge, Mozilla Firefox, Google Chrome, Opera, Safari etc.) benötigt, der üblicherweise zur Standardinstallation eines Systems gehört. Durch Frameworks wie Apache Cordova (siehe Abb. 1) ist es zudem möglich, den Browser in eine klassische Applikation einzubetten. Somit lassen sich Web-Apps auch in installierbare, klassische Applikationen „konvertieren", die im Play Store oder App Store angeboten werden. Durch den Kiosk-Modus können sie auch in dedizierten Embedded-Produkten eingesetzt werden.
- Web-Apps lassen sich einfach aktualisieren: Aufwendige Update-Prozeduren entfallen mit Web-Apps, denn beim Start wird automatisch die neueste Version geladen, ohne dass es der Nutzer bemerkt.
- Web-Apps sind betriebssystem- und geräteunabhängig: Da Web-Apps im Browser laufen und Browser für nahezu alle Betriebssysteme zur Verfügung stehen, können mit einer als Web-App implementierten Lösung alle Zielgeräte (z. B. Desktop-PC, Tablet, Handy etc.) auf einmal angesprochen werden.
- Web-Apps sind zukunftssicher: Web-Technologien existieren seit über 20 Jahren. Webseiten und Web-Apps von damals können auch heute noch in aktuellen Browsern geöffnet werden. Der Aufwand, 20 Jahre alte klassische Applikationen, beispielsweise basierend auf Windows 95, auf einem heute aktuellen Betriebssystem zu öffnen, ist dagegen ungleich höher.

Andererseits gibt es auch Situationen, für die klassische Applikationen besser geeignet sind als Web-Apps:

- Web-Apps haben nur beschränkt Zugriff auf spezifische Funktionen des Betriebssystems. So ist es beispielsweise nicht möglich, die Statusleiste in Windows zu manipulieren.

Web-Apps haben kein natives Look & Feel. Anstatt betriebssystemeigener GUI-Bausteine kommen HTML-Elemente zum Einsatz, was dazu führen kann, dass der Anwender die Bedienung von Web-Apps als „ungewohnt" empfindet.

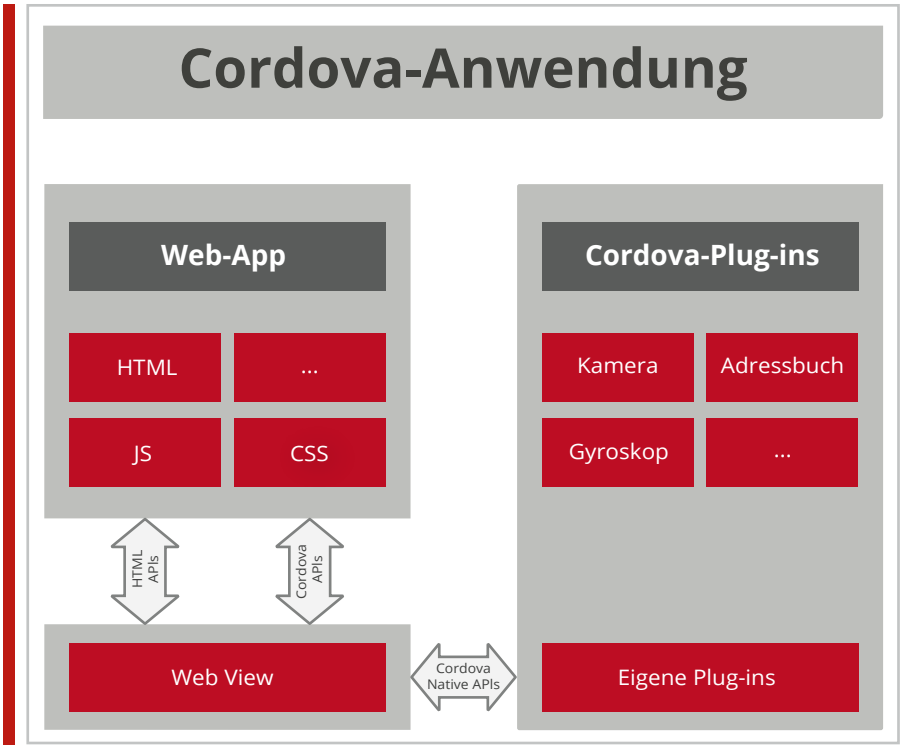

Abbildung 1:
Über das Cordova-Plug-in greifen Web-Apps auf Bereiche im Betriebssystem zu, die ursprünglich nicht zur Verfügung stehen.
Dazu zählen beispielsweise Adressbücher oder Kamerasysteme.

3 Herausforderungen bei Web-Apps

In den Anfangszeiten des Internets wurden Webseiten hauptsächlich zum Betrachten von Inhalten erstellt. Der Browser konnte die Webseiten ansteuern und den Inhalt anzeigen, wofür eine Client-Server-Architektur verwendet wurde (siehe Abb. 2):

Der Browser auf dem Computer des Nutzers (Client) stellt eine Anfrage an einen Server, die gewünschten Inhalte bereitzustellen. Der Server übermittelt diese Inhalte zurück an den Client, dessen Browser sie darstellt. Kennzeichnend für die Anfangszeit war, dass die Inhalte zumeist statisch auf dem Server hinterlegt waren, sich also nicht änderten. Die Aufbereitung der Inhalte war sehr schlicht und es bestanden kaum Möglichkeiten zur Interaktion zwischen Nutzer und Website.

Mit den Jahren wurden die Inhalte der Webseiten immer umfangreicher und aufwendiger aufbereitet. Die Inhalte lagen nicht mehr nur statisch auf Ser-

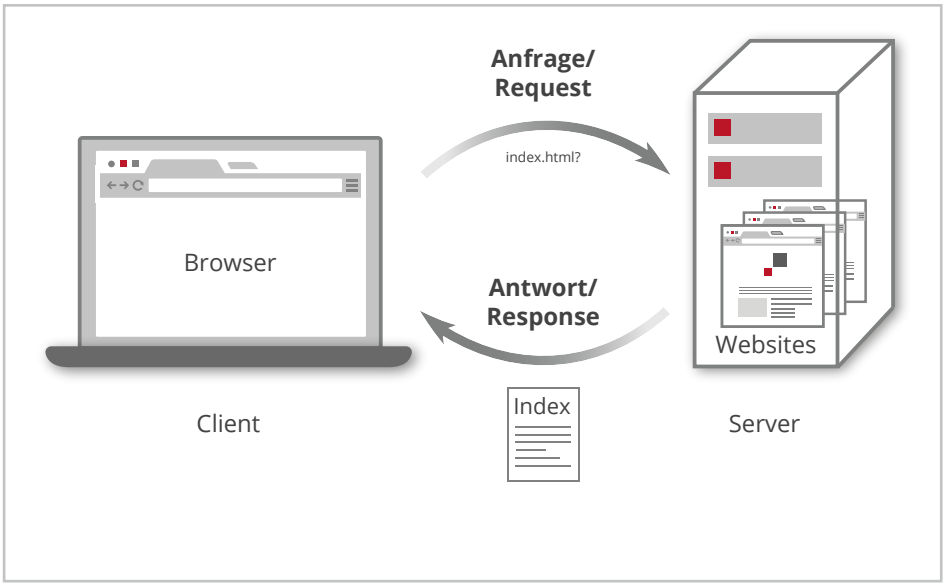

Abbildung 2:
Datenaustausch im Rahmen einer Client-Server-Architektur

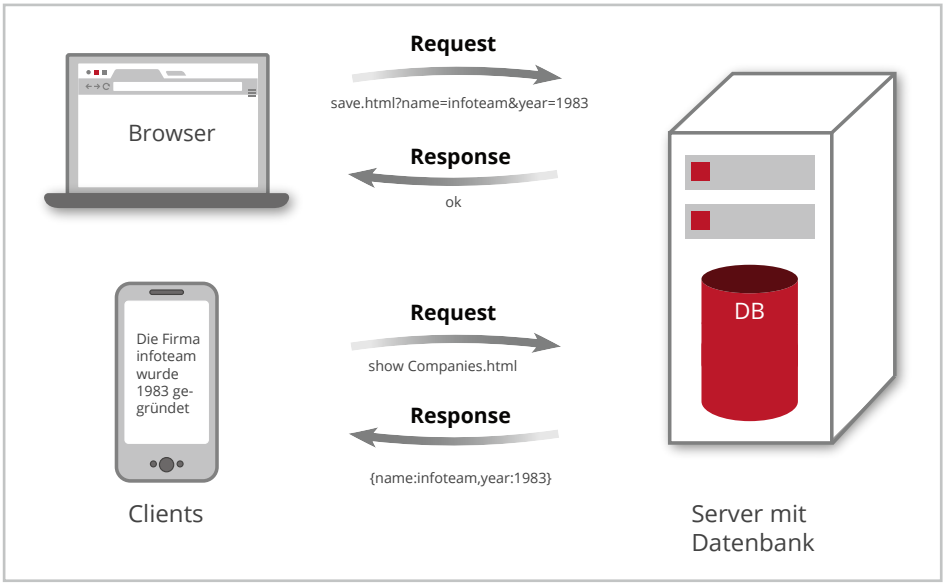

Abbildung 3:
Mit dem Web 2.0 stiegen die Anforderungen und die Komplexität in der Software-Architektur.

vern, sondern konnten von den Nutzern selbst erstellt und auf den Webservern abgelegt werden. Dadurch stiegen nicht nur die Anforderungen an die Server, die nun neue Inhalte abspeichern und wieder bereitstellen mussten (siehe Abb. 3). Auch die Interaktionsmöglichkeiten mit Webseiten wurden komplexer, da der Benutzer nicht mehr nur Inhalte konsumierte, sondern auch generierte.

3.1 Einschränkungen durch das Client-Server-Modell

Bemerkenswert ist, dass trotz aller technologischen Weiterentwicklungen das grundsätzliche Client-Server-Modell unverändert geblieben ist. Für das Entwickeln von Web-Apps, die im Grunde genommen nur sehr komplexe Webseiten sind, entstehen dadurch systembedingt einige Herausforderungen:

- Web-Apps benötigen immer einen Server. Ein reines Ausführen im Browser ohne Server ist nicht ohne Weiteres möglich, selbst wenn der eigentliche Funktionsumfang der Web-App ohne Server bereitstehen könnte. Es müssen daher besondere Maßnahmen getroffen werden, um eine Web-App offline-fähig zu machen (beispielsweise für den Fall, dass keine Online-Verbindung besteht).

Im LocalStorage lassen sich mittlerweile die meisten für den Offline-Betrieb nötigen Daten speichern. Dafür muss nur einmal eine Verbindung zum Server hergestellt werden, um die Web-App und den Inhalt für den LocalStorage zu laden. Dies ermöglicht den Einsatz von Web-Apps auch dort, wo nicht ständig eine Online-Verbindung zur Verfügung steht. Mit Frameworks wie zum Beispiel Electron lassen sich zudem auch Applikationen mit Web-Technologien wie HTML, JavaScript und CSS entwickeln, die auf keinen externen Server angewiesen sind und sich daher noch mehr wie klassische Applikationen verhalten.

- Im Client-Server-Modell kann ein Server nur auf Anfragen des Clients antworten. Von sich aus kann er keine Nachrichten direkt an den Client senden. Geht beispielsweise eine neue Nachricht ein, kann der Server den Client nicht direkt informieren, sondern der Client muss aktiv nachfragen. Der Nachrichtenweg ist also unidirektional.

  Techniken wie das sogenannte „long polling“ umgehen diese Einschränkung: Der Client sendet eine befristete Anfrage an den Server, der jedoch nicht sofort antwortet, sondern erst dann, wenn eine neue Nachricht eingegangen ist. Sobald die Frist abgelaufen ist, ohne dass der Client eine Antwort erhalten hat, sendet er eine neue Anfrage. Bei Daten geringer Größe wird diese Technik immer noch erfolgreich eingesetzt.

Erst eine bidirektionale Kommunikation in Echtzeit und mit großen Datenmengen ermöglicht das auf TCP basierende Netzwerkprotokoll WebSockets. Dieses Protokoll erlaubt es, einfach, performant und ohne gesonderte Clientanfrage Nachrichten vom Server zum Client zu versenden. Dadurch können beispielsweise komplexe Berechnungen auf einem zentralen Server ausgeführt und im Anschluss sehr viele Nutzer über die Web-App mit den neuesten Ergebnissen versorgt werden. Dies ermöglicht auch die Nutzung leistungsintensiver 3D-Technologien oder das Auswerten komplexer Data-Mining-Algorithmen auf mobilen Endgeräten direkt in der Fertigungshalle (siehe Abb. 4).

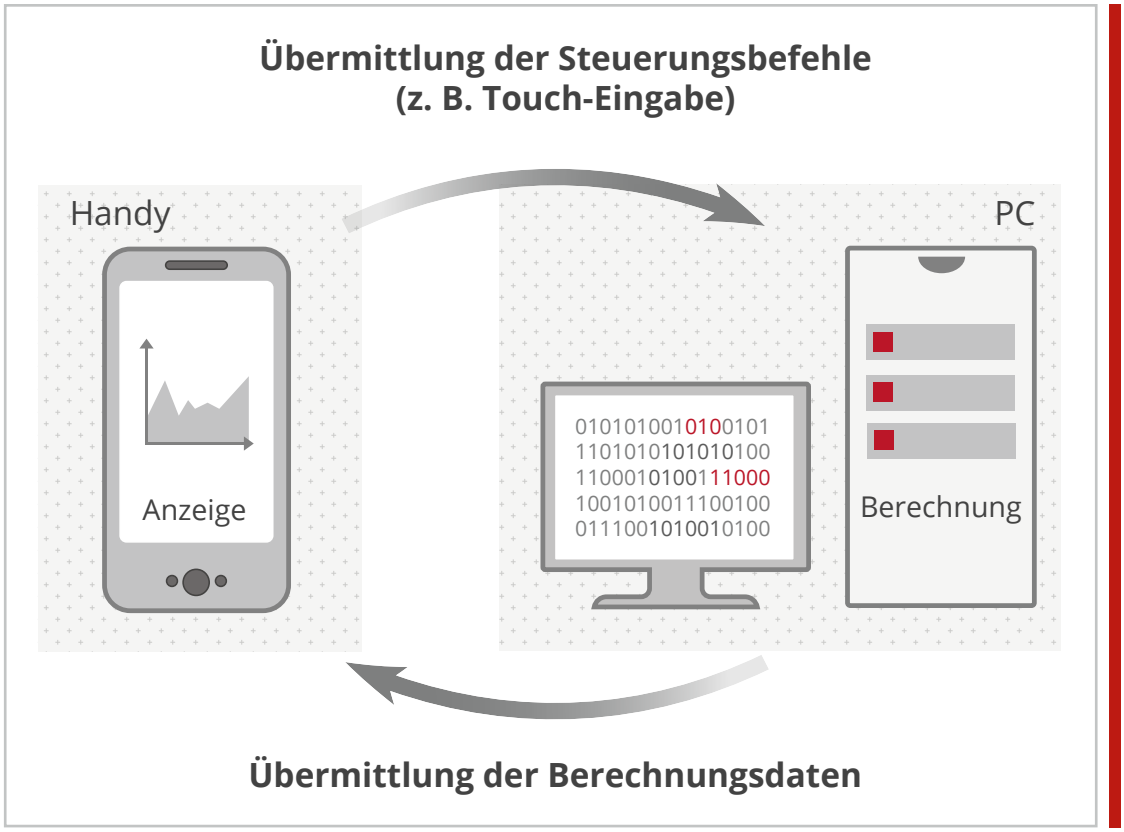

Abbildung 4:
Das Netzwerkprotokoll WebSockets ermöglicht die schnelle Verbreitung von Informationen. Auch bei großen Datenmengen.

3.2 Der Browser als Laufzeitumgebung

Wie bereits angedeutet, diente der Browser ursprünglich zur Darstellung einfacher und statischer Inhalte. Mittlerweile hat er sich zu einer kompletten, leistungsfähigen Laufzeitumgebung mit interaktiver Oberfläche für Web-Apps entwickelt. Trotzdem sind im Hinblick auf die grundsätzlichen Funktionsweisen der Browser verschiedene Aspekte bei der Entwicklung von Web-Apps zu berücksichtigen:

- Klassische Applikationen arbeiten in der Regel mit mehreren Threads (Ausführungssträngen). So steht ein Thread für komplexe Berechnungen zur Verfügung, während ein anderer auf Benutzereingaben wartet und entsprechend reagiert. Würden beide Aufgaben zusammen in nur einem Thread laufen, so bestünde die Gefahr, dass komplexe Berechnungen die Reaktion auf Benutzereingaben verzögern. Browser bieten einer Web-App bisher nur einen Thread an, sodass eine Aufteilung der Aufgaben auf mehrere Threads nicht ohne Weiteres möglich ist. Über Web Worker können rechenintensive Aufgaben im Hintergrund ausgeführt werden, was zeitaufwendige und komplexe Berechnungen ohne verlängerte Reaktionszeiten der Benutzeroberfläche zulässt.

Dies ermöglicht beispielsweise Motion Tracking im Browser: Eine Bewegungserkennung wird auf einem Kamera-Stream durchgeführt, um das Kamerabild mit den berechneten Daten live anzureichern. Die aufwendigen Motion-Tracking-Algorithmen laufen via Web Worker im Hintergrund ab, während die Benutzeroberfläche der Web-App im Browser so flüssig wie bei einer klassischen Applikation reagiert.

- Das Anzeigen komplexer 3D-Grafiken und Animationen ist eine weitere Herausforderung für Web-Apps. Für die Darstellung komplexer Visualisierungen wird in der Regel der Zugriff auf die Hardwarebeschleunigung des Systems benötigt. Dieser Zugriff war über den Browser lange Zeit nicht direkt möglich, weshalb klassische Applikationen gegenüber Web-Apps insoweit im Vorteil waren. Mit der Entwicklung und Unterstützung von WebGL ziehen die Web-Apps mit den klassischen Applikationen gleich, denn WebGL erlaubt nun auch im Browser den Zugriff auf die Hardwarebeschleunigung des Systems. Das Erstellen aufwendiger 3D-Grafiken ist somit kein Problem mehr und wird von allen gängigen Browsern unterstützt.

3.3 Programmiersprachen und JavaScript-Frameworks

Im Vergleich zu den Ursprüngen haben sich die heutigen Programmier- und Beschreibungssprachen für Web-Apps auf Client-Seite (HTML5, CSS3, JavaScript 6) stark weiterentwickelt. Dennoch sind bei der Entwicklung von Web-Apps historisch bedingte Eigenheiten zu beachten.

JavaScript, ursprünglich dafür entwickelt, gewöhnliche HTML-Elemente dynamisch zu verändern und somit mehr Interaktivität in die Anzeige von Webseiten zu bringen, ist mittlerweile die Standardprogrammiersprache für die Client-Entwicklung. Sie ist über Jahre hinweg zu einer sehr mächtigen Skriptsprache gewachsen, ist in der Bedienung durch ihre komplexen Strukturen und Operationen auf niedriger Abstraktionsebene aber wenig intuitiv. Große und leicht zu wartende Web-Apps lassen sich so kaum wirtschaftlich realisieren. Aus diesem Grund greifen Entwickler auf JavaScript-Frameworks zurück, die sowohl kommerziell als auch nichtkommerziell verfügbar sind.

Googles Angular 2 ist ein solches frei verfügbares JavaScript-Framework. Es ist besonders für das Erstellen sogenannter Single-Page-Applications (Web-Apps, die sich wie klassische Applikationen verhalten) ausgelegt und unterstützt den Entwickler dabei, die Web-App in einzelne leicht zu wartende Module zu unterteilen. Angular 2 wird häufig im Verbund mit TypeScript verwendet, einer eigenständigen Programmiersprache, die das objektorientierte Entwickeln mit JavaScript vereinfacht. TypeScript wird in JavaScript transkompiliert, sodass mit TypeScript geschriebene Web-Apps direkt im Browser lauffähig sind. Wegen des starken Fokus auf Modularisierung lassen sich mit Angular 2 schnell komplexe und gut strukturierte Web-Apps erstellen. Da Angular 2 von Google unterstützt wird, ist von einer längerfristigen Pflege und Weiterentwicklung des Frameworks auszugehen, sodass darauf basierende Web-Apps zukunftssicher sein dürften.

4 Fazit

Web-Apps werden kontinuierlich wichtiger und haben viele Vorteile gegenüber klassischen Applikationen. Trotz der rapiden Weiterentwicklung von Web-Technologien darf die Wartungsfähigkeit und Beständigkeit zukünftiger Web-Apps bei ihrer Programmierung nicht vernachlässigt werden. Hinzu kommt, dass Web-Apps meistens für unterschiedliche Endgeräte mit unterschiedlichen Leistungsprofilen und Displaygrößen ausgelegt sein müssen. Dementsprechend sind alle daraus resultierenden Anforderungen zu berücksichtigen; Gleiches gilt für Security-Aspekte im Web-Umfeld.

Für die Umsetzung dieses komplexen Zusammenspiels einer Vielzahl von Anforderungen und historisch bedingtem Fachwissen braucht es einen Softwarepartner, der einerseits über langjährige Erfahrung im Umgang mit Web-Technologien verfügt und andererseits neue Entwicklungen aufmerksam verfolgt und in seine Arbeit integriert. Die infoteam Software AG ist deshalb Ihr perfekter Entwicklungspartner für Ihre Softwarelösungen, implementiert als Web-App.

5 Glossar

Browser	Computerprogramm, ursprünglich zur Darstellung von Webseiten
Electron	Open-Source-Framework zur Erstellung nativer Applikationen mit den Web-Technologien JavaScript, HTML und CSS
GUI	Graphical User Interface (Benutzeroberfläche)
HTML	Hypertext Markup Language (textbasierte Auszeichnungssprache zur Strukturierung digitaler Inhalte)
Kiosk-Modus	Spezieller Modus für Computerprogramme wie z. B. Web-Apps, der die Rechte des Benutzers einschränkt. Der Modus kommt beispielsweise bei Informationsterminals zum Einsatz, damit der Nutzer den Browser mit der Web-App nicht verlassen kann.
LocalStorage	Client-seitige Speicherung von Daten u.a. für den Offline-Betrieb von Web-Apps
Nativ	Speziell für ein Betriebssystem entwickelt und dahingehend optimiert (im Gegensatz zur Web-Entwicklung, die betriebssystemunabhängig ist)
TCP	Transmission Control Protocol (Netzwerkprotokoll, das die Art und Weise des Datenaustauschs zwischen Netzwerk komponenten definiert)
Thread	Ausführungsstrang bzw. Ausführungsreihenfolge in der Abarbeitung eines Programms
UI	User Interface (Benutzerschnittstelle)
Web-App	Anwendungsprogramm nach dem Client-Server-Modell (im Gegensatz zur Desktopanwendung)
WebSockets	Auf TCP basierendes Netzwerkprotokoll für eine bidirektionale Verbindung zwischen einer Web-Anwendung und einem Webserver

Über die infoteam Software Gruppe

Die infoteam Software Gruppe realisiert seit fast 40 Jahren spezifische Softwarelösungen für ihre Kunden aus den Märkten Industry, Infrastructure, Life Science und Public Service. Das Kerngeschäft bilden die Teil- oder Gesamtentwicklung von Steuerungs- und Embedded-Software, Middleware und Anwendungssoftware – agil, modern und nach aktuellen Security-Anforderungen. Spezialdisziplinen sind u. a. normativ regulierte Software für den Einsatz in Medizin- und Laborgeräten (IVDR, MDR, FDA, ISO 13485, IEC 62304 etc.) sowie funktional sichere Software bis zur höchsten Sicherheitsstufe (IEC 61508, DIN EN 50128 etc.). Abgerundet wird das Leistungsportfolio durch langjährige Erfahrungen in den Bereichen Datenanalyse, KI und maschinelles Lernen.

Die infoteam Software Gruppe beschäftigt mehr als 300 Mitarbeiter und verfügt über Standorte und Tochtergesellschaften in Deutschland, Tschechien, der Schweiz und China. Stammsitz der Muttergesellschaft infoteam Software AG ist Bubenreuth bei Erlangen.

www.infoteam.de

Kontakt

infoteam Software AG
Am Bauhof 9 | 91088 Bubenreuth | Deutschland
Telefon: +49 9131 78 00-0
Telefax: +49 9131 78 00-50
info@infoteam.de | www.infoteam.de