## Grundlagen

Grundprinzipien des Web Engineering

* Klar definierte Ziele & Anforderungen
* Systematische Entwicklung einer Webanwendung in Phasen
* Sorgfältige Ausgestaltung dieser Phasen
* Kontinuierliche Überwachung des gesamten Entwicklungsprozesses
* Ein Bild, das Text, Screenshot, Kreis, Diagramm enthält.

  Automatisch generierte BeschreibungPlanung der Wartung

Standardisierung

* Internationale Standards sind wichtig für überregionale und large-scale Entwicklungen
* Proprietäre Lösungen sind verlockend weil
  + Verfügbar
  + Gut verbreitet und unterstützt
  + Werden womöglich zu einem Standard

W3C: Weiterentwicklung der technischen Grundlagen und Standards des Webs, Entwicklung von Standards

Webanwendung: „ A web application is a software system based on technologies and standards of the W3C that provides Web specific resources such as content and services through a Web browser”

Kategorien von Webanwendungen

Ein Bild, das Text, Screenshot, Diagramm, parallel enthält.

Automatisch generierte Beschreibung

|  |  |  |
| --- | --- | --- |
| Dokumentenzentrierte Webanwendungen |  |  |
| Vorteile | Nachteile | Beispiel |
| Einfachheit des Systems | Hoher Aufwand  Gefahr von Inkonsistenzen | Statische Homepages  Einfache Firmenpräsentation |

Interaktive Webanwendungen

* Simple Benutzerinteraktion über HTML Formulare
* Verarbeitung der Information via Common Gateway Interface
* Abhängig von Benutzereingaben können Inhalte dynamisch generiert werden

|  |  |  |
| --- | --- | --- |
| Transaktionsbasierte Webanwendungen |  |  |
| Vorteile | Nachteile | Beispiel |
| Effiziente und konsistente Datenverwaltung  Strukturierte Abfragen | Gesteigerte Komplexität | Interaktive Änderung, datenbank-basierte Systeme |

Workflow-basierte Anwendungen

* Abwicklung von Geschäftsprozessen
* Standardisierte Schnittstellen, lose gekoppelte Systeme, Interoperabilität zwischen den Systemen
* Voraussetzung: Strukturierung der zu automatisierenden Prozesse und Vorgänge
* Herausforderungen
  + Hohe Komplexität der einzubindenden Dienste
  + Autonomie der beteiligten Unternehmen
  + Notwendigkeit zur Robustheit und Flexibilität der Geschäftsprozesse

Kollaborative Webanwendungen

* Unstrukturierte, kooperative Umgebungen
* Kommunikation zwischen Beteiligten ist das Grundprinzip
* Bsp.: Wiki-Systeme, Blogs

Portal-basierte Webanwendungen

* Single Point of Access auf verteilte und meist heterogene Informationquellen und Dienste
* Beispiele: Suchdienste, Marktplatzportale, Community Portale

Universelle Webanwendungen

* Allgegenwärtiger Zugriff auf personalisierte Dienste
* Fokus meist auf Human Computer Interface
* Voraussetzung: Real Time Anpassung

Wissensbasierte Webanwendungen

* Semantic Web, Web 3.0
* Maschinelles Lernen, Künstliche Intelligenz

Social Web

* Förderung von sozialen Netzwerkgemeinschaften
* Vorteile
  + Nutzergesteuerte Seiten & Inhalte
  + „Bringing people together“
* Nachteile
  + Social Engineering
  + Dynamisch wachsende Anzahl von Nutzern
  + Große Anzahl unstrukturierter Daten

## Architektur von Web-Anwendungen

Ziele von Softwarearchitekturen

* Divide-and-Conquer: Zerlegung von Anwendungen in einzelne, voneinander abgeschlossene Teile mit spezifischen Eigenschaften
* Komplexitätsreduktion: Struktur einer Anwendung ohne Implementierungsdetails
* Vogelperspektive: Erzeugung eines gemeinsamen Verständnisses zwischen Entwicklern
* Vermittlung der Anforderungen & Eigenschaften zwischen Beteiligten
* Abbildung / Konzeption von Anforderungen

Prinzipien von SW-Architekturen

* Build to change instead to last.
* Single Responsibility

Architekturstile: Familie von Architekturen, die sich verschiedene Eigenschaften teilen

* Stellt Einschränkungen für Komponenten und deren Beziehung dar
* Einschränkungen bestimmen die Form einer Architektur
* Definierte Vor- und Nachteile pro Stil

n-Tier Architecture

* N-Tier: Einteilung in Schichten mit klaren Schnittstellen. Eine Schicht kann nur auf eine Schicht davor oder alle Schichten davor zugreifen
* Anwendung: Standard Webanwendungen, heterogene Systeme

|  |  |
| --- | --- |
| Vorteile | Nachteile |
| Portierbarkeit  Einfache Architektur  Natürliche Entwicklung der Anwendung | Schichtenübergreifende Funktionen  Flaschenhälse  Änderungen betreffen viele Schichten |

Event Driven Architecture

* Besteht Produzenten und Konsumenten, die lose miteinander verbunden sind. Ein Konsument verarbeitet ein Ereignis, sobald es verfügbar ist. Produzenten können neue Events generieren
* Anwendungsbereich: Verarbeitung von Ereignissen durch mehrere Subsysteme, Echtzeit

|  |  |
| --- | --- |
| Vorteile | Nachteile |
| Entkoppelte Module  Hohe Skalierbarkeit  Teilsysteme sind unabhängig | Verarbeitung in vorgegebener Reihenfolge  Garantierte Verarbeitung einer Nachricht  Kontrolle der Ausführung |

Microservices

* Bestehen aus einer Menge kleiner, autonomer Dienste
* Anwendungsgebiet: große, komplexe Anwendungen mit häufigen Änderungen und meist großen Entwicklungsteams

|  |  |
| --- | --- |
| Vorteile | Nachteile |
| Unabhängige Entwicklung & Deployment  Fehlerisolation  Heterogene Technologien | Dienstübergreifende Funktionen  Datenintegrität  Komplexität |

Ein Bild, das Text, Screenshot, Diagramm, parallel enthält.

Automatisch generierte Beschreibung

## Architektur von Web-Anwendungen

Logisches 3-Schichten Modell

1. Präsentationsschicht: User Interface (HTML, JavaScript, Cookies)
2. Anwendungsschicht: Geschäftslogik (WebServer, JSP, PHP, Session)
3. Persistenzschicht: dauerhafte, konsistente Datenspeicherung (Oracle, MySQL, Postgres)

Thin Client

* Geringere Komplexität
* Darstellungskomponenten werden am Server erzeugt

Rich/Fat client

* Komplexer Client Code
* Client is used to perform the majority of the processing

|  |  |
| --- | --- |
| Client-side scripting | Server-side “scripting” |
| Skript/ Code am Client ändert HTML  Ausführung im Browser des Benutzers  Einfachste Form | Skript / Programm auf dem Server generiert dynamisch HTML  Ausführung als eigenständiger Prozess  Spring, Ruby, PHP |

|  |  |  |
| --- | --- | --- |
| Statisches HTML |  | Webseiten statisch am Server  HTTP Anfrage führt zum Auslesen der Datei |
| Programmatische Erzeugung ohne Layers |  | Webseiten werden programmatisch am Server erzeugt  HTTP Anfrage führt zum Ausführen von ProgrammCode |
| Common Gateway Interface |  | Webseiten werden programmatisch am Server in einem eigenen Prozess erzeugt  Datenhaltung kann extern erfolgen  Vorteile   * Sicherheit durch eigenen Prozess * Volle Suchfunktionalität des DBMS   Nachteile   * Ein Prozess pro Anfrage * Keine Speicherung des Zustands |
| Programmatische Erzeugung mit APIs/Layers |  | Webseiten werden programmatisch am Server erzeugt  Server stellt Schnittstelle bereit, sodass Benutzerprogramme über diese Schnittstelle auf http anfragen reagieren können |

Skripting

|  |  |
| --- | --- |
| Vorteile | Nachteile |
| Erster Ansatz, Design von Logik zu trennen  HTML Designer kann Scripting einfügen | Web Server-abhängige proprietäre Erweiterung von HTML  Befehlssatz reicht für komplexe Anwendungen nicht |

## HTTP

Ein Bild, das Text, Screenshot, Schrift, Diagramm enthält.

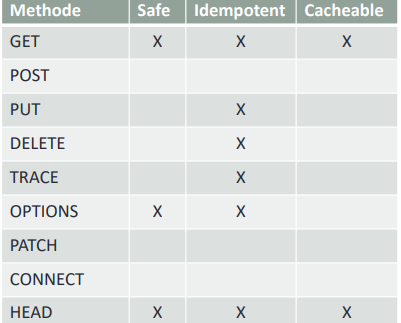
Automatisch generierte BeschreibungHTTP (Hypertext Transfer Protocol): Protokoll für verteilte, kollaborative Hypermedia Informationssysteme

Protokoll der Anwendungsschicht über zuverlässigem Transportschichtprotokoll

Design Guidelines

* Einfach & lesbar
* Erweiterbar
* Zustandslos, ermöglicht aber Sessions
* Datenübergang kontrolliert in Transportschicht

|  |  |
| --- | --- |
| Methodenname | Beschreibung |
| GET | Anfordern einer Ressource vom Webserver |
| POST | Schickt verschiedene, strukturierte Daten zur Verarbeitung an den Webserver |
| PUT | Lädt eine Ressource auf den Webserver |
| DELETE | Löscht eine Ressource auf dem Webserver |



Safe: Methode führt zu keiner Änderung der Ressource

Idempotent: wiederholter Request führt zum gleichen Ergebnis, d.h. der Request hat keine Seiteneffekte

Cacheable: Antwort kann zwischengespeichert werden

HTTP Status Codes

|  |  |  |
| --- | --- | --- |
| Code | Klasse | Beispiel |
| 1xx | Information | 100 continue |
| 2xx | Erfolg | 200 ok |
| 3xx | Umleitung | 301 moved permamently  304 not modified |
| 4xx | Client-Fehler | 404 not found |
| 5xx | Server-Fehler | 502 Bad Gateway |

HTTP Header Elemente

* Authentifizierung von Anfragen
* Caching von Anfragen
* Client Hints
* Conditionals (If-Match, If-Modified-Since)
* Content Negotiation (Abstimmung gewünschter Inhalt)
* Cookies
* Redirects

SSL

* Sicherheitsmechanismen sollen Vertrauen, Integrität sowie Authentifizierung sicherstellen
* SSL: Secure Sockets Layer
* TLS: Transport Layer Security
* Zwischenschicht zwischen HTTP und TCP/IP

HTTP cookies

* Erweiterung von HTTP um Daten im Browser abzuspeichern
* Server kann in Cookies eindeutige Session-IDs abspeichern, um den Client bei erneuten Besuchen identifizieren zu können

Übertragung von Cookies

* Domain: definiert, welcher Host das Cookie empfangen kann
* Path: URL Pfad der enthalten sein muss
* SameSite: definiert, wann Cookies bei Cross-Site Requests gesendet werden

First-Party Cookies: Cookie Domain Attribute entspricht der aktuellen Seite

Third-Party Cookies: Cookie Domain Attribute entspricht nicht der aktuellen Seite

## HTML5

Ein Bild, das Text, Screenshot, Diagramm, Plan enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Screenshot, Zahl, Rechteck enthält.

Automatisch generierte Beschreibung

Ein Bild, das Text, Diagramm, Screenshot, Design enthält.

Automatisch generierte Beschreibung

## Java Servlets

Servlets are the Java programs that run on the Java-enabled web server or application server. They are used to handle the request obtained from the webserver, process the request, produce the response then send a response back to the webserver.

Merkmale

* Server-seitige Klassen, die Applikationen nach dem Request-Response Modell (HTTP) ermöglichen
* Plattformunabhängigkeit durch Java Technologie und standardisierte API
* Wiederverwendung der Funktionalität von Java
* Sicherheitskonzept von Java

ArchitekturEin Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte Beschreibung

Lebenszyklus

Servlet wird in einem speziellen Servlet-Container verwalten (API basierter Web Server). Servlet Container übernimmt die Weiterleitung der Client Anfrage an das Servlet

1. Benutzer sendet Formular an Browser
2. Browser löst Aktion an eingabe.html aus
3. Aktion wird in .xml ausgeführt
4. Anfrage wird mitgegeben (Servlet)
5. Browser nimmt Antwort von Servlet entgegen
6. Browser gibt Antwort an Client

Servlet Lebenszyklus

1. Laden der Servlet Klasse
2. Instanziieren
3. Init(ServletConfig)
4. Service(HTTP-ServletRequest)
5. Destroy()

Interaktion zwischen Servlets

* Kontrollfluss: Delegation und Einbindung Weitergabe einer Client-Anfrage an andere URLs
* Datenfluss: Attribute

Session Tracking von Servlets

* Problem: Zustandslosigkeit von HTTP
  + Es gibt keine Sessions, also jeder Request wird individuell und unabhängig von anderen betrachtet
  + Es ist schwer in kontextabhängigen Dokumenten zu navigieren oder Ergebnisse einer Suche in Teilen zu präsentieren
* Verschiedene Varianten, Zustandsinformationen in Anfragen einzubauen
  + Autorisierung eines Benutzers
  + Versteckte Felder in Formularen
  + Zustandsinformationen in URL kodieren
  + Cookies
  + Java Servlet Tracking API

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte BeschreibungSichtbarkeit von Servlet Variablen