Arkitektur for begyndere

Denne artikel beskriver forskellige basale n-tier arkitekturer.

Som man bør kende og have valgt inden man går igang med at udvikle et system.

Den forudsætter kun en smule kendskab til IT termer.

Skrevet den **15. feb 2010** af **arne_v** I kategorien **Programmering / Generelt** | ★★★☆☆

Historie:
V1.0 - 01/02/2004 - original
V1.1 - 20/06/2005 - tilføj lidt om layers
V1.2 - 14/02/2010 - smårettelser

Indledning

Når man skal igang med et mere kompleks projekt er det meget vigtigt at man får styr på de forskellige tiers i ens arkitektur.

Man ved hvor mange der er. Man ved hvilken teknologi de skal baseres på. Man ved hvilken funktionalitet der ligger i hver af dem.

Denne oversigt over forskellige gængse arkitekturer skulle gerne hjælpe til at forstå mulighederne og dermed også hjælpe med beslutningen.

Det er ikke en komplet oversigt. Men dækker formentligt langt de fleste tilfælde.

Inden vi går igang lige en definition:
tier = lag som kan (men ikke behøver) køre på separat system
layer = lag som er en logisk opdeling af ens kode

Der er stor forskel.

Når man skal kalde fra et tier til et andet tier er der noget netværk og en netværks protkol involveret. Når man kalder fra et layer til et andet layer indenfor samme tier er det et normalt kald af noget kode.

En opdeling i tiers vil derfor normalt koste performance men kan give skalerbarhed og sikkerhed.

Forskellige tiers kan sagtens implementeres i forskellige teknologier. Forskellige layers indenfor samme tier vil typisk skulle udvikles i samme teknologi.

applikation (1-tier)

Dette er den simplest mulige arkitektur - en applikation (typisk en GUI applikation) som kører og evt. læser/skriver nogle filer.

Al funktionalitet ligger i sagens natur i applikationen.

Bruges til små systemer som kun skal bruges af en enkelt person.

Typiske sprog: Java, C, C++, C#, Pascal, Basic

applikation ---- database server (2-tier)

Dette er en udvidelse af den simple model, hvor applikationen (typisk en GUI applikation) i.s.f. at læse/skrive til filer læser/skriver til en database.

Logikken ligger i applikationen. Alle data gemmes på database serveren.

Bruges til data orienterede systemer som skal bruges af flere brugere internt i firmaet.

Typiske sprog: Java, C, C++, C#, Pascal, Basic

Typiske database servere: DB2, MS SQLServer

Database interfacet vil typisk være: ODBC, JDBC, OLE DB eller et database specifikt API

client applikation ---- server applikation (2-tier)

Dette er en klassisk client/server løsning. X clienter snakker med 1 server.

Logikken ligger på serveren. Bruger interfacet ligger i klienterne. Serveren har mulighed for at lade klienterne interaktere.

Bruges til ikke-data-orienterede systemer kunne f.eks. være noget chat lignende.

Typiske sprog på client side: Java, C#, Pascal, VB

Typiske sprog på server side: C, C++, C#, Java

Kommunikationen mellem client og server er typisk TCP/IP sockets. Evt. med et lag ovenpå i form af Java RMI eller .NET remoting.

browser ---- web server (2-tier)

Dette er en web løsning som oprindeligt tilbage i midten af 1990'erne med statisk indhold.

Web serveren server kun filer og sender filernes indhold ud til browseren uden at bekymre sig om indhold.

Filerne indeholder så HTML, CSS og JS (JavaScript) og de fortolkes af browseren og vises.

Der kan diskuteres om det egentlig er en programmerings løsning, fordi

der er ikke nogen egentlig applikations logik. Men vær opmærksom på at opgaven med at få siderne til at se ens ud og virke ens i alle browsere kan være en meget stor opgave.

Bruges til simpel publicering af information over internettet.

Typiske web servere: Apache HTTPD, MS IIS

client applikation ---- server applikation ---- database server (3-tier)

Dette er den oprindelige 3 tier løsning. En GUI client applikation som tager sig af bruger interfacet. En server applikation med logikken. Og en database server til at gemme data på.

Bruges til større systemer som skal bruges af flere brugere internt i firmaet.

Typiske sprog på client side: Java, C#, Pascal, VB

Typiske sprog på server side: C, C++, C#, Java

Typiske database servere: Oracle, DB2, MS SQLServer

Kommunikationen mellem client og server er typisk TCP/IP sockets. Evt. med et lag ovenpå i form af Java RMI eller .NET remoting.

Database interfacet vil typisk være: ODBC, JDBC, OLE DB eller et database specifikt API

browser ---- web server med ASP ---- database server (3-tier)
browser ---- web server med PHP ---- database server (3-tier)
browser ---- web server med ASP.NET ---- database server (3-tier)
browser ---- JSP/servlet container ---- database server (3-tier)
browser ---- web server med CGI ---- database server (3-tier)

Dette er en typisk web applikation.

I.s.f. at serve filer med statisk indhold, så genereres der helt eller delvist dynamisk indhold af HTML, CSS og JS (JavaScript) på server side.

Indholdet fortolkes og vises i browseren. Logikken og den egentlige præsentation ligger i server side scriptene. Databasen gemmer data.

Bruges til services på internettet som f.eks. forum'er og web butikker.

Typiske server side sprog: ASP (VBScript), PHP, ASP.NET

Lidt mindre typiske server side sprog: JSP/servlet (Java), Perl CGI

Typiske web servere: Apache HTTPD, MS IIS

Typiske JSP/servlet containere: Apache Tomcat, Caucho Resin

Typiske database servere: MySQL, MS SQLServer

Database interfacet vil typisk være: ODBC, JDBC, OLE DB eller et database specifikt API

browser ---- JSP/Servlet container ---- EJB container ---- database server (4-tier)

Indholdet fortolkes og vises i browseren. Den egentlige præsentation
ligger JSP og servlet. Selve logikken ligger i EJB. Databasen gemmer data.

Bruges til større e business løsninger.

Alt er Java baseret.

Typiske J2EE app servere: IBM WebSphere, BEA WebLogic, JBoss (inkl. Apache Tomcat)

Typiske database servere: Oracle, DB2, MS SQLServer

layers

Normalt vil man ønske at dele koden op i layers.

Hvis vi kigger på nogle af de mest bruge arkitekturer:

applikation ---- database server (2-tier)

vil man have opdelt applikationen i layers.

browser ---- web server med XXX ---- database server (3-tier)

vil man have opdelt web applikationen der kører på web serveren i layers.

Den helt klassiske opdeling er:

presentation layer = bruger grænsefladen
business logic layer = den bagvedliggende logik som er uafhængig af brugergrænsefladen
data access layer = koden til at tilgå databasen

I meget simple tilfælde vil business logic layer helt kunne undværes.

Bemærk at der ikke er grund til at have et layer som ikke gøre noget.

Personligt foretrækker jeg dog opdelingen:

presentation layer = koden til at vise noget for brugeren
controller layer = koden som ekskveres når brugeren gør noget
business logic layer = den bagvedliggende logik som er uafhængig af brugergrænsefladen
data access layer = koden til at tilgå databasen

Det er naturligvis vigtigt at layers er klart adskilte med simple
interfaces.

Ideelt består et lag af:
- data klasser
- interfaces
- en factory klasse til at hente instans af lag med

Nogen gange bruger man et IoC framework fremfor en egen factory klasse til
at instantiere med.

Kommentar af simonvalter d. 28. okt 2004 | 1

Meget lærerigt, jeg havde godt en forestilling om hvad det var men havde ikke undersøgt det nærmere.

Kommentar af elf d. 04. feb 2004 | 2

Spændende læsning

Kommentar af zatz d. 28. okt 2005 | 3

Kommentar af thundergod d. 25. maj 2004 | 4

Kommentar af poolo d. 06. feb 2004 | 5

Rigtig god!

Kommentar af mikkel_strack d. 18. maj 2004 | 6

Fin artikel

Kommentar af cdull d. 20. feb 2006 | 7