Avdelningen för datavetenskap

Conny Nykvist

Mikael Andersson

Utökning av PHP

Extension of PHP

Examensarbete 10 poäng

Dataingenjörsprogrammet

Datum/Termin:	HT-05
Handledare:	Robin Staxhammar
Examinator:	Stefan Lindskog
Ev. löpnummer:	2006:01

Karlstads universitet 651 88 Karlstad

Tfn 054-700 10 00 Fax 054-700 14 60

Information@kau.se www.kau.se

Utökning av PHP

Conny Nykvist

Mikael Andersson

Denna rapport är skriven som en del av det arbete som krävs för att erhålla en dataingenjörsexamen i datavetenskap. Allt material i denna rapport, vilket inte är mitt eget, har blivit tydligt identifierat och inget material är inkluderat som tidigare använts för erhållande av annan examen.

Conny Nykvist

Mikael Andersson

Godkänd, 2006-01-18

Handledare: Robin Staxhammar

Examinator: Stefan Lindskog

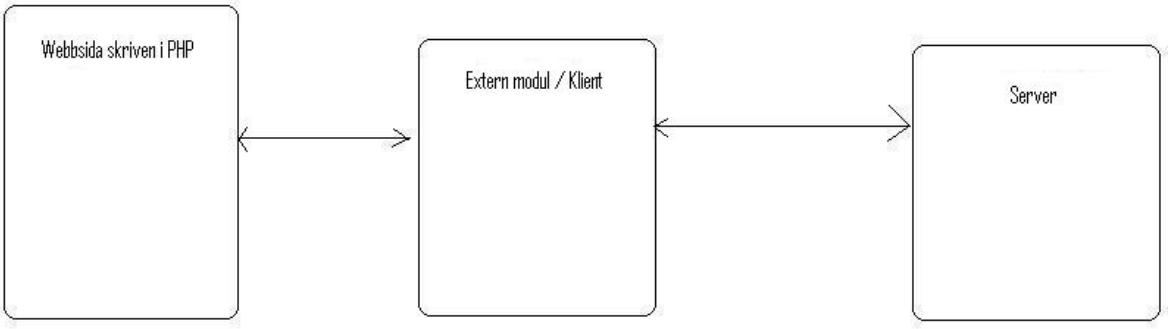
Sammanfattning

Detta examensarbete är gjort på uppdrag av ett företag som heter IT Resource AB i Årjäng Värmland.

Målet med detta examensarbete var att skapa ett API till PHP som skall utöka PHPs funktionalitet. APIet skall förenkla samt effektivisera IT Resource ABs arbete med att utveckla webbapplikationer. Den förenkling samt effektivisering som APIet erbjuder är möjligheten att återanvända funktioner samt att man enkelt kan lägga till nya funktioner i APIet vid behov.

Den funktionalitet som APIet kommer att tillhandahålla är en förbättrad hantering av variabler och strängar som genereras från en webbsida skriven i PHP. Med en förbättrad hantering menas bland annat att det skall vara möjligt att lagra variabler och strängar från en webbsida skriven i PHP.

Det skapade APIet består av 2 byggstenar, en klient och en server. Klienten i klient/server-systemet är en extern modul innehållande funktionalitet för kommunikation med både PHP och servern. Detta innebär att klienten i klient/server-systemet kommer att fungera som ett gränssnitt mellan PHP och servern.



Figur 1: En Illustration som visar kopplingen mellan PHP, klient och server

I serverdelen återfinns implementationen av den funktionalitet som APIet erbjuder. Den funktionalitet som implementerats i serverdelen är lagring av variabler och strängar i primärminnet.

Resultatet av arbetet är ett API som utökar PHPs funktionalitet. Detta API kan utökas ytterligare vid behov och kan med fördel användas av IT Resource AB för utveckling av webbapplikationer i PHP.

Extension of PHP

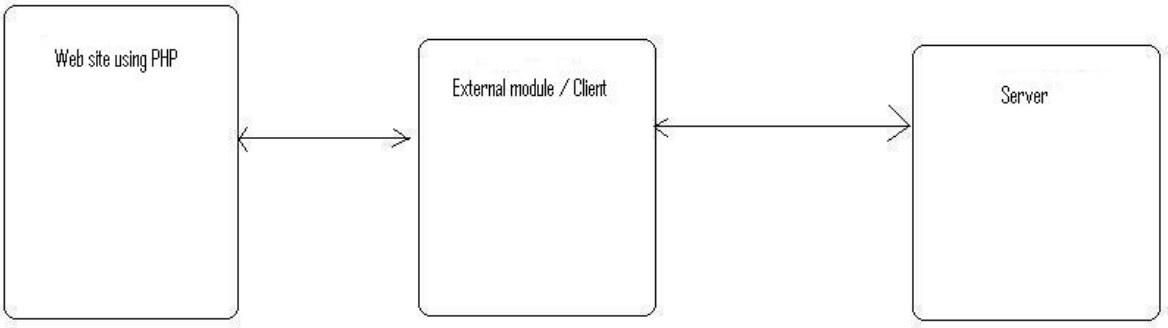
Abstract

This bachelor's report is based upon an assignment specified by the company IT Resource AB, located in Årjäng Värmland.

The goal of this bachelor's project was to create an API to extend the functionality of PHP. The API shall simplify and make the work more efficient for IT Resource AB when developing web applications. The simplification and the increased efficiency that the API offers are the possibilities to reuse and easily add new functions to the API when needed.

The functionality that the API will offer is an enhanced capability to handle variables and strings generated from a web page written in PHP. The enhanced capability refers to the possibility to store variables and strings from a web page written in PHP.

The API created consists of 2 parts, a client and a server. The client in the client/server system is an external module which contains the functionality for the communication with both PHP and the server. This means that the client in the client/server system will act as an interface between PHP and the server.



Figur 2: Illustration of the connection between PHP, client and server

The implementation of the API's functionality is found in the server part of the client/server system. The functionality implemented in the server is the possibility to store variables and strings in the main memory.

The result of this project is an API that increases the functionality of PHP. The API can, when needed, be extended further, and IT Resource AB can benefit from using it when developing web applications that make use of PHP.

Innehållsförteckning

Figurförteckning

1 Inledning

Detta examensarbete är utfört åt IT Resource AB som hade ett behov av att förenkla samt effektivisera sitt arbete med att utveckla webbapplikationer i PHP, PHP: Hypertext Processor. För att effektivisera samt förenkla IT Resource ABs arbete skall ett API, Application Program Interface, till PHP skapas för att bygga ut PHPs funktionalitet. Detta API skall lösa de problem som har lyfts fram av IT Resource AB. Mer om detta behandlas under bakgrund i avsnitt 1.1 samt under mål i avsnitt 1.2.

1.1 Bakgrund

IT Resource AB är ett företag som ligger i Årjäng i Värmland. De arbetar inom IT-området med support, integration, anpassning/utveckling av mjukvaror och erbjuder expertkompetens inom IT för företag. IT Resource AB har även en lång erfarenhet inom nätverk, drift och programmering. Deras senaste uppdrag rör webbapplikationer med exempelvis CRM, Customer Relationship Management, verktyg samt system för orderhantering och övervakning av utrustningar som kommunicerar via GSM. CRM är en affärsstrategi för att hantera kunder [9]. [1]

De flesta av IT Resource ABs kunder är företag inom den tillverkande industrin.
Vid utveckling av webbapplikationer använder IT Resource AB den så kallade LAMP arkitekturen (Linux, Apache, MySQL och PHP). LAMP är en öppen källkodslösning som använder Linux som operativsystem, Apache som webbserver, MySQL som relationsdatabas och PHP som programspråk.

IT Resource AB har uttryckt ett önskemål om att förenkla samt effektivisera sitt arbete inom utvecklingen av webbapplikationer, då med störst fokus på händelsestyrda webbapplikationer. En händelsestyrd webbapplikation är en webbapplikation som är direkt kopplad till en användarinteraktion. Ett problem vid utvecklingen av händelsestyrda webbapplikationer är att PHP hanterar variabler och strängar på ett icke tillfredställande sätt. IT Resource ABs förhoppning är att en utökning av PHP kan göra det möjligt att hantera variabler och strängar

på ett tillfredsställande sätt vilket skulle förenkla samt effektivisera deras arbete vid utvecklingen av händelsestyrda webbapplikationer.

Ett sätt att utöka PHPs funktionalitet är att skapa ett API. Ett API är ett gränssnitt som gör det möjligt att i program utnyttja funktioner som finns tillgängliga i ett annat program eller i en funktionssamling [2].

De specifika problem som lyfts fram av IT Resource AB som behöver en lösning är att när ett PHPskript har exekverat klart så tappas all temporär data. Med temporär data menas i detta sammanhang variablers och strängars data som återfinns i PHPskriptet under exekveringen. Detta är ett problem för när man till exempel vill uppdatera en webbsida i PHP så kommer all temporär data att tappas. När en användare fyller i en order på en webbsida i form av ett formulär och skall bekräfta sin order, då skall bekräftelsen på ordern visas, det vill säga webbsidan skall uppdateras. När webbsidan uppdateras kommer användarens order att försvinna eftersom PHPskriptets exekvering är klar och då är all temporär data tappad. Den lösning som IT Resource AB använder sig av idag för att komma runt detta problem är att man lagrar dessa temporära data i en superglobal array (eller automatiskt global array). En superglobal array har hela skriptet som avgränsningsområde och det medför att den kan användas i metoder och funktioner utan att deklareras.

1.2 Mål

Målet med detta examensarbete är att skapa ett API som är en utökning av PHP och som skall lösa de problem som har lyfts fram av IT Resource AB. APIet skall hantera lagringen av variabler och strängar från en webbsida skriven i PHP och skall kunna utnyttjas av IT Resource AB för skapandet av webbsidor.

1.3 Uppsatsens upplägg

De krav som finns på APIets utformning samt vilken funktionalitet som skall återfinnas i APIet behandlas under kravspecifikationen i kapitel 2.

Eftersom vi skall skapa ett API till PHP för att bygga ut PHPs funktionalitet kommer vi att ta upp vad PHP är samt hur PHPs kärna fungerar. Mera information om detta återfinns under PHP i kapitel 3.

Den konstruktionslösning som har valts för APIet samt en motivering till varför just denna lösning valts återfinns under beskrivning av konstruktionslösning i kapitel 4.

Implementationen av den valda konstruktionslösning för det API som skall skapas tillsammans med kodexempel, lösningar på direkta problem under implementationen och diverse tester som var nödvändiga återfinns under implementation i kapitel 5.

Resultaten från detta projekt med hänsyn på hur målen har uppnåtts diskuteras under resultat i kapitel 6.

Hur projektet har fungerat, om den lösning som konstruerats var bra, kontakt med uppdragsgivare, problem under projektet, nya erfarenheter och så vidare sammanställs under projektets sammanfattning i kapitel 7.

2 Kravspecifikation

I detta kapitel återges kraven på det API som skall skapas under projektet.

Det API som skall skapas skall minst innehålla en funktion i sin funktionssamling som hanterar lagring av variabler och strängar från en webbsida skriven i PHP.
Lagring av dessa variabler och strängar kan ske på följande sätt:

- Lagring i primärminne
- Lagring på sekundärminne
- Lagring i databas

Funktionen som kommer att skapas skall hantera lagring i primärminnet, det skall dock vara möjligt att vid ett senare tillfälle vidareutveckla funktionen alternativt lägga till ytterligare funktioner i funktionssamlingen för att hantera de övriga fallen, det vill säga lagring på sekundärminne och lagring i databas.

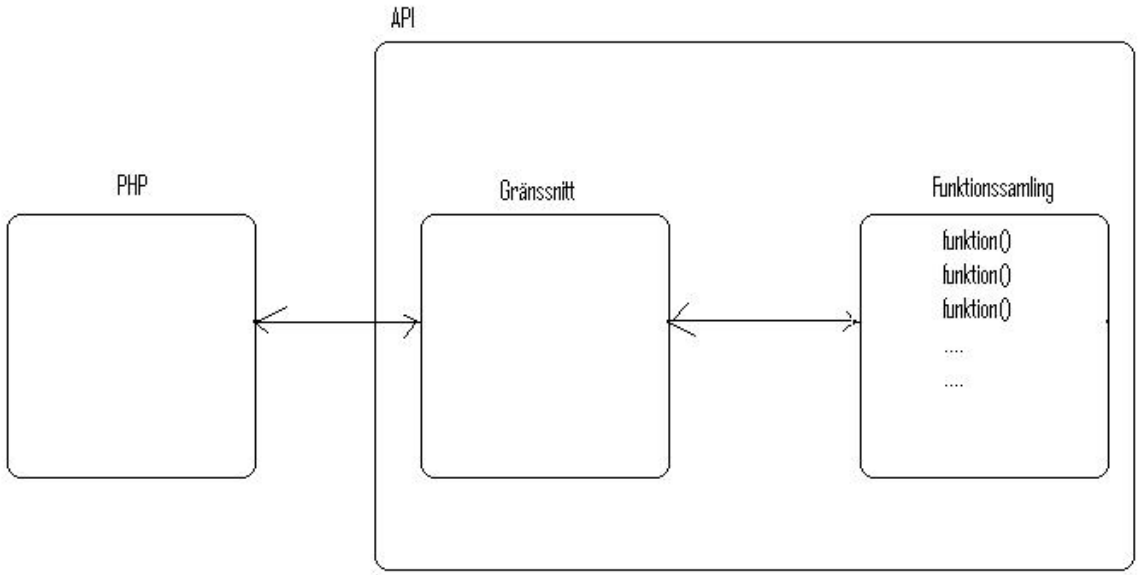
De grundkrav som ställs på det API som skall skapas är:

1. Utvecklingsmöjligheter
2. Säkerhet
3. Användarvänlighet

2.1 Utvecklingsmöjligheter

Kravet för vidareutvecklingsmöjligheter bygger på att man vid senare tillfälle kan ha en önskan om att utöka funktionssamlingen som APIet innehåller. Utökningen av funktionssamlingen kan till exempel ske genom tillägg av egendefinierade funktioner till den redan existerande funktionssamlingen. Genom att kunna utöka APIets funktionssamling med egendefinierade funktioner så kan användaren skräddarsy APIets funktionalitet.

Det skall även läggas viss vikt vid om det är möjligt att vidareutveckla gränssnittet mellan PHP och funktionssamlingen i APIet.



Figur 3: Illustration av koppling mellan PHP och API samt uppbyggnad av API

2.2 Säkerhet

De säkerhetsmekanismer som kommer att återfinnas hos det skapade APIet är:

- Validering av data
- Validering av användare

Validering av data avser data som är inparametrar till funktioner som återfinns i funktionssamlingen för det skapade APIet. Exempel på sådan validering kan vara att kontrollera om en variabel som används som inparameter håller sig inom ett bestämt intervall. Exempel på en inparameter är följande:

```
$param = 10;
$str = addValue($param);
```

$param håller det värde som det skall vara möjligt att validera, detta värde skickas som inparameter till funktionen addValue.

Validering av användare innebär att man skall kunna validera vilken användare det är som skall lagra eller alternativt hämta data. Denna validering behövs för att användaren skall kunna få fram de data som tidigare har lagrats för just denna användare. Validering kommer att bestå av att ett unikt id sparas tillsammans med de data (variabler och/eller strängar) som skall sparas för användaren. Detta unika id kommer att användas för validering av användaren. För att kunna återskapa webbsidan krävs det att det unika id som användaren har matchar det unika id som ligger sparad tillsammans med användarens data.

2.3 Användarvänlighet

Användarvänlighet innefattar att det skall finnas en utförlig dokumentation för APIet, denna dokumentation skall bland annat innehålla information om funktionssamlingen. Funktionssamlingsinformationen skall innehålla utförlig information om varje funktion som återfinns i funktionssamlingen, exempel på denna information är:

- Funktionsnamn
- Beskrivning av inparametrar till funktionen
- Beskrivning av lokala variabler och konstanter
- Funktionsbeskrivning med avseende på vad funktionen utför
- Beskrivning av returvärden från funktionen

Med en bra utarbetad användarbeskrivning blir användarvänligheten väsentligt förbättrad. Ytterligare delar som speglar användarvänligheten är hur möjligheten att lägga till samt ta bort funktioner ur funktionssamlingen kan utföras. Tillägg samt borttagning av funktioner ur funktionssamlingen skall kunna utföras genom att följa en användarbeskrivning för just detta ändamål. Användarbeskrivningen skall vara okomplicerad att följa utan tidigare erfarenhet av just detta API.

3 PHP

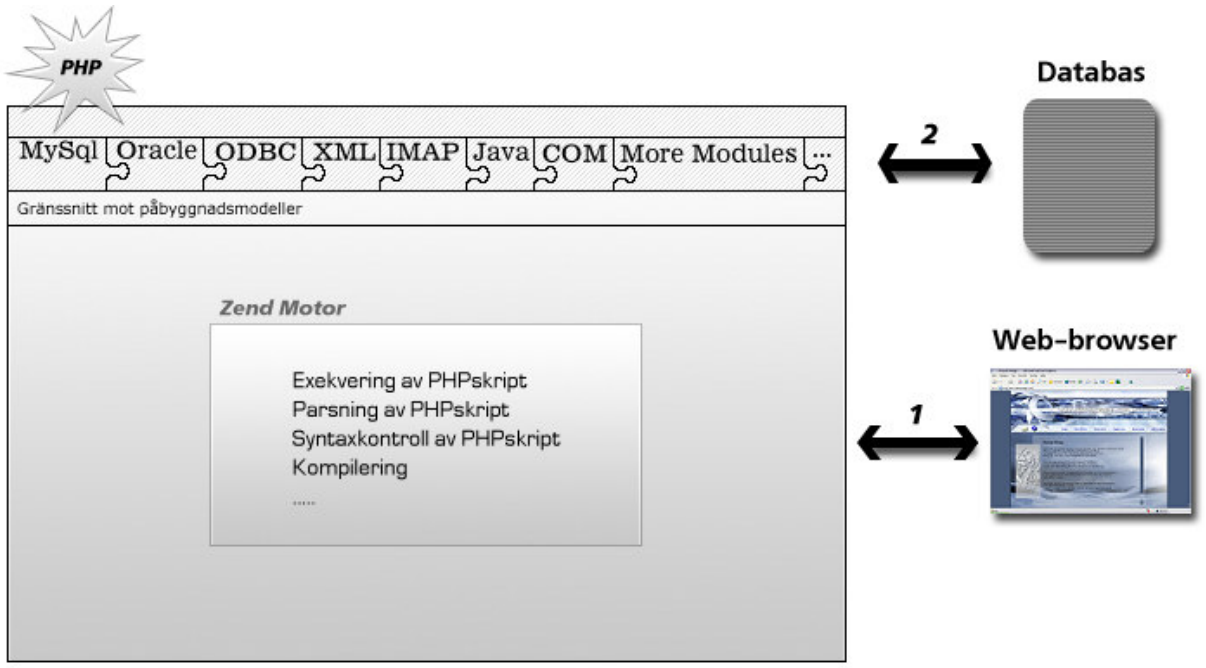
Under följande kapitel följer en kort beskrivning av PHP samt olika sätt att utöka funktionaliteten för PHP.

PHP, Hypertext Processor, är ett skriptspråk som bygger på en öppenkällkodlösning. PHP är skapat i programspråket C och dess syntax efterliknar C, Java samt Perl. [3]

Kärnan i PHP heter Zendmotorn (Zend Engine). Zendmotorn hanterar följande i PHP:

- Parsning, syntaxkontroll, kompilering och exekvering av PHPskript
- Implementation av alla standarddatastrukturer i PHP
- Gränssnitt till interna/externa påbyggnadsmoduler för ihopkoppling med externa resurser och protokoll, till exempel SQL, HTTP och FTP
- Tillhandahållning av all standardservice i PHP (inkluderat minneshantering, resurshantering med mera)

[4]



Figur 4: Översikt av PHP

3.1 Utökning av PHP

Under detta avsnitt presenteras de olika möjligheterna för utökning av PHP kortfattat.

3.1.1 Externa moduler

En extern modul är en extern fil innehållande programkod som används genom att den blir laddad under körning via användandet av funktionen dl() alternativt via tillägg i PHP.ini filen och detta resulterar i att dess funktionalitet blir tillgänglig för PHPskriptet under dess exekvering. När PHPskriptet avslutas rensas den laddade programkoden från minnet.
De fördelar samt nackdelar som finns med externa moduler är:

Fördelar:

- Externa moduler kräver ingen ny kompilering av PHP.
- Storleken på PHPs programkodsfiler ändras inte på grund av att man använder sig av programkod från externa filer.

Nackdelar:

- De externa filerna/filen behöver laddas varje gång skriptet exekverar och detta kan medföra att programmet/applikation blir tidskrävande.

[5]

3.1.2 Inbyggda moduler

Inbyggda moduler innebär att filer innehållande programkod kompileras samman med PHPs programkodsfiler, dess funktionalitet är tillgänglig för alla skript som körs direkt. De fördelar samt nackdelar som finns med inbyggda moduler är:

Fördelar:

- Inga filer behöver laddas, dess funktionalitet är tillgänglig direkt när de är kompilerade samman med PHP.

Nackdelar:

- En ny kompilering av PHP är nödvändig varje gång man vill utöka funktionaliteten i den inbyggda modulen alternativ ändra den nuvarande funktionaliteten i modulen.
- PHP ”växer” och kräver mer minne.

[6]

3.1.3 Zendmotorn

Att utöka PHP via Zendmotorn innebär att man ändrar direkt i PHPs kärna men det anses vara ett dåligt sätt på grund av stora negativa sidoeffekter. Exempel på dessa negativa sidoeffekter är:

- Gjorda modifikationer i kärnan skrivs över vid nästa uppdatering av PHP
- Kärnan blir inkompatibel gentemot resten av världens installationer av PHP

Eftersom det anses vara ett dåligt sätt att utöka PHP via Zendmotorn på grund av dessa stora negativa sidoeffekter kommer vi inte att ta upp denna variant för att utöka PHP mera i denna rapport.

[7]

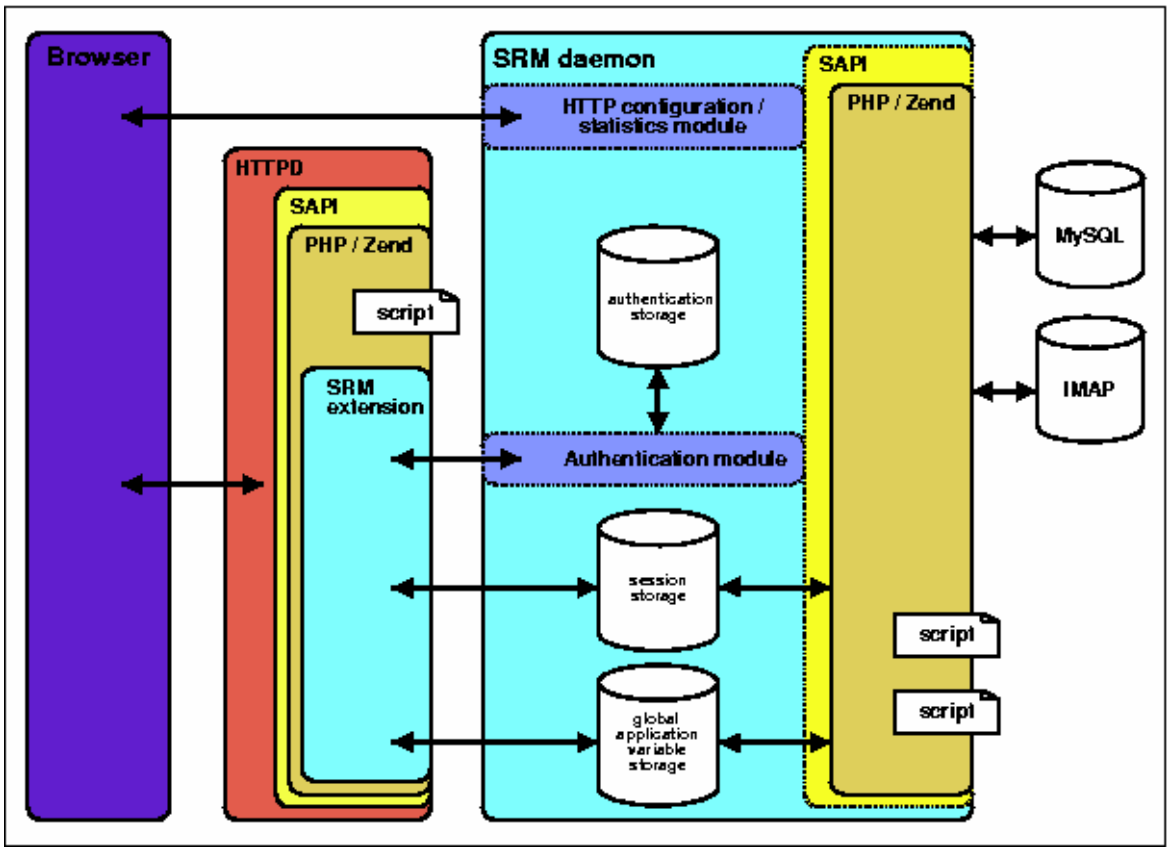
3.1.4 Utökning av PHP med hjälp av annan programvara

Ett exempel på en programvara som kan användas för att utöka PHPs funktionalitet är SRM, Script Running Machine. SRM skapades hösten 2000 av The Vulcan Logic Group. SRM kan installeras under operativsystemen Linux, Solaris/Intel, FreeBSD och OpenBSD. SRM är skrivet i programspråket C och befinner sig just nu i betastadiet (version 1.83 datum 2005-08-08).

SRM består av tre delar:

- SRMdaemon som är själva huvuddelen i SRM. Med uttrycket daemon menas en process som kör en tjänst i bakgrunden. SRMdaemon gör det möjligt för SRM att köra PHPskript.

- SRMs PHPutbyggnad är den delen av SRM som gör att SRM kan kommunicera med PHP.
- SRMs PHP SAPI, Server Application Program Interface, är det API i SRM som används när en klient anropar ett PHPskript som körs i SRMdaemon. Ett sådant PHPskript som körs i SRMdaemon kallas för en banan, en mera ingående förklaring av vad en banan är finns i beskrivningen av den funktionalitet som SRM erbjuder som följer nedan.



Figur 5: Illustrerar hur SRM är designat samt hur de olika delarna i SRM är sammankopplade och är hämtad från [8]. © 2006 by The Vulcan Logic Group

Den funktionalitet som SRM erbjuder är:

- Hantera sessionsdata

  SRM som sessionshanterare fungerar på det sättet att när en användare går in på en webbsida kontrollerar SRM om användaren har skickat med en sessionsidentifikation, som är en unik identifikation för varje session, om så är fallet så återskapas den tidigare sparade omgivningen annars så får användaren en sessionsidentifikation.

- Applikationsvariabler

  Applikationsvariabler är sparad data i SRM som kan användas över flera webbsidor eller flera användare till skillnad från PHP som tappar all temporär data efter att PHPskriptets exekvering är klar.

  Detta innebär att en användare kan återanvända data som tidigare har använts på en webbsida och nu använda det på en ny webbsida, det innebär även att flera användare kan använda sig av samma data samtidigt eftersom dessa data är sparade i SRM.

- RPC, Remote Procedure Calls: Fjärrstyrda proceduranrop

  Fjärrstyrda proceduranrop fungerar som ett vanligt proceduranrop med undantaget att anropet sker över ett nätverk. Ett proceduranrop kan även kallas funktionsanrop. Det finns två olika saker som kan ske vid ett fjärrstyrt proceduranrop i SRM.

  1. Funktionen finns definierad i en modul i SRM och kan exekveras som vanligt.
  2. Funktionen är definierad i ett av PHPs funktionsbibliotek. PHPs funktionsbibliotek laddas in i SRM när man startar SRM daemon. PHP skripten som är en del av dessa bibliotek kompileras och exekveras. Eftersom de kompilerade skripten inte tas bort, efter att de har laddats in i SRM, är det möjligt att anropa dem ifrån klientsidan av PHPskriptet utan att behöva parsa/kompilera dem igen.

- Bananer

  Fjärrstyrda objekt är PHPskript som körs permanent i SRMdaemon. Dessa fjärrstyrda objekt kallas för SRMApps eller bananer. Bananer kan instansieras precis som man gör med vanliga objekt. Klienter kan komma åt bananer genom att göra ett anrop till SRMs PHP SAPI.

[8] [11]

3.2 Sammanfattning av metoderna

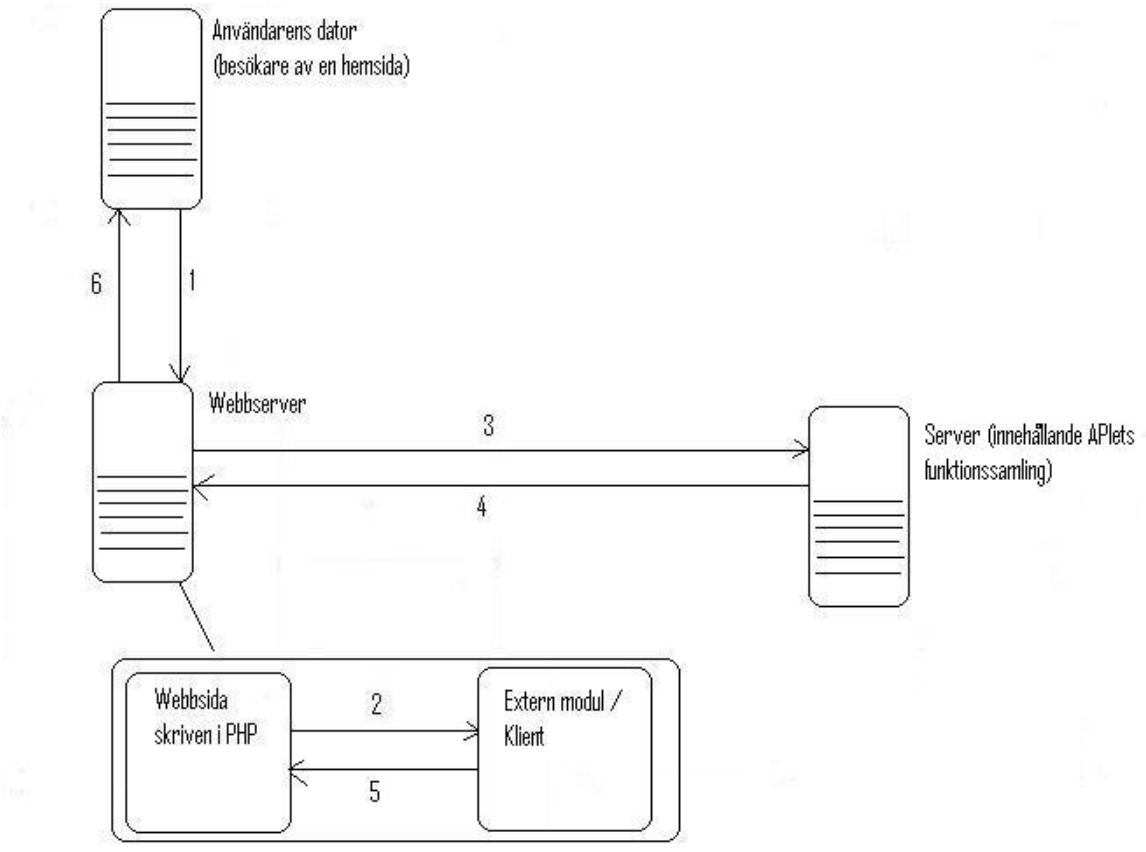
Av de metoder som diskuterats under avsnitt 3.1 känns inte utökning av PHP genom Zendmotorn som något alternativ. Följaktligen står valet mellan inbyggda moduler, externa moduler och SRM för att utöka PHP. SRM känns som den bästa lösningen i detta läge då SRM erbjuder den funktionalitet som behövs för att lösa de problem som IT Resource AB har presenterat, se avsnitt 1.1.

4. Beskrivning av konstruktionslösningen

Under följande kapitel diskuteras vald lösning för projektet samt varför just denna lösning valts.

4.1 Summering av konstruktionslösning

Här följer en kort beskrivning av hur konstruktionslösningen fungerar rent praktiskt när en användare besöker en webbsida.



Figur 6: Illustrerar hur konstruktionslösningen fungerar rent praktiskt.

1. En användare besöker en webbsida skapad i PHP. Användaren använder ett formulär för att ange diverse information. Användaren trycker på en ’OK knapp’ för att komma vidare till nästa webbsida som innehåller en verifiering av den information användaren precis har matat in.
2. Webbsidan skickar den information som användaren har angett till den externa modulen, den externa modulen paketerar informationen och kopplar upp sig mot servern.
3. Den externa modulen skickar den paketerade informationen till servern med hjälp av protokollet TCP/IP. Servern sparar undan den paketerade informationen i primärminnet.

Nu är informationen som användaren angav i formuläret sparad i primärminnet på servern och den nya sidan skall laddas. Webbsidan ber nu den externa modulen om den information som finns sparad för den unika användaren som just nu besöker webbsidan. Steg 2 och 3 utförs igen dock med skillnaden att istället för att paketera och skicka data så ber den om den sparade data som är kopplad till den unika användaren.

4. Informationen tillhörande användaren hämtas från primärminnet och skickas till den externa modulen med hjälp av protokollet TCP/IP.
5. Den externa modulen packar upp informationen och vidarebefordrar den till webbsidan.
6. Webbsidan kan nu visa verifieringsformuläret innehållande informationen.

4.2 Val av konstruktionslösningen

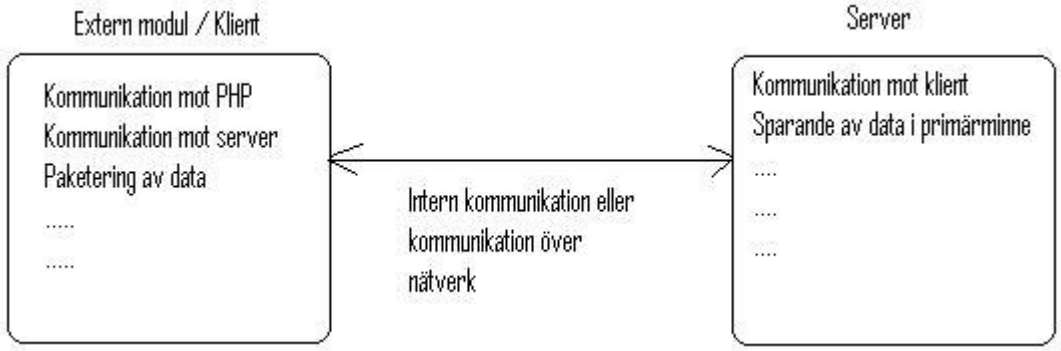
De lösningar för en utökning av PHP som diskuterats i rapporten är externa moduler, se avsnitt 3.1.1, inbyggda moduler, se avsnitt 3.1.2, Zendmotorn, se avsnitt 3.1.3, och SRM, se avsnitt 3.1.4.

Den ursprungliga idén var att använda SRM som en grund att bygga vidare på i projektet då SRM redan uppfyller flertalet av de krav som ställts på lösningen. SRM valdes dock bort eftersom vid en närmare granskning visade det sig vara en onödigt komplex konstruktion. De lösningsalternativ som då återstod var inbyggda moduler respektive externa moduler. Skillnaden mellan en extern moduls funktionalitet och en inbyggd moduls funktionalitet är

väldigt liten, med hänsyn på detta föll valet på att använda externa moduler vid utvecklingen därför att man då slipper att kompilera om PHP vid varje ändring i programkoden (enbart kompilering av modulen krävs). Om uppdragsgivaren senare upptäcker att inbyggda moduler passar bättre är det väldigt lätt att ändra till just detta.

Den lösning som valts är att konstruera en klient/serverlösning där klienten återfinns i form av en extern modul. Anledningen till att vi valt just denna lösning är att den på ett bra sätt kan tillgodose kraven om utökningsmöjligheter och säkerhet, en mer detaljerad beskrivning om detta följer nedan:

- Möjligheterna att vidareutveckla denna klient/serverlösning både på klient och på serversidan är mycket goda. Klientsidan kommer att tillhandahålla funktionaliteten för uppkoppling mot server samt paketering av data som skall sparas, se avsnitt 4.1. I detta projekt har vi valt att använda oss av en extern modul som klient, men denna externa modul kan ersättas mot till exempel en inbyggd modul eller annan programvara om så önskas så länge minst den funktionalitet som behövs för kommunikation mot PHP, kommunikation mot server och paketering av data finns. Serversidan kommer enbart att tillhandahålla funktionaliteten för sparandet av den paketerade data som klienten skickar, detta innebär att så länge man tar emot data från klienten på ett korrekt sätt så kan man utöka antal sätt att spara data nästan till det oändliga (till exempel spara i en databas, på sekundär minnet och så vidare). Klientsidan och serversidan kommer vara fristående från varandra förutom de regler som gäller för att skicka data samt hur data mottas. Detta innebär att modifiering av endera delen inte påverkar den andra, det vill säga man kan vidareutveckla på till exempel klientsidan utan att göra detta på serversidan.

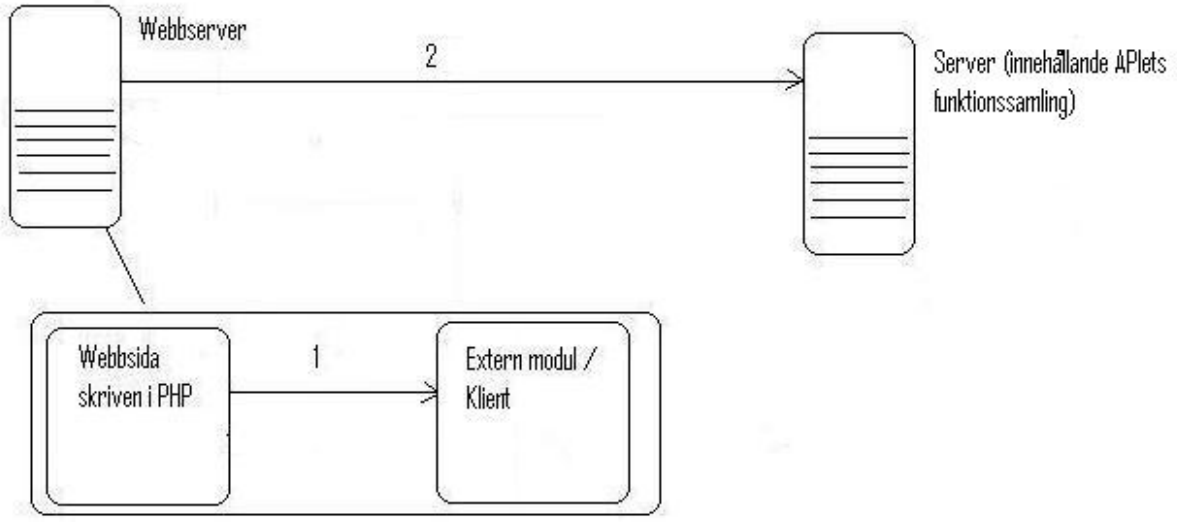


Figur 7: Illustrering av samband mellan klient och server

- Valideringen av data kommer att finnas på klientsidan i klient/serverlösningen. Anledning till att denna validering ligger på klientsidan är för att förhindra eventuella onödiga kommunikationer. Om data som skall skickas inte följer de kriterier som ställts på den så skall ett felmeddelande returneras till den anropande sidan i PHP. Klienten kommer att innehålla en funktion för att kontrollera att variabler håller sig inom ett bestämt intervall, denna funktion kan appliceras på de datavärden som skall kontrolleras. Klienten kommer även att innehålla en funktion för att kontrollera längd/storlek på textsträngar, denna funktion kan appliceras på de textsträngar som skall kontrolleras.
- Validering av användare kommer att ske både på klientsidan och på serversidan. Valideringen kommer att bestå av att ett unikt id för den besökande användaren på webbsidan paketeras tillsammans med de data som skall sparas åt användaren. Den datastruktur som används för paketeringen av de data som skall sparas för användaren kommer även att innehålla ett utrymme för att spara det unika id tillhörande användaren. Serversidan kommer att använda det för användaren unika id vid returnering av de sparade datastrukturerna vid en förfrågan om detta från klientsidan.

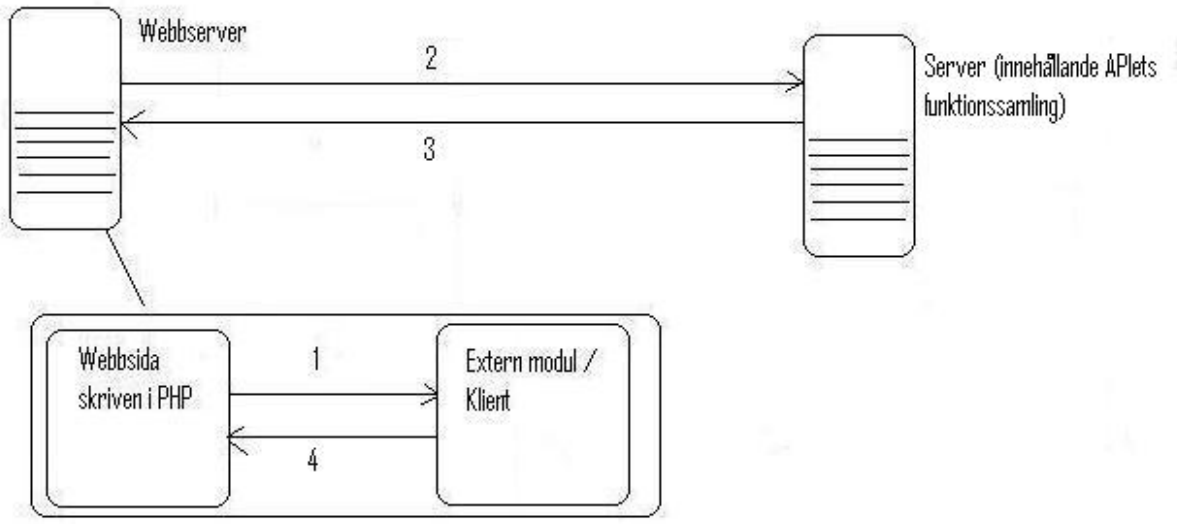
4.3 Klient

Klienten är skriven i programspråket C. Den klient som kommer att skapas är en extern modul innehållande funktionalitet för kommunikationen med både PHP och servern. Detta innebär att klienten i klient/server systemet kommer att fungera som ett gränssnitt mellan PHP och servern.



Figur 8: Illustrerar hur de olika delarna i konstruktionslösningen hänger ihop vid sparandet av data på servern.

1. Data (värden) skickas till den externa modulen från webbsidan skriven i PHP för paketering samt vidare transportering till servern.
2. Den externa modulen skapar en uppkoppling mot servern. De data som nu är paketerad i en datastruktur skickas över till servern för förvaring med hjälp av protokollet TCP/IP.



Figur 9: Illustrerar hur de olika delarna i konstruktionslösningen hänger ihop vid återhämtning av data från servern.

1. En begäran av återhämtning av sparad data skickas från en webbsida skriven i PHP till den externa modulen.
2. Den externa modulen skapar en uppkoppling mot servern. En datastruktur innehållande endast ett unikt id skickas över till servern med hjälp av TCP/IP protokollet.
3. Servern återhämtar den sparade datastrukturen som innehåller det matchande unika id och skickar detta till den externa modulen med hjälp av protokollet TCP/IP.
4. Den externa modulen paketerar upp den nu returnerade datastrukturens innehållande data och returnerar dessa till en webbsida skriven i PHP.

Den paketering av data som kommer att ske är att variabler och strängar som PHP skickar till den externa modulen kommer att läggas i en datastruktur tillsammans med ett unikt id som identifierar vilken användare (besökare på webbsidan) data tillhör innan den skickas över till servern för förvaring. Den datastruktur som kommer att användas är definierad på följande sätt:

```
typedef struct data{
        int     status;  //identifiera om det är skicka eller hämta som skall utföras
        int     id; //identifierare
        int     option; //vilket sätt att spara som skall användas, tex primärminnet
        int     value;  //utrymme för att spara ett värde
        char    text[255];  // utrymme för att spara en sträng, max 256 tecken lång
}Data;
```

De datavärden som skall sparas läggs in på sin korrekta plats i datastrukturen, till exempel så sparas en textsträng i text[255] (en text sträng innehållande max 256 tecken) och det unika id för användaren i id.

När data är placerad i datastrukturen kommer den skickas till servern med hjälp av protokollet TCP/IP.

4.4 Server

Serverdelen är skriven i programspråket C. Serverns uppgift är att spara de data som skickas från klienten samt att returnera de sparade data vid en förfrågan från klienten. När servern har mottagit dessa data i form av datastrukturer kommer de att sparas i primärminnet. De kommer att sparas i primärminnet genom att datastrukturen placeras i en indexerad tabell med ett unikt id som nyckel till rätt position i den indexerade tabellen för just denna datastruktur. Detta unika id korresponderar till en unik användare (besökare av webbsidan).

5. Implementation

I detta kapitel kommer konstruktionslösningens implementation att tas upp, hur problem som uppkommit under implementationen lösts och vilka kontinuerliga tester under körning som ansågs nödvändiga.

5.1 Beskrivning av filer

Klient/serverkonstruktionen kommer att innehålla 4 stycken filer:

1. Headerfil för den externa modulen (api.h)
2. Den externa modulen/klienten (api.c)
3. Serverapplikationens header fil (api-daemon.h)
4. Serverapplikationen (api-daemon.c)

5.1.1 api.h

Denna fil innehåller all inkludering av extern funktionalitet som behövs för att api.c skall fungera korrekt, definitionen av den datastruktur som används för att paketera data, definitionen av funktioner som skall exporteras till PHP och definiering av globala variabler. Inkluderingen av externa filer utlämnas här, om läsaren så önskar kan dessa studeras i källkoden.

Definitionen av de funktioner som skall exporteras till PHP kan se ut som följande exempel:

```
/*Funktionslista för exportering*/
    PHP_FUNCTION(addData);
    PHP_FUNCTION(getData);
```

Funktionerna vars namn är addData och getData definieras, källkoden för dessa funktioner återfinns i filen api.c under funktionsimplementation.

```
/*definiering av globala variabler */
    #define PORT 2001       //vilken port som servern skall kontaktas på
    #define DEBUG 1         //enbart för testning
    #define TRUE 1          //kontroll värde
    #define FALSE 0         //kontroll värde
    #define IP "127.0.0.1"  //serverns IP, här datorns interna IP
    #define REQ -1          //används vid hämtning av data
    #define POST 1          //används vid skickande av data
    #define RAM 1           //för sparning i primärminnet

    /* datastrukturen som används för paketering av användardata */
    typedef struct data{
            int         status;  //identifiera om det är skicka eller hämta som skall utföras
            int         id; //identifierare
            int         option; //vilket sätt att spara som skall användas, tex primärminnet
            int         value;  //utrymme för att spara ett värde
            char        text[255];  // utrymme för att spara en sträng, max 256 tecken lång
    }Data;
```

5.1.2 api.c

Filen api.c genereras av den i PHP5 medföljande automatiserade modulgenereringssystemet . Från och med PHP4 innehåller PHP ett automatiskt modulgenereringssystem, detta skapar alla nödvändiga filer för en inbyggd alternativt extern modul.

Följande delar av den skapade filen api.c är av intresse för utvecklingen av klient/server systemet.

Inkludering av filer

För att göra det så överskådligt som möjligt har det valts under detta projekt att endast inkludera filen api.h, för mer information om api.h se avsnitt 5.1.1, som i sin tur håller all ytterligare inkludering av filer och makron.

```
    /* inkludering av header filen api.h*/
    #include "api.h"
```

Funktionslista

Funktionslistan är en lista innehållande funktionsnamnen på de funktioner som skall vara möjliga att anropa från PHP. PHP tillhandahåller makron för definitionen av de funktioner som skall vara tillgängliga från PHP:

- PHP_FE(name,arg_types) – Definierar en funktion med namnet name som måste ha en korresponderande funktion med samma namn. arg_types skall alltid sättas till NULL.
- PHP_FALIAS(php_name,name,arg_types) – Definierar ett alias kallat name för php_name. Behöver inte ha en korresponderande funktion utan refererar name's källa istället. arg_types skall alltid sättas till NULL.
- PHP_NAMED_FE(runtime_name,name,arg_types) – Definierar en funktion som kommer att vara tillgänglig i PHP vid namnet runtime_name och denna korresponderar till en funktion vid namn name. arg_types skall alltid sättas till NULL.

[10]

En funktionslista kan se ut som följande exempel:

```
/* {{{ api_functions[]
 *
 * användarsynliga funktioner api_functions[].
 */
function_entry api_functions[] = {
        PHP_FE(addData,  NULL)      /* funktionsdeklaration */
        PHP_FE(getData,NULL) /* funktionsdeklaration */
{NULL, NULL, NULL}      /* Sista raden i api_functions[] */
};
```

Här definieras en funktion vid namnet addData och en funktion vid namnet getData, dessa funktioner måste ha korresponderande funktioner skapad i programspråket C (mer om detta under funktioner). {NULL, NULL, NULL} är terminering av funktionslistan och måste alltid ligga sist i listan.

Funktionsimplementation

Det kommer att skapas 2 huvudfunktioner, addData som används för att spara data på servern samt getData som används för att återhämta data från servern. Dessa funktioner innehåller funktionalitet för att få data från PHP samt returnera data till PHP. De kommer att anropa ytterligare funktioner för att få tillgång till funktionalitet för att validera data, paketera data, uppkoppling mot server och överföring av data till och från server.

PHP_FUNCTION(addData)

{

/* kod för närvarande under implementering samt testing, ej fullständig än just nu hanteras enbart strängar, en lista över vilken funktionalitet som kommer att vara tillgänglig ges här:

int validateData(…) – Ej implementerad än dock så kommer denna funktion kontrollera storleken på en textsträng samt kontrollera att värdet på en variabel befinner sig inom ett bestämt intervall om så önskan. Denna funktion kommer att returnera TRUE om det gått bra annars FALSE.

int pktDataForRam(char *text, int id) - lägger en användarens data (inparametern text) i en datastruktur samt lägger till ett unikt id (inparametern id) i datastrukturen för att kunna identifiera användaren. Sätter datastrukturens variabel option till önskat värde, variabeln option används för att kontrollera vilket sätt att spara data som önskas. Sätter datastrukturens variabel status till önskat värde, variabeln status används för att kontrollera om det är sparande av data eller hämtande av data som önskas. Returnerar TRUE om det gått bra annars FALSE.

int connectServer(void) – skapar en uppkoppling till servern och returnerar ”socketnummer”.

int sendData(int sockfd) – skickar data till servern. Variabeln sockfd innehåller det ”socketnummer” som kommer att användas. Returnerar TRUE om det gick bra annars FALSE.

*/ }

PHP_FUNCTION(getData)

{

/*kod för närvarande under implementation samt testing, en lista över vilken funktionalitet som kommer att vara tillgäng ges här:

int connectServer(void) – skapar en uppkoppling till servern och returnerar ”socketnummer”.

int getData(int sockfd, int id) – skickar en datastruktur till servern. Variabeln sockfd innehåller det ”socketnummer” som kommer att användas. Datastrukturen som skickas innehåller användarens unika id (inparametern id) samt så har datastrukturens variabel status satts till att hämta data. Klienten inväntar sedan svar från servern innehålland de data som begärts. Returnerar TRUE om det gått bra annars FALSE.

*/

/* uppackning av data sker direkt i funktionskroppen och behöver således inte någon egen funktion */

}

5.1.3 api-daemon.h

Denna fil innehåller all inkludering av extern funktionalitet som behövs för att api-daemon.c skall fungera korrekt. De delar av filen som tas upp här är definitionen av globala konstanter, definitionen av den indexerbara tabellen som skall spara datastrukturerna samt definitionen av datastrukturen Data.

```
/*globala konstanters definiering */
#define PORT 2001      //vilken port som servern skall kontaktas på
#define DEBUG 1        //enbart för testning
#define SIZE 20         //max antal unika id som kan sparas
#define REQ -1         //används vid hämtning av data
#define POST 1         //används vid skickande av data
#define RAM 1          //för sparning i primärminnet
#define TRUE 1         //kontroll värde
#define FALSE 0        //kontroll värde
```

```
/* datastrukturen som används vid mottagande av data från klient */
typedef struct data{
        int     status;  //identifiera om det är skicka eller hämta som skall utföras
        int     id; //identifierare
        int     option; //vilket sätt att spara som skall användas, tex primärminnet
        int     value;  //utrymme för att spara ett värde
        char    text[255];  // utrymme för att spara en sträng, max 256 tecken lång
}Data;

/* En indexerbar tabell av typen Data skapas, med SIZE (definierad som 20) antal platser
tillgängliga för att spara datastrukturer av typen Data*/
Data storage[SIZE];
```

5.1.4 api-daemon.c

Denna fil innehåller all funktionalitet för servern. Denna funktionalitet innefattar att hantera uppkopplingar från klienter, spara den data som skickas från klienten i primärminnet och returnera begärd data till klienten.

Det som kommer tas upp under detta avsnitt är de delar av filen som hanterar de datastrukturer som kommer från klientsidan till serversidan samt hur datastrukturer kan skickas tillbaka till klientsidan från serversidan. Då kommunikationen mellan server och klient är av mindre vikt, ty det enbart handlar om att transportera den paketerade data, kommer detta inte att tas upp.

Lagring av datastrukturer

När en datastruktur kommer till serversidan så behöver det kontrolleras om det är lagring av data eller återhämtning av data som skall utföras, för att kontrollera om det är lagring av data som skall utföras används följande test:

```
/*kontrollera om lagring är önskad samt vilken lagringsmetod som valts*/
if(data.status == POST){
switch(data.option){
                case RAM: storeInRam(data); //lagring i primärminnet
                                                break;
                        default:
                                                break;
                }
    }
```

data.status är den del av datastrukturen från användaren som anger om det är lagring av data eller hämtning av data som gäller. data.option är den del av datastrukturen som anger vilket sätt som datastrukturen skall lagras på, val av lagringsmetod väljs genom att värdet i data.option matchas mot olika fall i en ”switch-case” sats. Fallet RAM innebär att datastrukturen skall lagras i primärminnet. För att lagra datastrukturen i primärminnet används följande funktion:

```
/*lagra i primärminnet*/
int storeInRam(Data thisData){
memcpy(&storage[thisData.id],&thisData,sizeof(thisData));
return TRUE;
}
```

Funktionen storeInRam använder sig av memcpy (information om denna funktion återfinns i bilaga memcpy) för att kopiera datastrukturen till den indexerbara tabellen. Inparameteren thisData är den data som klienten skickade till servern för att bli lagrad.

Återhämtning av datastrukturer

När en datastruktur kommer till serversidan så behöver det kontrolleras om det är lagring av data eller återhämtning av data som skall utföras, för att kontrollera om det är återhämtning av data som skall utföras används följande test:

```
/*Om lagring inte var önskad så är det återhämtning av data önskad av klienten*/
else if(data.status == REQ){
        /* hitta rätt unika id*/
for(j = 0; j <= SIZE ; j++){
            if(data.id == j){ //temporär kontroll som kommer att ändras
                memcpy(&temp,&storage[j],sizeof(storage[j]));
            }
        } }
```

data.status är den del av datastrukturen från användaren som anger om det är lagring av data eller återhämtning av data som gäller. Om valet är att återhämta data så matchas det unika id som medföljde datastrukturen från klientsidan mot det unika id för den i primärminnet sparade datastrukturen. memcpy (information om denna funktion återfinns som bilaga memcpy) kopierar den i primärminnet lagrade datastrukturen till en temporär datastruktur som sedan skickas tillbaka till klientsidan.

5.2 Kontinuerliga tester under körning

APIet kommer att innehålla tester som utförs kontinuerligt under körning. De tester som kontinuerligt kommer att utföras under körning på klientsidan är:

- Kontroll av att variabelvärde befinner sig inom ett bestämt intervall (om så önskas av administratören för APIet).
- Validering av användare
- Kontroll av returvärde vid uppkoppling mot servern.
- Kontroll av returvärde vid sändning av data till servern.
- Kontroll av returvärde vid mottagning av data kommande från servern.

De tester som kontinuerligt kommer att utföras under körning på serversidan är:

- Validering av användare.
- Kontroll av returvärde vid skapandet av en socket.
- Kontroll av funktionen select(..) returvärde.
- Kontroll av datastrukturens variabler option och status.
- Kontroll av returvärde vid sändning av data till klienten.

6. Resultat

I detta kapitel kommer det att tas upp huruvida det skapade APIet uppfyller de krav som har specificerats i rapporten. Det kommer även tas upp problem med den valda lösningen.
De krav som fanns på APIet var:

1. Hantera lagring av variabler och strängar
2. Utvecklingsmöjligheter
3. Säkerhet
4. Användarvänlighet

6.1 Hantera lagring av variabler och strängar

Detta mål är delvis löst, det går i nuläget att lagra strängar från PHP i primärminnet och det är inga problem att vidareutveckla serversidan för att hantera ytterligare varianter av lagring. Det som saknas i vår lösning enligt de specifikationer som finns är att även lagringen av variabler skall vara möjlig. Att det i nuläget inte går att lagra variabler är helt och hållet en tidsfråga, implementationen av detta är delvis klar. Datastrukturen som används för lagring av strängar har även stöd för lagring av variabler i nuläget, dock så finns helt enkelt inte detta alternativ att välja vid lagringen än.

Ett av de stora problemen med lösningen som den är nu är att både klient och serversidan saknar dynamiska lösningar för hanteringen av data. På klientsidan hade en dynamisk datastruktur varit önskvärd då det nu är fasta maxgränser för hur stora dataelementen som lagras kan vara. På serversidan finns en maxgräns för hur många datastrukturer som kan lagras i minnet, detta hade också önskats vara dynamiskt. I detta problem innefattas också indexeringen av de datastrukturer som lagrats, vår önskan var att implementera en hash-tabell med ett unikt id som nyckel för varje lagrad datastruktur. Implementationen av detta har dock inte blivit av på grund av tidsbrist.

6.2 Utvecklingsmöjligheter

Detta krav har uppfyllts med råge, det finns stora möjligheter för vidareutveckling av lösningen i alla aspekter. Eftersom det skapats en klient/serverlösning som är nästan helt fristående från varandra så kan APIet skräddarsys vid både klientsidan och serversidan utan problem.

6.3 Säkerhet

Detta krav har inte uppfyllts än, implementationen av valideringen av både data och användare saknas till viss del. Valideringen av data finns det lösningar på, den är dock inte fullt testad än och kan därför inte räknas till att vara uppfylld i detta skede. Valideringen av användare är delvis implementerad, det som saknas är att bestämma vad som skall vara det unika id som behövs samt att anpassa datastrukturen efter detta. En följd av detta kan vara att implementationen av de valideringar som skall utföras på klient samt serversidan kan behöva justeras.

6.4 Användarvänlighet

Detta krav är delvis uppfyllt, användarbeskrivning för APIets funktionslista är klar i den mån funktionerna i funktionslistan är klara dock så saknas en utförlig användarbeskrivning för tillägg samt borttagning av funktioner i funktionslistan. Användarbeskrivningen för detta är inte klar på grund av att källkoden inte är klar, källkoden kan med andra ord komma att ändras till viss del efterhand och då kan eventuellt användarbeskrivningen inte längre vara aktuell.

7 Summering av projektet

IT Resource AB arbetslokaler befinner sig ca 10 mil från Karlstad, i staden Årjäng.
På grund av det stora avstånd mellan oss och uppdragsgivaren är arbetsformen som valts under projektet att arbeta på distans. Att arbeta på distans medför vissa fördelar och nackdelar.

De nackdelar som distansarbete medför är att man inte kommer i lika bra kontakt med företagets miljö samt anställda, man får en sämre inblick i hur företaget fungerar, det blir svårare att få svar på småfrågor som dyker upp under arbetets gång. Det blir även mindre diskussioner om arbetet eftersom det är lättare att diskutera när man träffas än att ta det genom telefonsamtal eller e-post.

De fördelar som distansarbete medför är att man inte är lika bunden av de arbetstider som gäller på företaget och kan i högre grad bestämma vilka tider man vill arbeta, man får även ett ökat ansvar för att ens egen arbetsmiljö skall fungera.

Från början hade vi tänkt göra en utvärdering av SRM istället för att skapa ett eget API. SRM skulle kunna lösa problemet med att en webbsida skriven i PHP tappar sina temporära variabler och strängar vid uppdateringen av webbsidan och på grund av detta så verkade SRM vara en bra lösning för IT Resource AB. Allt eftersom arbetet fortskred insåg vi att SRM inte var den bästa lösningen att använda. SRM är en applikation som inte är tillfullo dokumenterad, detta kan till stor del bero på att SRM fortfarande befinner sig i betastadiet. Ytterligare faktorer som spelade in i att detta inte kanske var en bra lösning var dess komplexitet, SRM var väldigt svårt att förstå till den nivå som behövs för att kunna vidareutveckla det.

När vi insåg att detta inte alls var den bästa lösningen, utan snarare en onödigt komplex och svår lösning, så valde vi att stanna upp med den lösningen och gå vidare på ett annat sätt.
Det som man ångrar nu i efterhand var att vi inte valde att gå vidare med en annan lösning tidigare än vad vi gjorde. Det hade varit en klar fördel om vi hade kommit igång med den andra lösningen tidigare än vad vi gjorde.

Det vi har kommit fram till under projektets gång är att vi kan med fördel utöka PHPs funktionalitet genom antingen inbyggda moduler eller externa moduler. Vi valde att gå vidare med en utökning av PHP genom externa moduler.

Den konstruktionslösningen som vi har använt oss av för att skapa APIet är en klient/serverlösning. I den lösningen kan man betrakta den externa modulen som klientdelen och serverdelen kan betraktas som APIets funktionssamling. Klienten fungerar som ett gränssnitt mellan PHP och servern. Klient/serverlösningen som vi skapade är bra eftersom det är enkelt att utöka funktionaliteten i APIet, det vill säga både i den externa modulen och i APIets funktionssamling.

Det största återkommande problemet under projektets gång är utan tvekan avsaknaden av erfarenhet att skriva större rapporter som denna, detta är också utan tvekan den mest värdefulla erfarenhet som vunnits under projektets gång.

Referenser

[1] http://www.itresource.se/ 2006-01-02

[2] http://www.nada.kth.se/dataterm/fkt.html#d20 2006-01-02

[3] http://se2.php.net/manual/sv/faq.general.php 2006-01-05

[4] http://www.zend.com/store/products/zend-engine-in-depth.php 2006-01-02

[5] http://se2.php.net/manual/sv/zend.possibilities.php#zend.possibilities.external 2006-01-02

[6] http://se2.php.net/manual/sv/zend.possibilities.builtin.php 2006-01-02

[7] http://se2.php.net/manual/sv/zend.possibilities.engine.php 2006-01-02

[8] http://www.vl-srm.net/ 2006-01-02

[9] http://www.crmnytt.com/intro.htm 2006-01-04

[10] http://se2.php.net/manual/sv/zend.structure.function-block.php 2006-01-05

[11] http://www.vl-srm.net/pres/linuxtag/img0.html 2006-01-05

[12] Deitel & Deitel. How To Program C++, second edition 1997

[13] Linux Programmerâs Manual 1993-04-10

Bilagor

Förkortningslista

API, Application Program Interface, ett gränssnitt som gör det möjligt att i program utnyttja funktioner som finns tillgängliga i ett annat program eller i en funktionssamling.

CRM, Customer Relationship Management, en affärsstrategi för att hantera kunder.

LAMP, (Linux, Apache, MySQL, PHP), en öppen källkodslösning som använder Linux som operativsystem, Apache som webbserver, MySQL som relationsdatabas och PHP som programspråk.

PHP, PHP: Hypertext Processor, ett skriptspråk som bygger på en öppenkällkodlösning.

SRM, Script Running Machine, en programvara som kan användas för att utöka PHPs funktionalitet.

memcpy

NAME

memcpy - copy memory area

SYNOPSIS

#include <string.h>

void *memcpy(void *dest, const void *src, size_t n);

DESCRIPTION

The memcpy() function copies n bytes from memory area src to memory area dest. The memory areas may not overlap. Use memmove(3) if the memory areas do overlap.

RETURN VALUE

The memcpy() function returns a pointer to dest.

[13]