Webbutveckling med PHP

Hur du uppfyller projektkrav

Skriven av: Wictor Kihlbaum (wk222as)

Inledning

Det primära steget innan själva programmeringen i ett projekt kan påbörjas är att komma på en stabil grundidé. Idén ska gärna lösa ett till flera problem för en eller flera personer. Min idé handlar kortfattat om att samla ihop de senaste utgivna nyheterna från flertalet webbsidor på en och samma webbplats. Gällande dessa nyheter valde jag i mitt projekt att specificera mig på webbsidor vilka publicerar spelrelaterade nyheter. Dessa typer av nyheter är något jag själv mer än gärna läser. Jag valde att gestalta min idé i form av en webbsida för att samla in samtliga av dessa inriktade nyheter. Min idé gynnar och underlättar sedermera nyhetsläsningen för människor vilka potentiellt besöker flertalet olika spelnyhetssidor. Oavsett om mina användare besöker nyhetssidor flera gånger dagligen eller enbart ett fåtal gånger i veckan kommer de, genom att jag samlar allt på en och samma plats, i synnerhet spara mycket tid som istället kan nyttjas till annat. Detta är en bekvämlighet jag tror många skulle uppskatta.

Användar-Fall

Det första konkreta jag visste att jag ville inkludera på min webbsida var en undersida där man kan samla nyheter från samtliga webbsidor. Dessa nyheter skall läsas in via RSS-flöden vilka sparats i en databas av typen "MySQL". När grundstrukturen samt själva navigationsmenyn, vilken möjliggör för användaren att ta sig mellan de olika undersidorna, väl var utförd inledde jag arbetet med ovan nämnda implementation. Sedan tidigare visste jag att jag på denna undersida ville möjliggöra för användaren ett antal inställningar vilka skall ge denne en så pass bra upplevelse som möjligt. Dock ville jag inte överösa användaren med massa inställningar. Detta främst på grund av två saker. Dels hade detta troligare resulterat i förvirring, och därmed tidsåtgång, innan man kan börja glädjas åt samt ta del av själva nyheterna, och dels hade detta resulterat i ett flertal fält på sidan vilket hade ingett en spontan känsla av såväl klottrighet som otydlighet. ”Mindre är mer” var ett koncept jag valde att följa när det kom till dessa användarinställningar.

De inställningar som implementerades var dels möjligheten att ändra antalet webbsidor som nyheter skall hämtas ifrån, och dels antalet nyheter som skall visas från var och en av dessa valda webbsidor. Dessa ovan nämnda inställningar placerade jag högst upp på undersidan vid namn "Newsfeed" för att man som användare skall uppmärksamma dessa först. En kortfattad kompletterande text placerades i närheten för att inge ytterligare klarhet i hur man som användare skall

interagera med dessa inställningar. Inställningarna var något jag valde att presentera och programmera som rullgardinsmenyer.

Eftersom antalet webbsidor samt antalet nyheter från dessa är ett förbestämt antal alternativ med fasta värden ansåg jag att det mest användarvänliga var att placera dessa alternativ i tydliga listor. I det fall mängden webbsidor och/eller nyheter istället varit helt valfritt hade givetvis ett bättre val varit att låta användaren mata in sitt val i klartext i ett textfält.
En extra inställning jag implementerade på denna undersida var att på samtliga av fälten vilka presenterar nyheter placera en rullgardinsmeny innehållandes de övriga namnen på nyhetssidorna. Detta för att man som användare direkt skall kunna ändra till en önskad webbsida utan att behöva bläddra längst ned på sidan.

Samtliga av ovan nämnda inställningar kräver att man som användare även klickar på en kompletterande knapp efter att man gjort sitt val i någon av de olika rullgardinsmenyerna. Detta för att den eller de inställningar man valt skall uppdateras och träda i kraft. Det var ett medvetet val från min sida. Ett än mer användarvänligt tillvägagångssätt hade varit att skippa denna typ av knapp överhuvudtaget för att istället låta sidan automatiskt uppdatera sig vid någon sorts ändring. Som jag förstod det vid vidare undersökning var att detta enbart går att lösa med programmeringsspråket JavaScript. Detta var något vilket inte var tillåtet att implementera i sitt projekt i och med att fokus enbart skulle ligga på språket PHP.
På grund av detta skäl valde jag att låta en uppdateringsknapp implementeras till var och en av de olika inställningsmöjligheterna. Man kan även argumentera för att detta skulle vara mer användarvänligt än en automatisk uppdatering med tanke på att användaren mycket väl eventuellt klickar fel alternativt ångrar sig i sista sekund. På det vis vilket det är utformat nu gör detta inget och ingen onödig tid samt beräkningskraft spenderas på att ladda in innehåll vilket användaren ändå kommer ändra snarast möjligast.

En mycket vanlig typ av undersida vilken återfinns på samtliga seriösa webbplatser är en kontaktsida. Detta var något även jag valde att implementera på min webbsida. Främsta anledningen till detta var att jag ville möjliggöra för besökare att kunna kontakta mig i syfte att rapportera eventuella buggar och tekniska problem vilka de upplevt. En ytterligare anledning till denna kontaktsida var även i syfte att kunna meddela mig om man som besökare önskar fler typer av nyhetssidor att bläddra och läsa bland.
Själva kontaktsidan är utarbetad i form av ett formulär där användaren fyller i ett antal fält. Det som krävs för att kunna kontakta mig är att man som användare i fälten fyller i sitt namn, sin mailadress, själva meddelandet man vill sända samt avslutningsvis svarar på en matematiskt slumpad fråga i syfte att förhindra eventuell spam.
En ytterligare sak vilken ingår i formuläret är själva ämnet för meddelandet. Detta är dock utformat, likt inställningarna på undersidan för nyheter, som en rullgardinsmeny

med ett redan förvalt ämne. Detta medför att meddelandet ändå skickas iväg på korrekt vis vid händelse av att användaren glömmer välja ett passande ämne. Anledningen till att ämnena är kategoriserade och inte går att skriva ut i klartext är främst i syfte att underlätta bearbetningen av respons för mig som utvecklare. Att på enkelt vis kunna hitta bland de olika meddelandena med dess innehåll och önskningar om förbättring samt förändring är att föredra. I och med denna kategorisering av meddelanden ger man sig själv möjlighet att kunna prioritera dem på ett betydligt effektivare och bättre vis. Prioritering av buggar, vilka eventuellt i värsta fall orsakar att nyheter inte visas korrekt, står högre i grad än exempelvis önskningar om implementation av fler nyhetssidor. Det viktiga är att samtliga besökare får en så pass angenäm upplevelse som möjligt.

Startsidan riktar sig främst till de besökare vilka dagligen besöker min webbsida. Det främsta syftet med startsidan är att presentera den absolut senaste nyheten från de olika nyhetssidorna. Om man dessutom är en registrerad samt inloggad användare som på sin egen profilsida valt en favorit bland nyhetssidorna så kommer man kunna läsa de senaste fem nyheterna från denna överst på startsidan. Om man dessutom är en registrerad samt inloggad användare så kommer man överst på startsidan även kunna läsa de senaste fem nyheterna från sin favorit bland nyhetssidorna. Favoritsidan är något man ställer in och sparar på sin profilsida.
Grundtanken med startsidan är således att kunna presentera den absolut senaste nyheten för någon som ofta, möjligen dagligen, uppdaterar sig bland nyheterna och därmed inte är intresserad av de, i förhållande, ”gamla” och äldre nyheterna.
Grundtanken med undersidan "Newsfeed”, som samlar flertalet av de senaste nyheterna från samtliga databas-sparade nyhetssidor, är snarare att erbjuda de besökare vilka kanske enbart veckovis går in och läser ska få en så pass översiktlig, sammanfattande och god upplevelse samt läsning som möjligt.

Test-process

I och med att programmeringsspråket PHP i grundutförande presenterar en vit och därmed totalt blank sida för programmeraren i det fall ett eller flera problem uppstår har jag under totala utvecklingsprocessen låtit samtliga felmeddelanden vara påslagna i syfte att ge mig chansen till att rätta till saker vilka gått fel. Detta är något jag ställer in i koden på Index-sidan för att upptäcka samtliga fel jag som programmerare och utvecklare råkar utföra ur en ren programmeringsaspekt.

När det kommer till fel vilka kan komma att uppstå som inte nödvändigtvis ligger i mina händer som utvecklare har ett antal undantag av diverse slag implementerats i koden. Dessa har främst implementerats för att undvika eventuella buggar.
De flesta undantagen är av typen ”Extended Custom Exceptions”, och därmed också namngivna efter deras specifika situation. Dessa kastas vid främst två olika typer av situationer. En av dessa situationer är då någonting går fel vid kommunikationen mellan webbsidan och databasen. Detta kan exempelvis ske vid inläsningen och hämtning av nyheter. Men det kan även röra sig om själva uppkopplingen mot

databasen i sig. Redan vid anrop mot databasen kan det gå fel av en eller flera olika orsaker. Det kan vara så ”enkelt” som att databasen just då ligger nere eller är under uppdatering/ombyggnad. Det kan också bero på att den tjänst som står för databasen, i mitt fall Microsoft Azure, har felat. Oavsett vad felet är kommer anropet med största sannolikhet misslyckas och därför är det mycket bra att på ett eller flera sätt meddela såväl användaren som webbutvecklaren vad som gått fel.

Den andra situationen är när användaren glömt alternativt missat att fylla i ett eller flera fält av information. Detta kan vara vid försök av inloggning eller registrering, men även vid försök till att skicka iväg ett kontaktformulär. Det handlar således om information vilken koden jag skrivit väntar sig för att kunna gå vidare i sin exekvering. De implementerade undantagen existerar följaktligen för att undvika att koden kraschar och förstör hela upplevelsen för användaren. Istället kastas ett eller flera undantag som för användaren presenteras i form av ett översiktligt och förklarande felmeddelande på lämpligt ställe på webbsidan. Presentationen av dessa felmeddelandena sker ofta intill det eller de fält vilka har uteblivande information. I det fall situationen berör någon typ av inmatning det vill säga.

Säkerhet

En viktig implementation i detta projekt ur en ren säkerhetsaspekt är att jag dels sparar användaruppgifterna vid registrering i en databas, och dels att en användares valda lösenord "hashas" innan det sparas i databasen. Detta är en stor förbättring gentemot ett av mina tidigare arbeten bland laborationerna då jag sparade en användares uppgifter dels i klartext, och dels enbart i en simpel textfil vilken dessutom gick att komma åt via adressfältet i webbläsaren. Detta är självfallet oförlåtligt i det fall någon skulle komma åt dessa uppgifter. Att skydda sina användares uppgifter är av yttersta och största vikt när det gäller att utveckla webbplatser vilka hanterar och kräver denna typ av information vid exempelvis registreringar och inloggningar.
Tillvägagångssättet att "hasha" lösenordet vid registrering är att föredra framför exempelvis kryptering. Syftet med kryptering är att också ge sig möjligheten att dekryptera. Detta är något man inte vill skall gå att utföra med lagrade lösenord i databasen. Den stora fördelen med "hashning" är att det inte går att återställa något man sparat med den metoden. Detta är en stor och mycket viktig säkerhetsåtgärd som försvårar att knäcka eventuellt läckta lösenord.

Det man riskerar utsätta sig för i det fall man sparar känslig data i klartext i databasen är för så kallad "Sensitive Data Exposure". Detta problem innebär helt enkelt att man inte krypterat den känsliga datan. I det fall känslig data, så som kreditkort, ID och verifikation uppgifter, inte skyddas ordentligt kan det innebära att angripare stjäl eller modifierar den känsliga och oskyddade datan. Detta för att i sin tur begå kreditkortsbedrägeri, identitetsstöld eller andra brott. Känslig data förtjänar extra skydd så som kryptering och speciella försiktighetsåtgärder.

Misslyckande i att skydda känslig data äventyrar ständigt ALL data som borde varit skyddad. Denna känsliga data innefattar bland annat sjukjournaler, användaruppgifter, personlig data och kreditkort. I detta specifika fall rör det sig om användarnamn samt tillhörande lösenord.
Rörande skydd av känslig data finns ett antal punkter vilka samtliga skall utföras. Man bör tänka över de olika hoten man planerar skydda sin data från, såväl interna som externa angrepp, för att sedan försäkra sig om att kryptera all känslig data på sätt som skyddar mot dessa angrepp. Man skall heller inte lagra känslig data i onödan. Det skall förkastas snarast möjligast. Data du inte innehar kan heller inte stjälas från dig. Som utvecklare skall man även se till att starka standard algoritmer och starka nycklar används, men också ordentlig nyckelhantering. Exempel på vad som kan användas är "FIPS 140 validated cryptographic modules". Det skall försäkras om att lösenord lagras genom en algoritm specifikt utformad för lösenordsskydd. Exempel på dessa är "bcrypt", "PBKDF2" och "scrypt". Ett par saker att föredra är även att dels inaktivera autofyll i HTML-forms vilka samlar känslig data, och dels inaktivera cachning för sidor vilka innehåller känslig data.
En viktig sak att tänka på, som kan ge en såväl bra som skrämmande bild av vad som kan hända ifall en eller flera användares uppgifter läcker ut, är att en användare varierar mycket sällan sina uppgifter mellan de många olika sidor denne besöker. Vilket i praktiken egentligen innebär att en människas konton från flertalet webbplatser nu är i en otroligt stor riskzon. Även ifall användaren inte har exakt samma lösenord på de olika sidor denne besöker lär det med största sannolikhet vara ett gäng smärre variationer av det läckta lösenordet, vilket i sin tur innebär att arbetet med att knäcka dessa är otroligt mycket lättare.

En annan sak jag implementerat i mitt projekt är i syfte att motverka spam-mail via min kontaktsida vilken erbjuder ett kontaktformulär för användaren. Det sista fältet vilket måste fyllas i från användarens sida för att kunna skicka iväg sitt meddelande är att besvara en så kallad "anti-spam"-fråga. I mitt specifika fall innebär detta en slumpad matematisk fråga som måste besvaras. Denna säkerhetsimplementation är till hjälp för att undvika spam-mail. Detta är för att motverka och försvåra arbetet för exempelvis olika webbskrapor som skulle kunna användas för att skicka iväg oönskad post. Det finns möjligen bättre alternativ för att motverka spam-mail, men detta är ett synnerligen enkelt sätt att göra det på.

I syfte att undvika attacker mot databasen med så kallade "SQL Injektioner" använder jag mig av lagrade procedurer som jag kallar på istället för dynamiska så kallade "queries" direkt i koden. Injektioner är ett problem som innebär att injektionsbrister, likt SQL, OS och LDAP, sker när ej betrodd data skickas till en interpreterare som del av ett kommando eller "query". Angriparens fientliga data kan lura interpreteraren till att exekvera oavsiktliga kommandon eller ge åtkomst till data utan ordentlig auktorisering. Dessa injektioner kan resultera i dataförlust, korruption eller nekad åtkomst. Injektioner kan ibland leda till totala övertaganden.

Att förebygga injektioner kräver att man håller opålitlig data separerat från kommandon och "queries". Det föredragna valet är att använda ett säkert API vilket antingen undviker användandet av en interpreterare helt och hållet eller förser ett parametriserat gränssnitt. Man skall dock vara försiktig med APIs som har lagrade procedurer och är parametriserade eftersom de fortfarande kan introducera injektioner under huven. I det fall ett parametriserat API inte finns tillgängligt skall man försiktigt städa undan specialtecken genom att använda den specifika syntaxen för den interpreteraren. Positiv eller "white list" inmatningsvalidering är även det rekommenderat. Det är dock inte alltid ett komplett skydd eftersom man i sin applikation kan kräva specialtecken i inmatningsfältet.