SimThebl Portfolio – Installationsmanual

SimThebl Portfolio är ett enkelt, lättviktigt presentationssystem för projekt på webben. Kom igång med din webb-portfolio direkt genom att följa dessa steg. Det enda som krävs är att du kör ett Linux-system du är bekväm med och har Python 3.3 och Flask 0.10 installerat (är detta inte fallet följer separata instuktioner senare i dokumentet). Nu kör vi!

1. Ladda ner simtheblportfolio.tar.gz till den katalog där du vill köra SimThebl Portfolio.
2. Kör **`tar -xf simtheblportfolio.tar.gz`** och ställ dig i mappen med **`cd simtheblportfolio`**
3. För att starta webbservern, kör: **`python3.3 server.py &`**
4. Nu kan andra nå din sida via din publika IP-adress.

Har du inte redan Python 3.3 och Flask 0.10 installerat på ditt Linux-system? Gör så här:

1. Gå till http://www.python.org/download/releases/3.3.2/, ladda ner och installera Python 3.3.
2. Gå till http://flask.pocoo.org/ och ladda ner Flask version 0.10 och packa upp tarballen.
3. Installera Flask till Python 3.3 genom att ställa dig i mappen och kör **`sudo python3.3 setup.py install`**.
4. Behöver installationen ytterligare program (t.ex. setuptools) för att slutföras kan du installera dem på samma sätt. Var bara noga med att använda Python 3.3 till samtliga program så kommer allt gå bra.
5. Nu är du redo!

SimThebl Portfolio – Användarmanual

SimThebl Portfolio består av följande sidor:

1. Startsida med information om sidans skapare, en välkomstskärm samt några små bilder från några av projekten.

2. En sida som listar alla projekt med kort och allmän information. Har även möjligheter för att söka i och filtrera projekten.

3. Projektsida unik för varje projekt, här visas allt om ett projekt upp.

4. En sida som listar alla tekniker i projektdatabasen och visar vilka projekt som innehåller respektive teknik.

Startsidan's innehåll är bl.a. ett sökfält. Klickar man på sök hänvisas man till /list.
Här visas bilder och information om sidans skapare.
Om man håller musen över någonting, visas det i en större, tydligare ruta.
Om man klickar på något hänvisas man till motsvarande projekts sida.

Söksidan, `/list`, visar alla projekt som matchar en sökning, eller alla projekt om inget sökord angetts.

Här finns kryssrutor för att bestämma vilka aspekter hos ett projekt man vill söka på och för vilka tekniker ett projekt måste ha för att visas. Det finns även en kryssruta för att bestämma om projekt ska visas i stigande eller fallande ordning baserat på en rullgardinsmeny som innehåller alla aspekter ett projekt kan ha.

OBS! Endast projekt som innehåller just de ikryssade teknikerna kommer att visas, om inte alla teknik-kryssrutor är avmarkerade, då de ignoreras.

Om ett sökord angetts på startsidan, eller i en tidigare sökning i /list, sparas det samt tillståndet för kryssrutorna till nästa sökning.

Klickar man på ett projekt i sökresultatet hänvisas man till det specifika projektets sida.

Projektsidan,`/project/<projectID>`, visar upp all information som finns tillgänglig för det aktuella projektet. Det innebär bilder, en kort/lång beskrivning av projektet, tekniker som använts osv.

Tekniksidan, `/techniques`, visar alla tekniker som finns i databasen. Om man håller musen över en teknik visas alla projekt som tekniken ingår i i en större, snyggare ruta.

Övrigt, allt som händer på servern loggas i en fil. för att lägga till eller ta bort projekt används ett admin-inlogg med admin-sidor.

SimThebl Portfolio – Systemdokumentation

SimThebl Portfolio är ett system för att lagra och presentera data om olika projekt på webben. Systemet är uppbyggt av två olika delsystem; ett datalager skrivet i Python 3.3 och ett presentationslager, som fungerar i symbios med varandra. Resultatet är en hemsida, bestående av fyra delsidor med dynamiskt innehåll, skrivna med HTML5, CSS och Jinja2. Systemet låter användaren navigera i en databas innehållande olika projekt med hjälp av webbläsaren. Man tillåts söka bland projekt, lista projekt enligt olika kriterier och sorteringsalternativ med mera. Denna sektion är till för att erbjuda en djupare förståelse för hur systemet är uppbyggt, så att eventuella förändringar eller tillägg underlättas.

Datalagret (Python 3.3, json)

Datalagret är vad man kan kalla för själva kärnan i systemet, eller åtminstone det som interagerar med systemets kärna: en json-fil (i detta fall data.json) där informationen om varje projekt finns lagrad. För att hämta information ur denna json-fil erbjuder datalagret från början en rad funktioner som gör detta, anpassat till olika behov. Eftersom datalagret är skrivet i Python går det även relativt snabbt och enkelt att skriva till fler funktioner efter behov, förutsatt att man har grundläggande kunskaper inom Python 3.3. Här följer en beskrivning av de olika funktionerna som finns inkluderade (i `data.py`):

- **`load(file_name)`**
  Tar emot namnet på en json-fil som en sträng, öppnar och kodar av filen och returnerar innehållet som ett listobjekt. Returnerar `None` om filen inte kan öppnas.

- **`add_project(db,file_name, project)`**
  Tar emot en projektdatabas som ett listobjekt, namnet på en json-fil som en sträng, samt en tabell som representerar ett projekt. Därefter tar den reda på det högsta projekt-ID:t i projektdatabasen och sätter ett nytt unikt projekt-ID som är högre. Därefter lägger den till det nya projektet på slutet av databasen med det nya ID:t och skriver den nya databasen till en json-fil med det givna filnamnet. Gick allt bra returneras listobjektet. Finns inte något matchande projekt eller om filen inte hittades returneras `None`.

- **`remove_project(db, file_name, ID)`**
  Tar emot en projektdatabas som ett listobjekt, namnet på en json-fil som en sträng, samt ett projekt-ID som ett heltal. Om ett projekts ID matchar det angivna tas det bort ur listan, varpå filen skrivs över med det nya listobjektet. Gick allt bra returneras listobjektet. Finns inte något matchande projekt eller om filen inte hittades returneras `None`.

- **`edit_project(db, file_name, ID, project)`**
  Tar emot en projektdatabas som ett listobjekt, namnet på en json-fil som en sträng, ett projekt-ID som ett heltal, samt ett projekt i tabellform. Sedan hittar funktionen projektet i databasen med motsvarande ID och ersätter det med det givna projektet. Sedan skrivs den nya databasen till en json-fil. Gick allt bra returneras listobjektet. Finns inte något matchande projekt eller om filen inte hittades returneras `None`.

- **`get_project_count(db)`**
  Tar emot ett listobjekt. Returnerar antalet inlägg i listan som ett heltal.

- **`get_project(db, id)`**

Tar emot en projektdatabas som ett listobjekt och ett projekt-ID som ett heltal. Returnerar hela tabellen för projektet vars ID motsvarar det angivna. Hittas inte projektet returneras `None`.

- **`search(db, sort_by=u'start_date', sort_order=u'desc', techniques=None, search=None, search_fields=None)`**
  Tar emot en projektdatabas som ett listobjekt, en post i projekttabellen att sortera efter (som ett strängobjekt), en sträng som bestämmer om sorteringen kommer att vara fallande eller stigande, ett listobjekt innehållande tekniker, en sträng med fritext och ett listobjekt innehållande fält att söka i.

  Returnerar ett listobjekt innehållande de projekttabeller som matchar sökningen.

  Funktionen använder sig av tre underfunktioner för att stegvis filtrera bort oönskade projekt ur sökresultatet. Alla indata i denna funktion, förutom projektdatabasen, är frivilliga. Resterande indata har standardvärden som representerar den mest troliga sökningen. Dvs att träffarna kommer att innehålla alla projekt och automatiskt sorteras efter startdatum i fallande ordning. Alla projekt visas i detta fall eftersom att standardvärdet för `techniques`, `search` och `search_fields` är `None` och respektive filtreringsfunktion bara körs om värdet inte är `None`. De tre underfunktionerna beskrivs nedan:

  - **`tech_search(search_result, remove_result, techniques)`**
    Tar emot ett sökresultat som ett listobjekt, en lista över projekt att ta bort ur sökresultat-listan samt ett listobjekt innehållande tekniker. Lägger till de projekt, som inte innehåller alla tekniker från tekniklistan, i ta-bort-listan.

  - **`free_search(search, search_result, remove_result)`**
    Tar emot ett sökord som en sträng, sökresultat som ett listobjekt samt en lista över projekt att ta bort ur sökresultat-listan. Lägger till alla projekt från sökresultatet, som inte innehåller sökordet, i ta-bort-listan.

  - **`field_search(search_result, search, search_fields, remove_result)`**
    Tar emot ett sökresultat som ett listobjekt, ett sökord som en sträng, ett listobjekt innehållandede fält som ska genomsökas samt en lista över projekt att ta bort ur sökresultat-listan. Lägger till de projekt, där innehållet i de angivna fälten inte matchar sökordet, i ta-bort-listan.

    Denna delfunktion innehåller två sökalgoritmer. Den första och vanligaste används för att genomsöka fält som innehåller strängar. I detta fall alla fält utom `'techniques_used'` som innehåller en lista. När funktionen stöter på detta fält körs algoritmen för att söka igenom en lista.

- **`get_techniques(db)`**
  Tar emot en projektdatabas som ett listobjekt. Returnerar en sorterad lista innehållande alla tekniker som finns listade i databasen.

- **`get_technique_stats(db)`**
  Tar emot en projektdatabas som ett listobjekt. Returnerar en tabell med de olika teknikerna i

databasen som nyckelvärden. Varje post innehåller en lista över projekttabeller i bantad form, där nyckelvärdets teknik har använts. Den bantade projekttabellen innehåller bara posterna med nyckelvärdena `project_name` och `project_no`.

- **`get_random_projects(db)`**
  Tar emot en projektdatabas som ett listobjekt. Väljer ut ett slumpmässigt projekt i listan och returnerar projektets bild, namn och kortfattade beskrivning som tre strängobjekt.

Presentationslagret (Python 3.3, Flask, Jinja2, HTML5, CSS)

Presentationslagret är användargränssnittet i systemet; dvs att det låter användaren använda funktionerna i datalagret med hjälp av enkla knapptryckningar i webbläsaren. Presentationslagret är ett hopkok av tre olika tekniker. Python 3.3 används för att köra lättvikts-webbservern Flask och låta den kommunicera med datalagret. Själva webbsidorna är byggda med hjälp av HTML5 och CSS för att visa och formatera datan på ett vettigt sätt och Jinja2 för att hämta datan från servern och generera lämplig HTML5 kod för att visa den i webbsidans kontext. Här följer en beskrivning av serverprogrammet (`server.py`):

- Först importeras alla moduler som behövs för serverns funktionalitet: `data` (vilket är filen `data.py` som beskrivs ovan), `flask` (själva webbservermodulen) och `jinja2` (templatemodulen). Modulen `logging` importeras för att kunna logga serveranrop och sökningar till en logfil (`log.txt`).

- Det första programmet gör är att initiera själva servermiljön vilken består av flask, jinja2 och loggingfunktionen.

- **`@app.route('/')` (funktionen `root()`)**
  Definerar vad som händer när användaren går till URL:en ”`/`”. Vad som händer här är att funktionerna `load()` och `get_random_project()` från data-modulen körs för att hämta in projektdatabasen och utvalda poster från ett slumpmässigt projekt för att visas på startsidan. Därefter anges vilken templatefil som ska användas (`root.jinja`) och renderar den. De variabler som hämtats från datalagret inkluderas även i den renderade mallen.

- **`@app.route('/list', methods=['GET', 'POST'])` (funktionen `mylist()`)**
  Definierar vad som händer när användaren går till URL:en ”`/list`”. Det första som händer är att projektdatabasen laddas in; `load()` anropas, sedan lägger `get_techniques()` alla tekniker i en lista och ett par variabler tilldelas standardvärden. Sedan kan två olika händelseförlopp utspela sig. Hamnar användaren på sidan för att denne har följt en länk alternativt skrivit in den exakta URL:en, körs sökfunktionen `search()` med endast projektdatabasen som argument (alla projekt listas).

  Hamnar användaren däremot på sidan för att denne använt ett sökformulär körs sökfunktionen med de agument som användaren angivit i hemsidans formulär. Alltså kan sökningen preciseras i detta läge. Vad som händer är att vi försöker hämta formulärdata för de olika sökargumenten och skickar med dem till sökfunktionen när de hittas. Hittas de inte skapas ett exception av respektive tryblock och ett standardvärde anges. Har användaren sökt på fritext loggas sökorden till loggfilen. Därefter anges vilken templatefil som ska användas (`list.jinja`) och renderar den med variablerna vi hämtat från datalagret och sökformuläret.

- **`@app.route('/techniques')` (funktionen `techniques()`)**
  Definerar vad som händer när användaren går till URL:en ”`/techniques`”. Först och främst laddas projektdatabasen in genom att funktionen `load()` anropas. Sedan hämtas alla tekniker i projektdatabasen med funktionen `get_techniques()`. Funktionen `get_technique_stats()` anropas för att kunna skicka med en tabell innehållande alla tekniker med relaterade projekt. Därefter anges vilken templatefil som ska användas (`techniques.jinja`) och renderar den med variablerna vi hämtat från datalagret.

- **`@app.route('/project/<projectID>')` (funktionen `project(projectID)`)**
  Definerar vad som händer när användaren går till URL:en ”`/project/`*`projektets ID`*”. Först och främst laddas projektdatabasen in genom att funktionen `load()` anropas. Sedan anropas funktionen `get_project(`*`projektets ID`*`)` för att hämta tabellen för det specifika projekt som har matchande ID som det som angivits i app.route:n. Därefter anges vilken templatefil som ska användas (`project.jinja`) och renderar sidan med medskickad projekttabell.

- De sista två kodraderna i dokumentet är till för att webbservern ska starta automatiskt när man kör filen (`server.py`) som ett program.

Här följer en beskrivning av webbsidorna:

- **`root.jinja`**
  Detta är startsidan för SimThebl Portfolio. Högst upp i body finns ett sökformulär som kör metoden `POST` till sidan `/list` och skickar med fritext från textrutan `'free_text'`. Innan man har skrivit in något i rutan ligger en placeholder där som visar ordet '`Sök`'. Under formuläret visas en välkomsttext följt av en länk till ett slumpmässigt utvalt projekt. Länken visar projektnamn, thumbnail och en kort beskrivning.

- **`list.jinja`**
  Denna sida listar alla projekt i projektdatabasen, alternativt skriver ut alla träffar i ett sökresultat. Längst upp på sidan finns ett sökformulär som låter användaren mata in en söktext. Placeholdern '`Sök`' visas innan användaren gör en inmatning. Om sidan visas som resultat av en sökning ligger söktexten som standardvärde i textfältet via en medskickad variabel.

  Nästa kodblock genererar en rullgardinsmeny och en kryssruta som låter användaren välja hur sökresultatet ska sorteras. En for-loop går igenom alla nyckelvärden i en projekttabell och lägger dem som alternativ i rullgardinsmenyn. Kryssrutan bestämmer om resultaten ska sorteras i stigande eller fallande ordning. Standardvärden är att sortera i fallande ordning efter startdatum för projektet.

  Nästa del i formuläret är en rad kryssrutor som låter användaren filtrera sin sökning genom att ange vilka fält i projektdatabasen som ska genomsökas. Rutorna skrivs ut genom att en for-loop går igenom alla nyckelvärden (motsvarar sökfält) i projekttabellen och genererar motsvarande kryssruta. Har användaren hamnat på sidan genom en sökning med ikryssade rutor är dessa rutor ikryssade som standard. Annars är inga rutor ikryssade, vilket innebär att alla fält genomsöks.

  En sista sektion i formuläret låter användaren kryssa i tekniker som ett projekt måste innehålla för att visas som en sökträff. Dessa rutor genereras genom att en for-loop går

igenom en lista med alla tekniker i projektdatabasen och genererar en kryssruta per teknik. Precis som med föregående formulärsektioner sparas värdet för tidigare ikryssningar.

Förutom ett sökformulär listar sidan även alla sökträffar som hittas för nuvarande sökning. Detta gör den genom att en forloop går igenom en medskickad lista med sökresultat och skriver ut länkar till de projekt som hittas. De fält i projekttabellen som finns representerade i länken är projektnamn, startdatum, slutdatum thumbnail och projekt-ID. Innehåller resultatlistan inga inlägg visas texten 'inga träffar :('

- **`techniques.jinja`**
  Denna sida listar alla tekniker i projektdatabasen samt alla projekt som hör ihop med respektive teknik. Detta gör den genom en nästlad for-loop. Den presenterar teknikerna i länkform genom att itererar genom en medskickad tekniklista. För varje teknik plockar den ut en lista med projekt från en tabell där nyckelordet motsvarar tekniken i fråga. Listan iteras sedan över och namn och projekt-ID skrivs ut som länkens `title`-värde. Detta innebär att projekten kommer att dyka upp när användaren hovrar över länken.

- **`project.jinja`**
  Den här sidan skriver ut en medskickad projekttabell som en sträng i webbläsaren.

SimThebl Portfolio - Systemtest

För att säkerställa Simthebl Portfolios driftsäkerhet har en mängd systemtest gjorts. Bland annat har datalagret testats automatiskt med den inkluderade filen `data_test.py`. Vill du som köpare se med egna ögon att allt är som det ska kan du själv köra filen och se testet bli godkänt. Alla funktioner har även testats manuellt upprepade gånger för att fånga upp eventuella undantagsfall.

I nuläget existerar inga kända buggar i systemet. Vi uppskattar att du som användare kontaktar simgu002@student.liu.se eller thebl297@student.liu.se om några problem eller buggar påträffas. Alla rapporter läses och följs upp med vederbörlig prioritet.