

# SimThebl Portfolio

Dokumentation  
Version 2.1

singu002 thebl297 2013

# Innehållsförteckning

- Installationsmanual: 3
- Användarmanual: 4-10
  - Startside: 5
  - Söksida: 6
  - Projektsida: 7
  - Tekniskida: 8
  - Adminsidor: 9-10
- Systemdokumentation: 11-16
  - Datalager: 12-13
  - Presentationslager: 13-16
- Systemtest: 17

## 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, Flask 0.10, Jinja2, Werkzeug och Git installerat (är detta inte fallet följer separata instruktioner senare i dokumentet). Nu kör vi!

### Installation:

1. Ställ dig i den mapp som du vill köra SimThebl Portfolio ifrån med  
**cd [mapp]**
2. Kör  
**git clone git://github.com/denfarligaslangen/portfolio.git**  
  
för att ladda ner portfolion. Ställ dig sedan i mappen med  
**cd portfolio**
3. För att starta webbservern, gå till mappen **web/** och starta servern med:  
**python3.3 server.py &**
4. Nu kan andra nå din sida via din publika IP-adress.

Har du inte redan Python 3.3, Flask 0.10, Jinja2, Werkzeug eller Git?

Gör så här:

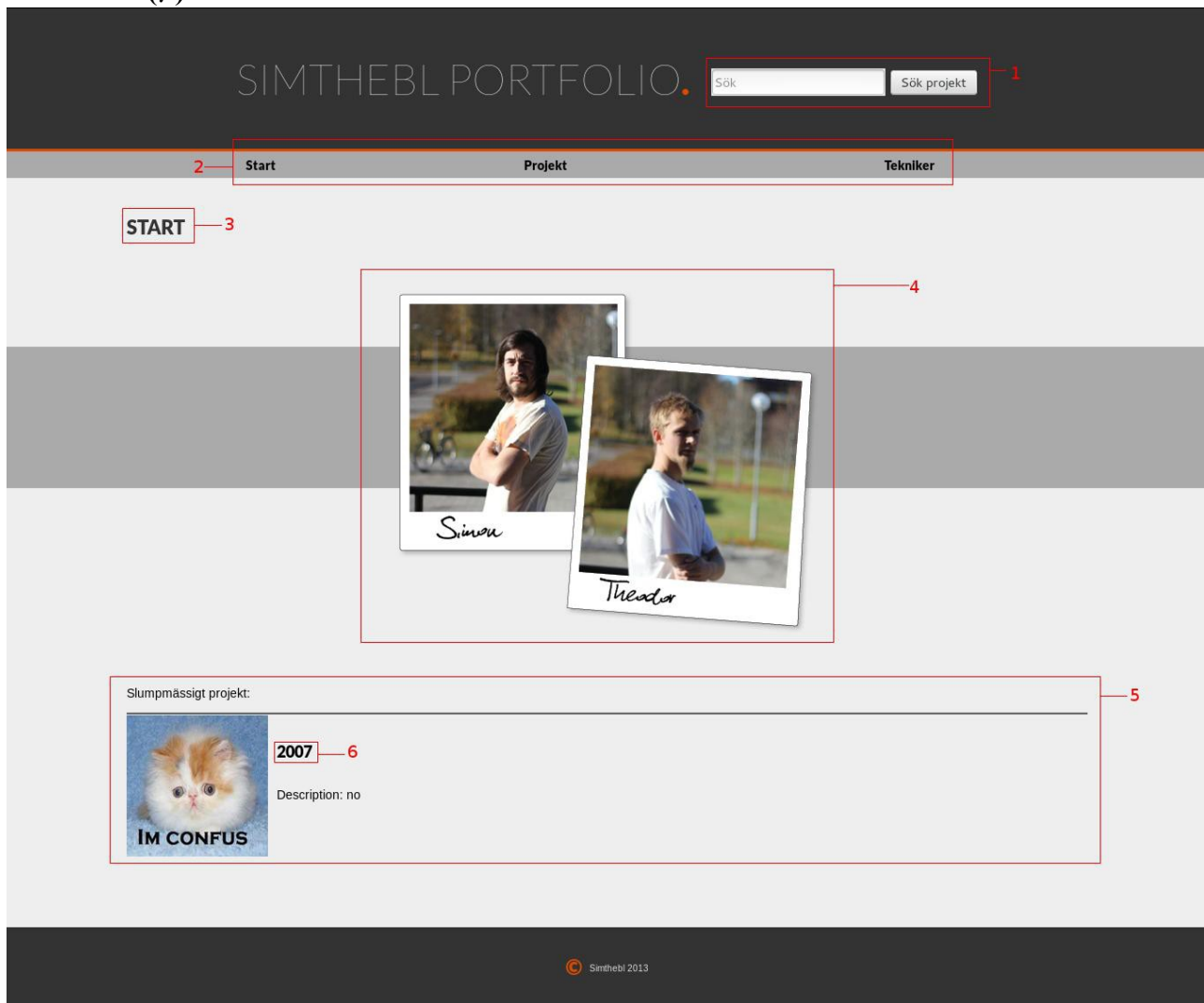
1. Gå till <http://www.python.org/download/releases/3.3.2/>, ladda ner och installera Python 3.3.
2. Gå till <http://flask.pocoo.org/> och ladda ner Flask version 0.10 och packa upp tarballen med:  
**tar -xf [filnamn].tar.gz**
3. Installera Flask till Python 3.3 genom att ställa dig i den uppackade mappen och kör  
**sudo python3.3 setup.py install**
4. Behöver du även installera Jinja2, använd samma tillvägagångssätt som med Flask. Hemsidan hittar du på <https://pypi.python.org/pypi/Jinja2>.
5. Behöver du även installera Werkzeug finns en tarball på <https://pypi.python.org/pypi/Werkzeug>, installeras även den på samma sätt.
6. Git installeras (på Ubuntu) genom att du kör  
**sudo apt-get install git**  
  
men kör du inte Ubuntu finns instruktioner här: <http://git-scm.com/download/linux>
7. Behöver installationen ytterligare program (t.ex. setuptools, MarkupSafe etc.) för att slutföras kan du installera dem från tarball på liknande sätt som ovan. Var bara noga med att använda Python 3.3 vid installation av samtliga program så kommer allt gå bra.
8. Nu är du redo!

## SimThebl Portfolio – Användarmanual

SimThebl Portfolio består av följande sidor:

- Startside (/) med information om sidans skapare, en välkomstskärm samt några små bilder från några av projekten.
- En sida (/list) som listar alla projekt med kort och allmän information. Har även möjligheter för att söka i och filtrera projekten.
- Projektsida (/project/<projectID>), unik för varje projekt, här visas allt om ett projekt upp.
- En sida (/techniques) som listar alla tekniker i projektdatabasen och visar vilka projekt som innehåller respektive teknik.

## Startsidan (/)



Startsidan fungerar dels som välkomstkärm men också som språngbräda till portfolions andra sidor.

1. Portfolions sökfunktion. Användaren kan skriva fritext i textrutan och klicka på sök. Användaren hamnar då på sidan **/list**. Detta fält finns på samma ställe på samtliga sidor, så i fortsättningen kommer den inte att förklaras.
2. Menyrad med länkar till portfolions andra delsidor. Även den ligger på samma ställe på alla delsidor och kommer härnäst inte att förklaras mer.
3. Här är du nu. För tillfället är du på sidan "Start".
4. En presentationsbild föreställande sidans skapare.
5. Ett slumpmässigt utvalt projekt från projektdatabasen där thumbnail på projektets bild visas, tillsammans med en kort beskrivning av projektet.
6. Länk som leder till det framlumpade projektets projektsida (**/project/<projectID>**)

## Söksidan (/list)

**SIMTHEBL PORTFOLIO.** Sök Sök projekt

Start Projekt Tekniker

**PROJEKT** — 1

Avancerad sök Sök

Sortera efter fält: start\_date Stigande




Sök i fält:

<input type="checkbox"/> project_no	<input type="checkbox"/> small_image	<input type="checkbox"/> end_date
<input type="checkbox"/> group_size	<input type="checkbox"/> long_description	<input type="checkbox"/> course_id
<input type="checkbox"/> project_name	<input type="checkbox"/> lulz_had	<input type="checkbox"/> external_link
<input type="checkbox"/> short_description	<input type="checkbox"/> start_date	<input type="checkbox"/> big_image
<input type="checkbox"/> course_name	<input type="checkbox"/> techniques_used	<input type="checkbox"/> academic_credits

Måste innehålla tekniker:

<input type="checkbox"/> csv	<input type="checkbox"/> ada	<input type="checkbox"/> c++
	<input type="checkbox"/> python	

**Träffar:**

 <b>IM CONFUS</b>	<b>2007</b> Startdatum: 2009-09-08 Slutdatum: 2009-09-09 Bild: imconfus.jpg
 <b>I cannot Brain today</b>	<b>NEJ</b> Startdatum: 2009-09-07 Slutdatum: 2009-09-08 Bild: dumb.jpeg
 <b>I have the dumb</b>	<b>,</b> Startdatum: 2009-09-06 Slutdatum: 2009-09-07 Bild: doge.jpg

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

1. Här är du nu. För tillfället sidan (/list) som bäst kan beskrivas med rubriken "Projekt".
2. Den avancerade varianten av portfolios sökfunktion. Användaren tillåts söka på fritext, kryssa i rutor för att ange speciella fält i databasen, samt kryssa i tekniker som sökträffarna måste innehålla för att visas. Användaren kan även specificera hur träffarna ska sorteras: stigande eller fallande ordning, samt vilket fält som ska användas vid sortering.
3. Här listas alla projekt som matchar sökningen, alternativt alla projekt om användaren gjort en tom sökning.
4. Projektets namn är en länk till sidan (/project/<projectID>) för projektet.

## Projektsidan (/project/<projectID>)

SIMTHEBL PORTFOLIO.

Sök  Sök projekt

Start Projekt Tekniker

2007 1

2 2

imconfus.jpg

2009-09-09

6

no no no

TDP003

2007

medium

YY

no

2009-09-08

XXX

OKÄNT

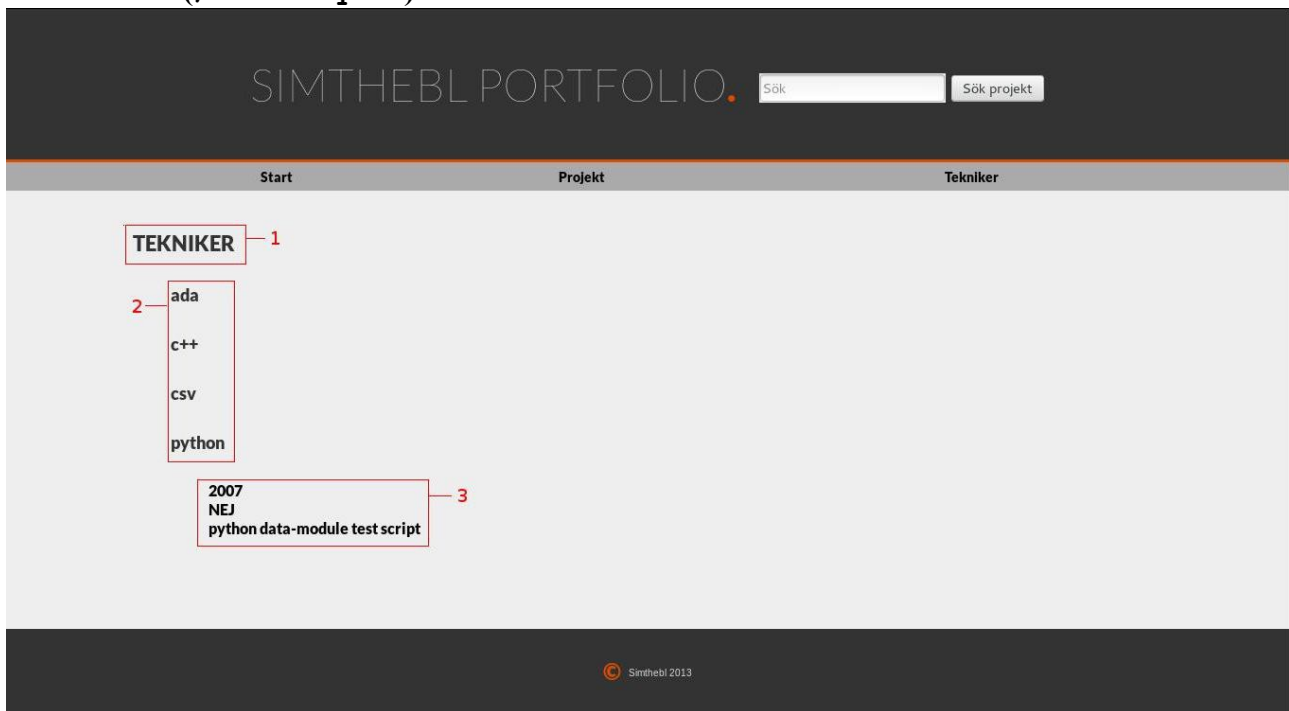
['ada', 'python']

WUT?

Projektsidan visar upp all information som finns tillgänglig för det aktuella projektet.

1. Det aktuella projektets namn.
2. All information från alla fält i projektet.

## Tekniksidan (/techniques)



Tekniksidan visar alla tekniker som finns i databasen. Om man håller musen över en teknik visas alla projekt som tekniken ingår i.

1. Här är du nu. Sidan som listar "Tekniker"
2. En lista med alla tekniker i teknikdatabasen.
3. Det som visas när användaren håller muspekaren över ett specifikt projekt, vilket är namn på och länk till alla projekt som innehåller respektive teknik.



## Adminsidan (/admin)

---

Du är inte inloggad!

Går användaren till URL:en **/admin** visas denna skärm. Här får administratören logga in med användarnamn **admin** och lösenord **h**. Då visas denna skärm:

[Add project](#)

python data-module test script fluffy.jpeg no

NEJ dumb.jpeg no

2007 imconfus.jpg no

, doge.jpg no

Här kan administratören lägga till ett projekt i databasen med knappen **Add project**. Administratören kan även ändra ett befintligt projekt och ta bort ett projekt med knapparna **Edit** respektive **Remove**.

Ändra/Lägg till projekt-sidan (/addit)

small_image	<input type="text" value="fluffy.jpeg"/>
end_date	<input type="text" value="2009-09-06"/>
group_size	<input type="text" value="2"/>
long_description	<input type="text" value="no no no"/>
course_id	<input type="text" value="TDP003"/>
project_name	<input type="text" value="python data-module test scri"/>
lulz_had	<input type="text" value="many"/>
external_link	<input type="text" value="YY"/>
short_description	<input type="text" value="no"/>
start_date	<input type="text" value="2009-09-05"/>
big_image	<input type="text" value="XXX"/>
course_name	<input type="text" value="OKÄNT"/>
techniques_used	<input type="text" value="['python']"/>
academic_credits	<input type="text" value="WUT?"/>
Project no : 1	<input type="button" value="Submit"/>

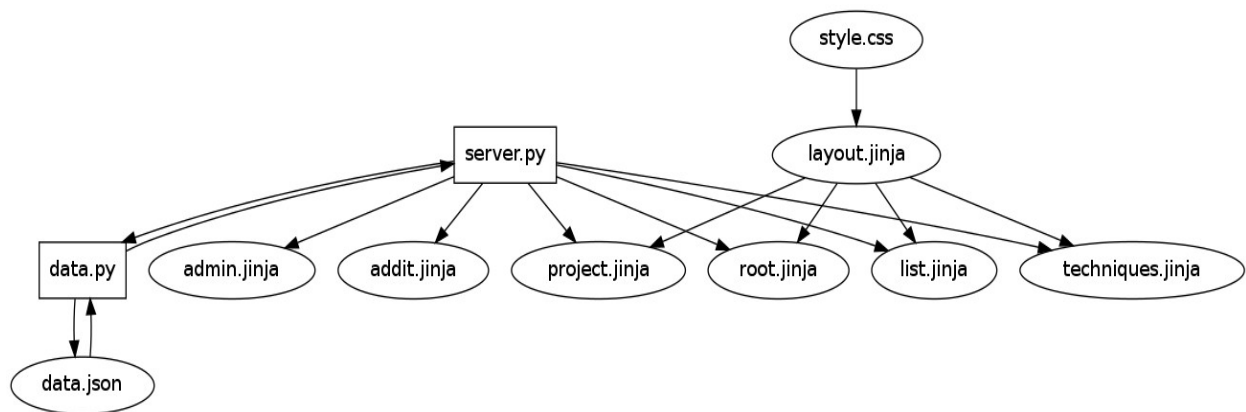
Med hjälp av denna vy kan administratören antingen lägga till ett projekt eller ändra i ett befintligt. Fälten representerar alla fält i projektdatabasen och administratören kan specificera alla data för projektet. Fältet **group\_size** måste ha ett heltal som inmatning och fältet **techniques\_used** måste ha en pythonlista med strängobjekt som inmatning. I övriga fält finns inga restriktioner.

**Övrigt**

Allt som händer på servern loggas i filen **web/log.txt**. 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. Användaren 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.



Bilden illustrerar hur systemet hänger samman som helhet. De noder som är fyrkantiga representerar hjärnan i systemet, dvs. de filer som styr systemet. Noderna som har en oval form representerar filer som är av mer dynamisk karaktär och styrs av "hjärnan". De innehåller information eller formatmallar.

Filerna som slutar med .jinja är de filer som användaren kommer att se i sin webbläsare. De innehåller formatmallar i HTML5, CSS och Jinja2 som bestämmer hur informationen från datalagret kommer att presenteras.

Här följer ett exempel på hur ett flöde i systemet kan gå till:

1. Användaren går till URL:en `/list` i sin webbläsare.
2. `server.py` ber `data.py` att köra sökfunktionen `data.search()` på filen `data.json`, utan några sökargument.
3. Sökfunktionen returnerar all data från `data.json` (vilket händer när inga argument ges i sökfunktionen).
4. `server.py` renderar sedan `list.jinja` som lägger sig själv i kontexten från mallen `layout.jinja` som i sin tur använder formateringen från `style.css`. `server.py` skickar med informationen från datalagret som ett listobjekt till `list.jinja`. Listobjektet itereras över och varje projekt i databasen visas som ett HTML5-kodblock i `list.jinja`, i användarens webbläsare.

## Datalagret (Python 3.3, json)

Datalagret är vad man kan kalla för 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 hantera informationen i denna json-fil erbjuder datalagret funktioner för att ladda in en json-fil, samt ändra i och lägga till projekt i den. Övriga funktioner som finns tillgängliga gör det enkelt att räkna antalet projekt i databasen, hämta ett specifikt eller slumpmässigt projekt, söka i databasen, räkna upp alla tekniker samt vilka projekt som använder en viss teknik. 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 alla de olika funktionerna som finns 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. Från början innehåller sökresultat-listan alla projekt. Allt eftersom underfunktionerna hittar projekt som inte matchar sökkriterierna läggs dessa till i en lista över projekt som ska tas bort. Slutligen tar den bort alla projekt som finns representerade i ta-bort-listan. 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 innehållande 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

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:

1. Flask är en expansion till Python, som låter användaren köra en lättvikts-webbserver. Eftersom serverns huvudfil är skriven i Python, är det lätt för den att kommunicera med datalagret.
2. Delsidorna är byggda med HTML5 och jinja2. Den största delen består av HTML5 medan logik, variabler och iteration sköts av jinja2, som genererar olika HTML5-kod till webbläsaren beroende på vilken data som behandlas. Varje sida blir som en flexibel HTML-mall som förändras efter behov av jinja2.
3. All design är löst via CSS. En CSS-fil anger var allting ska vara, färgscheman samt undermenyer som ska visas när användaren håller muspekaren över speciella rubriker. Filen laddas in av servern vid start och appliceras på samtliga sidor.

Här följer en beskrivning av serverprogrammet (`server.py`):

- Först importeras alla moduler som behövs för att servern ska fungera: `data` (alltså `data.py` som beskrivs ovan), `flask` och `jinja2`. Modulen `logging` importeras för att kunna logga serveranrop och sökningar till en logfil (`log.txt`).
- Det första programmet gör är att initiera servermiljön.
- **`@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 argument 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 fångas 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. Standardvärdet 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 hoverar över länken.

- **project.jinja**

Den här sidan itererar över ett specifikt projekt och skriver ut varje fält på en egen rad i webbläsaren.



## SimThebl Portfolio - Systemtest

Nedan följer en del av de systemtest som har gjorts för att säkerställa Simthebl Portfolios driftsäkerhet:

- Datalagret har 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. Vi har även manuellt gjort diverse anrop till **data.py** från python3-tolken som servern rimligtvis själv skulle kunna göra:
  - `load()` har testats med olika giltiga och ogiltiga filnamn.
  - `get_random_project()` har anropats med flera databaser.
  - `get_techniques()` har anropats med flera databaser.
  - `get_technique_stats()` har anropats med flera databaser.
  - `search()` har anropats med olika kombinationer av sökvilkor.
  - `remove_project()` har testats med flera projekt-ID:n.
  - `add_project()` har testats med flera projekt-tabeller.
  - `edit_project()` har anropats med olika projekt-ID:n.
- Sidan klarar av att användaren anger en felaktig URL. Användaren hänvisas då till lämplig "sidan hittades inte" sida.
- Projektsidan visar texten "inget projekt med detta id hittades" om användaren anger ett projekt-id i URL:en som inte finns med i databasen.
- Sidan kraschar inte vid någon känd kombination av sökkriterier eller söktexter.
  - Användaren kan söka på tekniker i fritext samtidigt som via kryssrutor.
  - Kryssrutor för att bestämma fält att söka i fungerar som tänkt.
  - Sortering i stigande / fallande ordning efter val i rullgardinsmeny fungerar som tänkt.
- Sökfunktionen kan med önskat resultat anropas från samtliga sidor.
- Alla länkar leder dit de ska.

Hemsidan som helhet har testats manuellt, både av utvecklare och av en dedikerad testgrupp.

Kända buggar:

- Om ett projekt läggs till via adminsidan och inte innehåller några tekniker, visas en tom ruta på söksidan. (Användaren hänvisas till projektet som inte innehåller några tekniker om användaren söker med hjälp av den tomma rutan, det är dock inte så tydligt).

Vi uppskattar att du som användare kontaktar [simgu002@student.liu.se](mailto:simgu002@student.liu.se) eller [thebl297@student.liu.se](mailto:thebl297@student.liu.se) om några problem eller buggar påträffas. Alla rapporter läses och följs upp med vederbörlig prioritet.