Systemdokumentation

Warhell Nasim Henrik Björs

Datum: 2014-10-16

Dokumentversion: 0.1

Projektnamn: Portföljsystem

Kurs: TDP003 Projektkurs - Egna datormiljön

Innehållsförteckning

Innehåll

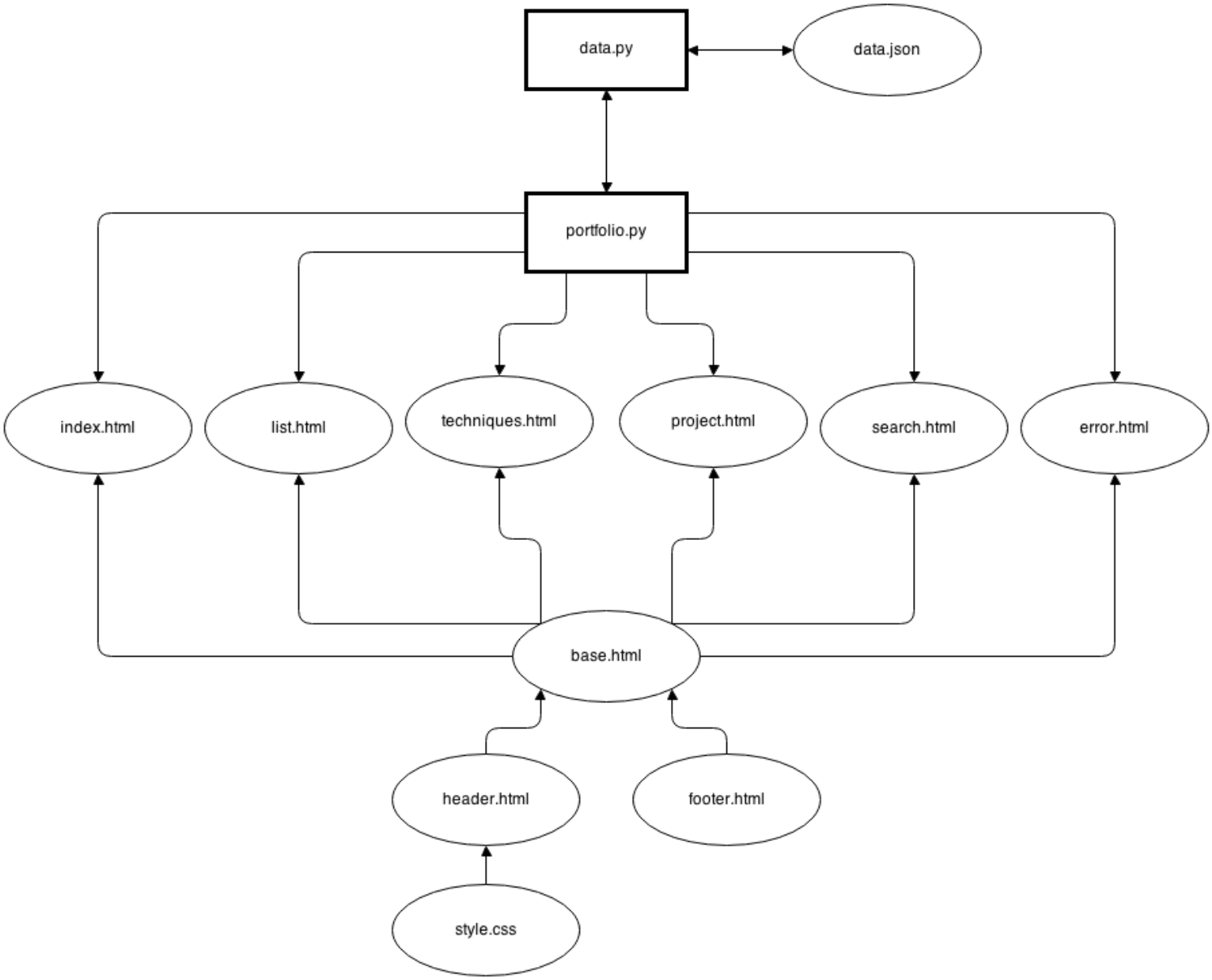
Översikt

Portföljsystemet är uppdelat i tre mindre delsystem. Vi har ett **datalager** som kommunicerar med databasen, i detta fall en JSON fil med all data. Vi har även **presentationsmallar** som byggs upp både utav statisk data samt dynamisk data som är argumenten vid anrop från presentationslagret. Till slut har vi **presentationslagret** som binder dem olika presentationsmallarna till webbaddresser och som även kommunicerar med datalagret för att vidarebefordra informationen till presentationsmallarna.

Man kan dra paralleller till ett generellt Model-View-Controller mönster. Datalagret är model, presentationslagret är controller och presentationsmallarna är view.

Presentationslagret är skrivet i Flask. Presentationsmallarna är skrivna i Jinja2, HTML5 samt CSS3.

Sekvensdiagram



Sekvensdiagramet illustrerar vilka delar det finns och hur dessa är sammankopplade. Noderna med lite mera ovala former är dynamiska. Rektangulära noder är var logiken sker.

Filer med filändelsen html är presentationsmallarna. Filen med filändelsen css kontrollerar utseendet på webbsidorna.

Datalagret, data.py, fungerar ungefär som ett API som kommunicerar med databasen, i vårt fall, en JSON fil med data.

Portfolio.py kontrollerar vilken data användas och vilken presentationsmall som skall presenteras beroende på vad besökaren skickar med som data. Exempelvis kan en besökare skicka GET och POST begäran med valfri data, exempelvis ”project_no=2”. En djupare förklaring hittar du under rubriken presentationslager. Alla anslutningar från användaren går igenom portfolio.py.

Anslutningsexempel

Om en användare exempelvis vill hämta webbaddressen /techniques med en GET begäran, börjar portfolio.py kolla om den webbaddressen.

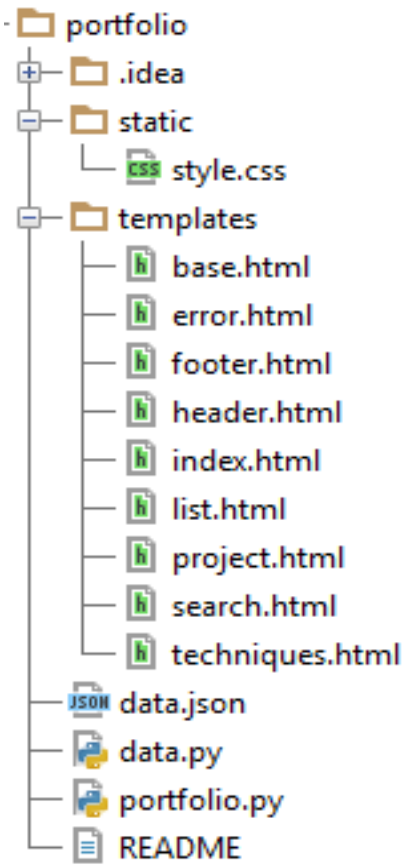
När portfolio.py hittar webbaddressen som matchar kör den igenom funktionerna, för webbaddressen /techniques börjar det med att data från databasen (JSON filen) hämtas via datalagret (data.py) och sparas i en variabel.

Sedan anropas ytterligare en funktion som tar en lista med projekt som argument och returnerar datatypen dictionary med alla olika tekniker som fanns med i argumentet tillsammans med respektive projekt som använder sig utav tekniken, det som returneras sparas i ännu en variabel.

Till slut anropas presentationsmallen med två argument, det ena är variabeln med alla tekniker respektive projekt och det andra argumentet är alla tekniker från första argumentet.

Nu har presentationsmallen renderats och visas upp för användaren. Utförligare beskrivning för hur presentationsmallen använder sig utav variablerna och hur renderingen går till hittar du under rubriken presentationsmallar.

Filstruktur



I roten utav portföljsystemet har vi två viktiga kataloger, static och templates. Under static infogas filer såsom bilder och stilmallar. I katalogen templates placeras alla presentationsmallar.

I roten finner vi även fyra olika filer, data.json, data.py, portfolio.py och README.

data.json innehåller alla projekt, det är i denna fil du lägger till och redigerar existerande projekt.

data.py är datalagret som innehåller flera olika funktioner som i sin tur används utav portföljsystemet för att kunna kommunicera med databasen (data.json).

portfolio.py är själva grunden för portföljsystemet, det är denna fil man startar upp servern med.

README innehåller upphovsrättsinformation tillsammans med värdefull information angående portföljsystemet.

Datalager

Datalagret är namnet på den modul i portfolioprogrammet som hanterar projektinformation. All data om projekten är lagrade i en JSON fil (data.json), och denna modul (data.py) hanterar bland annat inläsning av projekten, sökning och sortering bland dessa, och annat (mer detaljer senare). Notera att modulen endast hanterar inläsning av projektdata, så den sköter inget ändrande eller tilläggande av data. Funktionerna datalagret erbjuder är följande:

- **load(filename):** Läser in en JSON fil med projektinformation, där **filename** är en **string** som representerar namnet på den fil man vill använda, och returnerar filens innehåll som ett **list** objekt. Om något fel uppstår returnerar funktionen **None** istället.

- **get_project_count(db):** Hämtar ut antalet projekt ur en given lista med projekt, där **db** är ett **list** objekt som innehåller projektdata hämtad med hjälp av **load** funktionen, och returnerar antalet projekt som en **int**.

- **get_project(db, project_id):** Letar genom en lista med projekt och hämtar ut det projekt som har det ID-nummer man letar efter. **db** är även här en lista med projektdata hämtad via **load** funktionen, och **project_id** är en **int** som motsvarar det ID-nummer man letar efter. Funktionen returnerar en **dict** innehållandes all information om det projekt man hittade. Om inget hittades, returneras **None** istället.

- **get_techniques(db):** Hämtar ut alla använda tekniker ur en projektlista, där **db** är listan hämtad via **load**, och det som returneras är ett alfabetiskt sorterat **list** objekt med alla tekniker.

- **get_technique_stats(db):** Hämtar ut alla använda tekniker ur en projektlista, samt information om vilka projekt som använder dem. **db** är även här en lista av projekt hämtad via **load**, och det som returneras är ett **dict** objekt, där nyckeln är en teknik, och värdet är ett **list** objekt innehållandes **dict** objekt som har namnet och ID-numret för varje projekt som använder tekniken i fråga.

- **search(db, sort_by='start_date', sort_order='desc', techniques=None, search=None, search_fields=None):** Söker efter projekt i en lista innehållandes projekt. **db** är en lista av projekt hämtad via funktionen **load**, **sort_by** är en **string** som beskriver vilket informationsfält man ska sortera hittade projekt utifrån, **sort_order** är en **string** som representerar ordningen man ska sortera i, och kan endast ha värdena **'asc'** för stigande, och **'desc'** för fallande, **techniques** är ett **list** objekt som innehåller de tekniker hittade projekt måste använda (är det **None** struntar sökningen i tekniker), **search** är en **string** innehållandes en söksträng, och **search_fields** är en **list** med de sökfält som **search** ska söka i. om **search_fields** är **None** söker man i alla fält, medan om det är en tom lista så returneras inga sökresultat.

- **get_random_project(db):** Tar ut ett slumpmässigt utvalt projekt från en projektlista. **Db** är projektlistan returnerad via funktionen **load**, och det som returneras är ett slumpmässigt utvalt projekt ur den listan.

- **logger(functionname, *args):** Hanterar loggning av anrop gjorda i datalagret, där det som loggas är datum och tidpunkt för anrop, vilket anrop som gjordes, och vilka argument som paserades. **functionname** är en **string** som representerar namnet på funktionen som anropades, och ***args** är noll eller flera **string** argument som representerar vilka argument funktionen man anropade använde. Man kan dock logga vad som helst med denna funktion, och inte bara funktionsanrop, så länge man kan beskriva det man loggar med **string** objekt. **logger** har inga returvärden, men den skriver ut loggningen i filen **log.txt**.

Presentationslager

Översikt

Presentationslagret är ungefär som en dirigeringsmodul. Beroende på vilken webbaddress som användaren vill hämta, påkallar presentationslagret specifika funktioner och presentationsmallen som skall representera webbaddressen.

Presentationslagret importerar en del olika funktioner och moduler, detta för att allting ska fungera korrekt.

- ”Flask” importeras då det är grunden till mikroramverket.
- ”render_template” importeras för att kunna rendera de olika presentationsmallarna.
- Vi importerar ”request” för att kontrollera om användaren använt GET eller POST begäran samt för att hämta data som användaren skickat vid POST begäran.
- ”abort” importeras för att, vid behov, avbryta en begäran och dirigera om användaren till felkodssidan tillsammans med en HTTP felkod.
- ”data” importeras för att kunna kommunicera med databasen (JSON fil).

Webbadresser

/index

Hittas under ”@app.route('/')”.

Detta segment körs när användaren begär förstasidan ”/” eller ”/index”. Koden börjar med att tilldela en variabel ett slumpmässigt projekt genom att anropa funktionen get_random_project() från datalagret, argumentet är data.load() som läser in och returnerar alla projekt från data.json (databasen).

Därefter anropas renderingsfunktionen med två argument. Första argumentet är filnamnet för presentationsmallen som ska representera förstasidan, det andra argumentet är variabeln som tilldelades det slumpmässiga projektet.

/list

Hittas under ”@app.route('/list')”.

Segmentet körs när användaren begär webbaddressen ”/list”. Segmentet börjar med att tilldela variabeln ”projects” alla projekt från databasen med hjälp utav funktionen load() från datalagret.

Sedan tilldelas samma variabel, vilket skriver över tidigare data som variabeln tilldelats, resultatet som search() från datalagret returnerar. Vid första anblick skulle man kunna tro att search() funktionen returnerar projekt som matchar specifika sökkriterier, det är gör den också, men i vårt fall anropar vi funktioen med tre argument som gör att funktionen hoppar över sökningen och endast returnerar samma lista sorterad i stigande ordning. Första argumentet är variabeln ”projects” som innehåller alla projekt, andra argumentet anger att den fördefinierade variabeln ”sort_by” tilldelas strängen ”project_no” och till slut det tredje argumentet som anger att den fördefinierade variabeln ”listlength” tilldelas strängen ”asc”.

Sedan kontrolleras det om användaren begärt sidan med GET eller POST begäran.

Om användaren begärt sidan med GET begäran, anropas renderingsfunktionen med filnamnet för presentationsmallen som första argument, andra argumentet tilldelar variabeln "projectlist” alla projekt som hämtats från databasen, tredje argumentet tilldelar variabeln ”listlength” antalet projekt som ”projectlist” tilldelats.

Däremot, om användaren begärt sidan med POST begäran tilldelas variabeln ”searchedlist” det som returneras utav search() funktion från datalagret. Search() funktionen anropas denna gång med två argument. Det första argumentet är variabeln som innehåller alla projekt som sökningen skall appliceras på, det andra argumentet anger att den fördefinierade variabeln ”search” får strängen utav användarens inmatade sökord som angetts i input. Nu har variabeln ”searchlist” alla projekt som matchat sökordet.

Till slut anropas renderingsfunktionen, precis som vid GET begäran, men här tilldelas variabeln ”projectlist” datan från variabeln ”searchedlist” istället.

/project/<ID>

Hittas under ”@app.route('/project/<int:project_id>')”.

Segmentet körs när användaren begär webbaddressen ”/project/*”, där ”*” representerar ett tal. Koden börjar med att läsa in alla projekt till en variabel med hjälp utav funktionen load() från datalagret.

Sedan tilldelas ytterligare en variabel det som returneras från funktionen get_project() i datalagret. Första argumentet till get_project() är variabeln som tilldelats alla projekt, andra argumentet är talet från webbadressen. Om användaren exempelvis vill hämta webbadressen ”/project/2” så blir argumentet talet två. Ifall projektet hittas av funktionen returneras det tillsammans med uppgifterna för projektet, om projektet inte hittas returneras datatypen ”None” istället.

Därefter kollar vi om variabeln tilldelats ett projekt, om inte, avbryter vi anropet med HTTP felkoden 404, vilket innebär att sidan inte kunde hittas.

Om variabeln däremot tilldelats ett projekt, anropas renderingsfunktionen med filnamnet för presentationsmallen som första argument och andra argumentet anger att den fördefinierade variabeln ”project” tilldelas variabeln som håller projektet samt dess uppgifter för projektet.

/techniques

Hittas under ”@app.route('/techniques')”.

Segmentet körs när användaren begär webbaddressen ”/ techniques”. Koden börjar med att läsa in alla projekt till en variabel med hjälp utav funktionen load() från datalagret. Därefter anropas funktionen get_techniques_stats() från datalagret. Argumentet till funktionen är variabeln som innehåller alla projekt som laddades in. Vad som returneras är datatypen dictionary med unika tekniker som nyckel och värdet är en lista med ID samt namn för de olika projekten där nyckeln ingår i utnyttjade tekniker.

Till slut anropas renderingsfunktionen med filnamnet för presentationsmallen som första argument, andra argumentet anger att den fördefinierade variabeln techs tilldelas uppgifterna som returnerats från funktionen get_techniques_stats() och det tredje argumentet anger att den fördefinierade variabeln keys tilldelas alla unika tekniker som återfinns i uppgifterna som returnerats utav funktionen get_techniques_stats().

/search

Hittas under ”@app.route('/search)”.

Segmentet körs när användaren begär webbaddressen ”/ techniques”. Koden börjar med att läsa in alla projekt till en variabel med hjälp utav funktionen load() från datalagret. Därefter anropas funktionen get_techniques_stats() från datalagret. Argumentet till funktionen är variabeln som innehåller alla projekt som laddades in. Vad som returneras är datatypen dictionary med unika tekniker som nyckel och värdet är en lista med ID samt namn för de olika projekten där nyckeln ingår i utnyttjade tekniker.

Sedan kontrolleras det om användaren begärt sidan med GET eller POST begäran.

Om användaren begärt sidan med GET begäran, anropas renderingsfunktionen med filnamnet för presentationsmallen som första argument, andra argumentet tilldelar variabeln "keys” alla unika tekniker som återfinns i uppgifterna som returnerats utav funktionen get_techniques_stats().

Däremot, om användaren begärt sidan med POST begäran tilldelas variabeln ”searchfields” dem fälten som användaren valt att sökningsvillkoren skall appliceras för. Därefter kollar vi om variabeln överhuvudtaget tilldelats något fält, det vill säga att användaren valt något eller några fält. Om användaren inte valt något fält anger vi att variabeln ”searchfield” tilldelas datatypen ”None” istället för en tom lista.

Sedan tilldelas variabeln ”foundprojects” det som returneras från search() funktionen. När vi anropar search() funktionen har vi ett antal argument, första argumentet är variabeln ”projects” som innehåller alla projekt vi vill söka igenom, andra argumentet anger att den fördefinierade variabeln ”sort_by” skall ha värdet som representerar de fältet användaren vill att resultatet skall sorteras efter. Tredje argumentet anger att den fördefinierade variabeln ”sort_order” skall ha värdet som representerar den sorteringen som användaren valt att resultat skall sorteras efter. Fjärde argumentet anger att variabeln ”techniques” skall tilldelas alla tekniker som användaren valt att ett projekt skall ha utnyttjat för att inkluderas i sökningsresultatet. Femte argumentet anger att variabeln ”search” tilldelas sökordet som användaren valt att söka på. Sjätte argumentet anger att variabeln ”search_fields” tilldelas de fält användaren valt att sökningen skall ske i.

Till slut anropas renderingsfunktionen med filnamnet för presentationsmallen som första argument, andra argumentet anger att variabeln ”keys” får alla unika tekniker som återfinns i sökningsresultatet, tredje argumentet anger att variabeln ”projectlist” tilldelas alla projekt samt deras uppgifter från sökningsresultatet och till slut, fjärde argumentet anger att variabeln ”matches” tilldelas antalet projekt som matchade sökningsvillkoren.

Felkoder

Vi har även felkodshantering för tre olika HTTP felkoder. Detta innebär att webbsidan renderas som vanligt men innehållet kommer att vara information om felkoden istället.

500

Hittas under ”@app.errorhandler(500)”.

Detta segment sköter fel utav typen internt server fel. Renderingsfunktionen anropas tillsammas med felkodsmeddelandet.

400

Hittas under ”@app.errorhandler(400)”.

Detta segment sköter fel som uppstår vid fel/dålig begäran. Renderingsfunktionen anropas tillsammas med felkodsmeddelandet.

404

Hittas under ”@app.errorhandler(404)”.

Detta segment sköter fel som uppstår när en webbadress inte hittades. Renderingsfunktionen anropas tillsammas med felkodsmeddelandet.

Presentationsmallar

index.html

Index sidan visas när användaren besöker förstasidan utav porföljsystemet.

Presentationsmallen tar emot variabeln ”project” som innehåller ett slumpmässigt utvalt projekt och dess uppgifter.

Vad som sker i mallen är att detaljer såsom namn, kort beskrivning, tekniker start och slutdatum, grupstorlek för projektet presenteras för användaren. De olika teknikerna skrivs ut via en for loop. For loopen initieras med att ta varje teknik var för sig i tekniklistan för att sedan skriva ut tekniken.

list.html

Presentationsmallen tar emot variablerna ”projectlist” och ”listlength”. Den förstnämnda variabeln innehåller inga eller flera projekt med dess detaljer, den andra variabeln är tilldelad ett tal, talet representerar antalet projekt som variabeln ”projectlist” innehåller.

Mallen börjar med att initiera en for loop, loopen tar varje projekt som hittas i variabeln ”projectlist” och skriver ut dess detaljer såsom projektnamn, kortbeskrivning, tekniker, start och slutdatum, gruppstorlek, kurs-ID, kursnamn och så vidare. For loopen initieras inte ifall variabeln inte innehåller något projekt.

Därefter kollar vi om variabeln ”listlength” har värdet noll och om det är sant så skriver vi ut ett kort meddelande om att inga projekt hittades.

project.html

Presentationsmallen tar emot variabeln ”project” som innehåller ett utvalt projekt och dess uppgifter.

Vad som sker i mallen är att detaljer såsom namn, kort och lång beskrivning, tekniker start och slutdatum, grupstorlek för projektet presenteras för användaren. De olika teknikerna skrivs ut via en for loop. For loopen initieras med att ta varje teknik var för sig i tekniklistan för att sedan skriva ut tekniken.

search.html

Vid GET begäran får presentationsmallen variabeln ”keys”. Vid POST begäran får presentationsmallen variablerna ”keys”, ”projectlist” och ”matches”.

Variabeln ”keys” innehåller de olika teknikerna som projekten i databasen använd sig utav. Variabeln ”projectlist” innehåller projekt från sökningsresultatet. Variabeln ”matches” får sig ett tal tilldelat, talet representerar antalet projekt som variabeln ”projectlist” innehåller.

Presentationsmallen börjar med att skriva ut ett inmatningsfält för sökord. Därefter skrivs det ut ”checkboxar” för de olika sökbara fälten.

Sedan initieras en for loop, vi itirerar över variabeln ”keys” innehåll och skriver ut dessa med ”checkboxar”, detta tillåter användaren att specificera vilka tekniker som ett projekt skall ha utnyttjat för att inkluderas i sökningsresultatet.

När det iterationen är färdig skrivs alla fält som finns med i projekten tillsammans med ”radioboxar”, detta tillåter användaren att specificera vilket fält sorteringen skall appliceras på. Därefter skrivs det ut två ”radioboxar” som låter användaren välja om det ska vara stigande eller fallande ordning på sökresultatet. Till slut skrivs det ut en knapp, när denna klickas kör sökningen igång.

Vid POST begäran kommer nedanstående att köras. Vid GET begäran kommer nedanstående inte att köras.

Vi initierar en ”if” och ”elif” sats. Vi börjar med att kolla om variabeln ”matches” tilldelats ett positivt tal, då vet vi att en POST begäran gjort eftersom vid GET begäran får presentationsmallen inte ens den variabeln. Ifall variabeln tilldelats ett positivt tal skrivs ett horisontellt streck, vilket tydligt delar upp sökresultatet ifrån all inmatningsfält, radioknappar och bockningsrutor.

Om variabeln ”matches” inte tilldelats ett positivt, kollar vi om variabeln tilldelats talet noll så skriver vi ut ett horisontellt streck tillsammans med ett kort meddelande som talar om för användaren att inget projekt hittades utifrån sökningsvillkoren.

Till slut initieras en for loop, loopen tar varje projekt som hittas i variabeln ”projectlist” och skriver ut dess detaljer såsom projektnamn, kortbeskrivning, tekniker, start och slutdatum, gruppstorlek, kurs-ID, kursnamn och så vidare. For loopen initieras inte ifall variabeln inte innehåller något projekt.

techniques.html

Presentationsmallen tar emot variablerna ”techs” och ”keys”.

Den förstnämnda variabeln är utav datatypen dictionary som innehåller unika tekniker som nyckel och värdet är en lista med ID samt namn för de olika projekten där nyckeln ingår i utnyttjade tekniker. Den andra variabeln är utav datatypen list och innehåller alla unika tekniker som finns med i variabeln ”techs”.

Vad som sker i mallen är att en for loop initieras, vi itererar över variabeln ”keys” som då får varje teknik som är unik. Vi börjar med att printa ut tekniken med stor bokstav i början. Sedan initierar vi ytterligare en for loop. Denna gång itirerar vi igenom variabeln ”techs” och får varje projekt som har den pågående tekniken som nyckel. Inuti den nästlade for loopen skriver vi ut en projektnamnet samtidigt som vi länkar projektet till dess unika projektsida.

Såhär fortsätter det kontinuerligt tills alla unika tekniker respektive projekt har skrivits ut.

error.html

Presentationsmallen tar emot variablerna ”error” och ”message”.

Den förstnämnda variabeln innehåller felkoden och den andra variabeln innehåller beskrivning utav felkoden.

Vad som sker i mallen är att variabeln ”message” skrivs ut som rubrik och därefter skrivs även den andra variabeln ut, ”error”, som vanlig brödtext.

base.html

Denna presentationsmall inkluderas i alla presentationsmallar förutom ”header.html” och ”footer.html”. Presentationsmallen inkluderar presentationsmallarna ”header.html” och ”footer.html”.

Mallen anger basstrukturen för presentationsmallarna. Presentationsmallarna som inkluderar denna basstruktur får möjligheten att infoga data/kod i två olika segment. Det första är emellan taggarna ”<title>” som ger möjligheten att ange titel för webbadressen. Det andra segmentet är emellan taggarna ”<main>”, här skall huvudinnehållet infogas.

header.html

Presentationsmallen inkluderas i presentationsmallen ”base.html” som i sin tur är basstrukturen för portföljsystemets webbadresser.

Presentationsmallen skriver ut menyraden tillsammans med länkar till olika webbadresserna i portföljsystemet.

footer.html

Presentationsmallen inkluderas i presentationsmallen ”base.html” som i sin tur är basstrukturen för portföljsystemets webbadresser.

Presentationsmallen skriver ut upphovsrättsinformation samt namnen för upphovsmännen.

Felsökning

Felsökning sker främst genom analys av loggfilen som portföljsystemet genererar, samt genom de felmeddelanden som portföljsystemet ger när något fel händer. Denna fil är lagrad i samma mapp som portfolioprogrammet. Loggfilen är formaterad i sektioner, från den punkten när användaren skickar en förfrågan om att hämta en webbadress. Det första som loggas är vilken tidpunkt anropet gjordes, och sedan står det vilket sida man frågar om:

Request, GET eller POST, Adress till sidan

Därefter kommer en serie rader innehållandes vilka anrop som utförs till datalagret:

Namn på den funktion som anropas, argument #1, argument #2, …

Om allting gick bra avslutas loggningen med texten ”Success”, om inte står det ett felmeddelande istället.

Exempel

Användaren vill åt ett visst projekt

Request, GET, /project/13
load, filename=data.json
get_project, id=13, db=data.json
Success

Användaren vill åt ett projekt som inte finns

Request, GET, /project/6955
load, filename=data.json
get_project, id=6955, db=data.json
Page not found, 404: Not Found

Användaren vill söka på projekt

Request, POST, /search
load, filename=data.json
get_technique_stats, db=data.json
get_techniques, db=data.json
search, db=data.json, sort_by=project_no, sort_order=desc, techniques=[],
search=python, search_fields=[”project_name”]
Success.