LINKÖPINGS UNIVERSITET

TDP003 Projekt: Egna datormiljön

Systemdokumentation

Författare

Andreas Magnvall, andma228@student.liu.se
Patrick Lundberg, patlu055@student.liu.se

INSTITUTIONEN FÖR DATAVETENSKAP

Höstterminen 2018
Version 0.1

2018-10-18

1 Revisionshistorik

Ver.	Revisionsbeskrivning	Datum
0.1	Första utkastet av systemdokumentationen färdigt	181018

2 Innehållsförteckning

Innehåll

3 Introduktion

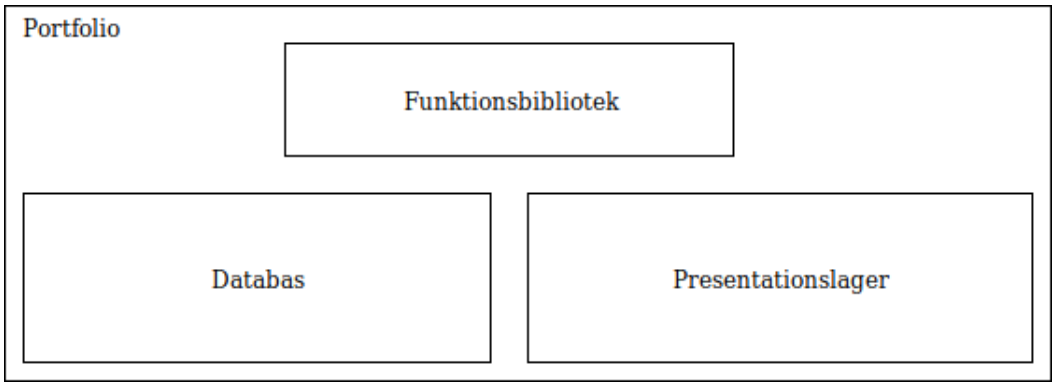
Detta dokument är en s.k. systemdokumentation för det tillhörande portfolio-systemet. Dokumentet är alltså en slags guide för systemet, vilken ska tydliggöra för hur systemets funktioner fungerar och hur de arbetar tillsammans för att bygga upp systemets struktur. Syftet med dokumentet är att redovisa funktionerna och strukturen på ett tydligt och enkelt sätt, men mer ingående beskrivningar kommer även att finnas.

Förutom de funktioner vi har skapat specifikt för portfolion använder vi även ett antal fördefinierade funktioner. Dessa består av Jquery, Bootstrap, och ett api för datalagret från LIU. Teknikerna som används för att sätta upp och köra webbservern är Python, Flask samt Jinja2. Länkar till alla dessa hittas i sektion 10 längst ner i dokumentet.

Vi har även lagt till några egna funktioner till datalagret som inte är beskrivna i länkarna. Dessa beskriver vi istället nedan i sektion 5.

4 Systemdiagram

4.1 Översiktsbild



Figur 1: Enkel bild över systemets delar

Funktionsbibliotek: Består av de enheter nämnda i sektion 3, Jquery, Bootstrap och datalagret. Datalagret består av ett antal funktioner för att underlätta lagring och hämtning av data till och från databasen. Jquery används för att förenkla användningen av javascript i presentationslagret och är även ett bibliotek som Bootstrap använder sig av. Bootstrap är ett CSS-bibliotek för att underlätta designen av presentationslagret. Exempel på detta är responsiv design.

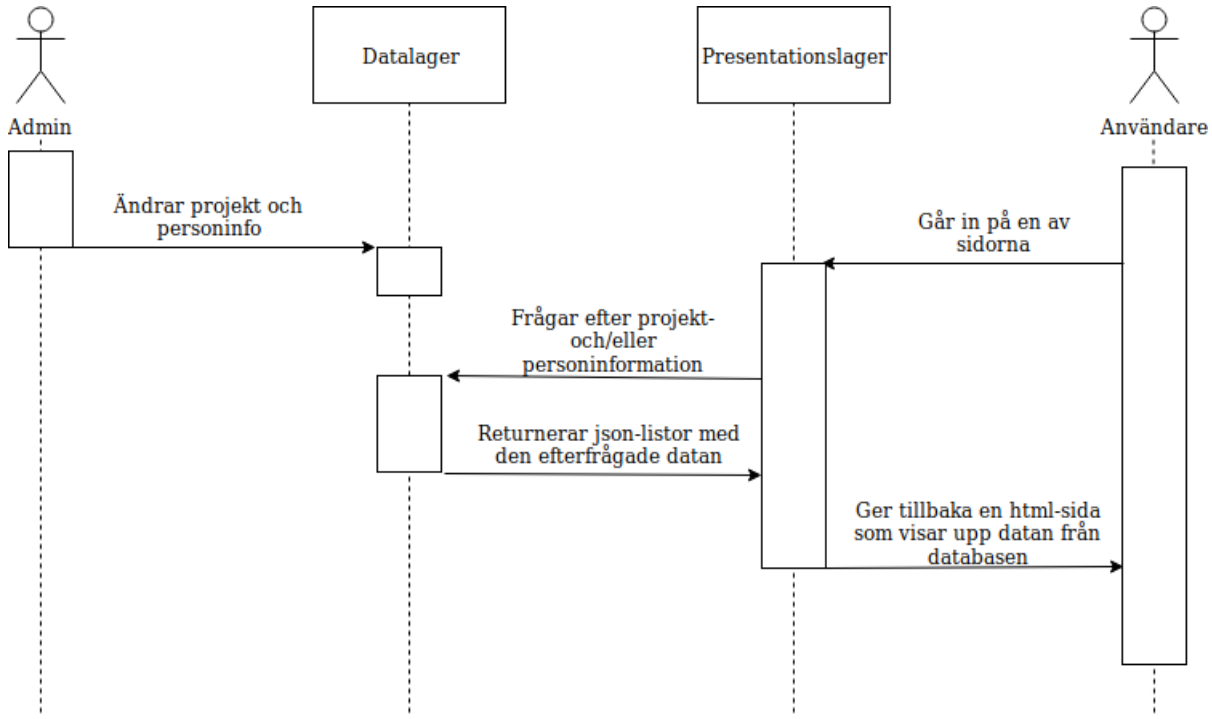
Databas: Består av två stycken json-filer, ”data.json” och ”user.json”. Dessa två json-filer används för att ge bra struktur till den data presentationslagret behöver, den ena för att lagra de projekt som presentationslagret ska visa och den andra för att lagra information om användaren av portfolion.

Denna data behöver alltså en specifik struktur, men om funktionerna i datalagermodulen används så är detta inget som behövs att tänkas på. En adt-liknande struktur har använts för databasen där en användare på ett kanske enklare sätt kan redigera datan i någon av databaserna. Som det är nu går det dock endast att bara tillföra data till databaserna med detta sätt. För att ta bort exempelvis ett projekt behövs fortfarande manuell redigering av databasen vilket kan vara krångligt. Likväl går det dock att även redigera projekt direkt i databasen.

Presentationslager: Webbsidans struktur bestäms av ett huvudprogram skrivet i Python med hjälp av Flask, Jinja2 och javascript. Mer specifikt: flask_site.py

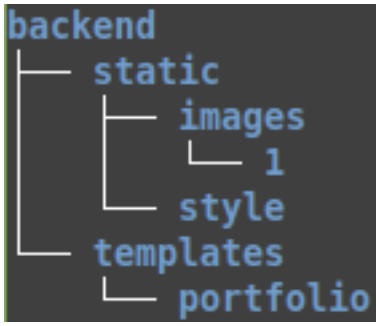
Programmet laddar in html-filer för navigationen enligt flask-teknik och använder funktioner från datalagermodulen för att hämta och filtrera data från databasen. Denna data skickas till html-filerna som laddas med Flask. Alla sidor utgår från en bassida, base.html, för framförallt headern. Men även stildokument och nödvändiga funktionsbiblotek laddas in på detta sätt till alla sidor. Utöver detta används både javascript och jinja2 för att generera innehåll på sidorna.

4.2 Sekvensdiagram



Figur 2: Enkelt diagram över sekvensföljden

4.3 Filstruktur



Figur 3: Exempel på katalogstruktur

backend: Innehåller python-filerna för datalagret, samt json-filerna som utgör databasen och de andra katalogerna.

static: Innehåller kataloger med grafiska filer, t.ex. bilder och css-filer.

static/images: Innehåller kataloger med projektens bilder. Katalogerna ska nämnas efter projektets id-nummer för att bilderna ska kunna laddas in. I detta fall, katalog ”1” innehåller bilder för projektet med id 1.

static/style: Innehåller lokala css-filer portfolion använder sig av samt de bilder som css-filerna använder sig av.

templates: Innehåller mer generella html-filer, vilka skulle kunna användas på många olika sätt.

templates/portfolio: Innehåller själva huvud-filerna, i detta fall index.html, list.html, techniques.html.

5 Datalagret

Datalagret har byggts upp av en redan förutbestämd api specifikation samt av diverse tillägg där det känts nödvändigt. Dess funktioner ligger filen data.py

Kursens specifikation av datalagret hittas i länk 3 i länksamlingen.

Utöver detta har följande funktioner använts:

get_keys(db)

Parameters:

db (list) - A list as returned by load.

Returns: list
The keys of all projects. Duplicates ignored.

get_user_info(db,key):

Parameters:

db (list) - A list as returned by load.
key(string) - A string specified by the user

Returns: string
A string value matched by the key.

6 Presentationslagret

Presentationslagret har byggts ihop av en blandning av html, css, javascript och jinja2. Mycket av de utseendemässiga bygger på bootstrap som är ett robust bibliotek för för websidedesign. Många av dess funktioner, exempelvis klasser, har använts på element på sidan. För det dynamiska som hämtas från en av de två databaserna renderas detta antingen med jinja2 eller javascript. Som hjälpbibliotek till javascript har jquery använts som förenkla användning av javascript, dels med många viktiga funktioner, dels för att enkelt jobba med element.

Javascript används till sådant som gömmer och ändrar element utifån vad användaren gör medans jinja2 är det som laddar in från databasen ursprungligen. Observera att detta inte gäller på söksidan varav all data hämtas med hjälp av javascript.

All javascriptkod är dokumenterad med kommentarer i koden. Dessutom är javascriptkodens struktur designad så att enbart kod som används på den enskilda sidan ligger i den filen.

7 Felhantering

En enkel 404-sida visas om en invalid webbaddress har skrivits in eller nåtts på något annat sätt.

Enkel felhantering finns även för de sidor som laddar data från ”data.json” och/eller ”user.json” där en felsida visas om ingen användbar data returneras från datalagerfunktionerna, med ett passande felmeddelande.

Felsida ges alltså även om något av filerna är tomma, även om det kanske inte typiskt skulle räknas som ett fel i systemet.

8 Kodstandard

Målsättningen har varit från början att i så god mån som möjligt att ha en bra standard på koden. Därför har vi exempelvis använt oss av av den kodstandard i python där variabel och funktionsnamn är i små bokstäver varav ord separerade av understreck.

9 Testning

9.1 Enhetstester

Det enda enhetstestet som har körts är data-test.py från länk 4 i länksamlingen.

Detta testar om datalagrets grundfunktioner som tidigare varit förspecifierade har blivit implementerade på rätt sätt. Testerna kör funktionerna och undersöker om det returnerar förväntat resultat.

9.2 Testning av kod

Koden har förövrigt skapats så att den överlag ska vara tänkt lätt att testa. Exempelvis är nästan all kod i funktioner. Projekt och user infon har även grundläggande funktionalitet som adt vilket skulle kunna byggas vidare på. Funktionerna ligger i filnamn som har ett tydligt beskrivande namn om vad koden där i handlar om. Exempelvis ligger inte datalagrets funktion i samma fil som flaskfunktionerna.

Detta gör det enkelt att testa kod vilket även gjorts löpande. Istället för att behöva felsöka mycket kod går det snabbt att centrera problemet till en fil och därefter undersöka vad som gått fel.

10 Länksamling

1. `http://flask.pocoo.org/`
2. `http://getbootstrap.com/`
3. `https://www.ida.liu.se/~TDP003/current/portfolio-api_python3/`
4. `https://gitlab.ida.liu.se/filst04/tdp003-2018-database-tests`