LINKÖPINGS UNIVERSITET

TDP003 Projekt: Egna datormiljön

Systemdokumentation

Författare

Philip och Ismail

INSTITUTIONEN FÖR DATAVETENSKAP

Höstterminen 2020
Version 1.1

2020-10-15

Innehåll

1 Revisionshistorik

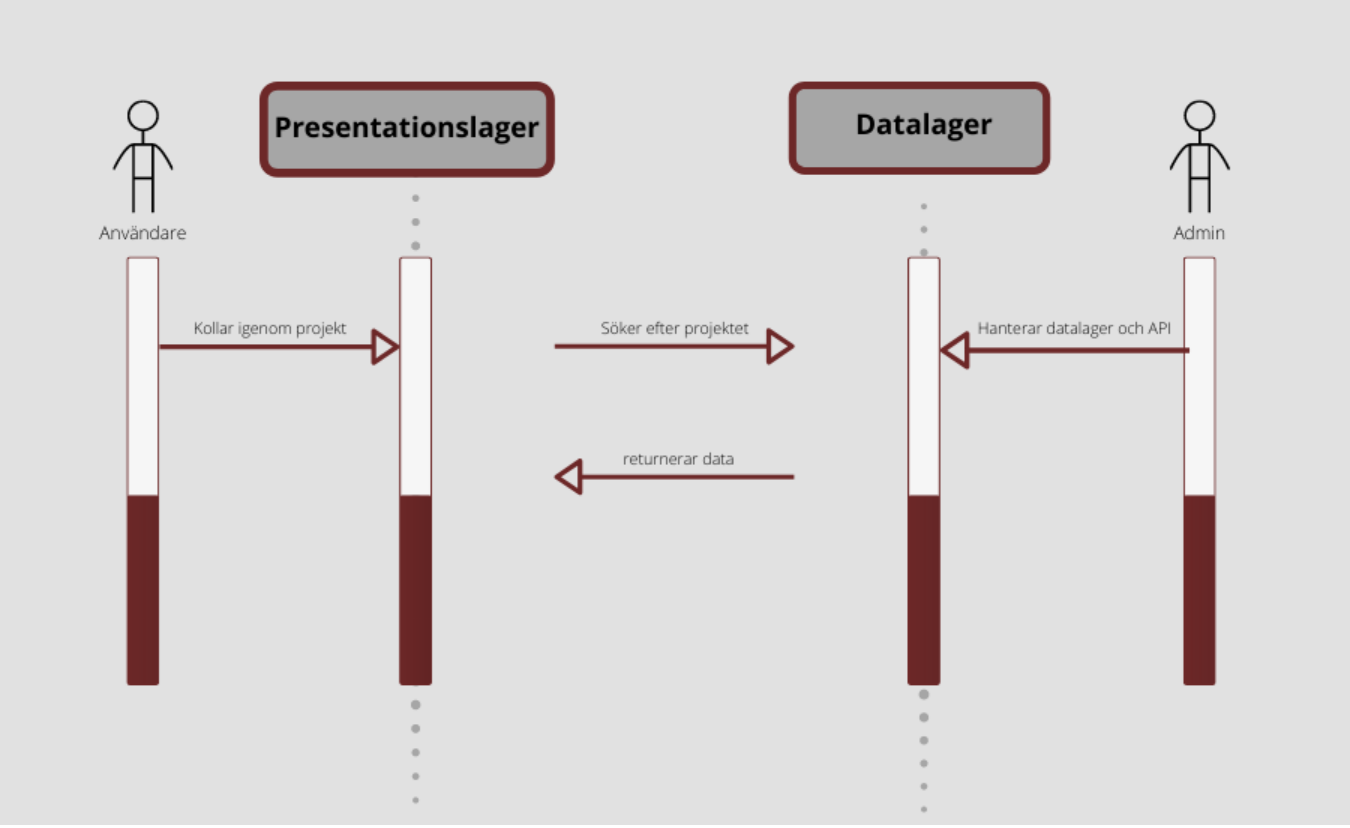
Ver.	Revisionsbeskrivning	Datum
1.0	Systemdokumentation skapad	15/10 2020
1.1	Systemdokumentation kompletterat	19/10 2020

2 Introduktion

Systemdokumentationen tillhör portfolio-systemet och tjänar som en guide för att förstå hur den är uppbyggd och fungerar. Presentationslagret är skriven i HTML, CSS, Python3, Flask och Jinja2 medan datalagret består av Python3-kod och en Json fil.

3 Översikt

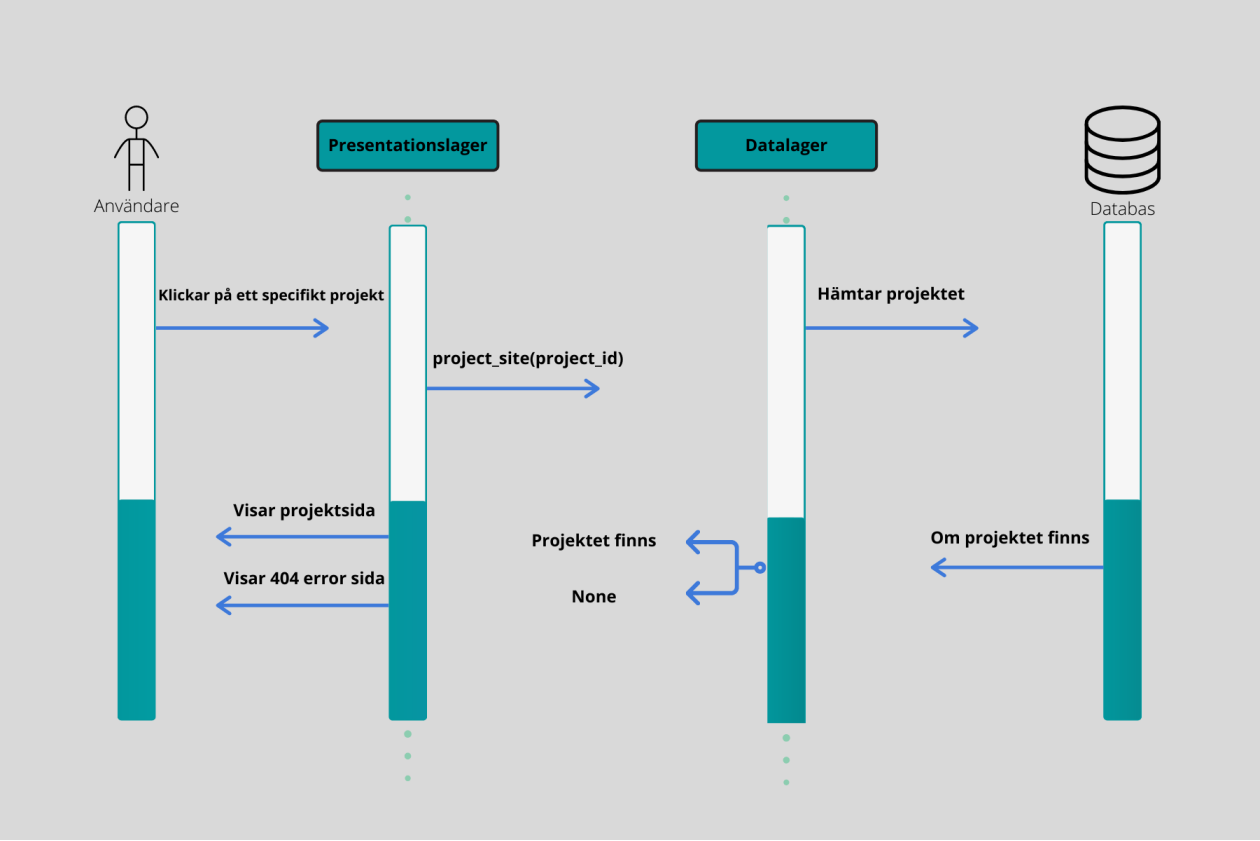
3.1 Översiktsbild



Figur 1: Översiktsbild över portfoliot.

I webbsidorna visas alla projekt. När användaren vill se och kolla igenom projekt efterfrågar presentationslagret information från datalagret, vilket datalagretsedan returnerar. Den som underhåller datalagret är admin.

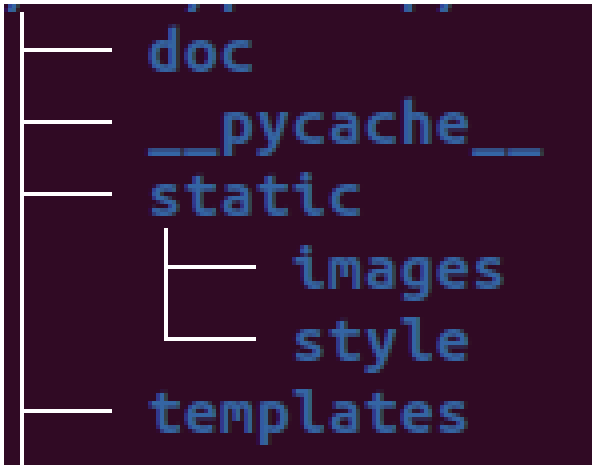
3.2 Sekvensdiagram



Figur 2: Sekvensföljd för individuella projektsidor.

Projektsidan laddar det specifika projektet som användaren har klickat på. Portfolion kommer då att efterfråga datalagret om projektet, vilket datalagret sedan letar igenom databasen (dvs. json-filen). Om den hittar den så returnerar datalagret projektet till presentationslagret. Om den inte finns visar presentationslagret error 404.

3.3 Filstruktur



Figur 3: Filstruktur.

- I huvudmappen ligger json-filen och all python-kod.
- I **doc/** finns dokumentationer så som systemdokumentation, installationsmanual osv.
- **static/** har två subkataloger: **images/** och **style/**. Images innehåller alla bilder som finns i portfoliot medan i style finns alla CSS-filer.
- I **templates/** finns alla HTML-filer.

4 Datalager och API

Datalagret består av en json-fil och sex API som tolkar datan i json-filen. För information om API:t, läs följande länk: `https://www.ida.liu.se/~TDP003/current/portfolio-api_python3/`.

5 Presentationslagret

Presentationslagret består av en python-kod och HTML-filer som samspelas med hjälp av Flask och Jinja2. Varje @app.route(), vilket bygger upp URL:en, har en funktion som returnerar HTML-filerna och datan som webbsidorna behöver. Vissa routes, som '/techniques' och '/list', använder sig av funktionerna i datalagret för att få fram projekt.

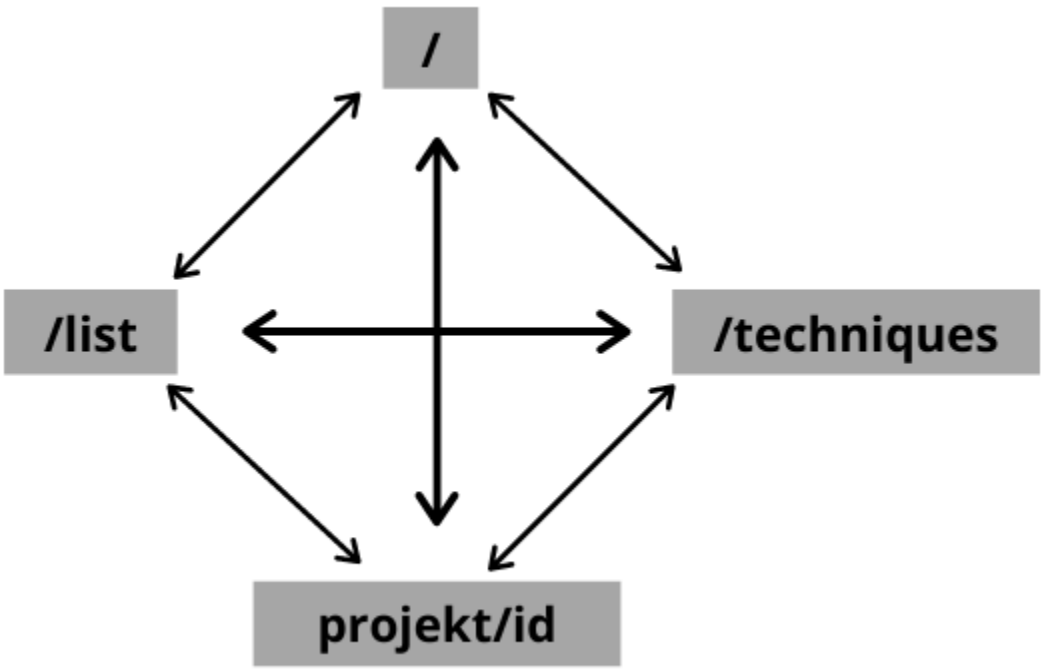
- Under **@app.errorhandler(404)** har vi en funktion som heter **page_not_found()**. Denna funktion returnerar pagenotfound.html, vilket bara säger "404 page not found". Detta behövs för att visa felmeddelanden till användaren så att användaren förstår vad som har gått fel.
- Under **@app.route('/')** har vi funktionen **index()**, vilket laddar datalagret och returnerar index.html—vår startsida.
- Under **@app.route('/techniques', methods=['POST', 'GET'])** har vi funktionen **techniques()**. Först laddar den upp datalagret och sedan har den en if och elif sats: om request.method är GET eller om request.method är POST. I båda fallen kollar den efter vilka tekniker som har klickats in. Skillnaden mellan dessa är att GET kommer aldrig ha några teknikar ifyllda, då GET-fallet sker när användaren hamnar i tekniksidan, så den kommer alltid att visa alla projekt. Det är inte förräns användaren klickar in tekniker och trycker submitdå filtreringen verkligen händer. Då hamnar den

under POST-satsen och samlar teknikerna med hjälp av request.form.getlist('techniques'), skickar in det i search-API:t och returnerar resultatet.

- Under **@app.route('/list, methods=['POST', 'GET'])** finns funktionen **get_search_fields()** (skall ej förvirras med search_fields inom datalagret). Precis som för techniques() laddar get_search_fields() först upp datalagret och har sedan en if och elif sats: request.method == 'GET' och request.method == 'POST'. I GET-satsen, vilket händer när användaren hamnar i sidan, kallar funktionen på search() med default-värden, dvs. utan sökord, utan tekniker och utan sökfält, och visar upp alla projekt. För POST-satsen sparar funktionen indata som användaren har skrivit in/klickat in i form av variabler med hjälp av request.form.get och request.form.getlist. Sedan stoppas variablerna in i search() och sist så returnerar funktionen resultatet från search().
- Under **@app.route('/project/<int:project_id>')** har vi funktionen **project_site(project_id)**. Olikt de andra funktionerna accepterar funktionen ett argument, vilket behövs för att kunna ladda upp det eftersökta projektet. Argumentet som stoppas in är projektets ID, vilket den får från URL:en. Funktionen laddar upp datalagret, letar efter det specifika projektet genom get_project(db, project_id) och returnerar all data om projektet.

6 Design

6.1 Översiktsbild



Figur 4: Översiktsbild över portfoliosidorna.

Portfoliot är uppbyggd i fyra URL:er: '/' (startsidan), '/list' (projektsidan), '/techniques' (tekniksidan) och 'project/id' (individuella projektsidor). I varje sida finns det en copyright-tagg längst ner och en navigerings-

fält längst upp. Navigeringsfältet länkar till startsidan, projektsidan och tekniksidan, vilket gör det lätt och smidigt att flytta sig igenom.

6.2 Startsidan

I startsidan ('/') listas alla projekt i form av bilder. Till höger finns det ett sidofält. Där ska det finnas en bild och en kort beskrivning av dig själv och ikoner som länkar till kontaktuppgifter, sociala medier, osv.

6.3 Projektsidan

I projektsidan ('/list') går det att söka igenom projekt. Sökfältet på sidan har följande funktionalitet:

- Sortering i fallande (descending) och stigande (ascending) ordning.
- Sortering efter olika fält (så som startdatum, projektnamn, osv.).
- Sökning i specifika fält (t.ex. projektnamn, kursnamn osv.).

6.4 Tekniksidan

I tekniksidan ('/techniques') listas alla projekt likt projektsidan. Istället för att ha ett sökfält har tekniksidan kryssrutor som filtrerar listan efter tekniker (så som Python3, C++, HTML etc.).

6.5 Projektsidorna

Varje individuell projektsida ('/project/id') har sitt eget ID, vilket definierar webbadressen. I den finns information om projektet: beskrivande text, en bild och ett översiktligt fält med kort information så som startdatum, slutdatum, kurs-id osv.

7 Felhantering

I presentationslagret används @app.errorhandler(404) för att visa användaren felmeddelande.

Vid underhåll och testning av presentationslagret, vilket körs med Flask i terminalen, används *debug mode*. Innan flask körs skrivs

```
$ export FLASK_ENV=development
```

i terminalen. Inom presentationslagret skrivs även

```
if __name__ == "__main__":
   app.run(debug=True)
```

8 Appendix

Figurer