

TDP003 Projekt: Egna datormiljön

Systemdokumentation

Författare

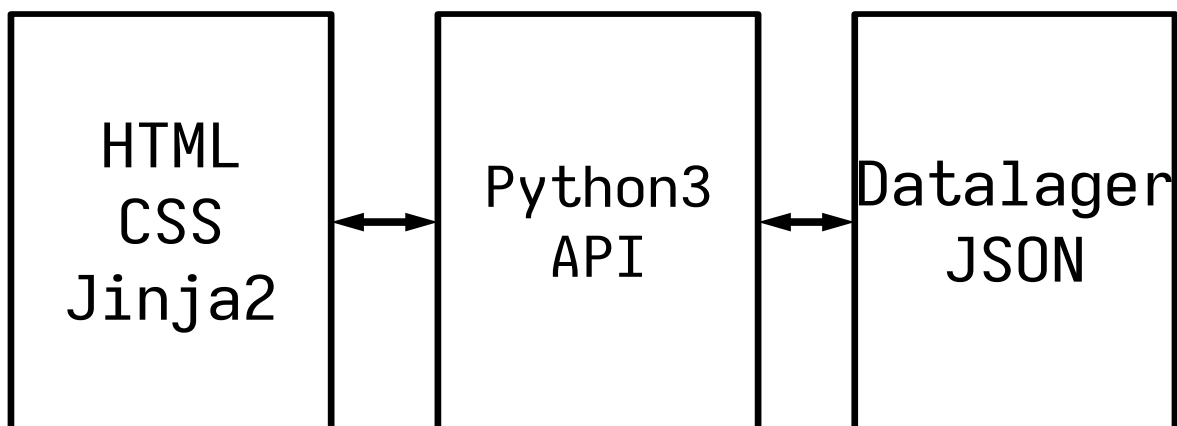
Daniel Huber, danhu849@liu.se
Jens Öhnell, jenoh242@liu.se

1 Revisionshistorik

Ver.	Revisionsbeskrivning	Datum
1.0	Systemdokumentation för Portfolio TDP003	151020

2 Översiktsbild

Hemsidans syfte är att presentera information om hemsidans ägare, och presentera projekt som denna har gjort på ett sökbart sätt. Hemsidan byggs upp av ett datalager samt ett presentationslager.



Figur 1: Översiktsbild för portfolion.

2.1 Datalagret

I datalagret görs information om olika projekt åtkommlig och sökbar. Det skrevs i Python3 och projekten sparas i JSON i en JSON fil.

2.2 Presentationslagret

I presentationslagret görs informationen tillgänglig för slutanvändaren på ett enkelt och intuitivt sätt. Lagret skrevs i Python 3 med ramverket Flask samt templatemotorn Jinja2.

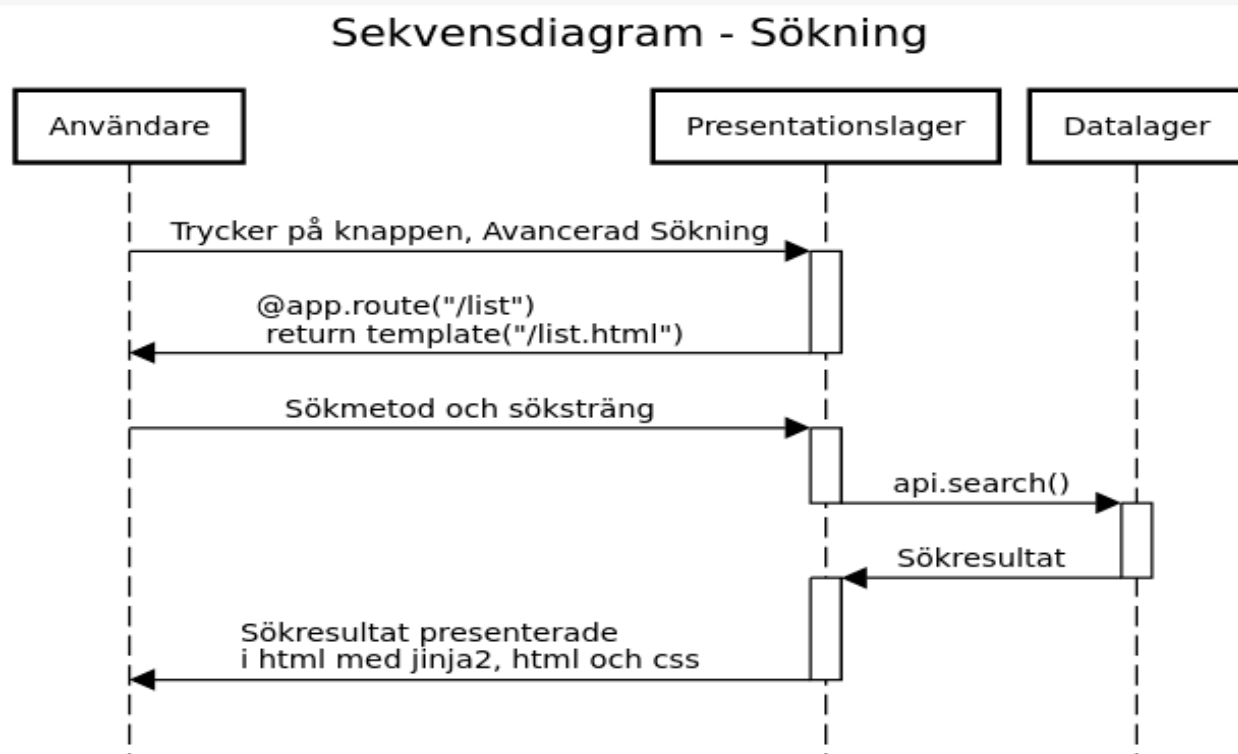
3 Specifikation - Presentationslagret

Skapandet av hemsidan har utgått från en kravspecifikation. Denna har varit vägledande i vilken funktionalitet sidan skall ha. Den fulla specifikationen kan hittas i dokumentet *Systemspecifikation av portfoliosystem* som återfinns på kurshemsidan.

Sammanfattningsvis utgörs hemsidan av följande:

- En förstasida med bilder.
- En söksida som gör det möjligt att söka och sortera på diverse fält för projekten.
- En tekniksida som gör det möjligt att sortera projekt på använda tekniker.
- Bilder, både thumbnails och fullstora, för alla projekt.
- För slutanvändaren begripliga felmeddelanden.
- Korrekta statuskoder.

4 Sekvensdiagram - Sökning



Figur 2: Sekvensdiagram för Sökning

I sekvensdiagram 2 visas det översiktligt hur användaren från (/home, /) presenteras med sökresultat.

5 Felhantering och Loggar

Samtliga portfoliehändelser loggas med hjälp av Pythons `logging.config` modul. Konfigurationen för loggningen definieras i filen `logging.cfg`. Loggar skrevs till filen `portfolio_log.log` i kronologisk ordning. Senast log längst ner. Båda filerna finns i root-katalogen för portfolio-appen. Loggar sparas med datumstämpel, viktighet, namn, tråd och meddelande. Alla loggar skrivs in i logfilen oavsett viktighet, men kan ändras vid behov genom att ändra `level=DEBUG` under `[logger_root]` i konfigurationsfilen `logging.cfg`. En filtrationsnivå över `DEBUG` rekommenderas ej då händelser som lett upp till kraschen kan ha filtrerats bort. Logfilen finns tillgänglig även när `flask` inte körs. Vid ny körning av `flask` skrivs inte `portfolio_log.log` över utan nya loggar skrivs in längst ner i filen.

Terminalkommandot `tail` med flaggan `-f` används vid `flask run` för att i realtid övervaka de senast tillagda loggarna i `portfolio_log.log`. Vid behov används flaggan `nN` där `N` definieras av antalet rader som visas. Utan flaggan `-n` visas de sista 10 raderna från filen i terminalen. Att alltid de 10 sista raderna visas från filen möjliggörs av flaggan `-f`.

```
tail -n15 -f /path/to/portfolio_app/portfolio_log.log
```

Utöver logfilen hanteras fel med hjälp av Pythons `print()` funktion som kan skrivas in i varje `@app.route()` funktion, men också egendefinierade funktioner. Informationen skrivs ut i terminalfönstret när funktioner aktiveras under en flask run körning. Med `print()` kan variabler från bland annat databasen komma åt. Flasks egen debug mode aktiveras genom att `FLASK_DEBUG` variabeln sätts till 1 innan flask run:

```
export FLASK_DEBUG=1
```

5.1 Testning/felsökning - Databas

Testning av datalagrets funktioner tillhandahålls av kursledningen och benämns enhetstester. Utöver de initiala enhetstesterna har fler med tiden lagts till. Enhetstesterna skrevs i Python3 där Pythons modul unittest (Unit testing framework) används. Testerna skrevs med kravspecifikationen i åtanke. Körning av testerna utan fel bevisar att funktionens krav uppfylls. Funktionerna för datalagret felsöks lättast med `print()` funktionen.

6 Kod

För fullständig dokumentation av kod, se dokumentet *Portfoliodokumentation*.