TDP003 Projekt:
Egna datormiljön

Systemdokumentation

Författare
Erik Lindow, erili445@student.liu.se
Erik Edling, eried975@student.liu.se

Examinator
Klas Arvidsson, klas.arvidsson@liu.se

Innehållsförteckning

1. Revisionshistorik

ver	Revisionsbeskrivning	datum
1.0	Första versionen klar.	14/10-14
1.1	Komplettering. La till rubrik 2 som introducerar det här dokumentet för läsaren. La till en länk till datalagrets specifikation och referenser till installationsmanualen. La även till en kort genomgång av system under rubrik 3. La till information om fel loggnign i rubrik 5.	24/10-14

2. Systemdokumentation

Du läser för tillfället systemdokumentationen för vårat system. Här kan du hitta detaljerad information om hur systemet fungerar och hur de olika delarna är beroende av varandra.

Det här dokumentet är menat för dig som är intresserad av att fortsätta utveckla vårt system.

Rubrik 3 beskriver arkitekturen av systemet översikligt, detta är en bild över hur de olika sidorna på hemsidan ligger i hierarkin och vilka delar som måste ha tillgång till vilka delar.

Rubrik 4 innehåller en bild som detaljerat visar vilka funktioner som kallas i vilken ordning när en specifik händelse sker på hemsidan, i det här fallet när användaren trycker på länken ‘Techniques’. Här hittar du även en kort genomgång av hur systemet fungerar i text.

Rubrik 5 går igenom felhantering, vart man ska leta när något går fel med systemet samt vart man kan hitta en lista med vanliga fel.

3. Översiktsbild av arkitekturen

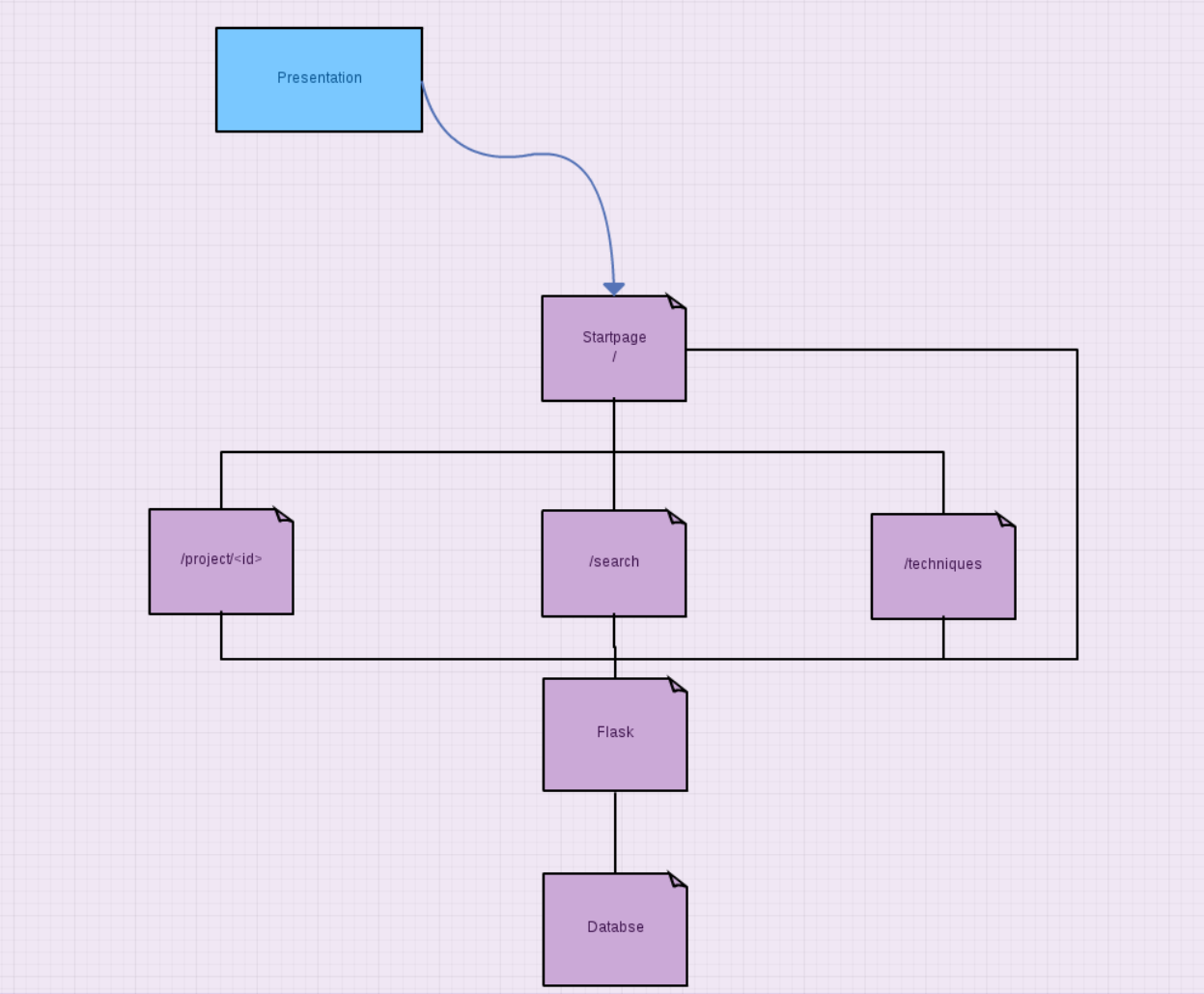
Arkitekturen är uppbyggd kring 3 grundpelare. Dessa är presentationslagret, datalagret, databasen med JSON filen. Det här avsnittet beskriver hur de olika lagrena ligger i relation till varandra för att ge en förståelse för systemets hierarki.
För information om katalogstrukturen, dvs. vart alla filer ska ligga för att systemet ska fungera, se installationsmanualen rubrik 2.1.

I kort fungerar systemet så här:

1. Användaren trycker på en länk (kan vara en sökknapp eller en statisk länk i huvudet) på sidan
2. Webbläsaren skickar då en request tillservern som tas emot av Flask
3. Flask kallar då på en eller flera funktioner i datalagret (om så krävs)
4. Datalagret returnerar önskad information till Flask som skickar ut det till respektive HTML
5. Jinja2 tar emot informationen och gör så att det visas ett snyggt HTML dokument för användaren

För att referera till bilden nedan skulle man kunna skriva punkterna ovan som så:
1.Presentationslagret -> 2.Flask -> 3.Database -> 4.Flask -> 5. Presentationslagret

En grafisk illustration över sidans arkitektur, kort info om varje del finns under bilden med referens till den här bilden inom parentes.



3.1. Presentationslagret (/, /project/<id>, /search, /techniques)

Presentationslagret är det som visar informationen rent visuellt för användaren. Det använder sig av Jinja2 för att kunna producera HTML kod utifrån information som det har får av Flask. All styling är gjord med CSS som vi har skrivit och den är helt statisk, vid intresse kan denna filen hittas i style mappen som ligger inuti static mappen (/static/style/main.css).

3.2. Datalagret (Flask)

Datalagret är den del av arkitekturen som laddar in en JSON fil och skickar informationen till presentationslagret som presenterar denna. Systemet kräver att datalagret stämmer överens med detta API “http://www.ida.liu.se/~TDP003/current/portfolio-api_python3/”.

3.3. Databasen (Database)

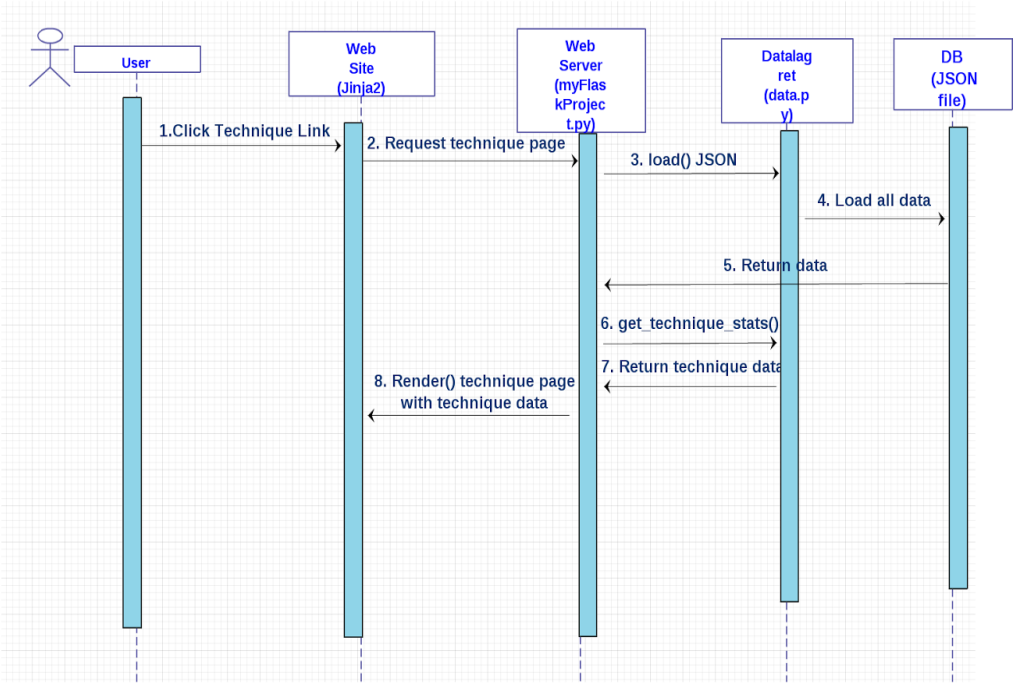
Databasen(JSON filen) används för att spara respektive projekts information. Informationen hämtas av datalagret som i sin tur kommer skicka vidare informatioen till presentationslagret som visar den snyggt för användaren.

4. Detaljerad genomgång av arkitekturen

Här nedan finns en bild på hur systemet jobbar när man trycker på Teknik sidan.

För mer detaljerade beskrivningar av hela datalagret, se sidan för datalagrets specifikation: http://www-und.ida.liu.se/~erili445/html/namespacedata.html

Nedan finns ett sekvensdiagram som grafiskt visar hur arkitekturen fungerar, hur de olika delarna skickar information och ger varandra information när användaren i det här fallet trycker på länken 'Technique'/'Teknik' på hemsidan.


User
Web Site (Jinja2)
Web Server (myFlaskProject.py)
Datalagret (data.py)
DB (JSON file)
1.Click Technique Link
2. Request technique page
3. load() JSON
4. Load all data
5. Return data
6. get_technique_stats()
7. Return technique data
8. Render() technique page with technique data

5. Felhantering

För en lista med vanliga fel, se installationsmanualen rubrik 2.3.2.

Den primära förebyggnaden av fel sker av tries och excepts. Blir något fel under "Try" kommer den gå vidare till Except och exekvera den kod som finns där. Det här är bra för att undvika fel men ännu viktigare är hur man hanterar de fel som inte går att förebygga.

Vi använder oss av en ganska vanlig logfunktion som finns i python och vi kallar den funktionen i varje viktig funktion så att man kan se vilken funktion som kallades precis innan det gick fel samt vilka parametrar den kallades med och i vissa fall vilka värden den returnerade. En oönskad men lycklig bonus vi upptäckte är även att Flask använder sig av denna logfunktion om den finns, och det gör den ju i vårat system, vilket betyder att alla requests till servern även loggas i loggfilen, dessa loggningar har ordet 'werkzeug' i början.

Ett bra ställe att påbörja felsökning är därför logfilen. Logfilen ligger i rotmappen i systemet, alltså på samma ställe som JSON filen och datalagret.
Generellt sett bör man inte behöva felsöka längre än så här eftersom nästa steg är att felsöka ren kod.

5.1 Avancerad felsökning

För att felsöka ren kod kan man använda sig av verktyget Python Debugger (PDb).
På det här sättet kan man med PDb gå igenom en python fil, förslagsvis datalagret (data.py), och felsöka koden genom att steg för steg gå igenom koden och se vad som faktiskt har hänt i exekveringen, vilka värden olika variabler har i olika skeden.
För att relativt snabbt komma igång med pdb kan man kolla:
http://pythonconquerstheuniverse.wordpress.com/2009/09/10/debugging-in-python/
vilket är en typ av quickstart till pdb och debugging i python.

Korta instruktioner om PDB
För att köra PDB måste du först i terminalen gå till mappen "MyPortfolio". Skriv sedan:
"python3 -m pdb data.py"

Olika användbara kommandon:

“continue”: Kör filen tills nästa stoppunkt påträffas eller ett fel inträffar. Påträffas ett fel kommer en lista upp med en traceback. Man kan då röra sig i tracebacken för att se vad som hänt. Detta kan göras med nästkommande två kommandon

“up”: Rör sig uppåt en frame i tracebacken

“down”: Rör sig nedåt ett frame i tracebacken.

Vill man kolla vad ett värde i en variabel har vid en specifik tillfälle kan man skriva variabelns namn för att se värdet.

Mer avancerad information om PDb finner du i pythons dokumentation “https://docs.python.org/2/library/pdb.html”.