LINKÖPINGS UNIVERSITET

TDP003 Projekt: Egna datormiljön

Systemdokumentation

Författare

Dennis Abrikossov, denab905@student.liu.se
Philip Bromander, phibr608@student.liu.se

INSTITUTIONEN FÖR DATAVETENSKAP

Höstterminen 2023
Version 1.2

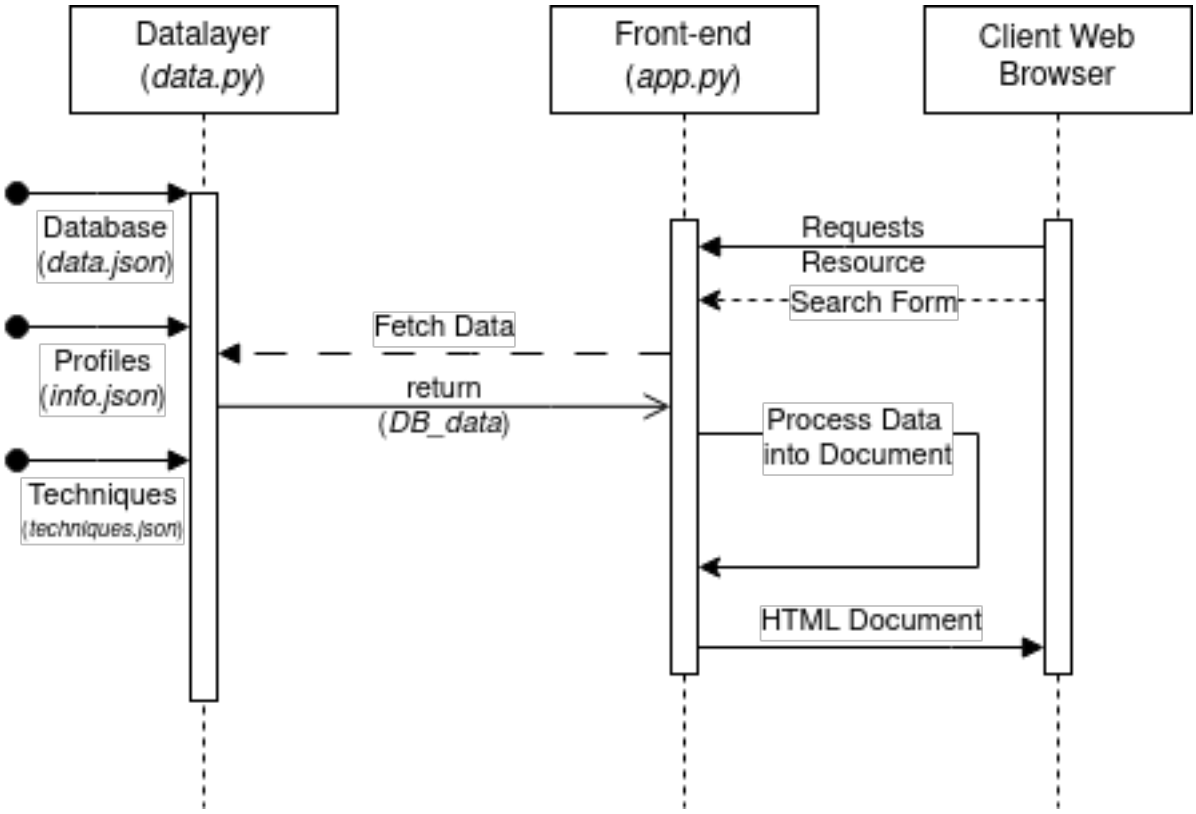
2023-10-12

1 Revisionshistorik

Ver.	Revisionsbeskrivning	Datum
1.2	Modifierade dokumentet för en första inlämning	12/10/23
1.1	Modifierade dokumentet för att hålla tabeller istället	05/09/23
1.0	Fyllt i dokument med deadlines & veckor	01/09/23

2 Flödesdiagram

I figur 1 kan vi observera flödesdiagrammet för det aktuella systemet.



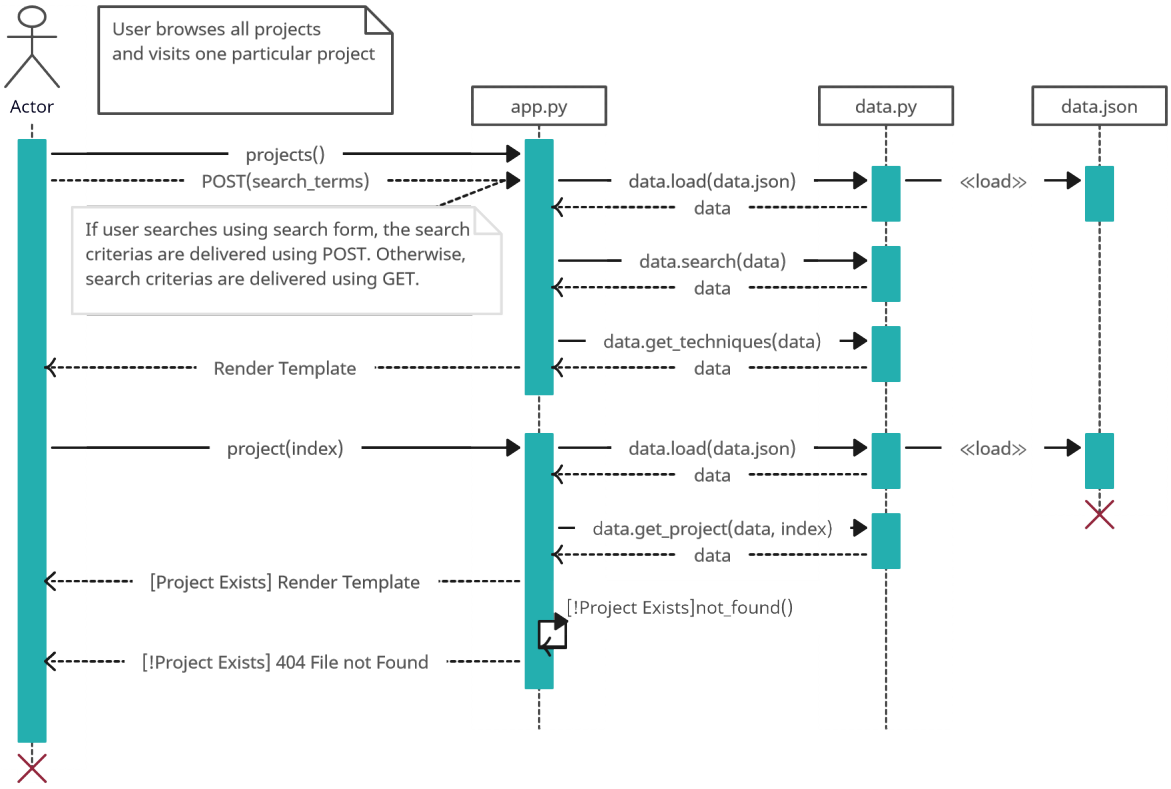
Figur 1: Flödesdiagram

När klienten önskar att hämta data initierar den en förfrågan. Front-end tar sedan hand om denna förfrågan och ansvarar för att interagera med användaren för att se till att vidare förfrågningar av användaren hanteras korrekt.

För att uppfylla användarens förfrågan hämtar front-end data från dataregistret via datalagret. Datalagret fungerar som en handhavare för data som behövs för att svara på klientens förfrågan. Här sparas all information och data som systemet behöver medans den hanterar en begäran.

Front-end bearbetar sedan datan datalagret hämtade och formaterade för att utföra ytterligare behandling eller formatering i mån för att skriva ut till en webbläsare som ett HTML-dokument. När front-end har hanterat datan godtyckligt skickar den tillbaka det färdigställda dokumentet till klienten, som ska presentera den till användaren som skapade begäran.

Flödesdiagrammet i figur 1 illustrerar hur informationen och datan rör sig genom systemet; från användarens förfrågan till hämtning och bearbetning av data från databasen i datalagret till den slutliga presentationen för klienten. Detta är en översiktlig illustration av hur systemet fungerar. Observera i figur 2 hur en användare kan söka upp ett specifikt projekt och hur interaktionen handskas mellan användaren, huvudprogrammet, datalagret, och databasen. I figur 2 börjar användaren med att klicka in sig på söksidan för projekt, antingen med eller utan sökkriterier, och sedan begär att få se ett specifikt projekt efter det att användaren har sett resultaten av sin sökning.



Figur 2: En användare söker efter ett specifikt projekt

3 Datalager

Följ URL:n för att se API:n för datalagret som erfordrat av Linköpings Universitet:
`https://www.ida.liu.se/~TDP003/current/portfolio-api_python3/data-module.html`

4 API för presentationlagret

4.1 Funktionsöversikt

Function	Function description
index()	Loads information from ”info.json” and renders it on the ”index.html” page.
projects()	Handles project search and sorting based on user input, rendering the results on the ”projects.html” page.
techniques()	Retrieves statistics on techniques used in projects and displays them on the ”techniques.html” page.
project(index)	Displays details for a specific project based on its index on the ”project.html” page.
not_found(err_message)	Handles 404 errors by displaying an error message on a custom error page.
generic_error(err_message)	Handles generic exceptions by displaying an error message on an error page.
log(level: str, message: str, source: request = ””)	This function is responsible for server logging, recording error messages, warnings, and debugging information, and saving them to a log file.

4.2 Funktionsdetaljer

index()	Loads information from ”info.json” using the ”data.load” function and renders the ”index.html” template with the extracted information as ”information_developers”.
Parameters:	None
Return value: Rendered Template Object	A rendered template for the ”index.html” page.

projects()	Handles GET and POST requests for the /projects route.
Parameters:	None
Return value: Rendered Template Object	A rendered template for the ”projects.html” page with search results.

techniques()	Loads information from ”techniques.json” and ”data.json”. Retrieves statistics for techniques used in projects using ”data.get_technique_stats”.
Parameters:	None
Return value: Rendered Template Object	A rendered template for the ”techniques.html” page with technique information and statistics.

project(index)	Handles GET requests for individual project details using the project's index.
Parameters:	
index: int	The index of the requested project.
Return value:	
Rendered Template Object	A rendered template for the "project.html" page with project details.

not_found (err_message)	Handles 404 errors by rendering an error page with a provided error message and a custom image.
Parameters:	
err_message: str	The error message to display.
Return value:	
Rendered Template Object	A rendered error template with the error message and image.

generic_error (err_message)	Handles generic exceptions by rendering an error page with a provided error message.
Parameters:	
err_message: str	The error message to display.
Return value:	
Rendered Template Object	A rendered error template with the error message.

log(level: str, message: str, source: request = "")	This function is responsible for server logging, recording error messages, warnings, and debugging information, and saving them to a log file.
Parameters:	
level: str	A fully capitalized string indicating the error level.
message: str	A string literal containing the log message
source: request = ""	The connection metadata (primarily for fetching IP).
Return value:	None

5 Felhantering och loggning

5.1 Felhantering

Om det uppstår ett fel i systemet medans det arbetar måste det hanteras på ett godtyckligt sätt så att inte förhindra servern ifrån att hantera nya förfrågningar. Detta hanterar systemet med enkel felhantering. Det finns två källor till problem: om Flask blir tillsagd att producera en sida som inte existerar, och om datalagret får en förfrågan som den inte kan hantera. Det första fallet är enkelt att hantera med en 404 ("not found") felkod. Får Flask en förfrågan om en undersida som inte existerar skickar den användaren till en speciell 404-felsida.

Den andra källan till fel är svårare att hantera godtyckligt eftersom systemet kan inte förutspå alla sätt datalagret kan misslyckas. Detta måste man hitta och underhålla genom att testa systemet. Ett fall som var under utvecklingen en källa till fel var när sökfunktionen fick en sträng som maskerades som en del av ett reguljärt uttryck och sökfunktionen blev förvirrad över hur den skulle söka. Detta är inte längre ett problem.

5.2 Loggning

För att hantera systemet i framtiden är det viktigt att veta vad som händer i systemet medans det körs och att kunna gå tillbaka i tiden för att läsa äldre felmeddelanden. Detta görs genom ett loggningsystem. I källkoden hanterar en funktion med namnet 'log' detta. Den skriver både ut i realtid felmeddelanden färgkodade i

terminalen och skriver samtidigt ner felmeddelanden till en fil med namnet 'log.md'. Logfilen är placerad tillsammans med huvudprogrammet och innehåller en lista av meddelanden, komplett med datumstämpel, felnivå, IP-address, och meddelande. Felnivå berättar hur allvarligt meddelandet är och kommer i fyra klasser: 'DEBUG' för användning under utveckling, 'INFO' för ofarliga körmeddelande, 'WARNING' för de mestadels ofarliga felmeddelanden, och 'ERROR' för de mest kritiska felmeddelanden.

6 Enhetstester

För att säkerställa att systemet fungerar som det ska under alla oförutserbara körsätt måste man testa programvaran. Det mest konkreta sättet att utföra detta på är genom enhetstester. För projektet var enhetstesterna för datalagret givna; dessa tester ser till att den mest grundläggande funktionaliteten fungerar. Till exempel testas'load'-funktionen om den laddar projekten rätt och är i korrekt ordning, vilket måste säkerställas manuellt. Den kollar också om det går att söka på ett ord med flera sökfält aktiverade samtidigt. Den testar däremot inte när man söker på flera tekniker samtidigt, om det kommer upp projekt som har alla markerade tekniker och inte bara första bästa.