HÖGSKOLAN DALARNA

GIK339 – Labb 2

API-kommunikation

Uppdaterad

2024-09-23

GIK339 – LABB 2

API-kommunikation

För 4hp i kursen Dynamiska Webbapplikationer ska två labbar och ett onlineprov utföras. Detta är den andra labben. Den utförs i labbgrupper, enligt de som finns på Learn (markerade Labbgrupp XX).

Läs igenom instruktioner för förberedelser, uppgifter och inlämning noggrant och tveka inte att höra av er till mie@du.se om det är några frågor.

Lycka till!

HÖGSKOLAN DALARNA

GIK339 – Labb 2

API-kommunikation

Uppdaterad

2024-09-23

Innehåll

HÖGSKOLAN DALARNA

GIK339 – Labb 2

API-kommunikation

Uppdaterad

2024-09-23

Förberedelser

1. Skapa ett lokalt repository på någon av er dator, exempelvis enligt instruktioner i *Git, GitHub, & GitHub Desktop for beginners*.
2. Döp det till **gik339-[gruppnummer]-labb2**.
    a. Ta inte med punkten ovan. Det blir fel på repositoryt om den slutar med en punkt.
    b. Om ni följer videon ovan kommer detta skapa en mapp av samma namn på en given position (angivet i fältet **Local path** i GitHub for Desktop).
3. Öppna mappen för repositoryt i VS Code.
4. Utför labben enligt instruktioner
    a. *OBS!* när ni gjort förändringar och ska *publicera ert repository till GitHub*, se till att inställningen **Keep this code private** *inte* är markerad.
5. Om gruppen vill arbeta från olika datorer, se till att båda jobbar utifrån samma repository på GitHub.
    a. Alltså, person 2 ska inte skapa ett nytt repository, utan hämta *det befintliga på GitHub* och ladda ner det lokalt till sin dator.
    b. Det ska alltså bara finnas ett (*1*) repository som ni ska arbeta mot, och vars länk ni ska lämna in enligt avsnittet Inlämning.

Startfiler

För att underlätta för er finns startfiler för denna labb på GitHub. Ni kan ladda ner koden direkt som en zip-fil på denna adress: *https://github.com/mie-du/gik339-ht23/archive/refs/heads/Labb2.zip*. Om ni eventuellt redan har laddat ner kursens repository i samband med tidigare lektioner, finns startfilerna i en branch vid namn **Labb2**.

Ni får denna föruppsatta miljö av några anledningar. Exempelvis har vi inte specifikt arbetat med SQL i denna kurs, och det är heller inget som examineras. Detta gör att jag vill att ni ska kunna fokusera på det som är viktigt istället.

Så börja med att ladda ner koden på valfritt sätt. Kopiera sedan startkoden och **klistra in den i er egen mapp** för Labb 2. Alltså den mapp som heter något i stil **gik339-[gruppnummer]-labb2**, och som skapades när ni satte upp git-repositoryt för labben enligt instruktionerna ovan.

Node.js och SQLite

Innan denna labb utförs behöver ni ha installerat Node.js och ha tillägget SQLite i VS Code. Postman är också fördelaktigt under arbetets gång. Om ni har följt med i **Lektion 4** ska ni redan ha detta förberett, annars hänvisar jag till instruktionerna i dokumentet för **Lektion 4**, under avsnittet **Förberedelser – Miljö**.

HÖGSKOLAN DALARNA

GIK339 – Labb 2

API-kommunikation

Uppdaterad

2024-09-23

Uppgifter

Nedan följer de uppgifter som ni ska ha utfört för att få godkänt på labben. Uppgifterna är *minimum*, vilket innebär att ni får labba runt med egna tillägg om ni vill, så länge *instruktionerna följs*. Instruktionerna i uppgifterna kanske inte alltid är helt logiska, men det finns en poäng med dem. Vid vissa tillfällen ska ni hitta och motivera egna lösningar, vid andra tillfällen *ska* ni använda er av något specifikt enligt mina instruktioner, exempelvis ett visst npm-paket.

Det är möjligt att jag gör större antagande gällande vad ni ska kunna vid denna labb, så ni får se till att gå tillbaka och repetera om det finns moment och begrepp som inte riktigt har satt sig från tidigare delar av kursen.

Jag hänvisar ibland till de föreläsningar och lektioner som behandlar ett givet område som ni ska arbeta med. Alla föreläsnings- och lektionsfiler finns här. Ni behöver titta i mapparna för respektive vecka för att hitta det hänvisade materialet. Länkar till material finns också i kursrummet.

Notering rörande generativ AI (GAI)

Vid utförande av uppgifterna är det *tillåtet* att använda er av GAI för att få idéer, eller förtydliga och förbättra redan skriven kod. Den färdiga koden måste dock måste vara er egen och ni måste kunna visa att ni förstår och kan kvalitetssäkra genererad kod.

Om ni har använt generativ AI behöver ni visa vilka delar som ni har fått stöd i att skapa och förklara hur processen som användes för att generera innehållet. Vidare detaljer rörande hur detta ska redovisas finns i avsnittet *Inlämning*.

Uppgift 1 Förberedelser, backend

Startfilerna
I startfilerna finns ett antal filer och mappar

1. En mapp vid namn **server** med
    a. En fil som heter **`server.js`**
    b. En fil som heter **`gik339-labb2.db`**. Detta är en SQLite-databas med en förkonfigurerad tabell med förifyllda data.

Hädanefter ska ni arbeta i mappen **server**, tills annat anges.

NPM-paket och .gitignore

1. Se till att terminalen utgår från mappen **server** (högerklick på mappen **server** → **Open in Integrated Terminal**)
    a. eller kommandot **`cd server`** i **Terminalen**.
2. Skriv följande kommandon i terminalen:
    a. **`npm init -y`**.
    Detta kommer att skapa filen **package.json**.

   b. `npm install express sqlite3 --save`.
      Detta kommer att installera paketen och lagra dem i en mapp vid namn **node_modules**.
3. Skapa en fil vid namn `.gitignore` i mappen **server**
4. Skriv `node_modules/` i filen och spara.

Steg 3 och 4 är *viktigt*, då det gör så att mappen **node_modules**, som kan bli att innehålla enormt mycket filer, inte följer med i ert repository och till GitHub. Jag vill *inte* se mappen **node_modules** i ert git-repository!

Nodemon
Om du inte redan har installerat det i samband med tidigare lektioner, se till att du har npm-paketet `nodemon` installerat.

1. Öppna en terminal och skriv följande kommando:
   a. `npm install -g nodemon`.
      Notera: Alternativet `-g` betyder att paketet installeras globalt. Om detta ger error, uteslut då `-g`.
2. I filen **package.json**, objektet `scripts`, ändra raden `"start": "node server.js"` till `"start": "nodemon server.js"`.

Uppgift 2 Skapa webbserver och första route

Nu ska själva webbservern skapas. Den ska framöver användas från ett **frontend** för att hämta data via en GET-request.

För att möjliggöra detta behöver servern ha ett API som tar emot GET-förfrågningar.

Se avsnitt **Routing med Express** i **Föreläsning 4** och dokument/inspelning för **Lektion 4** för detaljer om att skapa en server i Node.js med hjälp av Express.

All följande kod ska skrivas i filen I filen **server/server.js**.

Detaljer:

1. Importera **npm-paketet** `express` genom att skapa en variabel vid namn `express` och tilldela den värdet `require("express")`.
2. Skapa en variabel som **ska hålla själva serverobjektet**. Döp den till `server` och tilldela den värdet `express()`;
3. Sätt övergripande inställningar för servern med hjälp av `server.use()`. Denna kod behöver ni inte kunna i huvudet, så ni får den nedan (detta är inte en bild, det går att klippa och klistra):

```
server
  .use(express.json())
  .use(express.urlencoded({ extended: false }))
  .use((req, res, next) => {
    res.header('Access-Control-Allow-Origin', '*');
```

```
    res.header('Access-Control-Allow-Headers', '*');
    res.header('Access-Control-Allow-Methods', '*');

    next();
});
```

4. Starta webbservern genom att använda metoden `.listen()` hos **serverobjektet** (variabeln `server`) och ange **önskad port** som servern ska köras på (jag brukar använda port `3000`) som första argument. Ni kan också skicka ett valfritt andra argument i form av en callbackfunktion som kommer att köras när servern startat, ifall ni exempelvis vill ha feedback på att servern körs.
5. Skapa sedan en första route, som ska vara en get-route. Detta görs med hjälp av metoden `.get()` hos serverobjektet (variabeln `server`).
6. Ge metoden `get()` två argument:
    a. en **endpoint**, bestående av strängen "`/users`" , och
    b. en **callbackfunktion** i form av en **anonym arrowfunktion,** som tar parametrarna `req` och `res`. Lämna funktionen tom i övrigt för nu. Parametrarna måste tas emot i den ordningen.
7. Starta er server genom att i terminalen skriva `npm start`.
    a. Se till att terminalen utgår från mappen **server** (**högerklick på mappen server** → **Open in Integrated Terminal**).

Uppgift 3 Använd sqlite3 för att kommunicera med databas

Vi har inte gått igenom SQL specifikt för denna kurs, men detta behövs för att ni ska kunna fortsätta med labben. Ni har förberedda filer bland startfilerna. Där finns en databasfil vid namn **gik339-labb2.db**. I den databasen finns en tabell som heter `users`. Den ska ha kolumnerna `id`, `firstName`, `lastName`, `username` och color, och innehålla några rader av testdata att arbeta vidare med.

Om ni har SQLite-tillägget installerat i VSCode kan ni öppna databasen i VS Code för att se hur den ser ut (**högerklick på filen gik339-labb2.db** → **Open Database**), eller ställa en fråga mot den (**högerklick på filen gik339-labb2.db** → **New Query)** för att skriva SQL-frågor mot den.

Om ni inte utgår från startkoden, får ni själva skapa motsvarande utgångspunkt i en SQLite-databas.

All följande kod ska skrivas i filen I filen **server/server.js**.

Paketet sqlite3
I filen **server.js**.

Importera npm-paketet sqlite3 genom att skapa en variabel vid namn `sqlite3` och tilldela den värdet `require("sqlite3").verbose()`. Variabeln innehåller nu ett **databasobjekt**.

Hämta data för att skicka som svar vid GET-förfrågan
Resten av uppgiften ska skrivas i **callbackfunktionen** för serverns `.get()`-metod (skapad enligt *Uppgift 2, steg 6b*).

Använd **databasobjektet** (variabeln `sqlite3`) för att skapa en databaskoppling och lagra den i en variabel.

Hämta sedan alla rader ur tabellen **users**. Se avsnitt **SQL i Node.js** i **Föreläsning 4** för detaljer om att arbeta med SQLite i Node.js. För ytterligare information, se också *dokumentation för SQLite i Node.js*.

Detaljer:

1. Skapa en variabel vid namn `db` och tilldela den värdet `new sqlite3.Database("./gik339-labb2.db");`,
    a. Detta skapar en koppling till databasfilen med namnet **gik339-labb2.db**.
2. Använd databasobjektets (`db`) funktion `all()` för att ställa en fråga mot databasen.
3. Funktionen `db.all()` tar en **SQL-fråga** som första argument, och en **callbackfunktion**, som kommer att köras när förfrågan mot databasen är färdig, som andra argument.
    a. SQL-frågan som ska användas som första argument **ska vara** `SELECT * FROM USERS`.
    b. Callbackfunktionen ska ta parametrarna `err` och `rows`, där `err` kommer att innehålla eventuella felmeddelanden från databasen, och `rows` kommer att innehålla de faktiska data som hämtades ur databasen om allt gick bra. Parametrarna måste tas emot i den ordningen.
4. I callbackfunktionen, som alltså körs när frågan har behandlats hos databasen, ska svaret skickas tillbaka till klienten som gjorde förfrågan via `res.send(rows);`.
    a. *Välj själva* om ni vill hantera eventuella fel (innehåll i `err`-variabeln) genom att exempelvis skicka en annan **statuskod** och ett annat **meddelande** tillbaka till klienten, exempelvis `res.status(500).send(err);`.

Uppgift 4 Testa backend

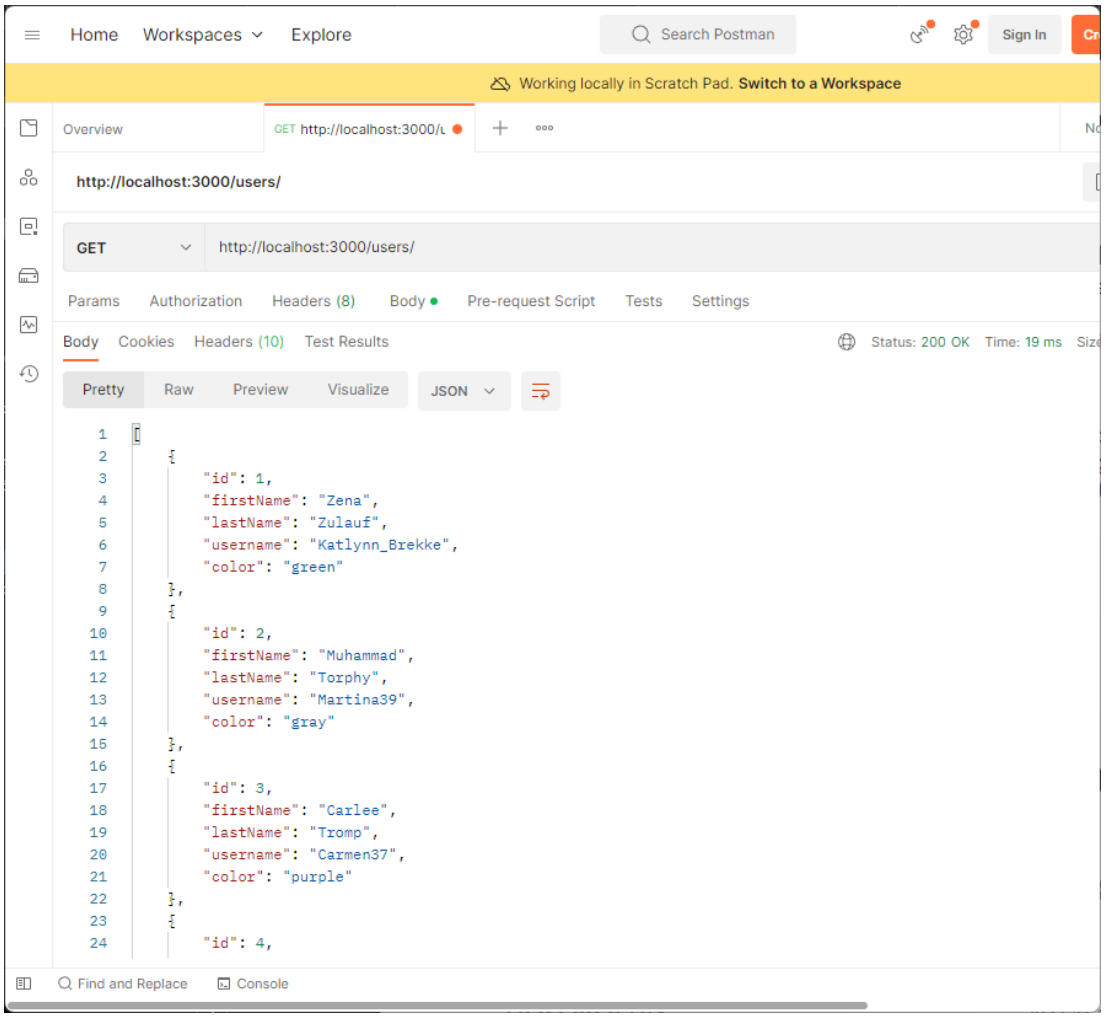
Innan ni går vidare bör ni testa det ni gjort såhär långt.

Eftersom vi bara gjort en GET-route, går det bra att testa den direkt i **webbläsaren**, men det går även bra att använda **Postman** eller annat verktyg för att testa routes.

Detaljer:

1. Öppna webbläsaren och gå till url:en **http://localhost:[port]/users** där **[port]** är den port du angav i *Uppgift 2 steg 4*. Url:en kan se ut exempelvis såhär: *http://localhost:3000/users*.
    a. *Alternativ*: använd en **webbläsare** för att navigera till samma url som ovan.
2. Output ska se ut något i stil med *Figur 1*.

Oavsett var ni testar bör ni få ut en **array** innehållande **tio objekt** som vart och ett representerar en **user** med egenskaperna **`id`**, **`firstName`**, **`lastName`**, **`username`** och **`color`**. Alltså innehållet i tabellen **users** i databasen.



Figur 1 - Bild på förväntat resultat i Postman (endast några av totalt 10 objekt är synliga)

Uppgift 5 Förberedelser, frontend

Lämna mappen **server** för nu. Skapa en mapp parallellt med mappen **server** som ni kallar **client**. Däri ska följande filer skapas:

1. **index.html**
   Skapa en HTML-sida ett `script`-element som hänvisar till filen **script.js**. Resten av innehållet på sidan får ni *välja helt själva*. Webbsidan kommer att till stor del få sina element via JavaScript, så den kan vara i stort sett tom i detta läge. Skriv gärna gruppnummer i exempelvis `title`-elementet eller något annat element på er site.
2. **Valfri CSS-fil/styling**
   Om ni vill kan ni skriva separat styling i en **.css**-fil eller `style`-element i HTML-filens `head`-element.
   - På det viset kan ni istället för att sätta all styling via JavaScript, hänvisa till **befintliga CSS-klasser** eller liknande för att styla de element som dynamiskt kommer att tillkomma via JavaScript.
   - Det går också bra att använda valfritt CSS-ramverk.
3. **script.js**
   Skapa filen **script.js** och dubbelkolla att den länkats korrekt via `script`-elementet i **index.html**. Resten av labben ska utföras där.

Uppgift 6 Fetch

All följande kod ska skrivas i filen **client/script.js**.

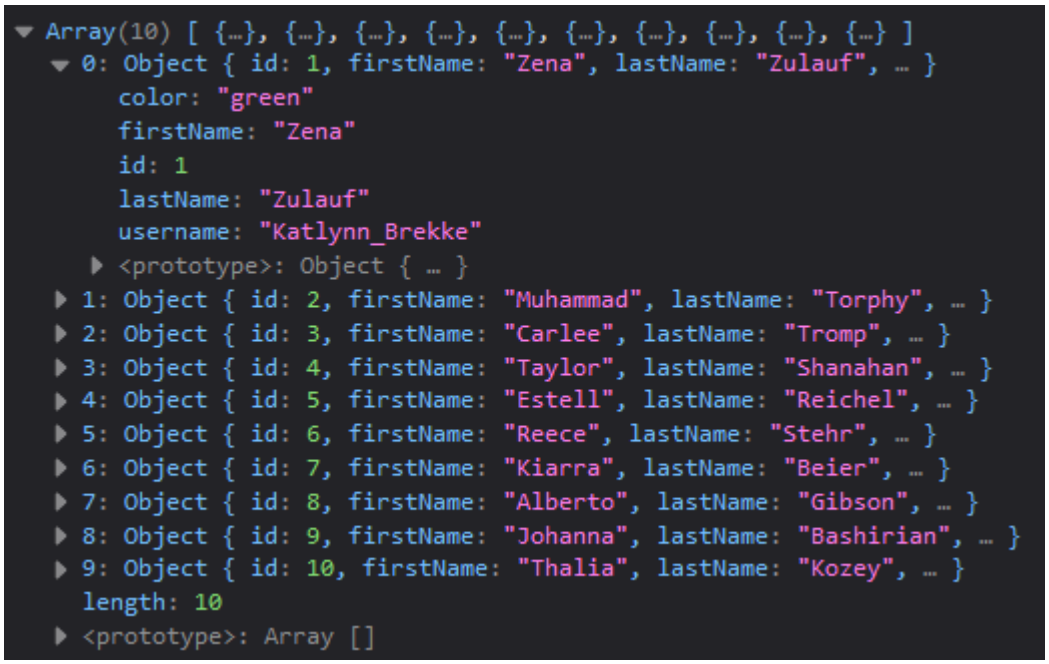
Nu ska en **koppling mot vår server** skapas för att hämta upp de users som ligger i databasen via en GET-förfrågan.

Viktigt! Se först till att servern körs. Ifall ni stängt VS Code vid något tillfälle stängs även servern, så se till att ni startat servern enligt *Uppgift 2 steg 7*.

Detaljer:

1. Använd JavaScripts inbyggda `fetch()`-API för att göra en fråga mot er **egen server**.
   a. Se avsnitt **API-kommunikation** i **Föreläsning 5** för mer om Fetch API.
2. Metoden `fetch()` ska som enda argument (i detta fall) ha url:en dit förfrågan ska skickas.
   a. Adressen till er server är densamma som ni använde för att testa ert backend i *Uppgift 4*.
3. Hämtningen av data mot er egen server sker asynkront, så *ni får välja* hur ni vill hantera detta asynkrona anrop, antingen med `then()` eller `async`/`await`.
   a. Se avsnitt **Asynkron JavaScript** i **Föreläsning 5** för detaljer om hur de olika varianterna att hantera asynkrona anrop.
   b. Notera att om ni använder `async`/`await` måste hela `fetch`-anropet i sin tur ligga i en funktion deklarerad med nyckelordet `async`.

4. Efter att svaret kommit tillbaka från servern behöver det ”översättas” till en **array av JavaScript-objekt**. Detta görs med hjälp av `response`-objektets `.json()`-funktion.
    a. Se avsnitt **API-kommunikation** i **Föreläsning 5** och dokument/inspelning för **Lektion 5** för mer om hur man tar emot och behandlar data från en server.
5. Funktionen `.json()` är i sig själv asynkron och måste hanteras därefter.
6. Det ni nu ska ha är en **array av JavaScript-objekt som representerar användare - user**. Varje **user**-objekt ska ha egenskaperna `id`, `firstName`, `lastName`, `username` och `color`.
7. Gör en `console.log` för att verifiera att du har fått ett resultat likt *Figur 2*.



Figur 2 - Output efter hämtning från server och ”översättning” av response

Uppgift 7 Skriv ut users i en HTML-lista

All följande kod ska skrivas i filen I filen **client/script.js**.

Nu ska ni använda **users**-arrayen för att skapa dynamisk HTML att lägga till på webbsidan.

Var just denna array av **users** finns tillgänglig vid denna punkt beror på hur ni löste *Uppgift 6 steg 3*. Det kan vara i en `then()`-callback om ni använde `then()` för att behandla data från server, eller i en variabel om ni använde `async`/`await`.

Se avsnitt **Påverka HTML via JavaScript** i **Föreläsning 3** för mer om att skapa HTML via JavaScript.

Notera: Jag kommer i vidare beskrivning instruera er att använda ett `ul`-element med enskilda `li`-element för att representera varje **user**, men ni får egentligen *välja själva* vilka element ni använder för att skriva ut **user**-objekten. Poängen är att innehållet i **users**-arrayen ska **läggas till i DOM-trädet** så att det **syns på webbsidan**. Min användning av `ul`/`li` är mer för att förenkla beskrivningen av stegen nedan.

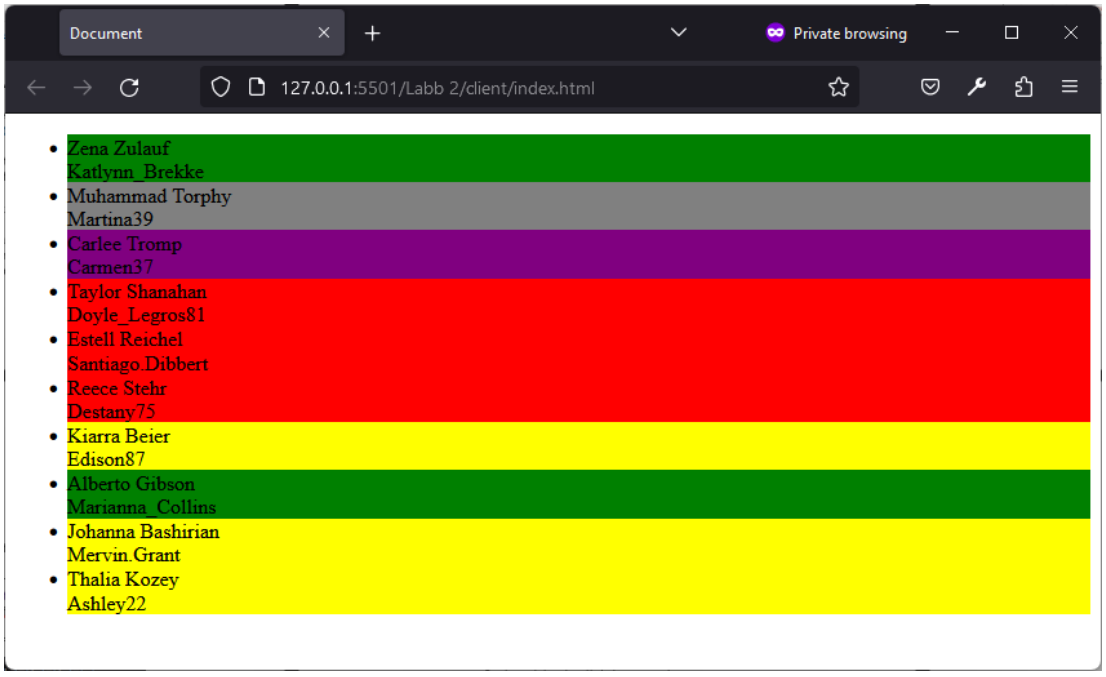
Detaljer:

1. Skapa ett `ul`-element, antingen via `document.createElement()` eller via en **templatesträng**.
    a. Styla `ul`-elementet med hjälp av egna klasser, ett CSS-ramverk eller styling direkt i JavaScript.
    b. Ge elementet andra attribut om så önskas.
2. Loopa igenom innehållande **users**.
    a. Använd valfri loop, exempelvis en traditionell `for`-loop, `forEach()` eller `map()`.
3. För varje element i **users-arrayen**
    a. Skapa ett `li`-element, antingen via `document.createElement()` eller via en **templatesträng**.
    b. Se till att **alla egenskaper** hos respektive `user`-objekt presenteras inuti `li`-elementet.
        i. Någon färg i `li`-elementet, exempelvis **bakgrund**, **text** eller **ram**, *ska vara* någon variant av färg som finns i egenskapen `user.color`. *[1]*
        ii. Använd i övrigt valfri styling och valfria ytterligare element, såsom `div`-element rubriker och paragrafer, för att organisera de olika egenskaperna inuti `li`-elementet på ett snyggt sätt.
    c. Lägg till `li`-elementet **sist** i `ul`-elementet på ett lämpligt sätt. *[2]*

[1] Ni kan exempelvis sätta en klass med samma namn som färgen i fråga, där ni själva då bestämmer detaljer kring hur den aktuella färgen ser ut. Ni kanske exempelvis inte gillar CSS förbestämda färg `red`, då kan ni skapa en CSS-klass för att göra en egen variant av röd att använda.

[2] Exempelvis `insertAdjacentElement`/`HTML` eller annan metod för att lägga till dynamisk HTML, enligt beskrivet i exempelvis **Föreläsning 3**.

4. Lägg till `ul`-elementet i DOM-trädet på ett lämpligt sätt. [2]
5. Kontrollera webbsidan. Den ska som minimum se ut något i still med Figur 3. Men jag *hoppas innerligt* att ni gör den snyggare än så. Alle bör ha lärt er bättre vid det här laget. 😊



Figur 3 - Otroligt fult exempel på HTML genererad från data hämtad ur databasen, via eget backend och dess API. Absolut minimum och godkänt, men gör det snyggare, tack!

HÖGSKOLAN DALARNA

GIK339 – Labb 2

API-kommunikation

Uppdaterad

2024-09-23

Inlämning

Nedanstående ska lämnas till uppgiften märkt Labb 1 i kursrummet. Vidare instruktion och datumangivelser finns i kursrummet.

Kod

- Lämna in länk till GitHub-repository som skapades och lades upp på GitHub under *förberedelserna*.
    - Länken ska se ut något i stil med: https://github.com/[**ditt-github-användarnamn**]/gik339-[**gruppnummer**]-labb2.
    - Det är möjligt att jag underkänner er om jag hittar en **node_modules**-mapp i ert repository.
- *Observera!* Lämna **inte** in en zip:ad mapp med er kod!

Loggar

Om generativ AI har använts ska ett dokument innehållande dokumentation om hur den använts inkluderas i inlämningen. Detta kan röra sig om exempelvis

- Loggar från **ChatGPT** eller andra chatverktyg
- Kommandon som ställts till **GitHub Copilot** och vilket svar som gavs.

Video

Gör en inspelning på *max 5 minuter*. Ni ska båda ha kameror på och namn synliga så att jag kan se vem som pratar (kan exempelvis uppnås genom att använda Zoom).

I videon ska ni:

- Välja två uppgifter *vardera* som ni demonstrerar och förklarar koden för.
    - Välj en av uppgifterna *Uppgift 2* eller *Uppgift 3* rörande **backend**.
    - Och en av uppgifterna *Uppgift 6* eller *Uppgift 7* rörande **frontend**.
- Om ni använt generativ AI ska ni i era uppgiftsredovisningar också peka på den specifika koden och förklara på vilket sätt den förbättrar er kod.
- Ni ska alltså båda demonstrera både frontend och backend. *[3]*

Detaljer rörande moment som ska förklaras för respektive uppgift

- För *Uppgift 2*, visa och förklara:
    - Vilka kommandon i terminalen och rader kod möjliggör att ni kan använda **Express** till er server?
    - Vilken rad kod **startar servern**?
    - När körs callbackfunktionen som anges som andra argument till **`get()`**?
        - Vilken typ av förfrågan? Till vilken **adress**/**endpoint**?)
    - Vad representerar objektet **`req`**?

[3] Alltså: en av er kan välja Uppgift 2 och 6, den andra Uppgift 3 och 7. Ni får inte välja samma, men ni får själva välja den kombination av frontend/backend-uppgift att demonstrera.

- För *Uppgift 3*, visa och förklara:
  - Vilka kommandon i terminalen och rader kod möjliggör att ni kan använda **SQLite** i ert Node.js backend?
  - Vilken/vilka rader kod skickar ett response till klienten som gjorde förfrågan?
  - Vad representerar objektet `res`?
- För *Uppgift 6*, visa och förklara:
  - Vad gör funktionen `fetch()`?
  - Vilket resultat får när man använt funktionen `fetch` och det **asynkrona anropet är färdigt** – vilka data kommer tillbaka?
  - Hur hanterar ni det asynkrona anropet, med `then()` eller `async`/`await`? Varför valde ni det ni gjorde?
- För *Uppgift 7*, visa och förklara:
  - Visa koden som skapar ny HTML. Förklara dess placering.
    - Ligger den i en **callbackfunktion**? Ligger den i en **annan funktion**, ligger den i ”roten” av JavaScript-filen? Varför ligger den där?
  - Beskriv **val av utseende på er webbsida**, hur valde ni att sätta olika stilar på elementen; egna CSS-klasser? Via `style`-egenskapen på `HTMLElement`-objekten direkt i JavaScript? Med hjälp av ett **CSS-ramverk**? Varför?
  - Visa slutresultatet av siten i webbläsaren.