PROJEKTBESKRIVNING

CRUD med en resurs

För 3,5 hp i kursen Dynamiska Webbapplikationer ska två labbar och ett onlineprov utföras. Detta är den första labben. Den utförs i labbgrupper, enligt de som finns på Learn (markerade Projektgrupp 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

Projektbeskrivning

CRUD med en resurs

Uppdaterad

2024-09-11

Innehåll

Förberedelser 3

Kravspecifikation 4

- Tekniska krav 4
- Gränssnitt 4
  - 1. Styling 4
  - 2. Lista 5
  - 3. Formulär 5
  - 4. Knappar/ikoner 5
  - 5. Meddelande 5

Tekniska detaljer 6

- Backend 6
  - 1. API-routes 6
  - 2. Gemensamt FunktionaliteT i Callbackfunktionen hos alla API-routes 6
  - 3. Databaskommunikation 7
  - 4. Databasen 7
- Frontend 9
  - 1. Generellt 9
  - 2. Visa alla (R i CRUD) 9
  - 3. Möjliggöra uppdatering av resurs 10
  - 4. borttagning av en resurs. (D i CRUD) 10
  - 5. Formuläret 11
  - 6. Skicka formulär (C och U i CRUD) 11
  - 7. Meddelanderuta 12

Inlämning 13

- Kod 13
- Video 13

HÖGSKOLAN DALARNA

Projektbeskrivning

CRUD med en resurs

Uppdaterad

2024-09-11

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]-projekt**.
    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.

HÖGSKOLAN DALARNA

Projektbeskrivning

CRUD med en resurs

Uppdaterad

2024-09-11

Kravspecifikation

Ni ska skapa en webbsida där man kan utföra **CRUD-funktionalitet** för en viss ***resurs***. Det kan vara vad som helst – generella produkter som t.ex. hos en webbshop, filmer, bilr eller något annat. Det får dock inte vara något stötande och det får inte vara användare (users). Dessa har vi jobbat tillräckligt med under kursen.

Eftersom jag inte vet vad er webbsida kommer att handla om, kommer jag vidare i dokumentet använda begreppet ***resursen*** för att prata om det som sidan ska handla om.

Webbsidan kommer, när den är färdig, att likna **användarregistrerings-applikationen** såsom den såg ut efter den tillgängliga **inspelade tutorial** i fem delar som finns i kursrummet.

Det kommer dock att finnas detaljer och krav som ni inte ännu har sett under kursens gång och som ni förväntas lösa på egen hand.

Tekniska krav

1. Datalagring ska ske i en SQL-databas, förslagsvis SQLite
    a. *Notera*: Detta kan behöva konfiguration av era datorer och VS Code, ifall ni inte arbetat med Lektion 4 eller Labb 2.
2. Backend ska skrivas i en Node.js-miljö med stöd av Express.
    a. *Notera:* Detta kan behöva installation av Node.js på era datorer, ifall ni inte arbetat med Lektion 4 eller Labb 2.
3. Backend och frontend ska kommunicera med varandra genom ett **API**.
4. Inga externa JavaScript-ramverk (såsom **React** eller **Vue**) får användas i frontend.
5. Endast *en* (*1*) HTML-sida ska använda. All information där ska skapas och ändras dynamiskt utifrån hur data hämtas och förändras.

Gränssnitt

Nedanstående delar **ska** finnas i er webbsidas gränssnitt. Delar kommer att skapas dynamiskt i JavaScript-kod, andra delar ska skrivas direkt i HTML-filen från början. Detaljer rörande dessa element följer under avsnittet

1. STYLING

Styling av webbsidan ska ske med hjälp av ett **CSS-ramverk**. Det kan vara *Bootstrap*, *Tailwind* eller något annat ramverk. Webbsidan måste vara **responsiv**, vilket ska lösas med hjälp koncept såsom *Grid* (länken är till Bootstrap, men konceptet finns i de flesta CSS-ramverk). Egen CSS-styling får användas, exempelvis rörande detaljer såsom färg, men i huvudsak ska ett ramverk användas.

Regler rörande styling ska tillämpas på såväl element i som från början finns i HTML-filen, som **dynamiskt genererade** HTML-element som läggs till via JavaScript.

2. LISTA

Det ska finnas en lista som **visar alla** lagrade poster för er valda ***resurs***, hämtade från ert backend. Listan ska alltid innehålla *aktuell information* från servern. Listan får se ut hur som helst. Respektive ***resurs*** kan presenteras vertikalt eller horisontellt och i vilken ordning ni vill.

3. FORMULÄR

Det ska finnas ett formulär för att möjliggöra **inmatning** av data för *all information* (utom `id`) hos er ***resurs***. Hur många och av vilken typ inmatningsfälten är bestäms av hur ni utformar er databas. Formuläret ska ha som minst ha en knapp för att skicka (`submit`) formuläret, men kan också ha knappar för att exempelvis rensa fält.

4. KNAPPAR/IKONER

Knappar för att möjliggöra **borttagning** och **ändring** av ***resurs*** ska finnas i gränssnittet, förslagsvis intill respektive ***resurs***. Ni väljer själva om detta är **knappar** eller **ikoner** i gränssnittet.

5. MEDDELANDE

När en ny resurs skapats, tagits bort eller uppdaterats ska ett meddelande, exempelvis i form av en *modal ruta* visas som feedback till användaren.

HÖGSKOLAN DALARNA

Projektbeskrivning

CRUD med en resurs

Uppdaterad

2024-09-11

Tekniska detaljer

Nedan följer tekniska detaljer. Dessa detaljer är *minimumkrav*, men ni väljer själva lösning kring hur ni skriver er kod för att uppfylla dessa krav. Ni får också fritt lägga till eller förändra kringliggande funktionalitet och kod, så länge minimumkraven uppfylls.

Backend

1. API-ROUTES

1. Det ska finnas **lämpliga API-routes** för att möjliggöra **alla funktioner** i frontend.
2. API-routes ska utformas och namnges enligt nedan (*RESTFul*-**standard**)
3. De API-routes som ska finnas är:
    - hämta alla (`GET` ***/resurs***)
    - uppdatera (`PUT` ***/resurs***)
    - skapa (`POST` ***/resurs***)
    - ta bort (`DELETE` ***/resurs/:id***).
4. Icke obligatoriskt API-route:
    - hämta en ***resurs*** (`GET` ***/resurs/:id***). Huruvida den behövs eller inte beror på hur ni utformar ert frontend.
5. Respektive route hanteras genom **express** och dess `.get`, `.post`, `.put`, `.delete`-funktioner respektive. Dessa tar alla en **callbackfunktion** där funktionaliteten för era respektive routes ska skrivas.
    - *Exempel:* en route för att hantera `GET` ***/resurs*** kan se ut såhär: `server.get(/resurs, () => { //callbackfunktion för att hantera förfrågan })`.

2. GEMENSAMT FUNKTIONALITET I CALLBACKFUNKTIONEN HOS ALLA API-ROUTES

1. `request`-objekt ska läsas **för att hämta relevant data som skickas från frontend**. Dessa kan finnas i exempelvis `body` eller `params` (data i url:en), beroende på hur förfrågan från frontend ser ut.
2. `response`-objektet **ska användas för att skicka svar från servern**.
3. Om ni vill kan även lämpliga **meddelanden** skickas med `response`-objektet (tillsammans med andra eventuella data) för att ge feedback till användaren.
4. Om det någonstans **uppstod ett fel**, ska status `500` skickas, tillsammans med ett felmeddelande i `response`-objektet.
    - Till detta kan ni exempelvis använda *try-catch*, eller på andra sätt fånga upp fel.

- Node-modulen **`sqlite3`** har också inbyggda funktioner för felhantering som kan nyttjas.

3. DATABASKOMMUNIKATION

1. Koppling till databasen görs i callbackfunktionen till **respektive route**.
2. Hur SQL-frågorna för respektive funktion ska se ut får ni själva ta reda på. Länkar till respektive SQL-fråga finns nedan.
    - Instruktioner rörande SQL specifikt för SQLite i Node.js finns på denna länk: *https://www.sqlitetutorial.net/sqlite-nodejs*.
3. Vid respektive route **ska förfrågan resultera i databas-funktionalitet enligt nedan:**
    - Vid **`POST`**-förfrågan ska **en ny rad** sparas (**`INSERT`**) i databastabellen för er ***resurs*** baserat på innehåll i **`request`**-objektets **`body`**.
    - Vid **`PUT`**-förfrågan ska en **befintlig rad** uppdateras (**`UPDATE`**) i databastabellen för er ***resurs*** baserat på innehåll i **`request`**-objektets **`body`**.
        - För att veta vilken resurs som ska uppdateras ska egenskapen **`id`** i **`request`**-objektets **`body`** användas som **`WHERE`**-villkor i SQL-frågan.
    - Vid **`DELETE`**-förfrågan (med **`id`** som parameter i url:en) ska en **befintlig rad** tas bort (**`DELETE`**) ur databastabellen för er ***resurs***.
        - För att veta vilken ***resurs*** som ska tas bort ska egenskapen **`id`** i **`request`**-objektets egenskap **`params`** användas som **`WHERE`**-villkor i SQL-frågan.
    - Vid **`GET`**-förfrågan (utan **`id`** som parameter i url:en) ska **samtliga rader** hämtas (**`SELECT`**) ur databastabellen för er ***resurs***.
    - Vid **`GET`**-förfrågan (med **`id`** som parameter i url:en) ska **en rad** hämtas upp (**`SELECT`**) ur databastabellen för er ***resurs***.
        - För att veta vilken ***resurs*** som ska hämtas ska egenskapen **`id`** i **`request`**-objektets egenskap **`params`** användas som **`WHERE`**-villkor i SQL-frågan.

4. DATABASEN

1. En **SQL-databas ska användas**, förslagsvis SQLite.
2. Motsvarande **npm-paket** ska användas för att kunna jobba med databasen i Node.js (**`sqlite3`** för att arbeta med en SQLite-databas).
3. Databasen **ska innehålla en (1) tabell**, motsvarande den ***resurs*** du arbetar med.

4. **Minst en kolumn** ska på sikt styra någon **grafisk aspekt** i frontend, såsom storlek, färg eller form på det element som ska representera den enskilda ***resursen***.[1]
5. I övrigt får tabellen får ha *vilka och hur många kolumner ni vill* (inom någon av rimlighet, förstås).
    - Kolumnerna kan döpas på samma sätt som t.ex. `id`- eller `name`-attributen i formulärets fält i frontend. Det är inte helt enligt korrekt namnstandard, men det förenklar mycket om JSON-objektets nycklar motsvarar tabellens kolumner.
6. Ni *bestämmer själva* vilka **datatyper** och andra eventuella restriktioner som finns hos kolumnerna.
    - Tabellen **ska** dock ha en `id`-kolumn, vars värde automatiskt sätts och ökar automatiskt för varje rad som läggs till i tabellen (`AUTOINCREMENT`). Ett `id` ska alltså aldrig skickas med en `INSERT`-query till databasen.

[1] Ett exempel kan vara om resursen är ***bilar*** kan bilens **färg** påverka utseendet i gränssnittet, men det behöver inte vara så konkret heller. Man kan ha att vissa kategorier representeras av ett visst utseende, t.ex. åldersgrupper har olika färger.

HÖGSKOLAN DALARNA

Projektbeskrivning

CRUD med en resurs

Uppdaterad

2024-09-11

Frontend

1. GENERELLT

Samtliga anrop för att **hämta eller förändra data** ska göras via `fetch()`-anrop till ett **backend som ni själva skapat**, via ett **API** som ni satt upp för att möjliggöra kommunikation till ert backend (se *Backend - API-routes*).

Innehållet på sidan ska alltid vara aktuellt och uppdateras (**utan sidomladdning** *[2]*) vid förändring av data.

2. VISA ALLA (R I CRUD)

1. En lista av **något slag** ska presentera **samtliga rader** av ***resursen*** som finns lagrade i databasen.
    - Listan kan vara av typen `ul` med tillhörande `li`-element för varje enskild ***resurs***, men andra lämpliga HTML-element får också användas. *[3]*
2. Listan får **inte finnas i HTML-koden från början**.
3. Listans innehåll ska hämtas (`GET` ***/resurs***) från ert backend via ert API.
4. När svar kommit från ert backend ska **HTML skapas och läggas till i DOM-trädet** för att **synas på webbsidan**.
    - Först nu kommer alltså listan att läggas till i DOM-trädet, eftersom den inte ska finnas där från början.
5. **Valfritt antal egenskaper** hos ***resursen*** ska synas i respektive listelement
    - Något eller några egenskaper ska dock påverka någon **designaspekt** hos listelementet.
    - Egenskapen `id` behövs också någonstans, men det ska **inte synas** i listan. Ett alternativ är att id-/klass-/*data*-attribut i en HTML-tagg för att lagra `id`:t på något sätt.
6. Listan ska **alltid ha aktuell data**, så den behöver uppdateras vid förändring.
7. Knappar/ikoner för att **ta bort** och **ändra *resurs*** kan med fördel också presenteras i respektive listelement intill övrig information om ***resursen***. De **behöver finnas någonstans**, och denna plats vore lämplig.

---

[2] Förtydligande: Omladdning av sidan kan ske på olika sätt exempelvis klick på **refresh**-knappen i webbläsaren, användning av `window.location.reload`, eller att Live Server uppdaterar sidan (en inställning hos Live Server som ni behöver vara uppmärksamma på). En ny förfrågan (`fetch()`) mot ert backend behöver göras *efter att något har förändrats i databasen* för att detta krav ska vara uppfyllt.

[3] Jag kommer att dock vidare använda begreppen **lista** och **listelement** för att prata om denna samling och dess enskilda element.

3. MÖJLIGGÖRA UPPDATERING AV RESURS

1. **Eventlyssnare** ska finnas som lyssnar efter **klick på knapp/ikon** för att **ändra *resurs*** (se *Gränssnitt - Knappar/ikoner*).
2. Vid klick på knappen ska **befintliga uppgifter** om vald ***resurs*** **presenteras i formulärets fält**.
    - *Notera:* Klick på denna knapp ska **inte skicka** en faktisk förfrågan till servern om att **uppdatera** uppgifter, utan endast **fylla i formulärets fält** med befintlig data så att man kan ändra dem där och sedan skicka data **via formuläret**.
    - *Notera:* Om inte alla egenskaper finns i listan redan behöver en **ny förfrågan** till API:et göras så att **samtliga fält i formuläret** kan fyllas på med data.
        - (`GET /resurs/:id` är en lösning. Det måste då förstås finnas en sådan route i ert API.
        - Formulärets fält kan då fyllas på **först när svar från backend** mottagits.
3. Någonstans behöver `id` sparas, för den resurs vars data **för närvarande finns i fälten**. Det behövs i *Punkt 6 - Skicka formulär (C och U i CRUD)*.
    - Exempel på lösning kan vara användning av `localStorage` eller att lagra `id` någonstans (**osynligt**) i anslutning till formuläret.
    - Egenskapen `id` ska *inte* synas någonstans i gränssnittet.
4. Någonstans måste alltså ***resursens*** `id` hämtas/finnas tillgängligt i eventlyssnaren.

4. BORTTAGNING AV EN RESURS. (D I CRUD)

1. **Eventlyssnare** ska finnas som lyssnar efter **klick på knapp/ikon** för att **ta bort *resurs*** (se *Gränssnitt - Knappar/ikoner*).
2. Vid klick på knappen ska **klickeventet fångas** upp och göra en förfrågan med metoden `DELETE` till ert API.
3. Någonstans måste alltså ***resursens*** `id` hämtas/finnas tillgängligt i eventlyssnaren, så att man kan veta vilket `id` som ska skickas med `DELETE`-förfrågan till API:et.
4. När svar från servern har mottagits ska ett meddelande visas, se *Punkt 7 - Meddelanderuta*.
    - Svaret får gärna komma från servern, dvs. finnas på serverns `response`-objekt.
5. När svar från servern har mottagits ska listan uppdateras dynamiskt, **utan** sidomladdning.

5. FORMULÄRET

1. Ett formulär med **fält som representerar de fält som er *resurs* har i databasen** ska finnas på webbsidan.
    - Alla fält utom `id` ska finnas representerade.
    - Antalet fält styrs alltså av hur ni utformat er databas.
2. Använd **lämpliga typer** av HTML-inputelement beroende på **datatyp i databasen**. Om något lagras som en `INTEGER` i databasen är det alltså vettigt att använda ett `input`-fält av typen `number` i formuläret.
3. Ni får validera fältets data genom HTML-validering eller JavaScriptfunktioner, men ni behöver inte.
4. Formuläret måste som minst ha en **knapp för att skicka** in (typen `submit`), men kan även ha knappar för att exempelvis rensa fält, om man vill.

6. SKICKA FORMULÄR (C OCH U I CRUD)

1. Endast **en (1)** funktion (**eventlyssnare**) ska finnas för att hantera formulärets `submit`-event.
    - Detta för att det är till väldigt stor **del samma funktionalitet** som behövs för att **skapa ny** respektive **uppdatera** en ***resurs*** och vi vill undvika duplicering av kod.
2. Ni behöver kunna kontrollera om det är en **ny** eller **befintlig *resurs*** som finns i formulärets fält och därigenom avgöra om det ska vara metoden **POST** (**ny resurs**) eller **PUT** (förändring av **befintlig resurs**) i förfrågan till API:et.
3. Detta ska ha hanterats i *Punkt 3 - Möjliggöra uppdatering av resurs*.
    - I övrigt skiljer sig inte funktionaliteten mellan att **skapa** ny eller **uppdatera** en ***resurs***.
4. Innehållet i formuläret (samtliga uppgifter hos ***resursen***) ska skickas i **JSON-format** via `request`-objektets `body`.
5. När svar från servern har mottagits ska ett meddelande visas, se *Punkt 7 - Meddelanderuta*.
    - Svaret får gärna komma från servern, dvs. finnas på serverns `response`-objekt.
6. När svar från servern har mottagits ska listan uppdateras dynamiskt, **utan** sidomladdning.

7. MEDDELANDERUTA

1. Det ska finnas en ruta som ska **visa feedback till användaren** när något händer med ***resursen***.
    a. Den kan med fördel vara en Modal från Bootstrap, eller liknande.
2. Rutan ska visas när en ***resurs*** har **skapats** eller **uppdaterats** eller **tagits bort** och svar från servern mottagits.
3. Texten/meddelandet kan komma från servern via ett `response`-objekt, men kan också vara något annat som passar den förändring som skett.

HÖGSKOLAN DALARNA

Projektbeskrivning

CRUD med en resurs

Uppdaterad

2024-09-11

Inlämning

Senast 17:00 fredagen den 12 januari ska nedanstående lämnas in 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**]-projekt.
- *Observera!* Lämna **inte** in en zip:ad mapp med er kod!

Video

Alla gruppdeltagare ska vara *aktiva och delaktiga i redovisningen*. Ni ska alla ha kameror på och namn synliga så att jag kan se vem som pratar (kan exempelvis uppnås genom att använda Zoom).

Gör en inspelning på *max 15 minuter* där ni

1. Visar webbsidan
    a. Visa hur ni uppfyller kraven rörande *Gränssnitt*.
    b. Visa att ni kan utföra samtliga CRUD-operationer (visa, lägga till, ändra och ta bort) på er webbsida.
    c. Kort beskriver val av CSS-ramverk och motiverar ert val.
2. Demonstrerar er kod genom att beskriva hur ni gjort för att implementera *Tekniska detaljer*.
3. Gruppens erfarenheter av arbetet
    a. Svårigheter & utmaningar
    b. Reflektion av samarbetet i gruppen
    c. Om ni vill: Kod eller lösningar som ni tyckte blev extra bra och vill framhäva