HÖGSKOLAN DALARNA

Projektbeskrivning

Webbshop

Uppdaterad

2025-01-14

PROJEKTBESKRIVNING

Webbshop

För 3hp i kursen Design av Webbapplikationer (GIK2XK) krävs ett grupparbete i form av ett utfört projekt.

Gruppuppsättning finns i kursrummet på Learn, i grupper markerade ”Projektgrupp xx”, där xx är ert gruppnummer.

För godkänt resultat krävs en lösning enligt nedanstående specifikation för minimumkrav, samt redovisning med individuellt deltagande från samtliga projektmedlemmar.

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

Webbshop

Uppdaterad

2025-01-14

Innehåll

Förberedelser 3

Kravspecifikation 4

- Tekniska krav 4
- Funktionskrav (användarfall) – minimum 4
  - 1. Som kund vill jag kunna: 4
  - 2. Som försäljare/administratör av webbshopen vill jag kunna: 5

Tekniska detaljer 6

- Backend & databasarkitektur 6
- Databas UML-diagram 7
- API-routes (backend) 8
  - 1. Produkter (`productRoutes`) 8
  - 2. Användare (`userRoutes`) 8
  - 3. Varukorg 8
- Services (backend) 8
  - 1. Lägga produkt i varukorg 8
  - 2. Hämta varukorg 9
- Frontend-struktur 9
  - 1. Förslag på vyer 9
  - 2. Övrigt 10

Ytterligare funktionalitet (ej krav) 11

- Databas, variant 11

Redovisning 12

- Inlämning av kod 12
- Loggar 12
- Presentationsseminarium 12

HÖGSKOLAN DALARNA

Projektbeskrivning

Webbshop

Uppdaterad

2025-01-14

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 **gik2xk-grupp[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 av kod*.
6. Ert repository *ska innehålla* en fil vid namn `.gitignore` innehållande texten `node_modules/` för att *exkludera* mappen **node_modules** från ert repository.

HÖGSKOLAN DALARNA

Projektbeskrivning

Webbshop

Uppdaterad

2025-01-14

Kravspecifikation

Ni ska skapa en webbshop. Produkterna i butiken kan vara vad som helst – blandat, filmer, bilar, spel, kläder eller i stort sett vad som helst. Det får dock inte vara något stötande eller opassande.

Väldigt mycket av funktionaliteten i detta projekt kommer ni att kunna finna i bloggen, som byggs under lektionerna. Koncepten är liknande, men ni behöver ”översätta” bloggens lösningar till en webbshop.

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. Det är mycket det som projektet går ut på – att tillämpa idéer från någon annan stans, men också i grupp lösa de problem som uppstår med de saker som inte löses genom något ni sett tidigare.

Tekniska krav

- Datalagring ska ske i en **SQL-databas** av valfri typ (MySQL, MariaDB, SQLite eller liknande).
- Backend ska skrivas i en **Node.js**-miljö med stöd av **Express**.
- **Sequelize** ska tillämpas som ORM-ramverk.
    - Det finns ett alternativt ORM-ramverk som heter *Prisma*, som är godkänt för den som vill ta sig an det, det är dock inget som kommer att gås igenom i kursen.
- Arkitektur för långsiktig och hållbar API-kommunikation ska tillämpas.
- Frontend ska skrivas i **React** med tydlig och logisk komponentstruktur.
- Webbsidan ska ha ett tilltalande, konsekvent och användbart gränssnitt.
- Färdiga **komponent-**/**CSS-bibliotek** från exempelvis Material UI eller Bootstrap bör användas.
    - Att göra helt egen styling med CSS eller andra tekniker för att styla React-applikationer kan vara svårt och bidra till att webbsidan inte blir tillräckligt väl utformad rent grafiskt, men anta gärna utmaningen.

Funktionskrav (användarfall) – minimum

Nedanstående delar **ska** kunna utföras. Detaljer kring hur denna funktionalitet tillämpas är upp till er, men det ska gå att göra allt nedanstående via ett gränssnitt i er webbshop.

1. SOM KUND VILL JAG KUNNA:

- se en **vy av flera produkter**.
- se en **detaljvy** för en given produkt med mer information och detaljer, inklusive
    - bilder och information om produkten
    - en lista av **betyg** på produkten
    - **snittbetyg** uträknat ifrån alla betyg
    - gränssnitt för att **betygsätta** en produkt.

- lägga valfritt antal av önskade produkter i en **varukorg**.
- **se min varukorg** med en lista över tillagda produkter, deras pris, antal och sammanlagda pris.

2. SOM FÖRSÄLJARE/ADMINISTRATÖR AV WEBBSHOPEN VILL JAG KUNNA:

- Skapa, ändra och ta bort produkter.

Notera: både besökare och administratör i webbshopen är en och samma modell/resurs – `user`. De särskiljs inte på något sätt förutom någon eventuell benämning eller text i gränssnittet. Det behövs alltså exempelvis ingen funktionalitet för att kontrollera någon slags användarroll och anpassa funktionalitet efter det. Ni får låtsas att vissa funktioner bara är synliga för vissa användare, även fast de kanske syns hela tiden.

Notera: För de delar som man inte kan administrera via frontend (t.ex. användare) används exempelvis Postman för att hantera data.

HÖGSKOLAN DALARNA

Projektbeskrivning

Webbshop

Uppdaterad

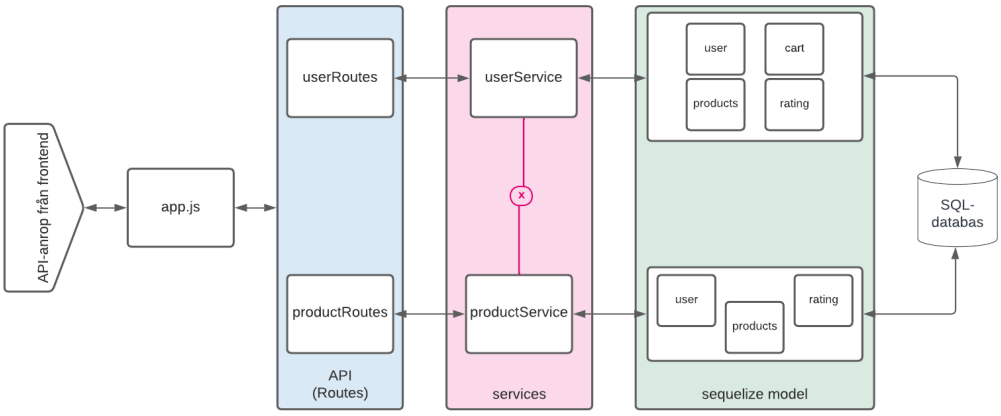
2025-01-14

Tekniska detaljer

Lösning på en stor del av de tekniska detaljerna kan ni se i lektionsvideorna. Nedan följer tips rörande hur ni *skulle* kunna lägga upp databas, arkitektur, routes och vyer. Ni är fria att följa eller ignorera dessa tips, så länge in uppnår minimumkrav.

Backend & databasarkitektur

Nedanstående är ett exempel på backendarkitektur. Det finns ett perspektiv till där det skulle kunna vara vettigt att även ha `cart`-routes och -servicar. Detaljerna är upp till er.



HÖGSKOLAN DALARNA

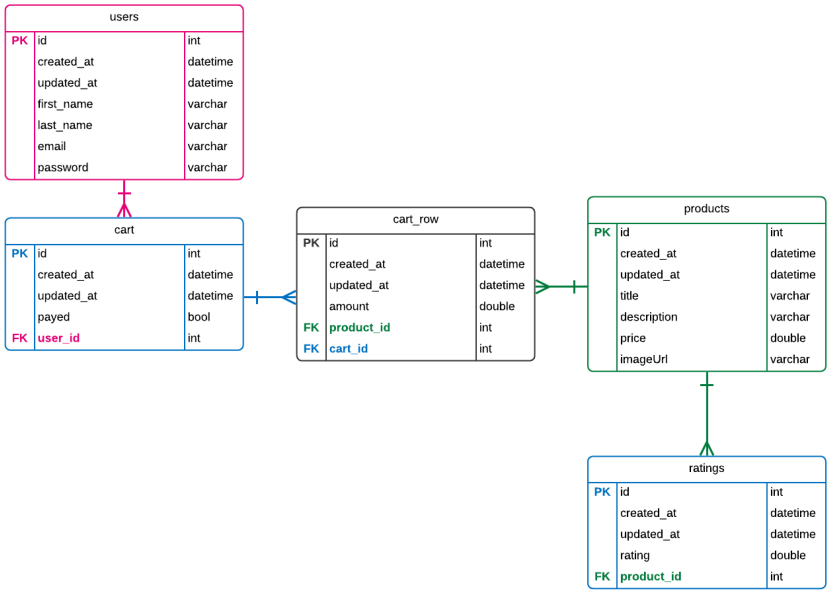
Projektbeskrivning

Webbshop

Uppdaterad

2025-01-14

Databas UML-diagram



Resonemang: En användare kan ha en varukorg. Varukorgen har information om ett enskilt köp och huruvida köpet är genomfört/betalt eller inte.

Varukorgen kan innehålla flera produkter. En produkt kan också finnas i en eller flera varukorgar – alltså ett många-till-många-förhållande mellan `cart` och `product`, vilket gör att `cart_row` kan behövas för att koppla samman varukorgar med produkter (motsvarande `post_tags` i bloggexemplet).

Produkterna i sig har information om sig själva och kan få betyg av användare.

HÖGSKOLAN DALARNA

Projektbeskrivning

Webbshop

Uppdaterad

2025-01-14

API-routes (backend)

Nedan följer förslag på de API-routes som kan exponeras till ert backend. Rörande

1. PRODUKTER (`productRoutes`)

- `get("/")`
  - Hämta alla produkter
- `get("/:id/")`
  - Hämta en specifik produkt.
  - Inkludera produktens samtliga betyg
- `post("/:id/addRating")`
  - Ge betyg till en produkt
- `post`, `put` och `delete`
  - Resterande CRUD för produkt

2. ANVÄNDARE (`userRoutes`)

- `get("/:id/getCart/")`
  - Använd för att hämta alla produkter som en användare lagt i sin senaste varukorg.
- `get`, `post`, `put`, `delete`
  - För att kunna hantera testdata, förslagsvis genom Postman.

3. VARUKORG

Varukorgen kan hanteras lite olika beroende på perspektiv. Man kan tänka sig en lösning där den är relaterad till `products` (`productRoutes`) eller att man gör en variant där det finns `cart`-routes och -services

- Cart: `post(/cart/addProduct)`
  - Innehåll i `body`: `userId`, `productId`, `amount`
  - Alternativ till att hantera att lägga till i varukorgen.
- Product: `post("/:id/addToCart)`
  - Innehåll i `body`: `userId`, `amount`.

Services (backend)

Detaljer rörande funktionalitet i backend som är utöver vanlig CRUD-funktionalitet:

1. LÄGGA PRODUKT I VARUKORG

Denna funktionalitet kan finnas i `productService` eller `cartService`, beroende på val av arkitektur hos era routes. Om ni exempelvis använder routen `/cart/addProduct` är det rimligt att använda `cartService`.

Exempel på steg för att lägga till en produkt i varukorgen.

- Hämta id (`cartId`) för senaste varukorg för given `user` om den finns.
    - Id för `user` ska komma tillbackend via exempelvis förfrågans `body`.
    - Sequelize-funktionen `findOrCreate` passar bra här.
- Sparar hämtar `cartId`, `productId`, `amount` i kopplingstabell `cart_row`.
    - Om kopplingen mellan en given produkt och en varukorg redan finns i kopplingstabellen, kan man istället endast uppdatera antalet. Funktionen `upsert` hos Sequelize kan fungera, men det är inte säkert att det stöds av alla SQL-varianter.

2. HÄMTA VARUKORG

Funktionalitet för att hämta upp en varukorg för en given användare kan placeras i `cartService` eller `userService`.

Förslagsvis hämtas senaste lagrade varukorgen för en användare.

Se till att nödvändig information om de produkter som finns i varukorgen kommer med. Dessa kan ”städas upp” likt upphämtning av inlägg (funktionalitet likt `_cleanPost` i lektionsvideorna).

Lämplig information att skicka tillbaka till användaren skulle vara något i stil med en array av objekt som består av produktens namn, pris och antal.

Frontend-struktur

Jag kommer inte ge er tips i form av t.ex. skisser och mockups för att jag vill att ni ska tänka helt själva rörande layout, utseende, val av MUI-komponenter, samt egen styling.

1. FÖRSLAG PÅ VYER

- Vy för att titta på en lista av alla produkter (`/`)
    - En mer kompakt variant av varje produkt visas.
    - Rating för produkt kan presenteras med hjälp av *Material UI:s "Rating"-komponent*.
- Vy för att titta se mer detaljer om produkt (`/products/:id`)
    - Samtliga detaljer om produkten visas.
    - Man ska kunna ge rating på produkten.
    - Rating för produkt kan presenteras och sättas med hjälp av *Material UI:s "Rating"-komponent*.
- Vy för att se sin varukorg med en lista av produkter och summa av deras pris
    - Detta kan vara en egen vy (`/cart`) eller någon typ av popup-ruta, exemelvis *Material UI:s ”Dialog”-komponent*.
- Vy för att lägga till och ändra produkter. (`/products/:id/edit`, `/products/new`)

2. ÖVRIGT

Navigering ska ske på ett logiskt och användarvänligt sätt, vissa delar via menyval och vissa delar via knappar och länkar på de olika sidorna. Sidan ska ha ett konsekvent och enhetligt utseende vad gäller färg, form och typsnitt.

Det ska också gå att skicka en `DELETE`-förfrågan till backend för en given produkt, även fast detta inte nödvändigtvis representeras av en vy. Ni bestämmer själva hur ni utformar dessa funktioner, men det ska gå att utföra samtliga CRUD-operationer för en produkt på ett eller annat sätt via gränssnittet i frontend.

Notera: För de delar som man inte kan administrera via frontend (t.ex. användare) används exempelvis Postman för att hantera data.

Ytterligare funktionalitet (ej krav)

Nedan följer tips på hur man kan utöka applikationen om man önskar en större utmaning.

- CRUD för användare i frontend.
- Mer detaljerad användarhantering och inloggning, vilket exempelvis skulle kunna innebära att:
    - Man måste vara inloggad för att skapa, ändra och ta bort produkter.
    - Man endast kan se sin egen varukorg.
    - För att hantera inloggning och dela information om exempelvis en användare i många olika komponenter kan det vara värt att kolla in *Reacts Context*.
- Utökat användargränssnitt med exempelvis:
    - Tema från Material UI för mer unik styling än standardinställningarna
    - Egen CSS och styling av siten
    - Paginering och sortering på produkter
    - Frontendvalidering på input-fält
    - Visa meddelanden från frontend/backendvalidering i *exempelvis MUI:s "Alert"-komponent*

Databas, variant

En lite utökad variant på ovanstående diagram kan vara att man inkluderar ett eller flera av dessa element:

1. Lagrar vem som gav betyget (`userId` i `rating`-tabellen)
2. Kan tillåta flera bilder till en produkt (ny tabell för att tillåta koppling mellan produkt och flera bilder).

HÖGSKOLAN DALARNA

Projektbeskrivning

Webbshop

Uppdaterad

2025-01-14

Redovisning

Senast **23:59 den 30 mars** ska nedanstående lämnas in i kursrummet.

Inlämning av kod

- Kommentera er kod så att det framgår vad ni har gjort och hur de olika tekniska kraven implementerats.
- 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**]/gik2xk-[**gruppnummer**]-projekt.
    - *Observera!* Lämna **inte** in en zip:ad mapp med er kod!

Loggar

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

- loggar från **ChatGPT** eller andra chatverktyg, eller
- kommandon som ställts till **GitHub CoPilot** och vilket svar som gavs.

Presentationsseminarium

Presentationsseminariet **sker enligt schema** och är obligatoriskt.

Alla gruppdeltagare ska vara *aktiva och delaktiga vid såväl egen redovisning som vid andras*.

Håll en kortfattad presentation (*max 5 minuter*) där ni

1. Visar er webbsida och demonstrerar att den uppnår *Funktionskraven* (minimum).
2. Berättar om gruppens erfarenheter av arbetet
    a. Svårigheter & utmaningar
    b. Vad har varit mest lärorikt?
    c. Reflektion av samarbetet i gruppen
    d. Om ni vill: Kod eller lösningar som ni tyckte blev extra bra och vill framhäva.

Därefter kommer det finnas tid (*max 5 minuter*) till förfogande för att lärare och andra studenter ska kunna ge synpunkter och feedback på ert projekt.

Det betyder också att ni förväntas att vara aktiva, intresserade och komma med frågor till era medstudenter vid deras redovisningar.