Min testprocess

Det var en väldigt givande uppgift som jag till en början hade svårt för att komma igång med. Måndagen och tisdagen då vi fick uppgiften var jag mycket osäker på vad jag skulle göra och hur. Det kändes som jag missat något grundläggande om hur man ”kopplar ihop” Visual Studio Code och postman. Därför förlorade jag mycket tid åt att kolla igenom lektioner de dagarna för att hitta bitarna som saknades. Men när jag väl kom igång på onsdagen gick det fort. Jag tänker inte förneka att vi samarbetade en del i Ronneby som grupp men alla gör såklart inte exakt likadant och vi alla lärde oss mycket av vårt samarbete.

Uppgifterna var klara på mindre än en vecka för att kunna lägga mer tid åt detta dokumentet och README.md filen.   
  
Det som var svårt med den här uppgiften var att flera av punkterna var svårlästa och otydliga med vad man ville ha exakt och många var även svåra att säga vad som är manuellt och automatiskt.

Jag är nöjd med det jag lämnar in. Kunde det varit bättre? Säkerligen, men mina kunskaper sträcker sig inte längre än det jag visat här. Uppgiften var väldigt bra och jag vet att jag kommer dra nytta av den i framtiden då min handledare på min LIA plats till våren-25 blev helt till sig när de hörde att jag hade ett projekt i MongoDB.

Med det sagt hälsar jag trevlig läsning!

Manuella

**Test 1:** Kontrollera att API:et returnerar rätt HTTP-statuskod (t.ex. 200 OK) för en lyckad GET-begäran.

**Motivering:** Syftet med testet är att manuellt överskåda att förfrågan får statuskod 200.

**Testprocess:** Jag skapade en **get-request** för att hämta alla authors och en **get-reguest** för att hämta alla books.

**Förväntat resultat:** Jag förväntar mig en statuskod 200.

**Resultat:** Jag fick en statuskod 200.

**Test 2:** Kontrollera att API:et returnerar det förväntade dataformatet (t.ex. JSON, XML) i svaret.

**Motivering:** Syftet med testet är att kontrollera manuellt att rätt dataformat visas i headers.

**Testprocess:** Jag skapade en **get-request** för att hämta alla authors och en **get-reguest** för att hämta alla books.

**Förväntat resultat:** Jag förväntar mig att under headers kunna se att Content-Type är application/json; charset=utf-8.

**Resultat:** Jag kan under headers se Content-Type är application/json; charset=utf-8 och därför i rätt dataformat.

**Test 3:** Se till att API:et returnerar rätt HTTP-statuskod (t.ex. 400 Felaktig begäran) för en ogiltig begäran.

**Motivering:** Syftet med testet är att manuellt kontrollera vad som händer om man fyller i exempelvis fel id och säkerställa att rätt statuskod syns.

**Testprocess:** Jag skapade en get-request för att hämta en author med id abc123, som inte finns.

**Förväntat resultat:** Jag förväntar mig ett felmeddelande som säger ”hittades inte” vilket bör innebära statuskod 400.

**Resultat:** Jag får statuskod 400.

**Test 4:** Testa om API:et returnerar rätt data när du frågar med specifika filter eller sökvillkor.

**Motivering:** Syftet med testet är att manuellt se att API:et kan visa enbart en del av ett objekt.

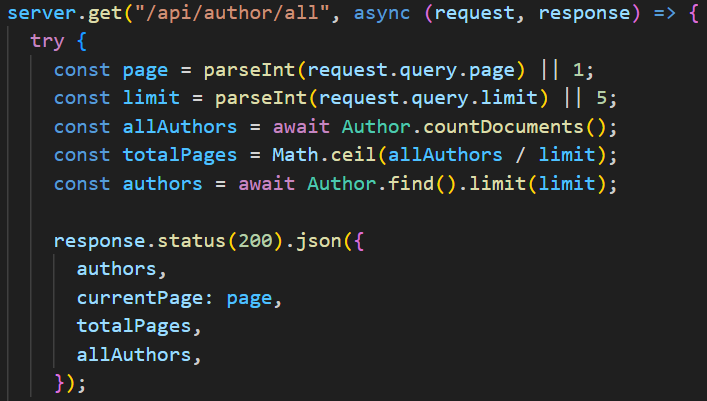
**Testprocess:** Jag börjar med en **post-request** där jag skapar en author och en **post-request** där jag skapar en book. Därefter skapade jag **4 get-requests** där jag söker specifikt efter bokens plot, genre, year och title. De 4 **get-requesten** har följande URL för att hitta de olika delarna [http://localhost:3000/api/book/{{bookId}}?fields=](http://localhost:3000/api/book/%7b%7bbookId%7d%7d?fields=) plot/year/genre eller title som endpoint.

**Förväntat resultat:** Jag förväntar mig att se de olika delarna av objektet.

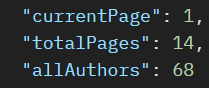
**Resultat:** Jag såg de olika delarna av objektet.

**Test 5:** Kontrollera att API:et returnerar sidnumrerade resultat när ett stort antal poster begärs.

**Motivering:** Syftet med testet är att kontrollera att resultatet enbart visar ett begränsat antal poster per sida för att inte visa alla i en ofantligt lång lista.

**Testprocess:** Jag valde att skapa en **get-request** för att hämta alla böcker och en **get-request** för att hämta alla författare. Jag använder följande URL för att få fram ett paginerat resultat: http://localhost:3000/api/book/all?page=1?limit=5?  
I js sätter jag nuvarande page till 1 och limiterar resultat per sida till 5. Sedan räknar jag ut antalet poster av authors och delar på 5 för att fram hur många sidor som finns totalt. Samma logik på get

**Förväntat resultat:** Jag förväntar mig att 5 poster visas och att det presenteras hus många poster som finns totalt och hur många sidor som finns.

**Resultat:** I responsen kan man längst ner se aktuell sida, antal sidor och antal authors sammanlagt. Jag får samma resultat på books

**Test 6:** Kontrollera om API:et hanterar specialtecken och icke-engelsk text korrekt i indata och returnerade svar.

**Motivering:** Syftet med testet är att manuellt kontrollera API:et kan visa special tecken och blir ett läsbart resultat.

**Testprocess:** Jag skapade en post-request för att skapa en author och en post-request för att skapa en book. Därefter skapade jag en **get-request** för boken jag nyss lagt in vars plot har å, ä och ö. Jag använde då fields som jag även använde i ett tidigare test.

**Förväntat resultat:** Jag förväntade mig att se å, ä och ö.

**Resultat:** Jag såg å, ä och ö.

**Test 7:** Testa API:ets svar när du skickar samtidiga begäranden för att säkerställa att det kan hantera flera användare och upprätthålla datakonsekvens.

**Motivering:** Syftet med testet är att manuellt kontrollera att API:et inte kraschar eller ger felaktiga svar när många förfrågningar sänds samtidigt.

**Testprocess:** Jag skapade en **get-request** för att hämta alla böcker och skrev sedan under scripts en for loop för att föra en **sendRequest** för att hämta samma **get-request** 5 gånger till.

**Förväntat resultat:** Jag förväntar mig att databasen kan hantera alla requests utan problem.

**Resultat:** Databasen klarar 6 requests utan problem.

**Test 8:** Testa om API:et hanterar olika HTTP-metoder korrekt (GET, POST, PUT, DELETE) för varje slutpunkt och returnerar lämpliga statuskoder och svar för varje metod.

**Motivering:** Syftet med testet är att manuellt kontrollera att de olika metoderna fungerar och når sina endpoints.

**Testprocess:** Jag skapade en post-request, get-request, put-request och delete-request där jag först skapade en author, hämtade jag samma author, ändrade namn och slutligen tog bort author. Under flödets gång ska jag se de korrekta statuskoderna.

**Förväntat resultat:** Jag bör se rätt statuskoder

**Resultat:** Jag såg alla statuskoder.

**Test 9:** Kontrollera om API:et hanterar uppdateringar av befintliga poster korrekt och se till att ändringarna sparas och återspeglas i efterföljande begäranden.

**Motivering:** Syftet med testet är att manuellt kontrollera att ett objekt ändras efter en **put-request.**

**Testprocess:** Jag skapade en book med en **post-request**, sedan skapade jag en **put-request** där jag ändrade på objektet. Sedan skapade jag en **get-request** där jag hämtar ner objektet igen.

**Förväntat resultat:** Eftersom testet är manuellt förväntar jag mig kunna se ändringarna i **get-requesten**.

**Resultat:** Jag kunde se ändringarna i responsen.

**Test 10:** Testa API:ets prestanda under hög belastning och simulera ett stort antal användare som gör förfrågningar samtidigt.

**Motivering:** Syftet med testet är att kontrollera att API kan hantera ett stort antal förfrågningar samtidigt.

**Testprocess:** Jag skapade en **get-request** för att hämta alla books och i scripts gjorde jag en for loop med en **sendRequest** för att simulera 30 **get-request** för all books.

**Förväntat resultat:** Jag förväntar mig att databasen kan hantera 30 **get-requests.**

**Resultat:** Databasen kunde hantera 30 **get-requests.**

**Test 11:** Kontrollera att API:et kan återställas på ett smidigt sätt från fel, till exempel problem med databasanslutningen utan att kompromissa med dataintegriteten.

**Motivering:** Syftet med testet är att manuellt kontrollera att API:et kan hantera att anslutningen till databasen kopplas ifrån utan att förlora data.

**Testprocess:** Jag testade denna uppgift genom att koppla bort anslutningen i mongoDB Compass och återansluta.

**Förväntat resultat:** All data bör finnas kvar.

**Resultat:** Resultatet av testet var all data fanns kvar.

**Test 12:** Testa API:ets förmåga att hantera edge cases, till exempel förfrågningar med saknade eller ogiltiga parametrar, och se till att lämpliga felmeddelanden returneras.

**Motivering:** Syftet med testet är att manuellt kontrollera att rätt felmeddelande visas när man använder parametrar som inte är giltiga.

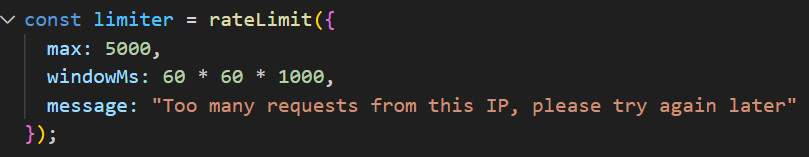
**Testprocess:** Jag återanvände mitt tidigare test där jag i en **get-request** söker efter en author med id abc123.

**Förväntat resultat:** Då det inte finns någon author med det id:et bör jag få ett felmeddelande.

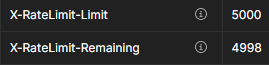
**Resultat:** Jag fick statuskod 400.

**Test 13:** Kontrollera att API:et implementerar hastighetsbegränsnings- eller begränsningsmekanismer korrekt för att förhindra missbruk eller överdriven användning av resurser.

**Motivering:** Syftet är att kunna kontrollera att API:et kan förhindra överbelastning med hjälp av en rate-limiter.

**Testprocess:** Efter att jag lagt till logiken för rate-limiter i js kunde jag hitta 3 nya headers i response.

**Förväntat resultat:** Efter varje sänt request bör jag därför se rate-limitern trappa ner.

**Resultat:** Jag kunde se i response headers att rate-limitern sjunker.

Automatiska

**Test 1:** Kontrollera att API:et returnerar rätt HTTP-statuskod (t.ex. 200 OK) för en lyckad GET-begäran.

**Motivering:** Syftet med testet är att automatiskt kontrollera att förfrågan får statuskod 200.

**Testprocess:** Jag skapade en **get-request** för att hämta alla authors och en **get-reguest** för att hämta alla books. Under scripts/test skrev jag test för automatiskt kontrollera statuskoden med pm.responde.to.have.status(200);

**Förväntat resultat:** Jag förväntar mig en statuskod 200.

**Resultat:** Jag fick en statuskod 200.

**Test 2:** Kontrollera att API:et returnerar det förväntade dataformatet (t.ex. JSON, XML) i svaret.

**Motivering:** Syftet med testet är att kontrollera manuellt att rätt dataformat visas i headers.

**Testprocess:** Jag skapade en **get-request** för att hämta alla authors och en **get-reguest** för att hämta alla books. Under scripts/tests skrev jag ett test för att automatiskt kontrollera att headers har rätt dataformat med koden pm.response.to.have.header(’responseheader’,’dataformat’).

**Förväntat resultat:** Jag förväntar mig att responsen är i rätt dataformat och testet går genom.

**Resultat:** Testet går igenom som förväntat och responsen är i rätt dataformat.

**Test 3:** Se till att API:et returnerar rätt HTTP-statuskod (t.ex. 400 Bad Request) för ogiltiga begäranden.

**Motivering:** Syftet med testet är att kontrollera att rätt statuskod visas vid felanvändning.

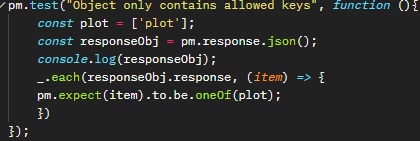
**Testprocess:** Jag skapade ett get-request där jag söker på en author med id abc123.I scripts/tests skrev jag ett test för att kontrollera att statuskoden är 400 med pm.response.to.have.status(400);

**Förväntat resultat:** Jag förväntar mig ett 400 Bad request meddelande och att testet går igenom.

**Resultat:** Testet gick som förväntat och statuskod var 400 Bad request**.**

**Test 4:** Skapa ett automatiserat test som skickar en begäran med specifika filter eller sökkriterier och kontrollerar om API:et returnerar rätt data.

**Motivering:** Syftet med testet är att kontrollera att kunna filtrera informationen som visas genom att enbart hämta ut en del av ett objekt.

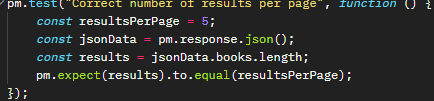
**Testprocess:** Jag skapade först en post-request för att skapa en author, en post-request för att skapa en book samt 4 get requests för att hämta olika delar boken jag precis skapade med fields. Jag använder då url:[http://localhost:3000/api/book/{{bookId}}?fields=](http://localhost:3000/api/book/%7b%7bbookId%7d%7d?fields=) plot/year/genre eller titel. Under tests/scripts kontrollerar jag att enbart rätt ”keys” syns.

**Förväntat resultat:** Jag förväntar mig att testet går igenom och att enbart plot/genre/year eller title delen av objektet syns.

**Resultat:** Som förväntat syns enbart plot/year/genre eller title och testet går igenom.

**Test 5:** Skriv ett automatiserat test för att kontrollera att API:et returnerar sidnumrerade resultat när ett stort antal poster begärs.

**Motivering:** Syftet med testet är att automatiskt kontrollera att API:et visar resultatet i ett begränsat antal poster.

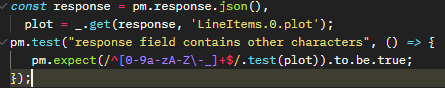
**Testprocess:** Jag skapade två get-requests, en för all books och en för all authors eftersom de båda endpoints resulterar i många resultat. I scripts/tests skapade jag ett test för att kontrollera att rätt antal poster per sida visas.

**Förväntat resultat:** Jag förväntar mig se 5 poster i response.

**Resultat:** Jag får 5 poster i response och testet går igenom.

**Test 6:** Testa om API:et hanterar specialtecken och icke-engelsk text korrekt i indata och returnerade svar med hjälp av ett automatiserat testverktyg.

**Motivering:** Syftet med testet är att automatiskt kontrollera att special tecken syns och är läsliga.

**Testprocess:** Jag skapade en post-request där jag skapar en book med mängder av å, ä och ö började jag med i sin plot, använder jag sedan en get-request för att hämta specifikt plot med hjälp av fields. Under scripts/testsanvänder jag testet pm.expect (/^[0-9a-zA-Z\-\_]+$/.test(plot)).to.be.true; för att säkerställa att response kan läsa de olika tecknen utan att förvränga texten.  **Förväntat resultat:** Jag förväntar mig att testet går igenom och att specialtecken är läsliga.

**Resultat:** Testet är godkänt och Å, Ä och Ö är läsliga.

**Test 7:** Utveckla ett automatiserat test som skickar samtidiga förfrågningar till API:et för att säkerställa att det kan hantera flera användare och upprätthålla datakonsekvens.

**Motivering:** Syftet med testet är att automatiskt kontrollera att flera request som sänds samtidigt inte påverkar databasen negativt.

**Testprocess:** Jag skapade en post-request för ny author och en post-request för ny book. Under scripts/tests skrev jag en for loop där jag loopar igenom fem omgångar av pm.sendRequest för att hämta alla böcker/författare och förväntar mig statuskod 200 varje gång.

**Förväntat resultat:** Jag förväntar mig att min simulering av flera requests sända samtidigt inte påverkar databasen negativt och att for loopen med testet går igenom felfritt.

**Resultat:** Databasen hanterar ovanstående test felfritt och är inte påverkad av mängden requests.

**Test 8:** Skapa ett automatiserat test och testa om API:et hanterar olika HTTP-metoder korrekt (GET, POST, PUT, DELETE) för varje slutpunkt och returnerar lämpliga statuskoder och svar för varje metod.

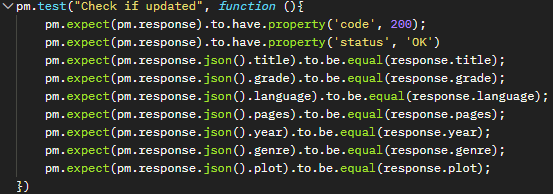
**Motivering:** Syftet med testet är att automatiskt kontrollera att varje request når sin endpoint.

**Testprocess:** Jag skapar post-, get-, put- och delete-requests där jag under scripts/tests kontrollerar jag responsens statuskoder med pm.response.to.have.status(201), pm.response.to.have.status(200), pm.response.to.have.status(204) samt. På Put-requesten skrev jag en expect som även förväntar sig att authorn har det nya namnet vi skrev i put requesten. På delete-requesten skrev jag även ett test för att kontrollera att det inte finns någon author med det id:et efter borttagningen med en sendrequest som förväntas få statuskod 404.

**Förväntat resultat:** Jag förväntar mig att rätt statuskoder visas.

**Resultat:** Testerna går igenom och rätt statuskoder visas.

**Test 9:** Skriv ett automatiserat test för att kontrollera om API:et hanterar uppdateringar av befintliga poster korrekt, vilket säkerställer att ändringar sparas och återspeglas i efterföljande förfrågningar.

**Motivering:** Syftet med testet är att automatiskt kontrollera att en **put-request** sparas korrekt.

**Testprocess:** Jag skapade först en **post-request** där jag skapar en book. Sedan skapar jag en **put-request** för att ändra delar av objektet. Under scripts/tests lägger jag till ett test för att kontrollera att de olika delarna av objektet är ändrade och sparade korrekt.

**Förväntat resultat:** Jag förväntar mig att testet går igenom och kunna se ändringarna i mongoDB

**Resultat:** Jag kan se ändringarna i mongoDB och testet går igenom.

**Test 10:** Utforma ett automatiserat prestandatest som simulerar ett stort antal användare som gör förfrågningar samtidigt för att kontrollera API:ets prestanda under hög belastning.

**Motivering:** Syftet med testet är att automatiskt kontrollera att databasen fungerar även under stora antal användare.

**Testprocess:** Jag använder samma test som i uppgift 7 förutom att jag ändrat antalet gånger for loopen hämtar alla authors till 30 istället för 5.

**Förväntat resultat:** Jag förväntar mig att databasen hanterar det plötsligt höga trycket och testet går igenom.

**Resultat:** Jag märker inga förändringar i hur databasen presterar och for loopen fungerar.

**Test 11:** Skapa ett automatiserat test som verifierar att API:et kan återställas på ett smidigt sätt från fel, till exempel problem med databasanslutningen eller avbrott i tjänster från tredje part, utan att kompromissa med dataintegriteten.

**Motivering:** Syftet med testet är att automatiskt kontrollera att API inte förlorar data av en oväntad från koppling.

**Testprocess:** Jag testade denna uppgift genom att använda postmans inbyggda performance test där man kan simulera en stor mängd användare under en begränsad tid. Under tiden requests rullade på bröt jag uppkopplingen mot mongoDB.

**Förväntat resultat:** Jag förväntar mig att requesten upphör och postman ger mig någon form av felmeddelande.

**Resultat:** Oväntat nog tycks postman ha en backup mot mongoDB då requesten fortsatte rulla på. Jag trodde det var en bugg men flera klasskamrater fick samma svar vilket får mig att tro att det inte är en bugg utan är menat så.

**Test 12:** Utveckla ett automatiserat test för att hantera edge cases, till exempel begäranden med saknade eller ogiltiga parametrar, och se till att lämpliga felmeddelanden returneras.

**Motivering:** Syftet är att automatiskt testa hur API reagerar på ogiltiga parametrar och ger rätt statuskoder.

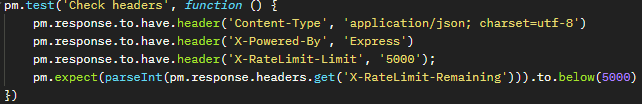
**Testprocess:** Jag återanvänder mitt test från uppgift 3 där jag gör en **get-request** för att hitta en author med id abc123. Under scripts/tests skriver jag ett test för att kontrollera att statuskod 400 syns.

**Förväntat resultat:** Jag förväntar mig att få statuskod 400 och att testet därmed går igenom.

**Resultat:** Jag får statuskod 400 och testet går igenom.

**Test 13:** Skriv ett automatiserat test för att kontrollera att API:et implementerar hastighetsbegränsnings- eller begränsningsmekanismer korrekt för att förhindra missbruk eller överdriven användning av resurser.

**Motivering:** Syftet med det automatiska testet är att kontrollera att rate-limitern är aktiverad.

**Testprocess:** Jag skapade en get-request för att hämta alla böcker och ville sedan kontrollera att en ratelimiter är med, satt till max 5000 och att den faktiskt räknar ner.

**Förväntat resultat:** Jag förväntar mig att testet går igenom och att rate-limitern är lägre än 5000.

**Resultat:** Testet går igenom och rate-limitern är längre än 5000.