Testrapport v3.0

Manuella tester

Test 1: *"Verify that the API returns the correct HTTP status code (e.g., 200 OK) for a successful GET request."*

Motivering: Att validera att en GET request får status 200 finns på flera olika tester i min postman collection för projektet eftersom det är simpelt att implementera testet i postman.

Testprocessen: För att testa om en request ger status 200 så kollar jag att responsen har status 200 med pm.response.to.have.status(200);

Förväntat resultat: att få status 200 för Get request som testet är implementerat på.

Resultat: resultatet av testet är status 200 när de GET-requests som jag implementerat testet på går bra, det vill säga att responsen ger ett resultat som har data.

Test 2: “Check if the API returns the expected data format (e.g., JSON, XML) in the response.”

Motivering: Att testa så att responses har headern Content-Type och att Content-Type har rätt typ r något som jag har ett test för. Mint test för Content-Type är implementerat på flera olika requests då det är simpelt att implementera dessa.

Testprocessen: För att kunna verifiera att responsen har content type application/json så kollade jag med ett test om responsen har headern Content-Type och sedan i samma test så kollar jag om Content-Type headern är application/json.

Förväntat resultat: Det resultat som är förväntat av testet är att hedern Content-Type finns och att Content-Type headern är application/json.

Resultat: Testet går igenom med godkänt resultat då de har både Content-Type headern och Content-Type headern är application/json.

Test 3: *"Ensure that the API returns the correct HTTP status code (e.g., 400 Bad Request) for an invalid requests."*

Motivering: För att kunna kolla om servern hanterar ogiltiga requests så har jag implementerat ett test som kollar hur servern reagerar på ofullständiga requests.

Testprocessen: Jag har implementerat ett test för att kolla responsen har status 400 om jag har ofullständig data(att jag saknar något som skulle vara i json objektet) i en POST-request.

Förväntat resultat: Att få status 400 om en POST-request saknar någon data.

Resultat: Resultatet som jag fått är att servern ger status 400 vid POST-request som saknar data.

Test 4: *"Test if the API returns the correct data when querying with specific filters or search criteria."*

Motivering: Det finns några tester för att testa så att de olika filter som API-et har fungerar som de är tänkta att fungera.

Testprocessen: De olika filterna har samma typ av tester men testerna kollar data som är specifikt för filtret. För genre filtret så kollar testet om genre responsen är si-fi eftersom testet är gjort för att kolla det. För title filtret så kollar testet i stället efter History of Tanzania men annars är processen likadan som för genre. För author filtret så kollar testet i stället efter Tammy Hessel men annars är processen likadan som för genre och title filterna.

Förväntat resultat: Testet för genre förväntar sig att responsen har si-fi som genre. Testet för title förväntar sig att responsen har History of Tanzania som title. Testet för author förväntar sig att responsen har Tammy Hessel som author.

Resultat: resultatet som jag fått på testerna för de olika filterna stämmer med de förväntade resultaten för dem.

Test 5: ”Verify that the API returns paginated results when a large number of records are requested.”

Motivering: Det finns tester för att se så att pagineringen som är implementerad fungerar som den är tänkt att den ska fungera.

Testprocessen: Testet kollar om responsen innehåller BooksPerPage som bara finns med på paginerade responses från APIet för att kunna se om resultatet är paginerat eller inte.

Förväntat resultat: Responsen innehåller BooksPerPage

Resultat: Resultatet på testet visar på att pagineringen fungerar som den är tänkt eftersom responsen innehåller BooksPerPage.

Test 6: *"Check if the API handles special characters and non-English text correctly in input data and returned responses."*

Motivering: Mitt test testar en typ av specialtecken Japanska Katakana tecken eftersom de skiljer sig helt från bokstäver.

Testprocessen: Testet som jag skrivit kollar om title, author, genre och info som matas in i request bodyn är likadan som de är i responsen. I request bodyn har jag skrivit author med katakana tecken och titel och genre med vanliga bokstäver och info med likamedtecken

Förväntat resultat: response bodyn har samma värden på title, author, genre och info som request bodyn har även om jag matar in katakana tecken.

Resultat: Resultatet som jag får visar att api-et kan hantera specialtecken som Japanska Katakana tecken efter som jag får testet att gå igenom.

Test 7:*”Test the API’s response when sending concurrent requests to ensure that it can handle multiple users and maintain data consistency.”*

Motivering: Testningen för att se till att det går att sända flera requests i rad går inte att göra helt manuellt men jag har tester som är implementerade för att sända flera requests i rad.

Testprocessen: För att se till att det går att sända flera requests i rad kollar att datan bibehålls och är konsekvent mellan requesten finns det några tester (ett för att kolla author och ett för att kolla title i responsen) som körs för varje request som görs i testningen.

Förväntat resultat: Att author och title inte har ändrats efter att all requesten är körda

Resultat: Resultatet av testningen visar att author och ett title är konsekventa från första requesten hela vägen till sista requesten.

Test 8: *"Test if the API correctly handles different HTTP methods (GET, POST, PUT, DELETE) for each endpoint and returns appropriate status codes and responses for each method."*

Motivering: För att veta om GET, POST, PUT, DELETE metoderna fungerar i de routes som dem är implementerade så har jag skapat tester för att kolla statuskoder på dem. OBS! DELETE requesten ska köras efter att test 9 har genomförts.

Testprocessen: För att testa om en Get request ger status 200 så kollar jag att responsen har status 200 med pm.response.to.have.status(200);. För att testa om en Post request ger status 201 så kollar jag att responsen har status 201 med pm.response.to.have.status(201);. För att testa om en PUT request ger status 200 så kollar jag att responsen har status 200 med pm.response.to.have.status(200);. För att testa om en DELETE request ger status 204 så kollar jag att responsen har status 204 med pm.response.to.have.status(204);

Förväntat resultat: att statuskoden ska vara 200 för GET, 201 för POST, 200 för PUT, 204 för DELETE.

Resultat: Resultatet visar att GET, POST, PUT, DELETE metoderna ger dem status koder som är förväntat av dem.

Test 9: ”Check if the API correctly handles updates to existing records, ensuring that changes are saved and reflected in subsequent requests.”

Motivering: För att säkerställa om uppdatering av data fungerar så behöver vi kolla om datan som vi vill uppdatera faktiskt blir uppdaterad genom att kolla med en ny request. I mitt test använder jag datorn från PUT-requesten som testades i test 8 eftersom jag implementerat att den sparar värdena i sig i variabler i postman.

Testprocessen: För att kunna kolla om en uppdatering av data är konsekvent eller bara har genomförts så skapar jag ett javascript objekt som innehåller datan i responsen och jämför den mot datan i variabeln som jag fått från test 8 PUT-requesten.

Förväntat resultat: Att all data i test 8 PUT-requesten och responsen i test 9 requesten är lika.

Resultat: Resultatet som jag fått visar att all data är konsekvent efter uppdatering.

Test 10: *”Test the API’s performance under heavy load, simulating a large number of users making requests simultaneously.”*

Motivering: För att testa hur API-et reagerar under belastning så behöver testningen simulerar många användare samtidigt vilket är en automatisk process och kan inte testat helt manuellt.

Testprocessen: Testningen för hård belastning har jag gjort genom att använda en request som hämtar data och sedan kört en simulering av hundra virtuella användare vilket postman har inbyggt vid run collection/folder, och låter postmans simulering av dem virtuella användarna senda requests för att kolla hur det reagerar

Förväntat resultat: Att svarstiderna blir längre med flera användare som använder systemet

Resultat: Resultatet var ok om mängden data som varje användare hämtade var liten eller paginerad men svarstiderna blev väldigt långa om det var mycket data som användarna ville ha. Prestandan för APIet är dåligt vid hård belastning om varje användare vill hämta mycket data.

Test 11: *”Verify that the API can recover gracefully from failures, such as database connection issues without compromising data integrity.”*

Motivering: För att se om API-et klarar av om databasen tappar kontakten och att API-et inte kraschar om databas kontakten återupptas så har jag skapat tester som kollar att API-et kan klara detta.

Testprocessen: Testningen som jag gjort på består av 4 request en som stänger db connectionen, en som försöker hämta data och testar om den får status 404 med pm.response.to.have.status(404);, en som öppnar db connectionen och en till som hämtar data och testar om den får status 200 med pm.response.to.have.status(200);.

Förväntat resultat: Requesten som hämtar data efter att databas connectionen är stängd ska ge status 404 och requesten som hämtar data efter att databas connectionen är öppnad igen ska ge status 200.

Resultat: Resultatet som jag fått är som förväntat. OBS! Om tiden mellan Requesten för att öppna databasen och att hämta data efter att den är öppnad är för kort så kommer resultatet att vara fel.

Test 12: *”Test the API’s ability to handle edge cases, such as requests with missing or invalid parameters, and ensure that appropriate error messages are returned.”*

Motivering: Testerna som jag har implementerat för detta påminner väldigt mycket om dem som är för test 3 eftersom test 3 kollar på invalid requests.

Testprocessen: testet kollar så att status koden blir 400 bad request med pm.response.to.have.status(400); i request bodyn så saknas två bitar av objektet samt så är en bit av objektet felaktig.

Förväntat resultat: Statuskod 400 bad request.

Resultat: Resultatet av testerna är godkända eftersom dem ger status 400.

Test 13: *”Verify that the API correctly implements rate limiting or throttling mechanisms to prevent abuse or excessive use of resources.”*

Motivering: För att det inte ska gå att sända hur många requests som under kort tid så har API-et rate limiting implementerat vilket måste testas så att det är implementerat på rätt sätt.

Testprocessen: Testet som finns för att kolla om rate limit är implementerat kollar efter X-RateLimit-Limit headern i responsen med pm.response.to.have.header("X-RateLimit-Limit");

Förväntat resultat: Att responsen har X-RateLimit-Limit headern.

Resultat: Resultatet som jag fått är att responsen har X-RateLimit-Limit headern.

Automatiserade tester

Dem automatiska tester behöver köras med minst 1000ms delay för att alla ska fungera.

Test 1: *"Validate that the API returns the correct HTTP status code (e.g., 200 OK) for a successful GET request."*

Motivering: Att validera att en GET request får status 200 finns på flera olika tester i min postman collection för projektet eftersom det är simpelt att implementera testet i postman.

Testprocessen: För att testa om en request ger status 200 så kollar jag att responsen har status 200 med pm.response.to.have.status(200);

Förväntat resultat: att få status 200 för Get request som testet är implementerat på.

Resultat: resultatet av testet är status 200 när de GET-requests som jag implementerat testet på går bra, det vill säga att responsen ger ett resultat som har data.

Test 2: *"Verify that the API returns the expected data format (e.g., JSON, XML) in the response."*

Motivering: Att testa så att responses har headern Content-Type och att Content-Type har rätt typ r något som jag har ett test för. Mint test för Content-Type är implementerat på flera olika requests då det är simpelt att implementera dessa.

Testprocessen: För att kunna verifiera att responsen har content type application/json så kollade jag med ett test om responsen har headern Content-Type och sedan i samma test så kollar jag om Content-Type headern är application/json.

Förväntat resultat: Det resultat som är förväntat av testet är att hedern Content-Type finns och att Content-Type headern är application/json.

Resultat: Testet går igenom med godkänt resultat då de har både Content-Type headern och Content-Type headern är application/json.

Test 3: *"Ensure that the API returns the correct HTTP status code (e.g., 400 Bad Request) for an invalid requests."*

Motivering: För att kunna kolla om servern hanterar ogiltiga requests så har jag implementerat ett test som kollar hur servern reagerar på ofullständiga requests.

Testprocessen: Jag har implementerat ett test för att kolla responsen har status 400 om jag har ofullständig data(att jag saknar något som skulle vara i json objektet) i en POST-request.

Förväntat resultat: Att få status 400 om en POST-request saknar någon data.

Resultat: Resultatet som jag fått är att servern ger status 400 vid POST-request som saknar data.

Test 4: *"Create an automated test that sends a request with specific filters or search criteria and checks if the API returns the correct data."*

Motivering: Det finns några tester för att testa så att de olika filter som API-et har fungerar som de är tänkta att fungera.

Testprocessen: De olika filterna har samma typ av tester men testerna kollar data som är specifikt för filtret. För genre filtret så kollar testet om genre responsen är si-fi eftersom testet är gjort för att kolla det. För title filtret så kollar testet i stället efter History of Tanzania men annars är processen likadan som för genre. För author filtret så kollar testet i stället efter Tammy Hessel men annars är processen likadan som för genre och title filterna.

Förväntat resultat: Testet för genre förväntar sig att responsen har si-fi som genre. Testet för title förväntar sig att responsen har History of Tanzania som title. Testet för author förväntar sig att responsen har Tammy Hessel som author.

Resultat: resultatet som jag fått på testerna för de olika filterna stämmer med de förväntade resultaten för dem

Test 5: *"Write an automated test to verify that the API returns paginated results when a large number of records are requested."*

Motivering: Det finns tester för att se så att pagineringen som är implementerad fungerar som den är tänkt att den ska fungera.

Testprocessen: Testet kollar om responsen innehåller BooksPerPage som bara finns med på paginerade responses från APIet för att kunna se om resultatet är paginerat eller inte.

Förväntat resultat: Responsen innehåller BooksPerPage

Resultat: Resultatet på testet visar på att pagineringen fungerar som den är tänkt eftersom responsen innehåller BooksPerPage.

Test 6: *"Test if the API handles special characters and non-English text correctly in input data and returned responses using an automated testing tool."*

Motivering: Mitt test testar en typ av specialtecken Japanska Katakana tecken eftersom de skiljer sig helt från bokstäver.

Testprocessen: Testet som jag skrivit kollar om title, author, genre och info som matas in i request bodyn är likadan som de är i responsen. I request bodyn har jag skrivit author med katakana tecken och titel och genre med vanliga bokstäver och info med likamedtecken

Förväntat resultat: response bodyn har samma värden på title, author, genre och info som request bodyn har även om jag matar in katakana tecken.

Resultat: Resultatet som jag får visar att api-et kan hantera specialtecken som Japanska Katakana tecken efter som jag får testet att gå igenom.

Test 7: *"Develop an automated test that sends concurrent requests to the API to ensure that it can handle multiple users and maintain data consistency."*

Motivering: Testningen för att se till att det går att sända flera requests i rad går inte att göra helt manuellt men jag har tester som är implementerade för att sända flera requests i rad.

Testprocessen: För att se till att det går att sända flera requests i rad kollar att datan bibehålls och är konsekvent mellan requesten finns det några tester (ett för att kolla author och ett för att kolla title i responsen) som körs för varje request som görs i testningen.

Förväntat resultat: Att author och title inte har ändrats efter att all requesten är körda

Resultat: Resultatet av testningen visar att author och ett title är konsekventa från första requesten hela vägen till sista requesten.

Test 8: *"Create an automated test and test if the API correctly handles different HTTP methods (GET, POST, PUT, DELETE) for each endpoint and returns appropriate status codes and responses for each method."*

Motivering: För att veta om GET, POST, PUT, DELETE metoderna fungerar i de routes som dem är implementerade så har jag skapat tester för att kolla statuskoder på dem. OBS! DELETE requesten ska köras efter att test 9 har genomförts.

Testprocessen: För att testa om en Get request ger status 200 så kollar jag att responsen har status 200 med pm.response.to.have.status(200);. För att testa om en Post request ger status 201 så kollar jag att responsen har status 201 med pm.response.to.have.status(201);. För att testa om en PUT request ger status 200 så kollar jag att responsen har status 200 med pm.response.to.have.status(200);. För att testa om en DELETE request ger status 204 så kollar jag att responsen har status 204 med pm.response.to.have.status(204);

Förväntat resultat: att statuskoden ska vara 200 för GET, 201 för POST, 200 för PUT, 204 för DELETE.

Resultat: Resultatet visar att GET, POST, PUT, DELETE metoderna ger dem status koder som är förväntat av dem.

Test 9: *"Write an automated test to check if the API correctly handles updates to existing records, ensuring that changes are saved and reflected in subsequent requests."*

Motivering: För att säkerställa om uppdatering av data fungerar så behöver vi kolla om datan som vi vill uppdatera faktiskt blir uppdaterad genom att kolla med en ny request. I mitt test använder jag datorn från PUT-requesten som testades i test 8 eftersom jag implementerat att den sparar värdena i sig i variabler i postman.

Testprocessen: För att kunna kolla om en uppdatering av data är konsekvent eller bara har genomförts så skapar jag ett javascript objekt som innehåller datan i responsen och jämför den mot datan i variabeln som jag fått från test 8 PUT-requesten.

Förväntat resultat: Att all data i test 8 PUT-requesten och responsen i test 9 requesten är lika.

Resultat: Resultatet som jag fått visar att all data är konsekvent efter uppdatering.

Test 10: *"Design an automated performance test that simulates a large number of users making requests simultaneously to check the API's performance under heavy load."*

Motivering: För att testa hur API-et reagerar under belastning så behöver testningen simulerar många användare samtidigt vilket är en automatisk process och kan inte testat helt manuellt.

Testprocessen: Testningen för hård belastning har jag gjort genom att använda en request som hämtar data och sedan kört en simulering av hundra virtuella användare vilket postman har inbyggt vid run collection/folder, och låter postmans simulering av dem virtuella användarna senda requests för att kolla hur det reagerar

Förväntat resultat: Att svarstiderna blir längre med flera användare som använder systemet

Resultat: Resultatet var ok om mängden data som varje användare hämtade var liten eller paginerad men svarstiderna blev väldigt långa om det var mycket data som användarna ville ha. Prestandan för APIet är dåligt vid hård belastning om varje användare vill hämta mycket data.

Test 11: *"Create an automated test that verifies the API can recover gracefully from failures, such as database connection issues or third-party service outages, without compromising data integrity."*

Motivering: För att se om API-et klarar av om databasen tappar kontakten och att API-et inte kraschar om databas kontakten återupptas så har jag skapat tester som kollar att API-et kan klara detta.

Testprocessen: Testningen som jag gjort på består av 4 request en som stänger db connectionen, en som försöker hämta data och testar om den får status 404 med pm.response.to.have.status(404);, en som öppnar db connectionen och en till som hämtar data och testar om den får status 200 med pm.response.to.have.status(200);.

Förväntat resultat: Requesten som hämtar data efter att databas connectionen är stängd ska ge status 404 och requesten som hämtar data efter att databas connectionen är öppnad igen ska ge status 200.

Resultat: Resultatet som jag fått är som förväntat. OBS! Om tiden mellan Requesten för att öppna databasen och att hämta data efter att den är öppnad är för kort så kommer resultatet att vara fel.

Test 12: *"Develop an automated test to handle edge cases, such as requests with missing or invalid parameters, and ensure that appropriate error messages are returned."*

Motivering: Testerna som jag har implementerat för detta påminner väldigt mycket om dem som är för test 3 eftersom test 3 kollar på invalid requests.

Testprocessen: testet kollar så att status koden blir 400 bad request med pm.response.to.have.status(400); i request bodyn så saknas två bitar av objektet samt så är en bit av objektet felaktig.

Förväntat resultat: Statuskod 400 bad request.

Resultat: Resultatet av testerna är godkända eftersom dem ger status 400.

Test 13: *Write an automated test to verify that the API correctly implements any rate limiting or throttling mechanisms to prevent abuse or excessive use of resources.*

Motivering: För att det inte ska gå att sända hur många requests som under kort tid så har API-et rate limiting implementerat vilket måste testas så att det är implementerat på rätt sätt.

Testprocessen: Testet som finns för att kolla om rate limit är implementerat kollar efter X-RateLimit-Limit headern i responsen med pm.response.to.have.header("X-RateLimit-Limit");

Förväntat resultat: Att responsen har X-RateLimit-Limit headern.

Resultat: Resultatet som jag fått är att responsen har X-RateLimit-Limit headern.