Testrapport v2.0

Manuella tester

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

Att validera att en get request får status 200 finns på flera olika tester i postman min postman collection för projektet eftersom det är simpelt att implementera testet i postman. 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. Testerna går igenom godkända och deras statuskod stämmer med vad postman visar för statuskod.

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

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 ett annat test så kollar jag om Content-Type headern är application/json. Mina test för Content-Type är implementerat på flera olika requests då det är simpelt att implementera dessa. Testerna går igenom med godkänt resultat.

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

Jag har implementerat ett test för att se så att API-et ger status 400 om jag har ofullständig data i en POST-request. jag har implementerat ett test som kollar om servern returnerar status 400 vid ofullständiga requeststst och testet visar på att API-et hantera invalid POST-requests korrekt.

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

Det finns några tester för att testa så att de olika filter som API-et har fungerar som de är tänkta att fungera. De olika filterna har samma typ av tester men testerna kollar data som är specifikt för filtret. Resultatet på testerna är godkänt då de går igenom utan fel.

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

Det finns tester för att se så att pagineringen som är implementerad fungerar som den är tänkt att den ska fungera. Testerna kollar av en del data(som sida som är vald) som bara finns med på planerade resultat för att se så att resultatet är paginerat. resultatet på testerna visar på att pagineringen fungerar som den är implementerad.

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

Testet som jag skrivit kollar om den data som matas in i request bodyn är likadan som den data responsen ger. Mitt test ger täckning för en typ av specialtecken som jag har stött på innan(Japanska Katakana tecken). Testresultatet som jag får visar att api kan hantera specialtecken som Japanska Katakana tecken.

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

Testningen för att se till att det går att sända flera requests i rad kollar att datan bibehålls och är konsekvent mellan requesten, detta är gjort genom att det finns några tester (för att kolla datan i responsen) som körs för varje request som görs i testningen. Resultatet av testningen visar att datan är konsekvent 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.”*

Det finns tester för GET, POST, PUT, DELETE metoderna som kollar så att status koden som responser ger är som dem ska vara(200 för GET, 201 för POST, 200 för PUT, 204 för DELETE). Testresultatet visar att GET, POST, PUT, DELETE metoderna fungerar som de ska och 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.”

testerna som jag skapat för att kolla om datan stämmer efter uppdatering av data kollar om alla delar som en användare kan sätta stämmer med varandra mellan request bodyn och en GET request response body, resultatet visar att datan är konsekvent efter uppdatering.

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

Testningen för hård belastning har jag gjort genom att använda en request som hämtar data och sedan kört en simulering av hundra virtuella användare som samtidigt använder api-et vilket gav ok resultat om datan som varje användare hämtade var liten men svarstiderna blev väldigt långa om det var mycket data som användarna ville ha. Resultatet för testningen är att 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.”*

Testningen som jag gjort på består av 4 request en som stänger db connectionen, en som försöker hämta data, en som öppnar db connectionen och en till som hämtar data. Resultatet som jag har fått tyder på att data integriteten bibehålls.

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.”*

Testerna som jag har implementerat för detta påminner väldigt mycket om dem som är för test 3 och dessa tester kollar så att status koden blir 400 bad request om du saknar bitar av ett json-objekt i en POST-request. resultatet av testerna är godkända efter som dem ger status 400 vilket dem ska ge.

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

Testningen som jag har för att testa rate limit kollar efter en header som bara finns om rate limit är implementerat som det ska och resultatet som jag får på testet visar att ratelimiten som jag implementerat fungerar som den ska.

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."*

Att validera att en get request får status 200 finns på flera olika tester i postman min postman collection för projektet eftersom det är simpelt att implementera testet i postman. 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. Testerna går igenom godkända och deras statuskod stämmer med vad postman visar för statuskod.

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

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 ett annat test så kollar jag om Content-Type headern är application/json. Mina test för Content-Type är implementerat på flera olika requests då det är simpelt att implementera dessa. Testerna går igenom med godkänt resultat.

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

Jag har implementerat ett test för att se så att API-et ger status 400 om jag har ofullständig data i en POST-request. jag har implementerat ett test som kollar om servern returnerar status 400 vid ofullständiga requeststst och testet visar på att API-et hantera invalid POST-requests korrekt.

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."*

Det finns några tester för att testa så att de olika filter som API-et har fungerar som de är tänkta att fungera. De olika filterna har samma typ av tester men testerna kollar data som är specifikt för filtret. Resultatet på testerna är godkänt då de går igenom utan fel.

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

Det finns tester för att se så att pagineringen som är implementerad fungerar som den är tänkt att den ska fungera. Testerna kollar av en del data(som sida som är vald) som bara finns med på planerade resultat för att se så att resultatet är paginerat. resultatet på testerna visar på att pagineringen fungerar som den är implementerad.

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.”*

Testet som jag skrivit kollar om den data som matas in i request bodyn är likadan som den data responsen ger. Mitt test ger täckning för en typ av specialtecken som jag har stött på innan(Japanska Katakana tecken). Testresultatet som jag får visar att api kan hantera specialtecken som Japanska Katakana tecken.

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.”*

Testningen för att se till att det går att sända flera requests i rad kollar att datan bibehålls och är konsekvent mellan requesten, detta är gjort genom att det finns några tester (för att kolla datan i responsen) som körs för varje request som görs i testningen. Resultatet av testningen visar att datan är konsekvent 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.”*

Det finns tester för GET, POST, PUT, DELETE metoderna som kollar så att status koden som responser ger är som dem ska vara(200 för GET, 201 för POST, 200 för PUT, 204 för DELETE). Testresultatet visar att GET, POST, PUT, DELETE metoderna fungerar som de ska och 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."*

testerna som jag skapat för att kolla om datan stämmer efter uppdatering av data kollar om alla delar som en användare kan sätta stämmer med varandra mellan request bodyn och en GET request response body, resultatet visar att datan 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."*

Testningen för hård belastning har jag gjort genom att använda en request som hämtar data och sedan kört en simulering av hundra virtuella användare som samtidigt använder api-et vilket gav ok resultat om datan som varje användare hämtade var liten men svarstiderna blev väldigt långa om det var mycket data som användarna ville ha. Resultatet för testningen är att 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."*

Testningen som jag gjort på består av 4 request en som stänger db connectionen, en som försöker hämta data, en som öppnar db connectionen och en till som hämtar data. Resultatet som jag har fått tyder på att data integriteten bibehålls, om requestsen körs för snabbt inpå varandra kommer testerna att misslyckas eftersom databasen tar tid att starta.

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."*

Testerna som jag har implementerat för detta påminner väldigt mycket om dem som är för test 3 och dessa tester kollar så att status koden blir 400 bad request om du saknar bitar av ett json-objekt i en POST-request. resultatet av testerna är godkända efter som dem ger status 400 vilket dem ska ge.

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.*

Testningen som jag har för att testa rate limit kollar efter en header som bara finns om rate limit är implementerat som det ska och resultatet som jag får på testet visar att ratelimiten som jag implementerat fungerar som den ska.