Test av IT-system

Individuell uppgift 4: Interaktion och testning av klient-server-API:er

Luleå Tekniska Universitet
I0015N
VT25

Noah Alvandi

Innehållsförteckning

Inledning

Syftet med denna uppgift var att automatisera API-testning genom att använda TypeScript tillsammans med Axios-biblioteket och testverktyget Jest. Systemet som testades är ett REST-API som finns publicerat på adressen *https://ltu-i0015n-2024-web.azurewebsites.net*, och som innehåller två huvudsakliga ändpunkter: **`/images`** och **`/images/search`**. Målet med testerna var att verifiera både funktionell korrekthet och att identifiera eventuella inkonsekvenser i hur API:et hanterar autentisering och felaktig indata.

Testfallen designades för att täcka både positiva scenarier (korrekt data och autentisering) och negativa scenarier (felaktiga API-nycklar, saknade data eller tomma fält). Detta gjordes i syfte att kontrollera att API:et svarar med förväntade HTTP-statuskoder i varje situation.

Testplan

Testplanen fokuserade på två HTTP-metoder: **`GET`** för att hämta alla bilder via **`/images`**, samt **`POST`** för att söka bilder via **`/images/search`**. Samtliga API-anrop krävde autentisering genom en HTTP-header med nyckeln **`X-Api-Key`** och värdet **`super-secret-api-key-abc123`**. För att undvika upprepning och säkerställa konsistens skapades en gemensam Axios-instans där **`baseURL`** och headern för API-nyckeln sattes in som standard.

Testerna för **`/images`** kontrollerade om anrop med giltig API-nyckel fungerade korrekt, samt om felaktig eller utebliven nyckel gav rätt svarskod. För **`/images/search`** verifierades att en korrekt sökfråga gav resultat, medan olika typer av ogiltig indata testades för att se om API:et returnerade **`400 Bad Request`** eller **`422 Unprocessable Entity`**. Även autentiseringstester ingick för att jämföra beteendet mellan de två ändpunkterna.

Alla testfall kördes isolerat, och varje testfall skilde tydligt mellan autentiseringslogik och validering av indata.

Testfall

Tabell 1. Följande tabell sammanfattar de testfall som genomfördes under uppgiften. Varje rad representerar ett enskilt testfall med tillhörande syfte, den data som användes vid testkörningen, samt det förväntade och faktiska utfallet. Målet med tabellen är att ge en tydlig översikt över hur systemet beter sig i olika scenarier, både vid korrekta och felaktiga indata, samt vid varierande autentiseringsförhållanden. Tabellen visar också om varje testfall passerade eller misslyckades, vilket ger en direkt indikation på API:ets stabilitet och eventuella inkonsekvenser i dess implementation.

Testfall ID	Syfte	Testdata	Förväntat resultat	Faktiskt resultat	Status
TF-01	Kontrollera att `GET /images` returnerar bilder med giltig API-nyckel	GET `/images` med header `X-Api-Key: super-secret-api-key-abc123`	HTTP 200 OK, svaret innehåller en lista av bilder	HTTP 200 OK, `results`-arra y returneras	Godkänd
TF-02	Kontrollera att `GET /images` utan API-nyckel returnerar 401	GET `/images` utan header	HTTP 401 Unauthorized	HTTP 401 Unauthorized	Godkänd
TF-03	Kontrollera att `GET /images` med felaktig nyckel returnerar 403	GET `/images` med `X-Api-Key: wrong-key`	HTTP 403 Forbidden	HTTP 403 Forbidden	Godkänd
TF-04	Kontrollera att `POST /images/search` med korrekt sökfråga fungerar	POST `/images/search` med `{ searchQuery: "cat" }`	HTTP 200 OK, svaret innehåller relevanta resultat	HTTP 200 OK, `results`-arra y returneras	Godkänd
TF-05	Kontrollera att `searchQuery` som saknas ger 400	POST `/images/search` med `{}`	HTTP 400 Bad Request	HTTP 400 Bad Request	Godkänd
TF-06	Kontrollera att tom `searchQuery` ger 400	POST `/images/search` med `{ searchQuery: "" }`	HTTP 400 eller 422	HTTP 400 Bad Request	Godkänd
TF-07	Kontrollera att API-nyckel som saknas vid sökning ger 401	POST `/images/search` utan `X-Api-Key`	HTTP 401 Unauthorized	HTTP 401 Unauthorized	Godkänd
TF-08	Kontrollera att felaktig API-nyckel vid sökning ger 403	POST `/images/search` med `X-Api-Key: invalid-key`	HTTP 403 Forbidden	HTTP 401 Unauthorized (inkonsekvent beteende)	Underkänd

Initiering och körning

Projektet initierades med kommandot **`npm init`**, och därefter installerades nödvändiga beroenden: **`axios`**, **`jest`**, **`ts-jest`**, **`ts-node`** och **`typescript`**. En gemensam Axios-instans konfigurerades i en separat modul (**`axiosInstance.ts`**) där både **`baseURL`** och **`X-Api-Key`** angavs. Denna instans importerades i samtliga testfall för att minska koduppprepning och säkerställa konsekvenser mellan testerna.

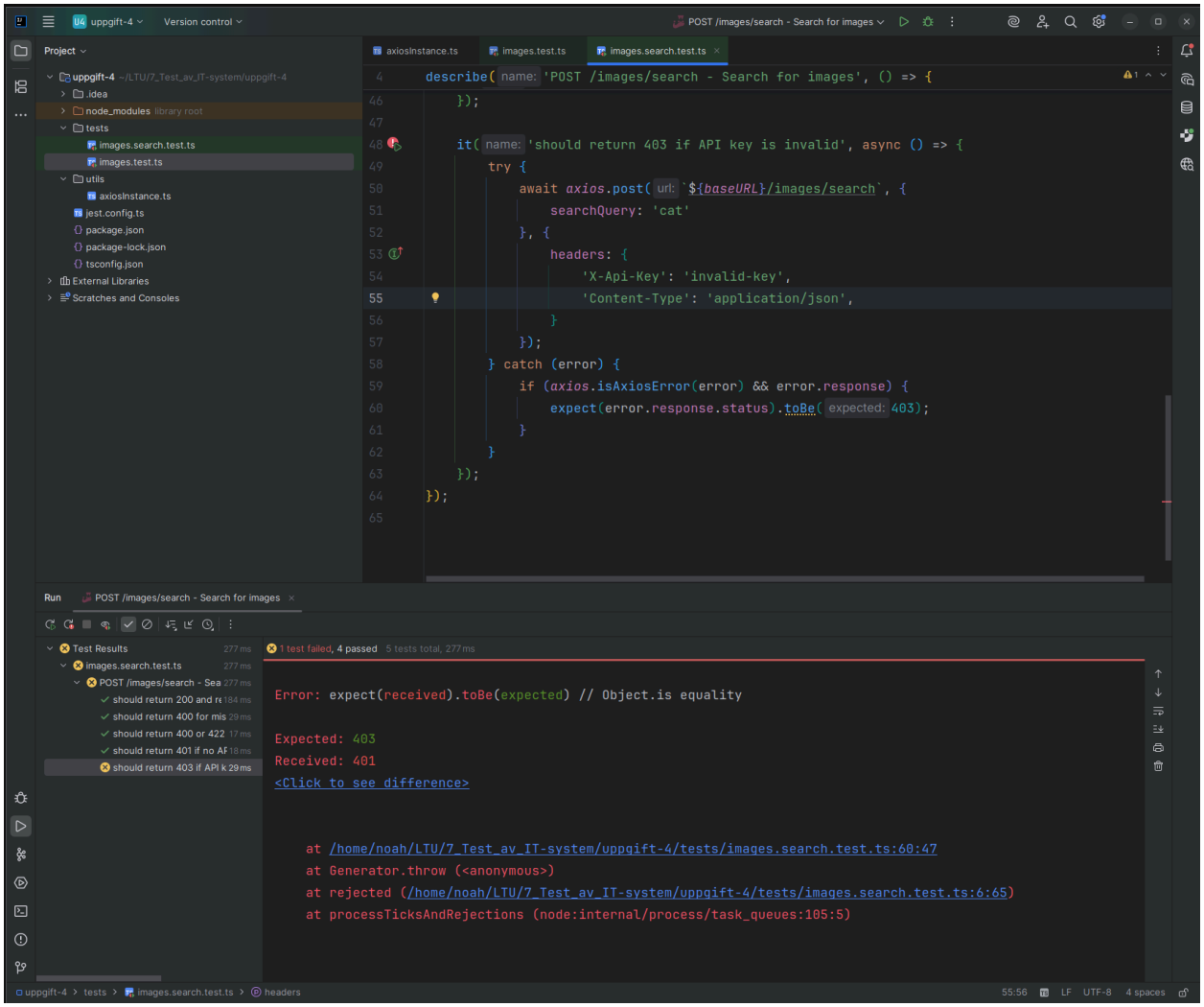
Testfallen skrevs i TypeScript med filändelsen **`.test.ts`** och placerades i mappen **`tests`**. Projektets inställningar hanterades via **`tsconfig.json`** för TypeScript och **`jest.config.js`** för Jest. När alla beroenden var installerade kunde testerna köras genom kommandot **`npm test`** i terminalen. Vid körning visades testresultaten direkt i konsolen med tydlig information om vilka tester som lyckades och vilka som misslyckades, inklusive felmeddelanden och radnummer i koden.

Ett exempel på ett misslyckat test visas i figur 1 nedan. Det aktuella testfallet hade som syfte att verifiera att ett **`POST`**-anrop mot **`/images/search`** med en ogiltig API-nyckel skulle resultera i statuskoden **`403 Forbidden`**. I praktiken returnerade dock API:et statuskod **`401 Unauthorized`**, vilket ledde till att testet underkändes. Denna avvikelse är problematisk eftersom statuskod **`401`** normalt indikerar att autentiseringsuppgifter saknas, medan **`403`** signalerar att autentisering har genomförts men att åtkomst nekas. I detta fall är **`X-Api-Key`** visserligen ogiltig, men den är fortfarande närvarande, vilket gör **`403 Forbidden`** till den mer korrekta och semantiskt riktiga responsen. Det inkonsekventa beteendet mellan **`/images`** och **`/images/search`** i detta avseende kan indikera en brist på hur API:ets autentisering är implementerad mellan olika ändpunkter.

För att visa testkörningen i praktiken har jag spelat in en kort video (under en minut) där hela testfallen körs och resultaten presenteras i terminalen. Videon är uppladdad som olistad på YouTube och kan ses via följande länk:

YouTube-länk till testkörning: https://youtu.be/J9puZK-Be0s
(Observera att länken är olistad och endast tillgänglig för personer som har tillgång till adressen.)

Jag kommer även att bifoga hela projektmappen till denna rapport så att koden, testerna och konfigurationen enkelt kan granskas, köras och återanvändas vid behov.



Figur 1. Skärmdump från det felande testfallet TC-08 i testfilen **`images.search.test.ts`**. Testet förväntade att API:et skulle returnera HTTP-statuskoden **`403 Forbidden`** vid anrop med en ogiltig API-nyckel, men fick istället **`401 Unauthorized`**. Detta indikerar ett inkonsekvent beteende i API:ets hantering av autentisering, där nyckeln är närvarande men ändå behandlas som om autentisering saknas. Figuren visar både kodraden som definierar förväntat resultat samt fel utmatningen från Jest i terminalen.

Reflektion kring lärdomar och testapproach

Genom arbetet med denna uppgift har jag utvecklat en djupare förståelse för hur API-testning fungerar i praktiken, och hur viktigt det är att arbeta strukturerat med automatiserade tester. Jag har lärt mig att ett tydligt definierat testupplägg med enhetliga metoder för att validera svarskoder, hantera headers och skicka begäranden bidrar till att bygga mer robusta och pålitliga system.

Att arbeta med Jest som testramverk har varit särskilt lärorikt. Det har gett mig insikt i hur testfall kan struktureras, hur fel fångas upp och rapporteras, och hur man arbetar med asynkrona HTTP-anrop i en testmiljö. Kombinationen med Axios gav dessutom en naturlig och flexibel modell för att simulera verkliga API-anrop, vilket gjorde att testerna inte bara verifierade logik utan även interaktionen med själva tjänsten.

En viktig insikt är att automatiserad testning inte enbart handlar om att bekräfta att saker fungerar, utan också om att avslöja brister i logik, design och konsekvens. Tester gör det möjligt att upptäcka sådant som annars kan gå obemärkt förbi under manuell testning eller användning.

Utifrån dessa erfarenheter rekommenderar jag starkt den testapproach som har använts i denna uppgift till Bothniabladet. Att ha en automatiserad teststruktur med Jest och Axios gör det möjligt att kvalitetssäkra API:er kontinuerligt, snabbt och tillförlitligt. Det skapar trygghet i utvecklingsprocessen och ger möjlighet att snabbt upptäcka regressionsfel eller inkonsekvenser, särskilt i ett system som kan komma att växa eller integreras med andra tjänster över tid.