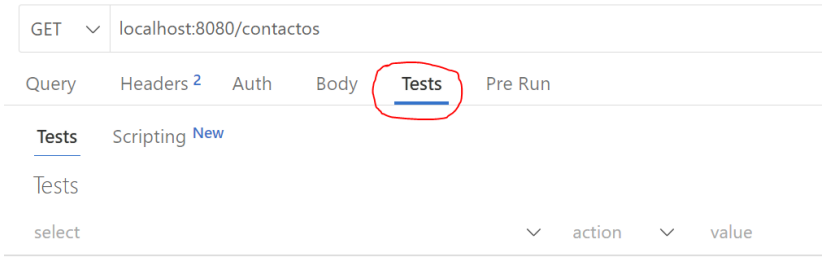
Testing de serveis REST

En este document donarem unes nocions bàsiques de com fer *testing* de serveis REST, i passar un conjunt de proves sobre una API REST i recollir els resultats.

1. Testing amb *ThunderClient*

En documents anteriors ja hem explicat com utilitzar l'extensió *ThunderClient* de Visual Studio Code per a realitzar proves de serveis REST. No obstant això, estes proves que vam fer llavors eren proves aïllades i "manuals": nosaltres creàvem la petició, construíem la URL i miràvem la resposta. El que veurem ací és com podem configurar col·leccions perquè s'acte-executen i avaluen els seus resultats.

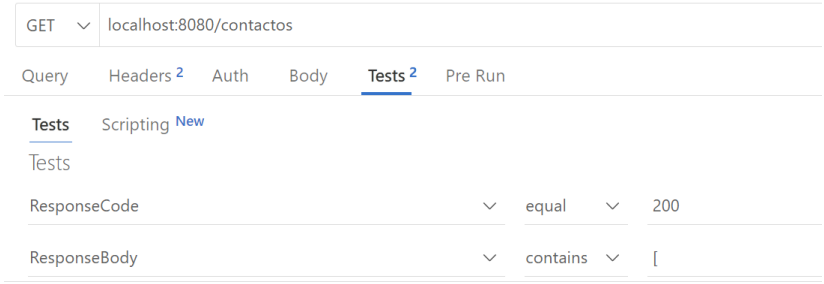
Els tests es gestionaran des de la pestanya *Testing* de cada petició o *Request*. Ací definirem el paràmetre (o paràmetres) que volem chequear de cada resposta.



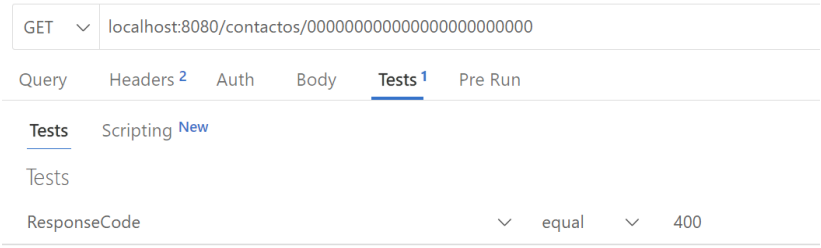
1.1. Definint tests bàsics

Definirem un test bàsic sobre el llistat de contactes, del nostre exemple *ContactosREST_v2*. En la col·lecció de proves, anem a la prova del llistat de contactes (*GET /contactes*) i en la seua pestanya *Testing* indiquem dos paràmetres de comprovació:

- El codi d'estat de la prova ha de ser 200
- El contingut de la prova ha de contindre un array (claudàtors), encara que estiga buit.

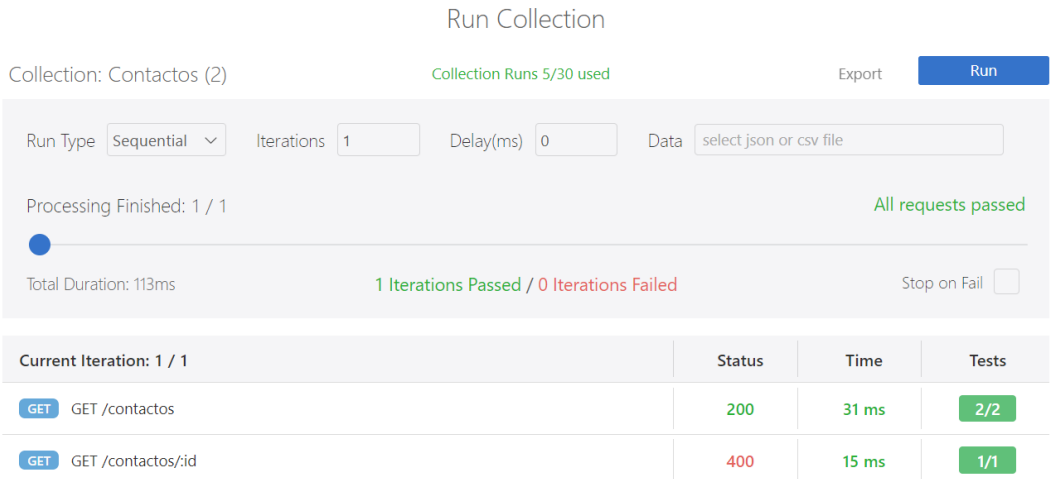


També podem fer altres tests senzills en altres peticions (*requests*). Per exemple, en la fitxa d'un contacte que sapiem que és incorrecte podem veure que el codi d'estat siga un 400, si ho hem especificat així en la resposta:

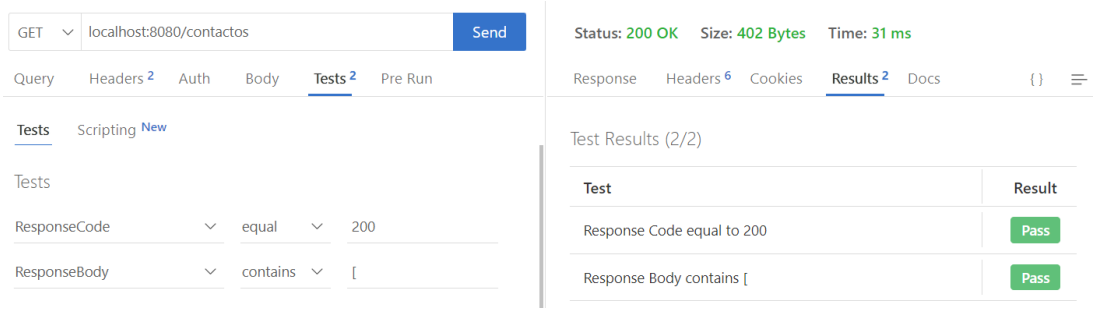


1.2. Execució i comprovació de tests

Per a llançar de colp tots els tests que hem definit fem clic dret sobre la col·lecció (en el panell esquerre) i triem *Run All*, i després premem el botó de *Run* que apareixerà. Això posarà en marxa tots els tests i, en el panell dret, podrem veure els resultats generals i de cada test:



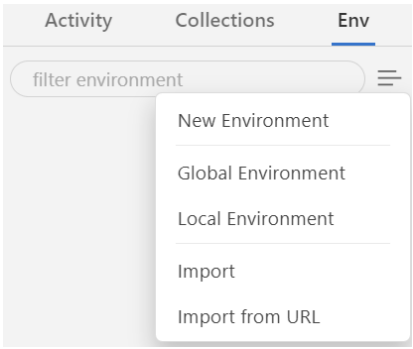
Fent clic en cada test podrem veure el desglossament de les proves o paràmetres que hem demanat verificar:



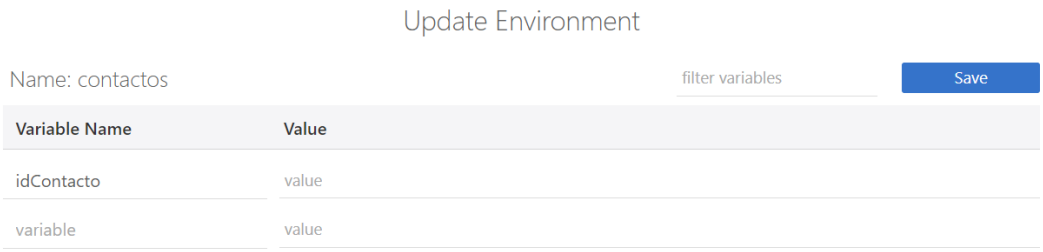
1.3. Utilitzant variables en els tests

En algunes ocasions llançar un test no és tan senzill, i necessitem una part variable que no podem controlar per endavant. Per exemple, si inserim un document que després volem modificar, com podem guardar-nos l'*id* d'eixe document per a la prova de modificació posterior?

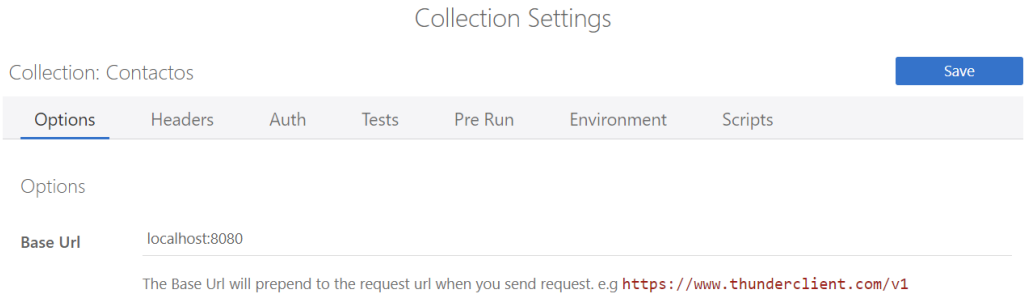
Per a això farem ús de la pestanya ***Env*** del panell esquerre, on podem crear variables d'entorn, o ben globals, o ben associades a una col·lecció en particular.



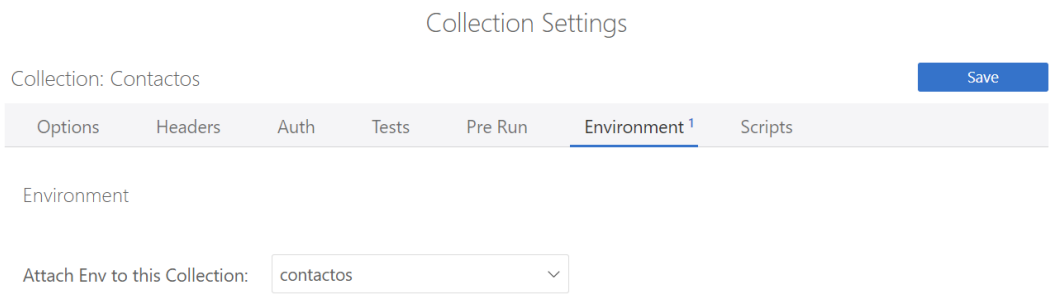
Crearem un nou entorn (*New Environment*) i li posarem un nom. Per exemple, *contactes*. Dins de l'entorn podem crear variables i deixar-les preparades per a emmagatzemar dades que necessitem. Per exemple, l'*id* del contacte amb el qual vulguem treballar:



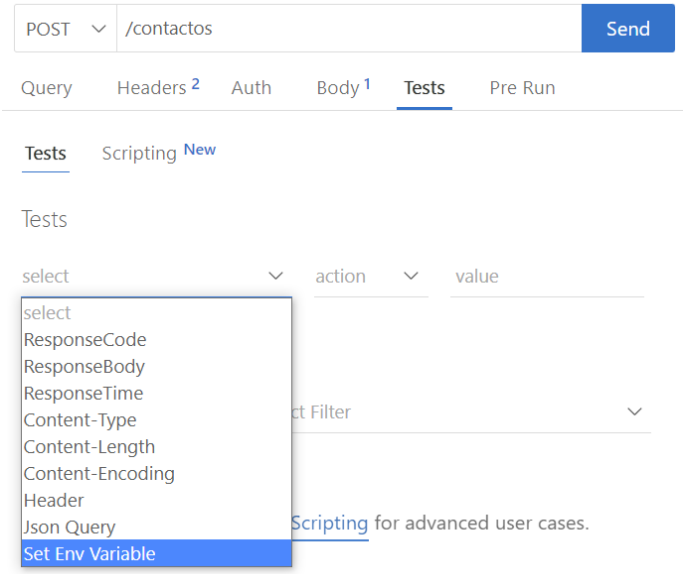
Ara hem d'anar a la col·lecció de proves de contactes, fer clic dret en ella i anar a ***Settings***. En la secció *Options* podem establir (si volem) una URL base per a totes les peticions de la col·lecció, de manera que totes començaran per este prefix (i podrem canviar fàcilment, per exemple, el número de port o la URL base en totes elles):



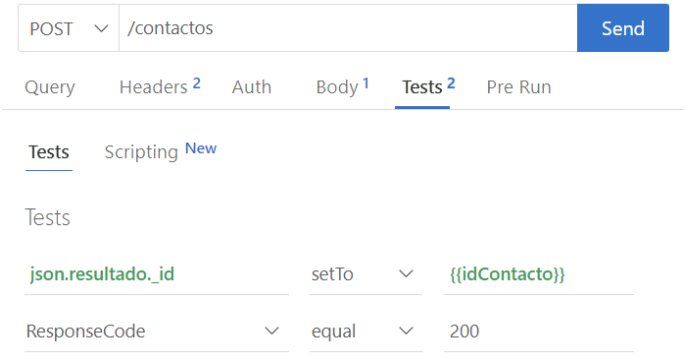
En la secció *Environment* podem associar un entorn a esta col·lecció (en el nostre cas, l'entorn *contactes*) que hem creat abans. D'aquesta manera qualsevol variable que hàgem definit en eixe entorn es podrà utilitzar directament en esta col·lecció.



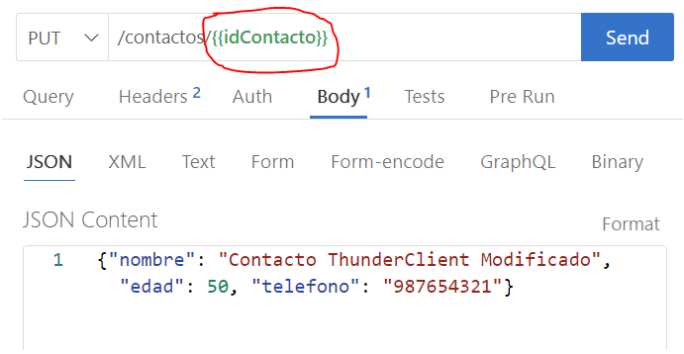
Vegem com funciona això. En la prova de POST per a inserir un nou contacte, anem a la pestanya *Tests* i triem l'opció *Set Env Variable*.



Després triem quin paràmetre JSON volem recollir (en el nostre cas, l'*id* del contacte inserit) i en quina variable volem guardar-lo (en la nostra variable d'entorn *idContacto*, expressada entre dobles claus). A més, podem indicar altres paràmetres de testatge, com que el codi de resposta siga 200:



A partir de la prova d'este servei, els que vinguen després ja tindran un valor emmagatzemat en la variable *idContacto*, i podem utilitzar-lo. Per exemple, així podríem modificar alguna dada del contacte recentment inserit, siga com siga el seu *id*:



1.4. Exportant tests

Si volem exportar un test per a poder-lo utilitzar en altres sistemes, haurem de tindre en compte si té variables d'entorn associades. En este cas haurem d'exportar tant l'entorn (pestanya *Env*) com la col·lecció, i importar-les en el sistema de destí.

2. Testing des de JavaScript: *Axios*

Un dels principals inconvenients que podem trobar a l'hora de provar les nostres aplicacions REST amb eines com Thunder Client o Postman és que el seu ús gratuït té unes certes limitacions, quant al nombre de vegades que podem executar una col·lecció.

Per a pal·liar esta deficiència i complementar millor les nostres proves podem fer ús d'alguna llibreria específica en JavaScript que permeta llançar-les de manera gratuïta i il·limitada des del codi. És el cas d'Axios.

2.1. Instal·lació i configuració

Les proves que fem en Axios les podem definir en un projecte a part del qual vulguem provar, o bé podem definir alguna carpeta *tests* dins del propi projecte. Encara que, això sí, necessitarem executar el programa de tests des d'un terminal, i pot ser que el terminal del projecte principal ja estiga ocupat executant Express.

En el projecte on anem a fer les proves necessitem instal·lar el mòdul *axios* amb la corresponent comanda:

```
npm install axios
```

Una vegada instal·lat ho incorporem al fitxer font (o fitxers) on anem a fer els tests (per exemple, `tests.js`), i realitzem una configuració prèvia on indiquem la URL base a la que volem connectar (aquella on estarà escoltant el projecte a provar), i el tipus de contingut que utilitzarem (en el nostre cas, contingut JSON):

```
const axios = require('axios');

const axiosInstance = axios.create({
    baseURL: 'http://localhost:8080',
    headers: {
        'Content-Type': 'application/json',
    }
});
```

2.2. Definició de proves simples

Imaginem que volem provar el llistat de contactes, que està escoltant per GET en la URI */contactes*. Podríem definir un mètode en el nostre fitxer de `tests.js` que llance la petició i reculla la resposta. En el propi codi JavaScript podem avaluar quins paràmetres considerem correctes i quins un error. Per exemple, esperem un codi 200 de resposta d'estat, i una llista (de grandària 0 o superior).

```
const obtenirContactes = async () => {
    try {
        const resposta = await axiosInstance.get('/contactes');
        if(resposta.status == 200 && resposta.data.resultat.length >= 0)
            console.log("OK - Llistat contactes");
        else
            throw new Error();
    } catch (error) {
        console.log("ERROR - Llistat contactes");
    }
};
```

En la variable `resposta` on obtenim la resposta disposem tant del codi d'estat retornat (propietat `status`) com de l'objecte retornat en la petició (propietat `data`). En el nostre cas tindríem `resposta.data.resultat` amb el llistat d'habitacions, o `resposta.data.ok` amb el booleà amb el resultat de l'operació. També podríem consultar les capçaleres de resposta a través de la propietat `headers` (`resposta.headers`).

Anem amb una altra prova simple: imaginem que provem de buscar un contacte que no existeix. En este cas esperem un codi 400 com a resposta:

```
const contacteIncorrecte = async () => {
    try {
        const resposta = await axiosInstance.get('/contactes/000000000000000000000000
        console.log("ERROR - Contacte incorrecte");
    } catch (error) {
        if(error.response.status == 400)
            console.log("OK - Contacte incorrecte");
        else
            console.log("ERROR - Contacte incorrecte");
    }
};
```

Notar que, en este cas, l'esperable és que es produïsca una excepció, perquè s'envia un codi d'estat d'error (400). Si no és així, la prova falla. I, en cas de rebre un error, també fallarà si el codi d'error que es rep (`error.response.status`) no és l'esperat 400.

2.3. Execució de proves

Per a executar les proves que anem definint podem definir una funció addicional que les aglutine totes i invocar-la des del programa principal de `tests.js` :

```
// Funció principal d'execució de proves
const executarProves = async() => {
    await obtenirContactes();
    await contacteIncorrecte();
    ...
}

// Main
executarProves();
```

Els missatges d'ERROR i OK que posem en cada prova són arbitraris, podem mostrar la informació que considerem. Però és recomanable que es puga veure còmodament en la consola d'eixida quines proves han eixit bé i quins no:

```
OK - Llistat contactes
OK - Contacte incorrecte
...
```

2.4. Guardar dades entre proves

Igual que hem vist en el cas de *Thunder Client*, en alguns casos ens pot interessar guardar-nos una part del resultat d'una prova per a utilitzar-lo en proves successives. Per exemple, l'*id* d'un contacte inserit per a poder-lo després buscar, modificar o esborrar. O el *token* de validació d'un usuari per a poder llançar proves que requerisquen autenticació.

La següent prova es guarda el *token* de l'usuari autenticat en un servei de *login*. Observeu com enviem el login i password de l'usuari en la petició, i com recollim i guardem el token en una variable que retornem.

```
const autenticarUsuari = async () => {
    try {
        const resposta = await axiosInstance.post('/auth/login', {
            login: 'nacho',     // Reemplaçar amb el nom d'usuari real
            password: '1234567' // Reemplaçar amb la contrasenya real
        });
        if (resposta.estatus == 200)
        {
            console.log("OK - Login");
            return resposta.data.resultat; // Retornem el token del resultat
        }
        else
        throw new Error();
    } catch (error) {
        console.log("Error - Login");
        return null;
    }
};
```

Podem definir una funció auxiliar que ens ajude a guardar-nos el token en la capçalera corresponent si és correcte, o a esborrar-ho d'aquesta capçalera si no ho és:

```
// Configuració del token d'autenticació per a les següents sol·licituds
const setToken = (token) => {
 if (token) {
    axiosInstance.defaults.headers.common['authorization'] = `Bearer ${token}`;
 } else {
    delete axiosInstance.defaults.headers.common['authorization'];
 }
};
```

Ara utilitzarem el token retornat per esta prova per a inserir un contacte, enviant el token d'autorització en la capçalera corresponent.

```
const nouContacte = async (token) => {
    // Usem la funció anterior per a guardar el token
    setToken(token);
    const contacte1 = {
        nom: "Axios",
        edat: 20,
        telefon: "688888888"
    };

    try {
        const resposta = await axiosInstance.post('/contactes', contacte1);
        if(resposta.status == 200)
        {
            console.log("OK - Nou contacte");
            return resposta.data.resultat._id;
        }
        else
            throw new Error();
    } catch(error) {
        console.log("ERROR - Nou contacte");
        return -1;
    }
}
```

En este cas retornem *-1* com a identificador del contacte. Podríem passar este *id* com a paràmetre a les proves que ho necessiten, com per exemple la fitxa del contacte que acabem d'inserir:

```
const fitxaContacte = async (id) => {
    try {
        const resposta = await axiosInstance.get(`/contactes/${id}`);
        if (resposta.status == 200 && resposta.data.resultat)
            console.log("OK - Fitxa contacte");
        else
            throw new Error();
    } catch (error) {
        console.log("ERROR - Fitxa contacte");
    }
};
```

Afegim ara estes proves a la funció principal:

```
const executarProves = async() => {
    await obtenirContactes();
    await contacteIncorrecte();
    let token = await autenticarUsuari();
    let id = await nouContacte(token);
    await fitxaContacte(id);
}
```