Basis apps

Realiseren

hoofdstuk

4

Consumeren van een API

Algemene informatie

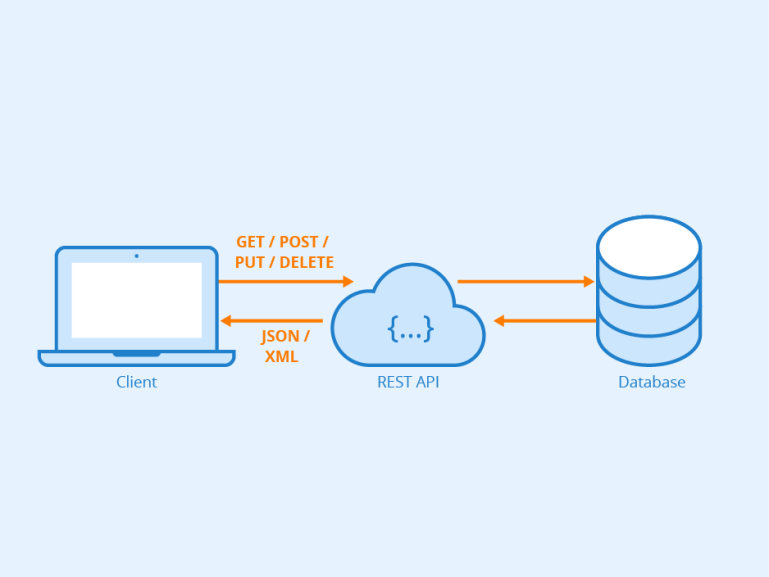
Onderwerp	Consumeren van een API
Leerdoel(en)	1. De student kan het concept API benoemen 2. De student kan een externe API consumeren (uitlezen) 3. De student realiseert data verwerking binnen een MVC-architectuur
Vereiste voorkennis	1. De student heeft uitgebreide voorkennis van MVC 2. De student heeft uitgebreide voorkennis van C#
Kwalificatiedossier	☐ B1-K1-W1: Plant werkzaamheden en bewaakt de voortgang ☐ B1-K1-W2: Ontwerpt software ☒ B1-K1-W3: Realiseert (onderdelen van) software ☐ B1-K1-W4: Test software ☐ B1-K1-W5: Doet verbetervoorstellen voor de software ☐ B1-K2-W1: Voert overleg ☐ B1-K2-W2: Presenteert het opgeleverde werk ☐ B1-K2-W3: Reflecteert op het werk

Inhoudsopgave

Introductie

Een van de grote voordelen van een mobile app is het mobiel kunnen meedragen van een toegang tot het internet. Dit betekend dat je, waar je ook bent, altijd de meest recente data van over een bepaald onderwerp kunt laten zien.

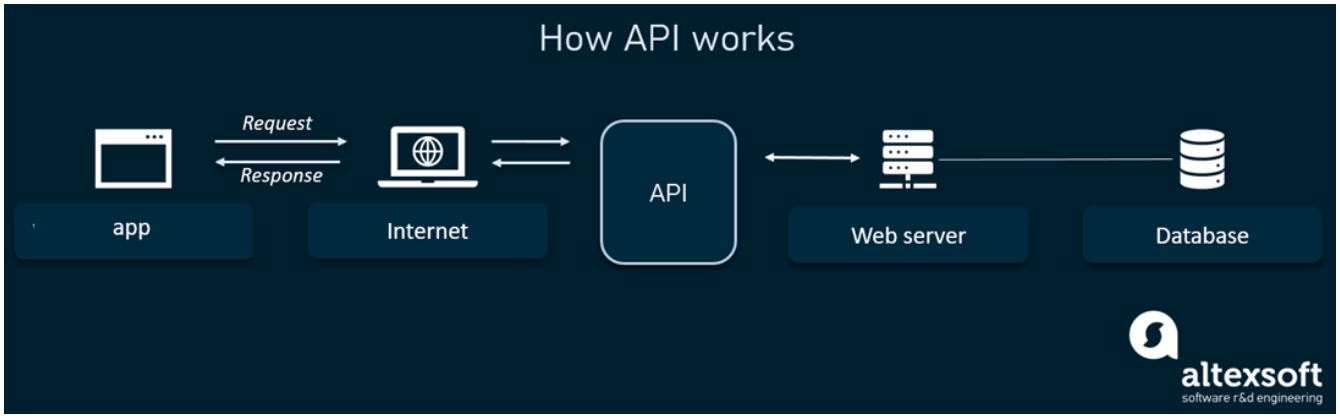
Deze data vraag je op door gebruik te maken van een zogeheten API (**A**pplication **P**rogramming **I**nterface). Een API vraag je aan de server een bepaalde set aan data, waarna deze server jou deze data zal toesturen (als alles goed gaat). Hiermee ben jij in staat om in jouw app data te gebruiken die jij zelf niet te onderhouden. Denk bijvoorbeeld aan het weerbericht, waarbij jij zelf natuurlijk niet de weersvoorspellingen hoeft te maken, maar deze kunt uitlezen van een externe partij.



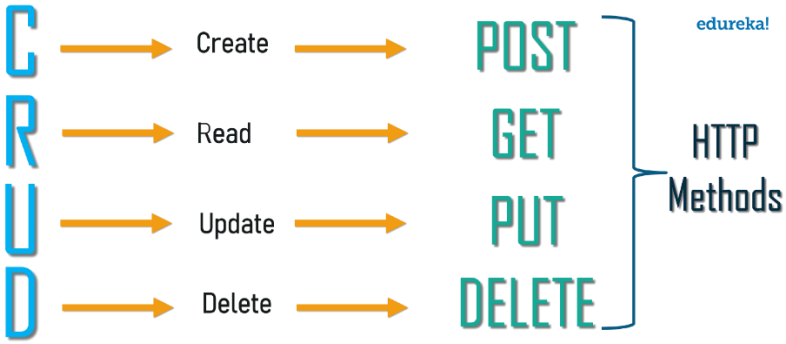
1. Introductie API

De term API staat dus voor (**A**pplication **P**rogramming **I**nterface). Een API heeft een Gebruiker (Consumer genoemd) en een Server. De gebruiker leest/gebruikt dus de API die de Server aanbiedt. Bijvoorbeeld het uitlezen van het weerbericht of de laatste AEX beurskoersen zijn mooie voorbeelden van het gebruik van een API.

Hieronder een schematische tekening van de werking van een API. Je ziet dat de app in kwestie via het internet een request maakt, waarna de server zal reageren in een bepaald formaat. Deze afspraken staan beschreven in een set aan afspraken. Wij houden één specifieke set aan afspraken aan die beschreven zijn onder de noemer **RESTful API's**.



Een correcte RESTful API request kan in verschillende 4 HTTP-methods komen. Namelijk GET, POST, PUT en DELETE. Ieder soort HTTP-method heeft een aparte functie en uitwerking. Hieronder een diagram waarin de types per CRUD-actie worden uitgelegd.



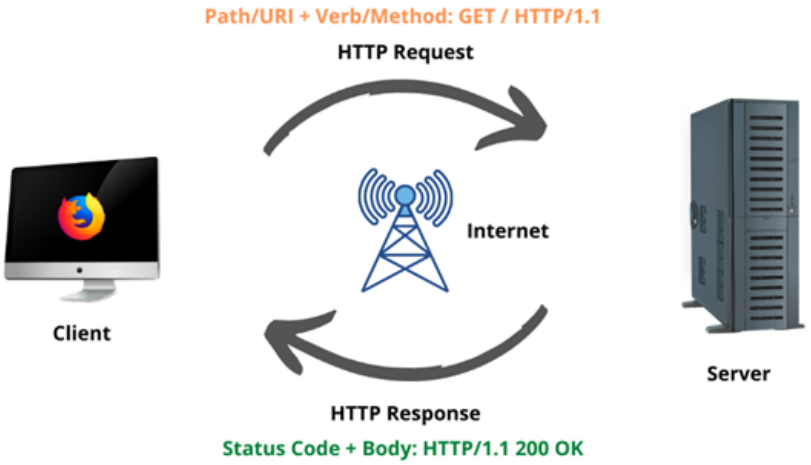
📖 Wil je meer weten over HTTP-requests? Lees de W3 schools documentatie:

https://www.w3schools.com/tags/ref_httpmethods.asp

HTTP Statuscodes

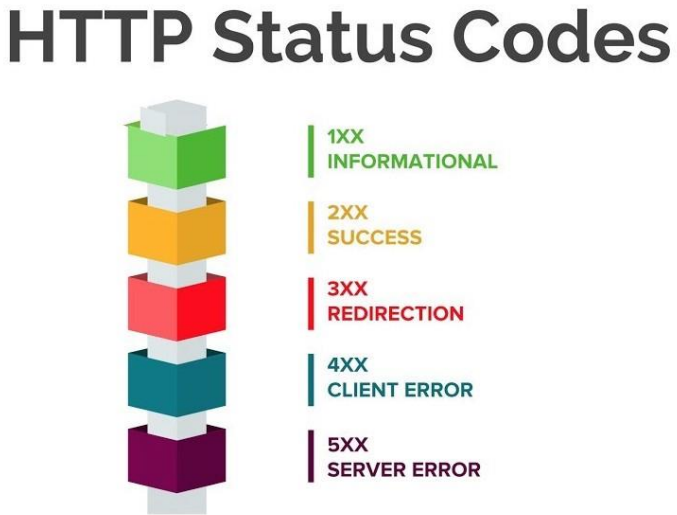
In de diagram hieronder zie je de zogeheten lifecycle van een HTTP verzoek (bezoeken van een website of het aanroepen van een API). Je ziet dat het HTTP verzoek uiteindelijk een bepaalde statuscode dient op te leveren. Een statuscode van 200 zegt dat het HTTP verzoek OK afgerond is (en dus succesvol), terwijl een statuscode van 404 betekend dat de webserver niet gevonden kan worden (404. Not Found).

De 404 error heb je waarschijnlijk vaker gezien, tijdens het bezoeken van een website die niet meer bestond.



De opbouw van statuscodes:

Statuscodes kun je per 100-tal rangschikken (zie afbeelding hieronder). Alle statuscodes binnen de 200-299 range betekenen een succes, terwijl alle codes binnen de 400 – 499 range een zogeheten Client error opleveren.



2. Implementeren van een GET request

Via de HTTP-method genaamd GET kunnen we aan een API data opvragen. Bijvoorbeeld het inlezen van het weerbericht of de huidige Bitcoin prijs zijn mooie voorbeelden waarbij een GET-request voldoende is. Het uitlezen van een bestaande API noemen we dus een API consumeren.

Hiervoor gebruiken we een door Microsoft geleverde class genaamd **HttpClient.**

 Hieronder leer je hoe je een GET Request maakt. Lees het kopje "Consume a REST based web service ".

https://learn.microsoft.com/en-us/dotnet/maui/data-cloud/rest?view=net-maui-8.0

Voorbeeldcode van een API request

```
string apiURL = "https://anapioficeandfire.com/api/houses/";

// HTTP Request aanmaken
HttpClient client = new HttpClient();

// Leeg EpisodeModel aanmaken
EpisodeModel result = new EpisodeModel();

// De URL als een URI verpakken + ID toevoegen
Uri uri = new Uri(apiURL);

// Het is tijd voor het: Versturen!!!
HttpResponseMessage respons = client.GetAsync(uri).Result;

// Checken of Request Geslaagd is
if(respons.IsSuccessStatusCode == true)
{
    // Ophalen van het EpisodeModel uit de respons
    var content = respons.Content.ReadAsStringAsync().Result;

    // Omzetten van de content string naar een Object
    result = JsonConvert.DeserializeObject<EpisodeModel>(content);
}
else
{
    // Request mislukt. Geef error.
}


// Het result returnen
return result;
```

Voorbeelduitwerking:

 https://github.com/saebuabu/MauiApp1/archive/refs/heads/master.zip

In bovenstaande link is de voorbeeldcode (zie Bitcoin Page) ingepakt voor het consumeren van een REST API. Neem deze goed door.

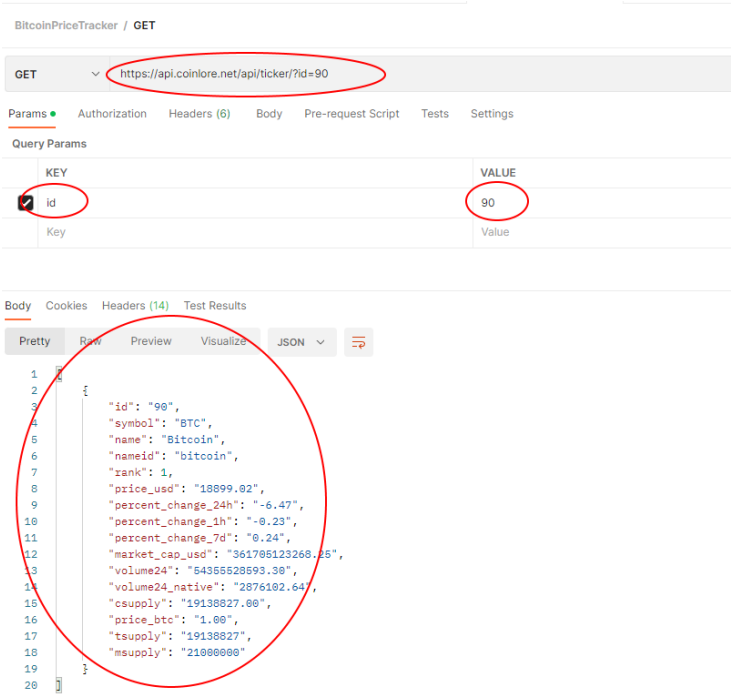
Met de voorbeelduitwerking kun je nu Oefening 4.1 maken.

 Je kunt nu **oefening 4.1** maken.

3. API-Endpoints inspecteren via Postman

Voordat je een API gaat consumeren (inlezen) is het belangrijk om eerst de *Response* waarde van de API goed te bestuderen. Zo kan een server bijvoorbeeld JSON of XML terug sturen als Response. Dit is vitaal om op voorhand te weten.

Hiervoor is gebruiken we een (gratis) software pakket genaamd POSTMAN.



Screenshot van een POSTMAN request

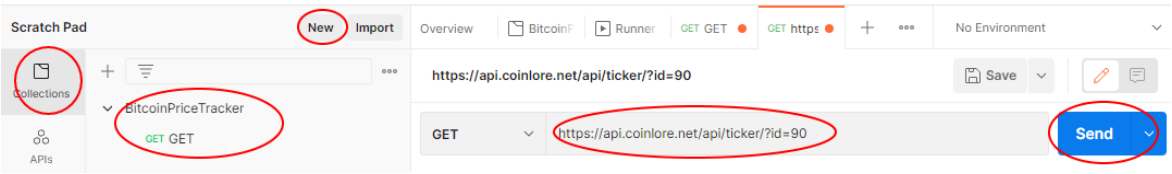
Installeren en gebruik van Postman:

→ Installeer Postman via https://www.postman.com/downloads/

→ Voer via Collections → New Collection → New Request

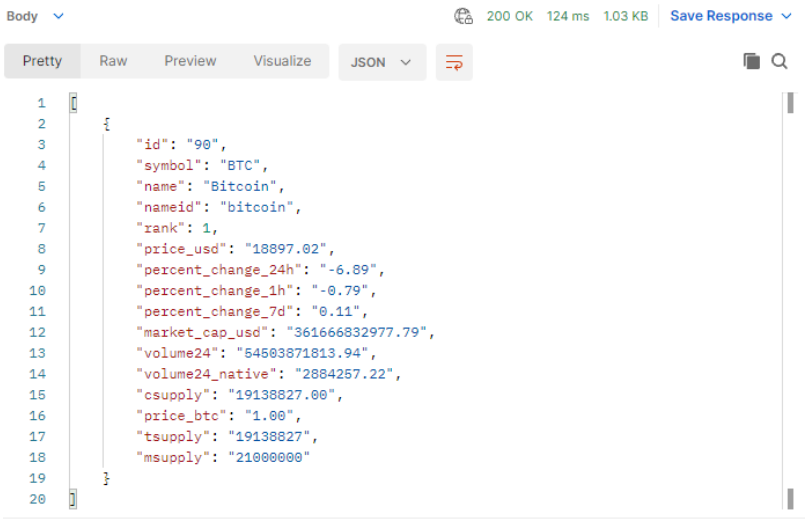
Voer deze GET URL in: https://api.coinlore.net/api/ticker/?id=90

→ Maak onderstaande screenshot na en druk op Send



Een nieuw GET request aanmaken en uitvoeren binnen Postman

→ Je krijgt onderstaande Response van de server hieronder te zien



→ Neem deze goed door. Wat zie je? Wat valt op (bijvoorbeeld: Welke velden? Welke datatypes? Waarom staan er blokhaken bovenaan en onderaan?).

4. Uitlezen van een (deel van) de API-respons via Jobjects

Het komt geregeld voor dat je een API respons terug krijgt die niet direct te converteren is naar een Model. Zie onderstaande voorbeeld API-respons:

5. Model View Controller architectuur

Je hebt eerder de MVC-architectuur gebruikt om een applicatie te bouwen waarvan de code beheersbaar, gestructureerd en makkelijk uitbreid baar is.

1. Model

Het uitlezen van een API is een taak dat typisch binnen de MVC-architectuur valt. Omdat je werkt met data, moet je eerst de data classificeren via een (of meerdere) **Modellen** (Gebruik hiervoor Postman, zie hoofdstuk 2).

Met bovenstaande Bitcoin Price voorbeeld kom je dus uit op onderstaand Model:

```
public class BitcoinPriceModel
{
    0 references | 0 changes | 0 authors, 0 changes
    public int Id { get; set; }
    1 reference | 0 changes | 0 authors, 0 changes
    public string Symbol { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public string Name { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public string NameID { get; set; }
    1 reference | 0 changes | 0 authors, 0 changes
    public double Price_usd { get; set; }
```

Je ziet hierboven dat de inhoud van een Model dus overeenkomt met de data die je API je aanlevert.

2. Controller

Daarna ga je aan de gang om een Controller te bouwen van ieder gemaakt Model. Aangezien we in deze reader alleen de API zullen consumeren (inlezen) spreken we dus alleen over de **Read()** functionaliteiten van een controller.

Belangrijk is om te bepalen of je API-response één **Model object** returned of juist meerdere **Model objecten** returned. Want in je controller moet je namelijk bepalen of je een **List<>** of een normaal **Object.**

Voorbeeld van een *ReadOne(int)* methode, waarin één specifiek Model object *gereturned* wordt:

```
public BitcoinPriceModel ReadOne(int id)
{
    BitcoinPriceModel result = new BitcoinPriceModel();
    // API code hieronder

    return result;
}
```

Een voorbeeld van een *ReadOne()* BitcoinPriceController methode

Hieronder een voorbeeld van een *ReadAll()* method binnen de Controller. Let op, deze returned dus een List:

```
public List<BitcoinPriceModel> ReadAll()
{
    List<BitcoinPriceModel> result = new List<BitcoinPriceModel>();
    // API code hieronder

    return result;
}
```

Voorbeeld uitwerking van een API-request:

```
// HTTP Request aanmaken
HttpClient client = new HttpClient();

// Leeg EpisodeModel aanmaken
EpisodeModel result = new EpisodeModel();

// De URL als een URI verpakken + ID toevoegen
Uri uri = new Uri(apiURL + id);

// Het is tijd voor het: Versturen!!!
HttpResponseMessage respons = client.GetAsync(uri).Result;

// Checken of Request Geslaagd is
if(respons.IsSuccessStatusCode == true)
{
    // Ophalen van het EpisodeModel uit de respons
    var content = respons.Content.ReadAsStringAsync().Result;

    // Omzetten van de content string naar een Object
    result = JsonConvert.DeserializeObject<EpisodeModel>(content);
}
else
{
    // Request mislukt. Geef error.
}


// Het result returnen
return result;
```

Voorbeelduitwerking (inclusief Model View Controller)

 In https://github.com/saebuabu/MauiApp1/archive/refs/heads/master.zip (zie CryptoCoinsPage) vindt je een voorbeeld uitwerking. Neem deze goed door.

 Je kunt nu **oefening 4.2** maken.