Onderzoeksrapport

MIJN JACHTVELD

De kracht van Mijn Jachtveld

Datum: 28-03-2024

School: Fontys University of Applied Sciences

Naam: Cédric Berden

Versie: 2.0

Assessor: Thijs Naus

Stakeholder: Jaap Verhofstad

Inhoudsopgave

Introductie

In dit onderzoeksrapport worden de onderzoeksvragen die van tevoren zijn opgesteld beantwoord. Waarom dit rapport wordt opgesteld is omdat het belangrijk is om goed te weten wat je gaat doen en dat het vanaf het begin goed gebeurt moet worden om een goed werkend project op te leveren. Voordat de onderzoeksvragen uitgewerkt gaan worden gaat er gekeken worden wat een API überhaupt is. Daarna worden de onderzoeksvragen beantwoorden die opgesteld zijn in het projectplan. De vragen die beantwoord gaan worden zijn:

- Welke API-technologieën worden vaak gebruikt in vergelijkbare toepassingen, en wat zijn hun sterke punten?
- Wat zijn de verschillende methoden voor het opzetten van een API?
- Hoe kan de gegevensuitwisseling tussen Mijn Jachtveld en andere applicaties geoptimaliseerd worden voor efficiëntie en nauwkeurigheid?
- Wat zijn de potentiële uitdagingen en overwegingen bij het uitbreiden van API-oproepen voor een volledig functioneel webapplicatie?
- Hoe kan de API veilig worden gemaakt voor het uitwisselen van gegevens met anderen?

De programeer taal dat er binnen dit bedrijf gebruikt wordt en waar Mijn Jachtveld mee gemaakt is is PHP. Met het onderzoek moet daar aandacht aangeven worden zodat dat de uitkomst ook voor dit project geld.

Wat is een API?

Om te beginnen is het handig om te weten wat een API is en welke functies het vervult. Daarom wordt hieronder een kleine samenvatting gegeven van wat een API is.

API staat voor application programming interface en het is een software-interface die communicatie mogelijk maakt tussen twee verschillende applicaties. Je kunt het zien dat er een bezoeker is die een antwoord vraagt aan een provider die dan antwoord teruggeeft. Het kan zowel eenrichtingsverkeer zijn waarbij de bezoeker informatie opvraagt, als tweerichtingsverkeer waarbij de provider aanvullende informatie nodig heeft voordat een antwoord wordt gegeven.

Als je API's gebruikt met een API-laag is het mogelijk om met verschillende front-end applicaties te gebruiken zonder dat de backend aangepast hoeft te worden. Zo zou je dan naast je website er makkelijk een app bij creëren. Naast dat kun je makkelijk data wisselen met een extern bedrijf wat nodig is voor het project dat we gaan maken.

Kortom, API's vormen een brug tussen verschillende softwareapplicaties en maken geavanceerde en flexibele integraties mogelijk, wat tegen woordelijk belangrijk is bij het ontwikkelen van moderne en schaalbare projecten.

Bronnen

1. Wat is een API en wat kan je ermee? (23-10-2019) Lisanne schoenmaker https://www.salesforce.com/nl/blog/wat-is-een-api/
2. What is an API? (02-06-2022) https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces

Welke API-technologieën worden vaak gebruikt in vergelijkbare toepassingen, en wat zijn hun sterke punten?

Er zijn verschillende soorten API-technieken die bij vergelijkbare applicaties worden gebruikt, de 2 die het meest voorkomen zijn REST en SOAP API's waarvan REST de laatste jaren het meest gebruikt wordt. Naast die 2 ga ik ook GraphQL onderzoeken omdat het een opkomende API-techniek is. In de conclusie ga een keuze maken welke techniek er gebruikt gaat worden en waarom ik ervoor heb gekozen.

REST

Representational State Transfer ofwel REST is een API-specificatie. REST is een ontwerpprincipe, dit betekent dat er geen officiële standaard is voor REST full-web-API's. De data bij REST wordt verstuurd via een link en de data die teruggestuurd wordt gaat via een JSON, XML of een tekstindeling. Met een REST API verbreekt de verbinding nadat de API heeft gereageerd op het verzoek van de client.

Data komt binnen via JSON maar zou ook via een XML of een tekstbestand binnen komen. Waarom REST vooral gebruik maakt van JSON is omdat het makkelijk is en het vrij licht is om data te transporteren.
REST API's zijn niet automatisch beveiligd maar kan wel beveiligd worden door externe.
Werkt met HTTP en HTTPS.
Is gedesigned voor mobile apparaten in gedachten en hoeft minder gestructureerd te worden.
REST heeft een architectuur stijl waardoor er meer flexibiliteit is met de designs.
REST maakt gebruik van GET, POST, PUT, DELETE.

SOAP

Simple Object Access Protocol ofwel SOAP is een andere API-specificatie. Het maakt gebruik van XML om de data door te sturen en het krijgt de berichten vanuit HTTP of SMTP. Vergeleken met een REST API is SOAP een protocol in plaats van een ontwerpprincipe.

De data komen binnen via XML. Hoewel het zwaarder is dan JSON vanwege de tags, is het nog steeds veel gebruikt, vooral in integraties waar een rigide structuur nodig is.
SOAP API's zijn standaard beveiligd om de berichten te beveiligen.
Werkt met HTTP, HTTPS, SMTP en XMPP.
Is gedesigned voor grootte applicaties en moet goed gestructureerd zijn.
SOAP is een protocol waardoor een set van regels zijn voor de berichten in de web-service.
SOAP werkt met WSDL (XML-taal) omdat het gebruik maakt van XML.

GraphQL

Als we naar de toekomst kijken van API's komen er een aantal ontwikkelingen aan waar we rekening mee kunnen houden. Een van de ontwikkeling is dat GraphQL een opkomst begint te maken [5]. GraphQL is ontwikkeld als een reactie op REST API's. Het idee erachter is dat er nauwkeuriger syntaxis uitgevoerd kunnen worden. Het wordt ook vaker gebruikt als er veel en ingewikkelde data getransporteerd moeten worden.

De data komen binnen in een JSON via PersistGraphQL
GraphQL is niet standaard beveiligd.
Werkt met HTTP en HTTPS.
Is gedesigned om zodat de client zelf kan aangeven welke dat hij wilt ontvangen
GraphQL wordt beschouwd als een flexibelere benadering dan REST.
In GraphQL is er maar een eindpunt, meestal een HTTP POST waar alle query's naar gestuurd worden.

Frameworks

Om een API te maken in PHP is het aan te raden om een framework te gebruiken. De voordelen om een framwork te gebruiken zijn dat losse koppeling, schaalbaarheid, interoperabiliteit, prestaties, flexibiliteit, eenvoudig onderhoud, taalonafhankelijkheid en kan in sommige gevallen meehelpen met de beveiliging van de API.

Laravel

Laravel is een open-source fullstack PHP framework en is de populairste. Het volgt de model-view-controller-architectuur (MVC) waardoor de applicatie in 3 delen wordt afgescheiden.

Voordelen

- Laravel heeft een groot en actieve community hierdoor is er veel te vinden online.
- Laravel is vanaf PHP 7 beschikbaar, dit zou een voordeel kunnen hebben als de applicatie PHP 7 gebruikt.
- Laravel heeft een goed gedesigned architectuur dat bij de moderne applicaties past.
- Laravel is redelijk makkelijk te leren.
- Ingebouwde test support.
- Laravel heeft uitgebreide beveiligingsfuncties

Nadelen

- Laravel kan complexer zijn dan anderen frameworks, dit komt omdat ze veel features en libaries hebben.
- Omdat Laravel zo uitgebreid is kan het in sommige gevallen langzamer zijn.

Slim

Slim is een open-source PHP micro-framework. De bedoeling van Slim is dat het makkelijk te gebruiken is en snel is voor de eindgebruiker. Slim maakt ook gebruik van de MVC-architectuur.

Voordelen

- Slim is geen complex framework en daarom is het niet heel lastig om te leren.
- Slim staat erom bekend dat het een kleiner framework en is daarom sneller dan andere frameworks.
- Slim heeft een ingebouwde test support

Nadelen

- Slim heeft geen uitgebreide communicatie.
- Slim is niet geschikt voor grote applicaties.

Lumen

Lumen is een opensource micro-framework voor het maken van microservices en API's. Lumen is gecreëerd door dezelfde persoon en bedoeld om een alternatief te geven om een lichtere framework te zijn.

Voordelen

- Lumen staat er ook om bekend dat het een kleiner framework is waardoor het licht en snel is.
- Lumen is vooral gefocust op API's dus daar licht de specialiteit van het framework.

Nadelen

- Lumen is niet geschikt voor grote applicaties.
- Lumen heeft gelimiteerde documentatie.
- Lumen kan lastiger te leren zijn als er geen ervaring is in Laravel

Symfony

Symfony is een populaire open sourced fullstack PHP framework voor het maken van webapplicaties. Het maakt ook gebruik van de MVC-architectuur wat de applicatie in 3 delen opsplits.

Voordelen

- Symfony is een uitgebreid framework.
- Symfony heeft ingebouwde test support.
- Symfony heeft een uitgebreide documentatie.
- Symfony heeft uitgebreide beveiligingsfuncties.

Nadelen

- Symfony is lastiger om te leren.
- Symfony ingebouwde functies niet echt flexibiliteit en er kan daardoor minder controle zijn over de code.
- Symfony is zwaarder en kan daarom langzamer zijn dan andere frameworks.

Conclusie

De conclusie die ik in het eerste deel van het onderzoek haal is dat REST API de beste API-techniek is voor dit project. De reden om voor REST te kiezen is gebaseerd op de eenvoudige implementatie, beschikbaarheid van PHP-architectuur voor REST, en de brede acceptatie binnen de PHP-gemeenschap. SOAP is vooral vrij ingewikkeld om toe te passen in een project en wordt bij dit soort applicaties bijna niet gebruikt. De techniek GraphQL wordt tegenwoordig vaker gebruikt als een alternatief van REST en daar zat misschien nog een twijfel in.

In het tweede deel van het onderzoek hebben we de frameworks besproken die er gebruikt kunnen worden om de API's mee te kunnen maken. Er zijn veel verschillende soorten en we hebben er 4 uitgekozen om verder te onderzoeken op basis van aanbevelingen online. Degene die er het meest bij mij uit zijn gekomen zijn Laravel alleen het nadeel daarvan is dat het een volledig framework is. Dit is omdat de rest van het project gemaakt is in gewoon PHP en alleen de API een framework moet hebben en Laravel te groot is het alleen te gebruiken voor het maken van API's. Daarnaast is het vrij lastig om Laravel te leren en dat het framework gebruikt moet worden door anderen binnen het bedrijf is het beter voor een ander framework te gaan. Daarom gaan we op basis van het project waar ingewerkt moet gaan worden de keuzen naar Slim. Dit is dan ook voornamelijk omdat Slim gemaakt is voor API's te bouwen en daarom een kleiner framework is. De enige twijfel die er was om Lumen te gebruiken alleen omdat lumen een gebaseerd is op Laravel en als er nog nooit met Laravel gewerkt is is het lastiger om te leren, en omdat er binnen het bedrijf niemand ervaring heeft met Laravel is het beter om Slim te gebruiken vanwege zijn eenvoudigheid.

Prototype

Hieronder laat ik zien hoe een REST API gemaakt wordt. Ik laat een paar voorbeelden zien van vanlila PHP, Lumen en Slim om te laten zien wat de verschillen zijn. Ik heb gekozen om een simpele API te maken als voorbeeld waar je de tekst hello world mee gestuurd wordt.

Vanilla PHP

Om een API te maken in vanilla PHP wordt er in het begin met een if statement gekeken of er in de URL een gelijk staat aan hello-world. Het andere wat er in de if statement wordt gecontroleerd is of er een GET, POST, PUTH, DELETE, extra. Als de is statement correct is stelt de HTTP-koptekst in om aan te geven dat de respons een eenvoudige tekst is. Daarna wordt de tekst “Hello, World” teruggestuurd.

```php
<?php
if ($_SERVER['REQUEST_URI'] === '/hello-world' && $_SERVER['REQUEST_METHOD'] === 'GET') {
    header('Content-Type: text/plain');
    echo "Hello, World";
}
```

Lumen

In het begin wordt er een nieuw Lumen-applicatie-object gemaakt. Dit object wordt gebruikt om routes te definiëren en de applicatie uit te voeren. Daarna wordt er via het object gekeken of er een get wordt aangeroepen met de route hello-world. De functie is een anonieme functie die wordt uitgevoerd wanneer het verzoek binnenkomt op de '/hello-world'-route. Het accepteert een $request object, dat informatie bevat over het inkomende HTTP-verzoek, en een $response object, dat wordt gebruikt om een HTTP-antwoord op te bouwen. In dit geval schrijft de functie eenvoudigweg "Hello, World" naar de body van het antwoord.

Als de functie is aangeroepen wordt er "Hello, World" naar de body gescreven van het HTTP-antwoord. Dan wordt er met een return het antwoord terug gestuurd. And op het laatst staat er app->run() en dit start de Lumen applicatie.

```php
<?php
$app = new Application();

$app->get('/hello-world', function (Request $request, Response $response) {
    $response->getBody()->write("Hello, World");
    return $response;
});

$app->run();
```

Slim

Het maken van een API in Slim ziet er bijna hetzelfde uit als met Lumen, Het enige verschil is het aanmaken van een Slim-applicatie-object.

```php
$app = AppFactory::create();

$app->get('/hello-world', function (Request $request, Response $response) {
    $response->getBody()->write("Hello, World");
    return $response;
});

$app->run();
```

Dit is het minimale om een API te maken en dat is de reden dat het nu veel op elkaar lijkt maar als het project zich verder ontwikkeld zullen de verschillen groter worden.

Bronnen

1. What is an API? (02-06-2022) https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces
2. What is REST API (g.d) https://www.uptrends.nl/wat-is/rest-api
3. Difference between REST API and SOAP API (22-09-2024) https://www.geeksforgeeks.org/difference-between-rest-api-and-soap-api/
4. SOAP vs. REST APIs: The Key Differences Explained for Beginners (04-05-2023) Anna Fitzgerald https://blog.hubspot.com/website/rest-vs-soap
5. The Future of API Technology: Trends to Watch (08-12-2023) David Heath https://www.linkedin.com/pulse/future-api-technology-trends-watch-david-heath-ixjnc
6. GraphQL vs. REST APIs (12-12-2023) Estaban Herrera https://blog.logrocket.com/graphql-vs-rest-api-why-you-shouldnt-use-graphql/
7. GraphQL vs. REST: What You Didn't Know (g.d) Adil Sikandar https://www.mobilelive.ca/blog/graphql-vs-rest-what-you-didnt-know
8. 5 Popular PHP REST API Frameworks (01-04-2023) Roniee https://medium.com/@akashjha9041/5-popular-php-rest-api-frameworks-66f67b8d7e53
9. Top 5 PHP REST API Frameworks (03-01-2023) Preet Kaur https://www.moesif.com/blog/api-product-management/api-analytics/Top-5-PHP-REST-API-Frameworks/
10. Best PHP Frameworks for Web Development in 2023 (12-07-2023) Inshal Ali https://www.cloudways.com/blog/best-php-frameworks/
11. How does XML work in SOAP? (03-01-2024) https://www.linkedin.com/advice/0/how-does-xml-work-soap-skills-programming-izohc
12. SOAP vs REST vs JSON - a 2023 comparison (17-10-2022) Anna Monus https://raygun.com/blog/soap-vs-rest-vs-json/

Wat zijn de verschillende methoden voor het opzetten van een API?

Het opzetten van API's zijn verschillende methodes voor. In het vorige onderdeel hebben we besloten om REST API's te gaan gebruiken dus daar zullen nu verder op gefocust worden. Een API aanmaken is niet heel moeilijk maar het op een goeie manier opzetten is het meest ingewikkeld. Er zijn verschillende onderdelen om een API/API laag op te zetten dus we gaan alle onderdelen die belangrijk zijn hier behandelen.

Met REST API heb je een aantal methodes die je kunt gebruiken. Hiermee kun je aangeven wat de API moet doen wanneer je hem aanroep.

GET: Wordt gebruikt om een representatie van een bron op te halen.

POST:Wordt gebruikt om nieuwe bronnen en sub bronnen te creëren.

PUT: Wordt gebruikt om bestaande bronnen bij te werken.

PATCH: Wordt gebruikt om bestaande bronnen bij te werken.

DELETE: Wordt gebruikt om bestaande bronnen te verwijderen.

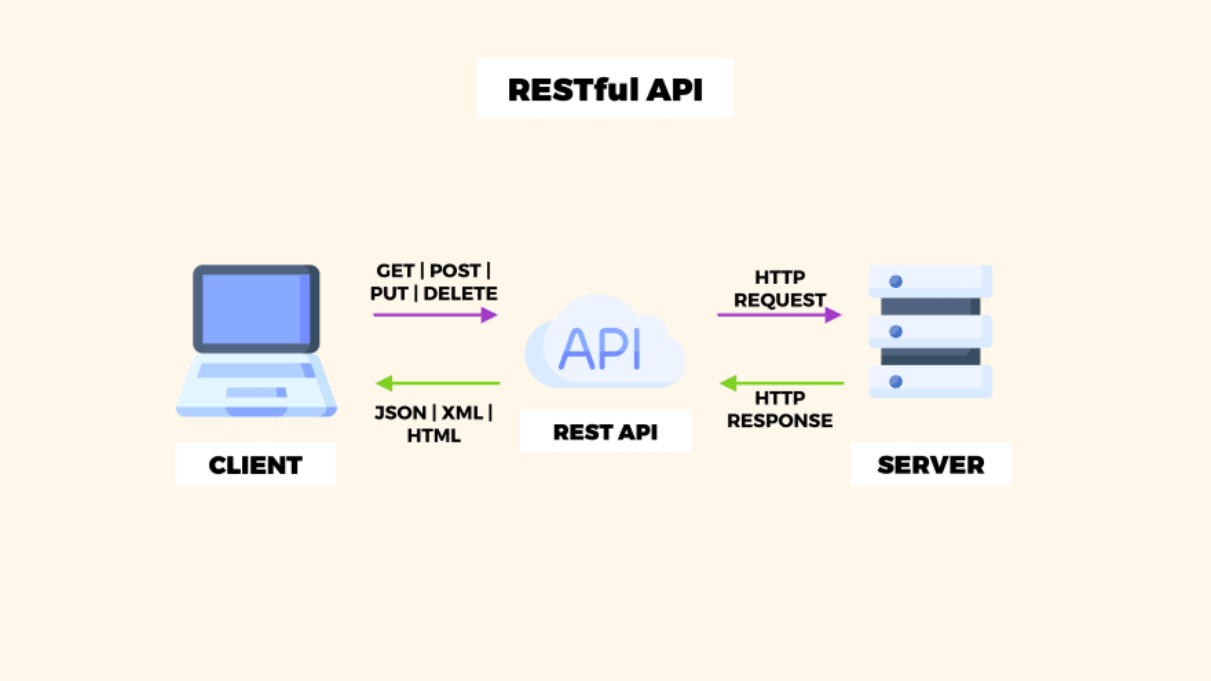
Er zijn verschillende API's maar over het algemeen komt het op de komende 3 neer.:

1. **Privé API**
   Dit zijn API's die je intern gebruikt in je eigen project. De API's worden niet buiten een bedrijf gedeeld met derde.
2. **Publieke API**
   Ook wel open API genoemd worden gebruikt om met andere te kunnen communiceren. Waar een privé API gebruikt wordt om binnen een bedrijf te gebruiken worden publieke API gebruikt
3. **Partner API**
   Een partner API lijkt veel op een publieke API alleen heb je rechten nodig om met deze API kunnen communiceren. De rechten worden meestal verleend aan derde of aan derde die betalen om de API te mogen gebruiken.

Als we kijken naar het project van Mijn Jachtveld en wat voor soort API we nodig hebben dan valt de privé API-gelijk af omdat de API's door derde gebruikt moet gaan worden. Voor het project willen we niet dat iedereen de API's kan gebruiken dus dan is de enige die nog overblijft de partner API.

Client-server

Om een API goed te gebruiken is het handig om te weten wat de methode van REST inhoudt. Hieronder staat een afbeelding waarop wordt weergeven hoe een REST API werkt.



Figuur 1 https://www.incodom.kr/REST

De REST API en de foto houdt in dat servers en clients alleen bron-URL's(specifiek webadres) mogen kennen, en dat is alles. Makkelijker gezegd de client applicatie en de server applicatie moeten apart kunnen werken zonder van elkaar af te hangen. Alles kan onafhankelijk met elkaar worden aangepast zolang de interface ertussen in ongewijzigd blijft.

Dataformaten

Met het communiceren van een API is het essentieel om een gestandaardiseerd formaat te gebruiken voor de uitwisseling van gegevens. Hier zijn enkele dataformaten die met een REST API gebruikt kunnen worden:

JSON (JavaScript Object Notation):

JSON is een van de meest populaire dataformaten voor het structureren van gegevens in een leesbaar formaat. Het is licht van gewicht, gemakkelijk te begrijpen en te schrijven, en wordt vaak gebruikt in webontwikkeling.

JSON

```
{
  "naam": "John Doe",
  "leeftijd": 30,
  “adres”: {
  “straat”: “Sample Street”,
  “stad”: “Sample City”
  }
}
```

XML (eXtensible Markup Language):

XML is een opmaaktaal die wordt gebruikt voor het structureren van gegevens in een hiërarchische indeling.

xml

```
<persoon>
  <naam>John Doe</naam>
  <adres>
  <straat>Sample Street</straat>
  <stad>Sample City</stad>
</adres>
</persoon>
```

Bij het kiezen van een dataformaat moet rekening worden gehouden met de vereisten van het project, zoals leesbaarheid, efficiëntie, en de compatibiliteit met de technologieën die worden gebruikt. JSON blijft echter een veelgebruikte keuze vanwege zijn eenvoud en brede ondersteuning in veel programmeertalen en frameworks.

Conclusie

De conclusie die ik uit dit onderdeel haal is dat een API (REST API) op verschillende manieren gemaakt kan worden. We hebben onderzocht wat voor verschillende methodes REST heeft. Daarna is er gekeken welke verschillende API's er zijn en hebben we de beste voor dit project uitgekozen en is er afgesloten om te kijken hoe een REST API werkt. Naast dat is er ook onderzocht welke dataformaten er zijn voor REST.

Bronnen

1. Best Practices in API Design (g.d) https://swagger.io/resources/articles/best-practices-in-api-design/
2. What Is a REST API? (20-11-2020) https://www.akana.com/blog/what-is-rest-api#:~:text=A%20REST%20API%20is%20an,interaction%20with%20RESTful%20web%20services.
3. REST Architectural Constraints (06-11-2023) Lokesh Gupta https://restfulapi.net/rest-architectural-constraints/
4. API bouwen? Zó maak je een API-koppeling (g.d) https://www.lucidchart.com/blog/nl/api-bouwen
5. API-koppeling (g.d) https://linku.nl/technieken/api-koppeling/#:~:text=Soorten%20API's,De%20publieke%20API.
6. REST API: Key Concepts, Best Practices, and Benefits (19-11-2024) https://www.altexsoft.com/blog/rest-api-design/
7. How to Design an API – Application Programming Interface Best Practices (29-06-2022) Sophia Iroegbu https://www.freecodecamp.org/news/design-an-api-application-program-interface/
8. REST API and Data Formats (27-09-2023) Alrazak https://medium.com/@alrazak/rest-api-and-data-formats-cd842e81d914

Hoe kan de gegevensuitwisseling tussen Mijn Jachtveld en andere applicaties geoptimaliseerd worden voor efficiëntie en nauwkeurigheid?

Om de gegevensuitwisseling tussen Mijn Jachtveld en andere applicaties te optimaliseren, is het cruciaal om een efficiënte en nauwkeurige API te hebben. Trage en onnauwkeurige API's kunnen de gebruikerservaring negatief beïnvloeden, de schaalbaarheid belemmeren en de algehele applicatieprestaties vertragen.

API Performance Metrics

API Performance Metrics is essentieel om de prestaties van de API te meten en te verbeteren. Door statistieken zoals latentie, netwerkefficiëntie, CPU en geheugengebruik te volgen, kunnen knelpunten worden geïdentificeerd en proactieve maatregelen worden genomen om de prestaties te optimaliseren.

Latency:

Latentie omvat de tijd die een client nodig heeft om een verzoek te verzenden en het antwoord te ontvangen. Het omvat ook de tijd die de server nodig heeft om het verzoek te verwerken en het antwoord terug te sturen.

Netwerkefficiëntie:

Bij het optimaliseren van netwerkefficiëntie is het belangrijk om rekening te houden met de hoeveelheid gegevens die tussen de client en de server wordt overgedragen. Dit kan worden bereikt door de payload groottes te optimaliseren om het gebruik van bandbreedte te minimaliseren. Het hergebruiken van bestaande verbindingen is ook een goede praktijk om de overhead van het opzetten van nieuwe verbindingen te verminderen.

CPU/Memory (RAM):

Identificatie van prestatieknelpunten: Monitor de CPU en het geheugen om mogelijke knelpunten te identificeren en zorg voor optimale serverprestaties voor API-verzoeken.

Proactieve probleemdetectie: Houd pieken of hoge gebruiksniveaus in CPU en geheugen in de gaten om problemen proactief te detecteren en aan te pakken voordat ze de prestaties van de API beïnvloeden.

Resourcegebruik voor schaalvergroting: Analyseer trends in CPU- en geheugengebruik om beslissingen te nemen over het toewijzen van bronnen en schaalstrategieën om efficiënte API-bewerkingen te behouden.

Reactietijd:

De reactietijd van de API is van cruciaal belang voor de effectiviteit ervan. Een snelle reactie is essentieel om een goede gebruikerservaring te bieden en vertragingen in de applicatie te voorkomen. Door de reactietijd te monitoren, kunnen mogelijke codeproblemen of uitdagingen in de serverprestaties worden geïdentificeerd en aangepakt.

Beschikbaarheid van de API:

Beschikbaarheid verwijst naar de mate waarin de API operationeel en toegankelijk is voor gebruikers. Het wordt vaak uitgedrukt als een percentage van de tijd dat de API beschikbaar is in vergelijking met de totale tijd.

Een hoge beschikbaarheid is cruciaal om ervoor te zorgen dat gebruikers te allen tijde toegang hebben tot de functionaliteiten van de API. On beschikbaarheid kan leiden tot verstoringen in dienstverlening en een negatieve impact hebben op de gebruikerservaring.

Fouten:

Fouten in de context van API-prestaties verwijzen naar onverwachte problemen of foutieve verzoeken die de API ontvangt of genereert. Fouten kunnen variëren van eenvoudige syntaxfouten tot complexere operationele problemen.

Het monitoren van fouten helpt bij het identificeren van potentiële zwakke punten in de implementatie van de API. Een hoog aantal fouten kan wijzen op problemen met de code, de infrastructuur of de gegevensverwerking, en het is van vitaal belang om deze snel te detecteren en te corrigeren.

Aantal verzoeken per seconde:

Het aantal verzoeken per seconde geeft aan hoe vaak gebruikers de API aanroepen gedurende een specifieke tijdsperiode. Het meten van de belasting op de API in termen van verzoeken per seconde biedt inzicht in de vraag en het gebruik.

Het aantal verzoeken per seconde is van belang voor het schatten van de capaciteitsbehoeften van de API. Het helpt bij het plannen van resources en schaalbaarheid. Bij piekbelastingen kan het aantal verzoeken per seconde snel stijgen, en het is essentieel om ervoor te zorgen dat de infrastructuur deze belasting aankan zonder prestatieproblemen.

Monitoring van de beschikbaarheid, fouten en verzoeken:

Gebruik monitoringtools om de beschikbaarheid van de API continu te meten. Houd een logboek bij van fouten en analyseer deze om patronen en oorzaken te identificeren. Gebruik ook tools om het aantal verzoeken per seconde te meten en trends in gebruik te volgen.

Stel waarschuwingen in voor ongewone patronen, zoals een plotselinge toename van fouten of verzoeken. Een snel reactie- en herstelmechanisme is essentieel om downtime te minimaliseren en de gebruikerservaring te behouden.

Optimaliseer efficiëntie en nauwkeurigheid

Nu dat we weten hoe we kunnen zien of er iets minder werkt met een API willen we nu ook weten hoe we een API dan ook efficiënter te maken. Om de efficiëntie en nauwkeurigheid te verbeteren heb je een paar dingen die je kunt doen.

Caching mechanisms:

Met een caching mechanisme kan er veel gebruikte data opgeslagen worden waardoor het niet de hele tijd opnieuw opgehaald hoeft te worden. Afhankelijk van het project kan het op verschillende niveaus in een webapplicatie worden geïmplementeerd.

Optimaliseer de database query's:

Hiermee zorg je ervoor dat er zo min mogelijk tijd in zit om data uit de database te halen. Dit zorgt ervoor dat de API-prestatie vooruitgaat omdat ie niet meer zolang hoeft te wanten voordat er iets terug gestuurd kan worden.

Het kan ook helpen om de verbinding met de database te optimaliseren voor een betere en snellere verbinding.

Minimizing unnecessary data transfer:

Door zo min mogelijk data te gebruiken bij een API-call wordt de API sneller. Het moet wel data zijn die je niet gebruikt en daarmee onnodige dat is. De data worden niet gebruikt dus om een manier te vinden om de data niet mee te sturen wordt de API geoptimaliseerd.

Server

Door een betere server te hebben waar de API-laag op zit gaat de API's sneller worden. Het verschild per applicatie hoe goed de server moet zijn, bij een kleinere applicatie hoeft de server niet zo goed te zijn dan bij een andere applicatie. Het is iets waar je weinig controle op hebt, het kan ook zijn dat de server goed genoeg is voor de applicatie.

Conclusie

De conclusie die ik hier uit trek is dat er een aantal manieren zijn om een API beter te laten werken. Het is namelijk belangrijk dat de ervaring van de gebruiker goed is en dat de API vloeiend werkt. Het zijn voornamelijk onderdelen waar je bij het opzetten van een API goed aan denkt.

Wanneer de applicatie uitgewerkt wordt is het goed om naar de API-performance metrics te kijken, omdat het goed is om te weten of de API goed werkt en als dat niet zo is kan er gezien worden waar dat aan ligt. Hierdoor zien we niks over het hoofd en kunnen we sneller de API op een goeie manier aan te passen.

Bronnen

1. API Performance Optimization: A Complete Guide to Metrics, Terminology, and Optimization Techniques (23-04-2023) Prudvi Tarugu https://medium.com/@prudvi.tarugu/api-performance-optimization-a-complete-guide-to-metrics-terminology-and-optimization-techniques-26f92d0fbfb2
2. 13 API Metrics That Every Platform Team Should be Tracking (26-01-2022) Derric Gilling https://www.moesif.com/blog/technical/api-metrics/API-Metrics-That-Every-Platform-Team-Should-be-Tracking/
3. 5 Caching Mechanisms to Speed Up Your Application (15-09-2022) Pragati Verma https://hackernoon.com/5-caching-mechanisms-to-speed-up-your-application
4. 8 Tips for Optimizing an API (19-09-2023) Terence Bennett https://blog.dreamfactory.com/8-tips-for-optimizing-an-api/
5. The Power of AI Detector API: Enhancing Efficiency and Accuracy (06-11-2023) https://aicontentfy.com/en/blog/power-of-ai-detector-api-enhancing-efficiency-and-accuracy
6. Top API Metrics for Different Teams That You Should Monitor (16-08-2022) Janani https://www.atatus.com/blog/api-metrics/
7. The Most Important Metrics for API Performance Monitoring (01-10-2023) https://apitoolkit.io/blog/the-most-important-metric/

Wat zijn de potentiële uitdagingen en overwegingen bij het uitbreiden van API-oproepen voor een volledig functioneel webapplicatie?

Het uitbreiden van API's voor een volledig functionele webapplicatie brengt verschillende uitdagingen en overwegingen met zich mee die van cruciaal belang zijn voor een succesvolle implementatie. In dit hoofdstuk worden deze aspecten besproken om een dieper inzicht te bieden in de complexiteit van het uitbreidingsproces.

Schaalbaarheid

Een van de voornaamste overwegingen bij het uitbreiden naar API's is de schaalbaarheid. Hoe goed kunnen de API's omgaan met een toenemend aantal gebruikers? Het is essentieel om maatregelen te implementeren die de prestaties handhaven en schalen naar de groeiende vraag.

Beveiliging en Gegevensbescherming

Uitbreiden naar API's verhogen het belang van beveiliging en gegevensbescherming. Het is noodzakelijk om potentiële beveiligingsrisico's te identificeren en mitigatiestrategieën te implementeren. Maatregelen voor de bescherming van gevoelige gegevens tijdens de overdracht en verwerking via API's moeten worden geoptimaliseerd.

Gegevensconsistentie en Integriteit

Bij het uitvoeren van meerdere API's is het handhaven van gegevensconsistentie een cruciale overweging. Hoe kan inconsistentie in de database worden voorkomen? Er moet worden nagedacht over transactiebeheer en mechanismen om ervoor te zorgen dat gegevens integraal blijven, zelfs bij parallelle API-verzoeken.

Foutafhandeling en Feedback aan Gebruikers

Een effectieve foutafhandeling is essentieel bij het uitbreiden van API's. Gebruikers moeten duidelijke feedback ontvangen in geval van fouten. Het implementeren van robuuste fout afhandeling mechanismen en het vastleggen van fouten voor latere analyse zijn belangrijke stappen.

Documentatie, Versiebeheer en Communicatie

Het bijwerken van documentatie en het beheren van versies zijn cruciaal om een soepele uitbreiding te waarborgen. Hoe wordt de documentatie georganiseerd en bijgewerkt bij uitbreidingen? Overwegingen met betrekking tot versiebeheer moeten worden geïntegreerd om bestaande implementaties niet te breken.

Monitoring en Analyse

Effectieve monitoring van API-prestaties is nodig om prestatieproblemen snel te identificeren en op te lossen. Het implementeren van analysemechanismen om het gebruik van API's te begrijpen, draagt bij aan continue optimalisatie van het systeem.

Compliance met Wet- en Regelgeving

Het is van belang om ervoor te zorgen dat de uitgebreide API’s voldoen aan alle wettelijke en regelgevende vereisten. Regelmatige controles en aanpassingen zijn nodig om te garanderen dat het systeem in lijn blijft met de geldende wetgeving.

Tijdlijn en Budgettaire Overwegingen

Tot slot moeten realistische verwachtingen worden vastgesteld met betrekking tot de tijdlijn voor implementatie en eventuele budgettaire beperkingen moeten in overweging worden genomen.

Conclusie

In dit hoofdstuk hebben we de belangrijkste potentiële uitdagingen en overwegingen besproken bij het uitbreiden van API’s. Nu weten we waar er goed opgelet moet gaan worden met het maken de API’s.

De uitdaging waar het meest opgelet moet gaan worden is het beveiligen van de API. Dit kan niet zomaar worden afgerafeld worden maar er moet goed nagekeken worden want dit is belangrijkste gedeelte van de API.

Bronnen

1. Overcoming API Integration Challenges: Best Practices and Solutions (g.d) https://www.theneo.io/blog/api-integration-challenges
2. How to Turn Any Website into an API (17-10-2023) Emily https://visualping.io/blog/turn-website-into-api/
3. Een complete gids voor API-ontwikkeling (07-04-2023) https://appmaster.io/nl/blog/een-complete-gids-voor-api-ontwikkeling
4. API bouwen? Zó maak je een API-koppeling (g.d) https://www.lucidchart.com/blog/nl/api-bouwen
5. Challenges of Working with APIs from Developers Perspective (20-02-2024) Rahul Deo Rathore https://binmile.com/blog/challenges-of-working-with-apis-from-developers-perspective/

Hoe kan de API veilig worden gemaakt voor het uitwisselen van gegevens met anderen?

Nu we weten welke soort API er gebruikt gaat worden en we bekend zijn met de opzet van een API, kunnen we onze aandacht richten op de beveiliging ervan. In het geval van Mijn Jachtveld, waarbij externe partijen toegang zullen krijgen tot de API, is het cruciaal om de beveiliging op een hoog niveau te waarborgen. De externe aard van deze API's brengt externe risico's met zich mee, die buiten onze directe controle liggen.

Aanvallers kunnen proberen gebruik te maken van API-kwetsbaarheden die niet afdoende worden aangepakt door generieke beveiligingsmaatregelen. Dit benadrukt de noodzaak van gespecialiseerde API-beveiligingsoplossingen.

Algemene API-beveiliging

Gebruik maken van HTTPS

Zorg ervoor dat je API alleen via HTTPS wordt benaderd. Dit versleutelt de communicatie tussen de client en de server, waardoor het moeilijker wordt voor kwaadwillende om gegevens te onderscheppen. Dit betekent dat zelfs als iemand erin slaagt het dataverkeer te onderscheppen, de inhoud onleesbaar blijft zonder de juiste decryptiesleutel.

Wachtwoord Hash

Door wachtwoorden te hashen wordt het systeem beschermd. Het proces van wachtwoordhashing is het omzetten van een wachtwoord in een unieke reeks tekens, de hash genoemd. Als het wachtwoord toch onderschept wordt is het daardoor toch nog beveiligd en kan er weinig mee gedaan worden als ze het niet kunnen ontcijferen. Het hashen kan ook voor gevoelige informatie waarvan je niet wilt dat iemand anders het in handen krijgt.

OAUTH 2.0

OAuth 2.0 is een industriestandaard protocol voor autorisatie. Het stelt gebruikers in staat om een programma of website toegang te geven tot hun privégegevens die zijn opgeslagen op een andere website, zonder hun gebruikersnaam en wachtwoord te delen. Hier zijn de belangrijkste concepten:

Verificatie:

Verificatie bevestigt de identiteit van een gebruiker of app die toegang wil tot de API.

Dit kan gebeuren via referenties zoals gebruikersnaam en wachtwoord, een certificaat of eenmalige aanmelding (SSO).

Autorisatie:

Autorisatie bepaalt of een gebruiker of app gemachtigd is om toegang te krijgen tot een specifieke API.

OAuth 2.0 gebruikt tokens om toegang te verlenen zonder gevoelige gegevens zoals gebruikersnamen of wachtwoorden te delen.

OAuth zonder gebruikersaccounts

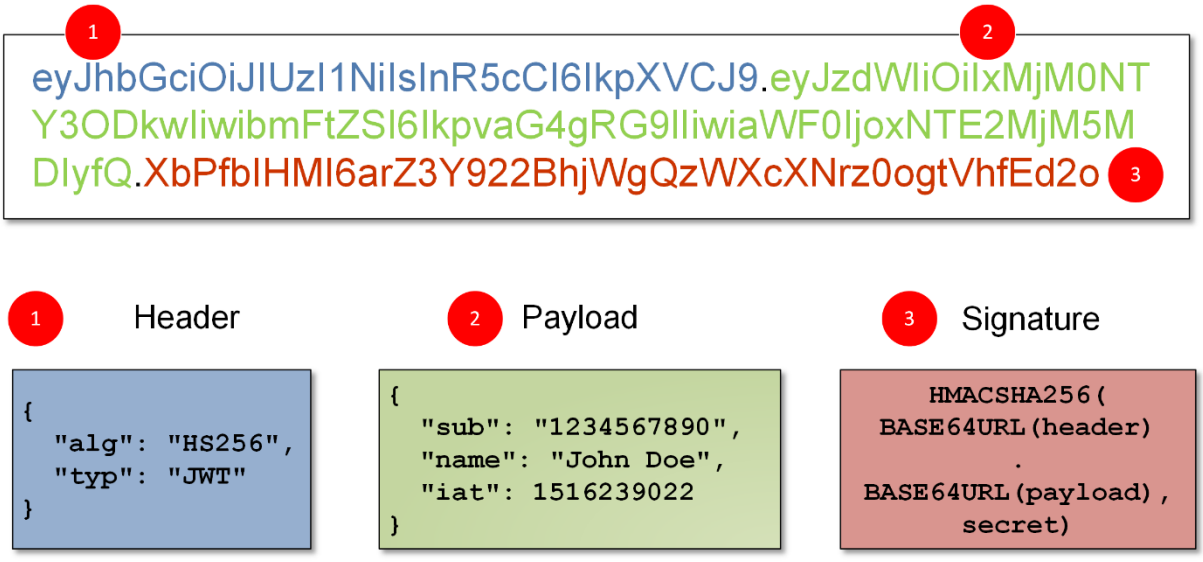
Als er geen gebruikersaccounts in uw systeem aanwezig zijn (of als het beheer van gebruikersaccounts vermijdt wil worden), kan OAuth nog steeds gebruikt worden voor scenario's zoals integratie met externe services of API's.

JWT (JSON Web Token)

Het is een compacte en URL-veilige manier om claims tussen twee partijen weer te geven. Een JWT is een gestructureerd JSON-formaat dat claims tussen partijen vertegenwoordigt. Ze worden vaak gebruikt voor authenticatie en autorisatie. De compactheid van JWT's maakt ze efficiënt voor overdracht over netwerken, terwijl de ondertekening van de token de integriteit ervan garandeert.

JWT's zijn populair in web-ontwikkeling en API-beveiliging vanwege hun compacte en veilige aard. Door hun gestandaardiseerde formaat en de mogelijkheid om claims veilig te transporteren, vormen JWT's een betrouwbare methode voor het veilig authentiseren van gebruikers in web- en API-omgevingen.

Om een beter idee te geven hoe zoon token eruitziet en hoe het in elkaar zit heb ik hieronder een afbeelding geplaats van een JWT.



Figuur 2 https://research.securitum.com/jwt-json-web-token-security/

Bearer Token

Een Bearer token is een string token dat wordt gebruikt om cliënten of servers te authentiseren en autoriseren. Het wordt vaak gebruikt in OAuth 2.0-authenticatieflows, waarbij de token wordt opgenomen in het autorisatieheader veld van HTTP-verzoeken. Een bearer token is bijna hetzelfde als een JWT-token allen het verschil is dat een JWT-token extra informatie bevat in de vorm van claims, terwijl een bearer token alleen dient als een sleutel voor authenticatie.

CORS (Cross-Origin Resource Sharing)

CORS is een beveiligingsmechanisme dat browsers gebruiken om te bepalen of een webpagina op een bepaald domein toestemming heeft om bronnen zoals data op te halen van een ander domein dan het domein waarop de pagina is geladen.

CORS is handig voor het beveiligen van je API omdat het voorkomt dat kwaadwillende websites ongeautoriseerd toegang krijgen tot de gegevens die worden aangeboden door je API. Zonder CORS-beleid zou elke willekeurige website toegang kunnen krijgen tot gegevens die worden geserveerd door de API. Met CORS kun er aangeven worden welke domeinen gemachtigd zijn om toegang te krijgen tot de API. Hierdoor kan je API vertrouwelijk blijven en alleen toegankelijk zijn voor de toegestane webpagina's.

Rate Limiting

Rate Limiting is een manier om ervoor te zorgen dat de aantal verzoeken dat een gebruiker, applicatie of IP-adres binnen een bepaalde tijdsperiode kan maken, te beperken is. Het is namelijk belangrijk voor de veiligheid maar ook voor de kwaliteit controle van de methodes. Dit wordt gedaan om overmatig gebruik, misbruik of onbedoelde belasting van een API te voorkomen. Door het implementeren van rate limiting kunnen API-providers de beschikbaarheid, stabiliteit en prestaties van de diensten behouden. Er zijn een aantal verschillende manieren om rate limiting toe te passen op een API:

Token Bucket:

Met deze methode zorg je ervoor dat een gebruiken in een bepaalde tijd een aantal verzoeken mag doen. Door elke keer na een verzoek een token uit de bucket te halen en na de periode weer terug te vullen, als de bucket op is zal de gebruiker geen verzoeken meer kunnen verzenden.

Leaky Bucket:

De leaky bucket is bijna hetzelfde als de token bucket het enige verschil is dat je de tokens bij je bucket krijgt en als de bucket overloopt krijg je een time-out.

Fixed Window Counter:

Met deze methode kun je in een bepaalde tijd (uur of een dag) een aantal keer een request sturen naar de API en als de periode is afgelopen worden de requests gereset.

Sliding Window Counter:

Deze methode is bijna hetzelfde als de fixed window counter alleen gaat het hier om een korte tijd wanneer de requests worden gereset. De tijd zal ongeveer een minuut of een paar minuten zijn tot wanneer de requests worden gereset.

IP Throtting:

Met deze methode wordt ervoor gezorgd dat er vanaf een bepaald IP-adres te veel verzoeken gedaan kunnen worden. Als de limiet is bereikt kan er geen request meer gedaan worden tot dat het limiet is gereset. Deze methode is speciaal voor bots gemaakt.

API-beveiligingssoftware

Er zijn een aantal bedrijven die het makkelijker maakt om een API te beveiligen. Ze bieden uitgebreide oplossingen die specifiek zijn ontworpen om de veiligheid van uw API's te waarborgen. Hier zijn een aantal functionaliteiten die worden aangeboden door bedrijven (niet alles wordt overal aangeboden):

- **Authentication & Authorization:** deze aanbieders gebruiken technieken zoals API keys, OAuth of JWT om de gebruiker toegang te kunnen geven tot de data van de API. Het handhaven van strikte toegangscontrole en autorisatieregels is een kernaspect van API-beveiliging.
- **Encryption:** sommige aanbieders bieden aan om encryption protocollen te gebruiken zoals SSL en TLS.
- **Logging en Monitoren:** deze aanbieder bieden inzicht in het API-verkeer en detecteren afwijkingen van normaal gedrag. Dit stelt u in staat om snel te reageren op mogelijke bedreigingen en verdachte activiteiten te identificeren.
- **Threat Detection en Preventie:** Deze aanbieders implementeren geavanceerde mechanismen voor het detecteren en voorkomen van bedreigingen, zoals DDoS-aanvallen, SQL-injecties en schadelijke bots. Dit zorgt voor een robuuste bescherming van uw API's tegen diverse aanvalsscenario's.

Het ligt aan het bedrijf dat de software aanbiedt welke functies er allemaal inzitten. Dus als er een software gebruikt gaat worden moet er goed gekeken worden welke software er gekozen wordt. Voor het beveiligen van de API zelf is het alleen maar een hulpmiddel het kan ook zonder gedaan worden alleen kost het meer tijd. En het laatste aandachtspunt maar misschien wel de grootste is de prijs. De prijzen zouden kunnen variëren van 10 euro per maand tot wel 1.000 euro per maand.

Conclusie

De conclusie die ik hier uitmaak is dat een API op verschillende manieren beveiligd kan worden en dat het ook belangrijk is om dit serieus te nemen, het laatste wat je wilt is dat er data gejat wordt. Het is noodzakelijk om gebruik te maken van HTTPS, daarnaast is het belangrijk om goed te kijken welke beveiliging methodes er gebruikt moeten gaan worden voor dit project. Waarschijnlijk wordt er gebruik gemaakt van bearer/JWT-tokens voor de autorisatie van het project vanwege de werking met Slim. Daarnaast moet er gekeken worden om rate limiting toe te toevoegen om spam tegen te gaan.

Om beveiliging software te pakken ligt vooral aan de stakeholder. Dit is omdat het veel geld kan kosten en het niet per se een must is om te pakken. Het maakt ontwikkelen van de API een stuk gemakkelijker. Daarnaast moet het softwarepakket ook werken met het Slim framework.

Ik zou eerst aanbevelen om te kijken of het beveiligen van een API ook zonder een externe beveiliging software. Als de beveiliging eenmaal staat hoeft er ook niet meer nagekeken te worden en dat kan op de langere termijn veel geld besparen. Als het toch lastiger is dan gedacht is het handig om toch naar een pakket te kijken zodat je zeker weet dat de API goed beveiligd is.

Bronnen

1. REST API Security Essentials (04-11-2023) Lokesh Gupta
   https://restfulapi.net/security-essentials/
2. Best practices for REST API security: Authentication and authorization (06-10-2021) Sam Scott/Graham Neray
   https://stackoverflow.blog/2021/10/06/best-practices-for-authentication-and-authorization-for-rest-apis/
3. REST API Security (15-04-2023) Chandan Singh
   https://medium.com/@chandan3611/rest-api-security-fea121198d66
4. Understanding OAuth2 (02-08-2023) Albert Starreveld
   https://medium.com/web-security/understanding-oauth2-a50f29f0fbf7
5. 4 Most Used REST API Authentication Methods (29-07-2019) Guy Levin
   https://blog.restcase.com/4-most-used-rest-api-authentication-methods/
6. How to secure a REST API using JWT authentication (12-08-2022) Fernano Doglio
   https://blog.logrocket.com/secure-rest-api-jwt-authentication/
7. API Security 101: Understanding the Risks and Implementing Best Practices (15-12-2023) Vinugayathri Chinnasamy
   https://www.indusface.com/blog/what-is-api-security-and-why-is-it-important
8. CORS (Cross-Origin Resource Sharing) (23-08-2023)
   https://appmaster.io/nl/glossary/cors-cross-origin-resource-sharing-0
9. Best API Security Software (g.d)
   https://www.peerspot.com/categories/api-security
10. Lock & Code: The Ultimate Guide to the 27 Best API Security Tools of 2024 (04-01-2024) Paulo Gardini Miguel
    https://thectoclub.com/tools/best-api-security-tools/
11. API Security: Authorization, Rate Limiting, and Twelve Ways to Protect APIs (18-05-2023) Phani Deepak Akella
    https://www.indusface.com/blog/api-security-guide-ways-to-protect-apis/
12. What is API Rate Limiting and How to Implement It (04-08-2020)
    https://datadome.co/bot-management-protection/what-is-api-rate-limiting/
13. API Rate Limiting — Everything You Need to Know (22-08-2023) Marguel Ellis
    https://blog.hubspot.com/website/api-rate-limit
14. Bearer Token, JWT Bearer Token, OAuth2 (13-03-2023) Tushar Ghosh
    https://tusharghosh09006.medium.com/bearer-token-jwt-bearer-token-oauth2-193d24574038
15. What is Bearer token and How it works? (07-05-2021) Rajesh Kumar
    https://www.devopsschool.com/blog/what-is-bearer-token-and-how-it-works/