



Web Servislerine Giriş

Zafer CÖMERT



**Bölüm
6**

REST ve RESTful API

Web Servislerine Giriş



REST

1. REST
2. REST Kısıtlamaları
3. HTTP Metotları
4. Basitlik
5. Sayfalama
6. Filtremele ve Sıralama
7. Kaynak isimlendirme
8. Idempotence
9. Oturum Açma ve Yetkilendirme
10. REST API Olgunluğu
11. REST API Dokümantasyonu

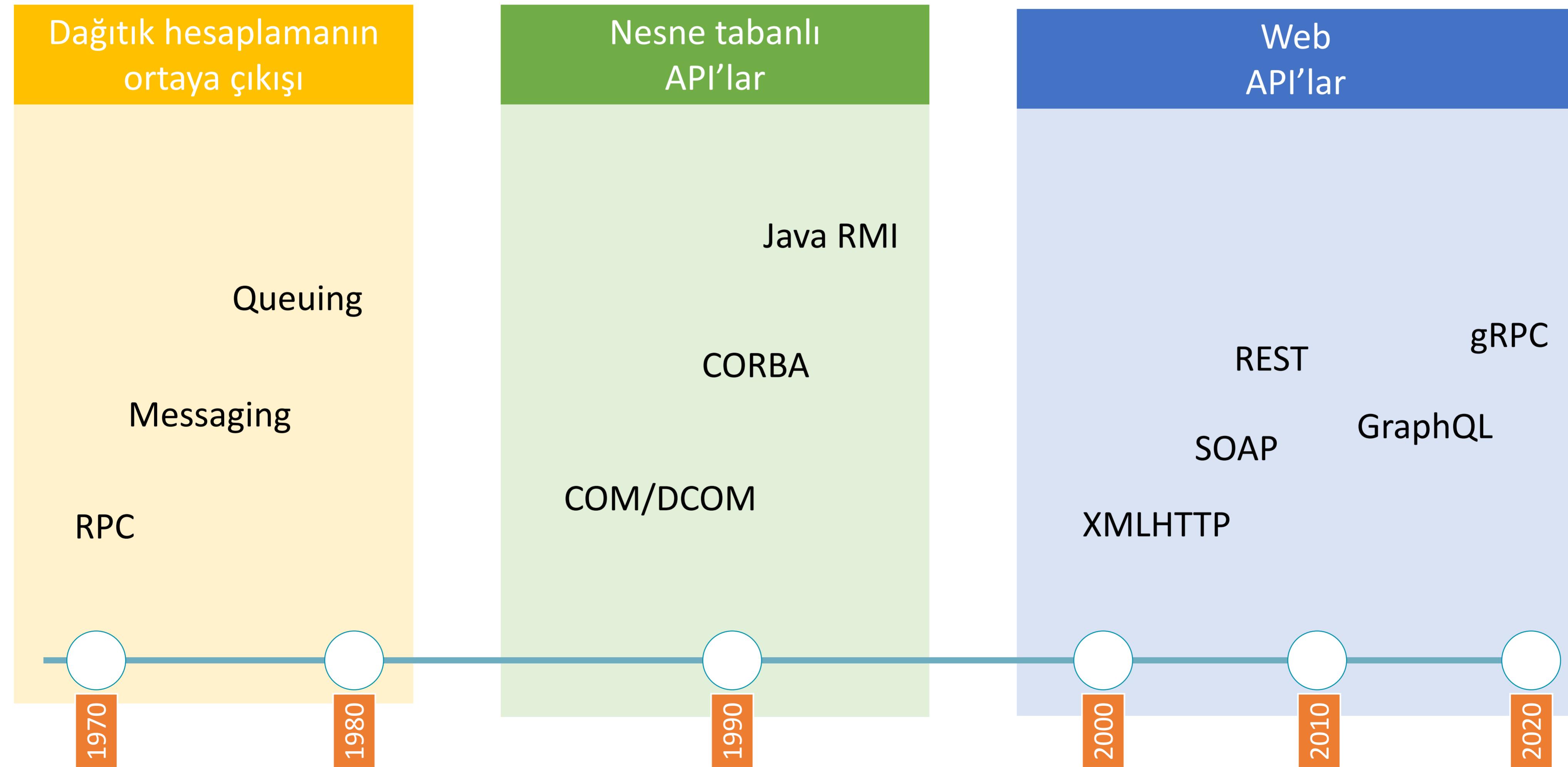
API

Application Programming Interface

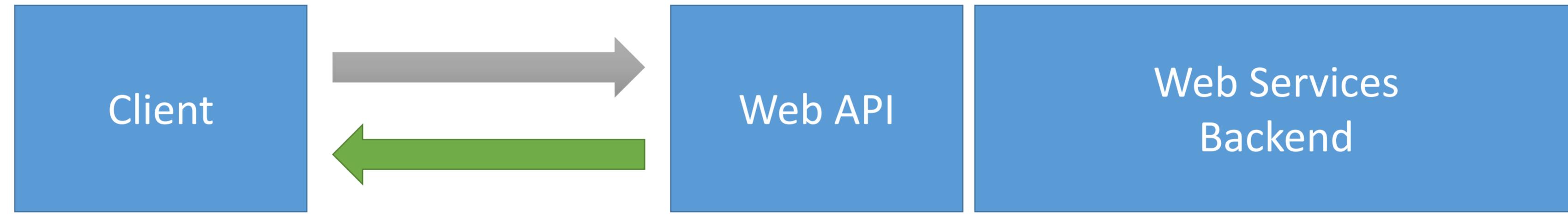
API

- **Application Programming Interface (API)**, bir yazılımın diğer yazılımlar veya uygulamalar tarafından kullanılmasını sağlayan belirli bir kural seti içeren bir arayüzdür.
- API'ler, bir uygulamanın belirli işlevlerine veya verilere erişim sağlar, böylece farklı yazılım bileşenleri veya hizmetleri birbiriyle iletişim kurabilir.

APIs



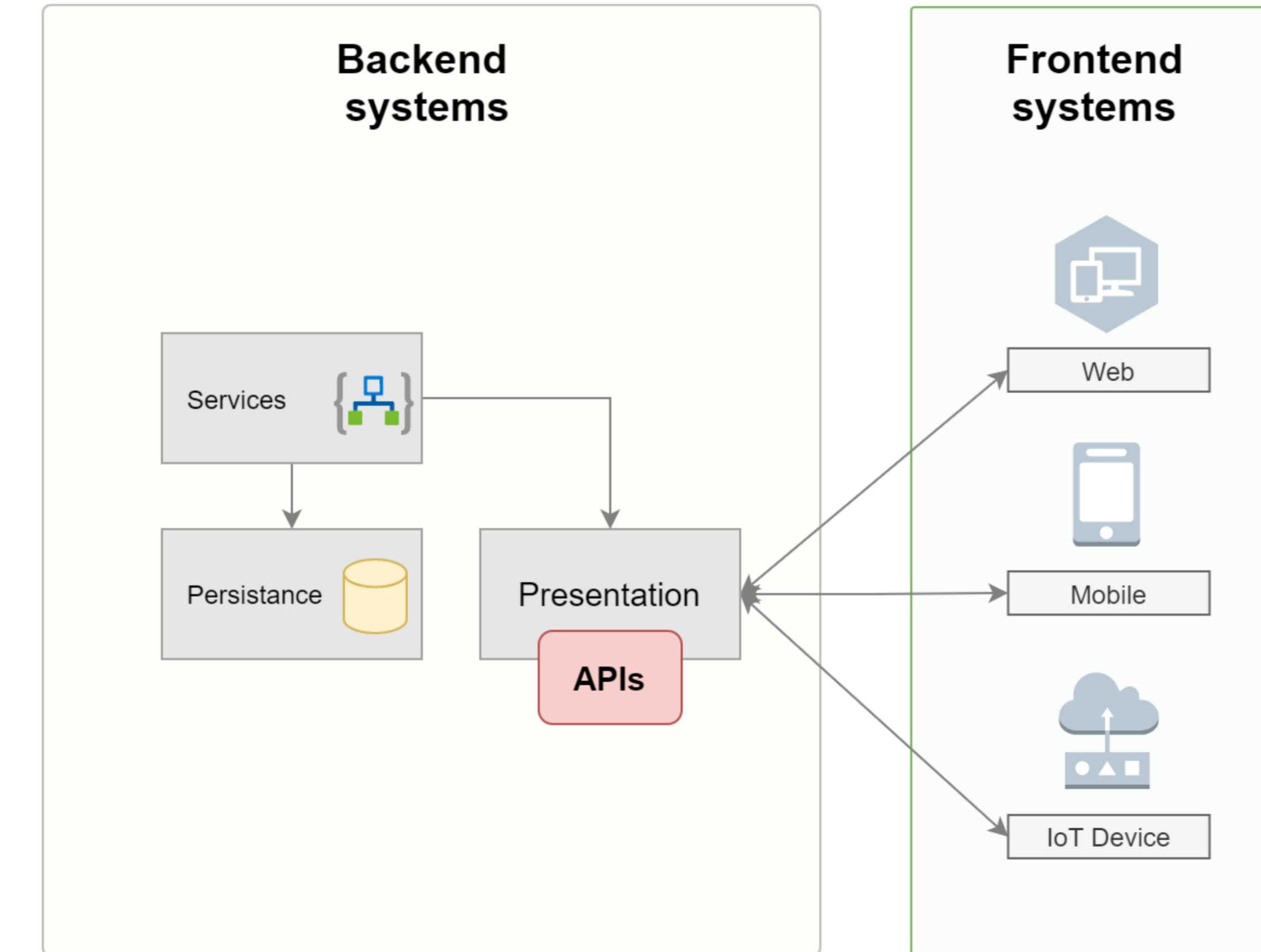
API



- Web servisleri, bir web sitesi veya başka bir uygulamanın ihtiyaçlarını destekleyen özel web sunucularıdır. İstemci programlar, web servisleri ile iletişim kurmak için uygulama programlama arayüzlerini (API'ler) kullanır. Genel olarak, bir API, bilgisayar programları arasındaki etkileşimleri kolaylaştmak ve bilgi alışverişi yapmalarına olanak tanımak için bir dizi veri ve işlevi ortaya koyar.

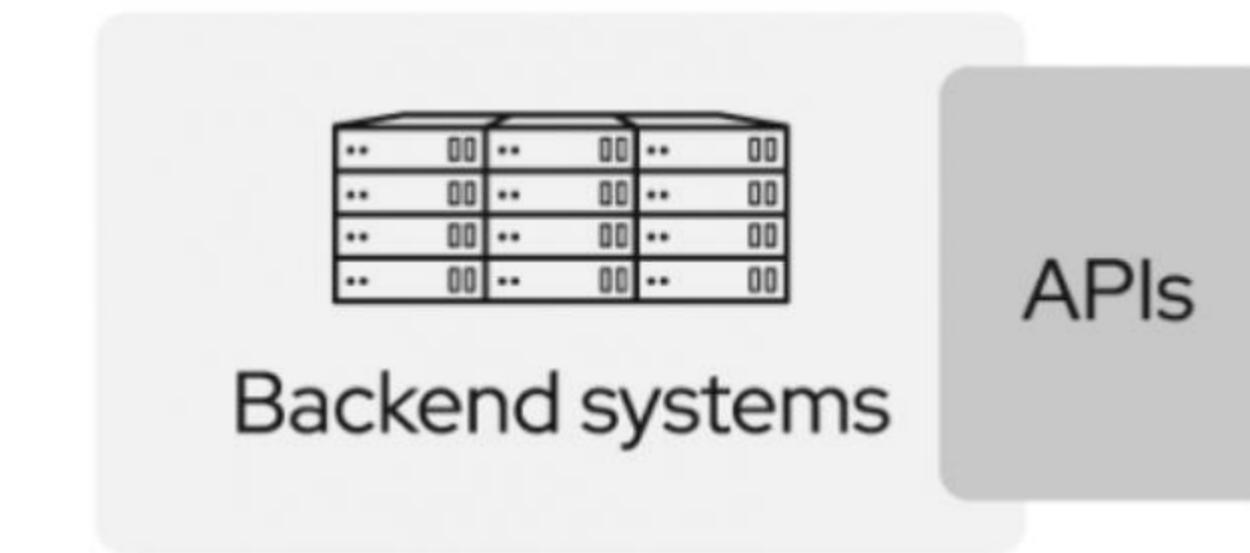
Web Servisler

API



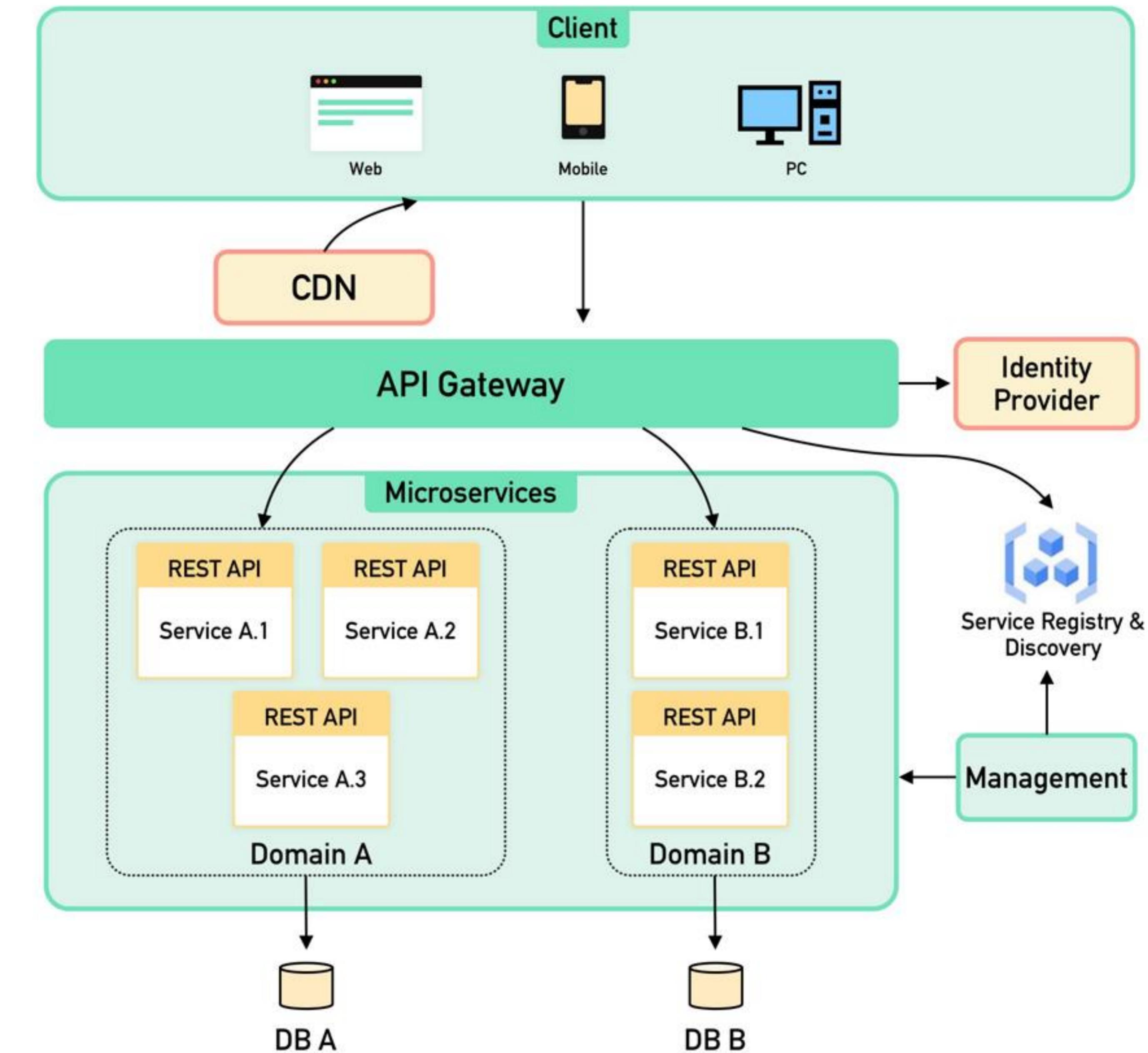
Web Servisler

API



1. Backend as a service (BaaS)
2. Backend as a function (BaaF)

Microservices





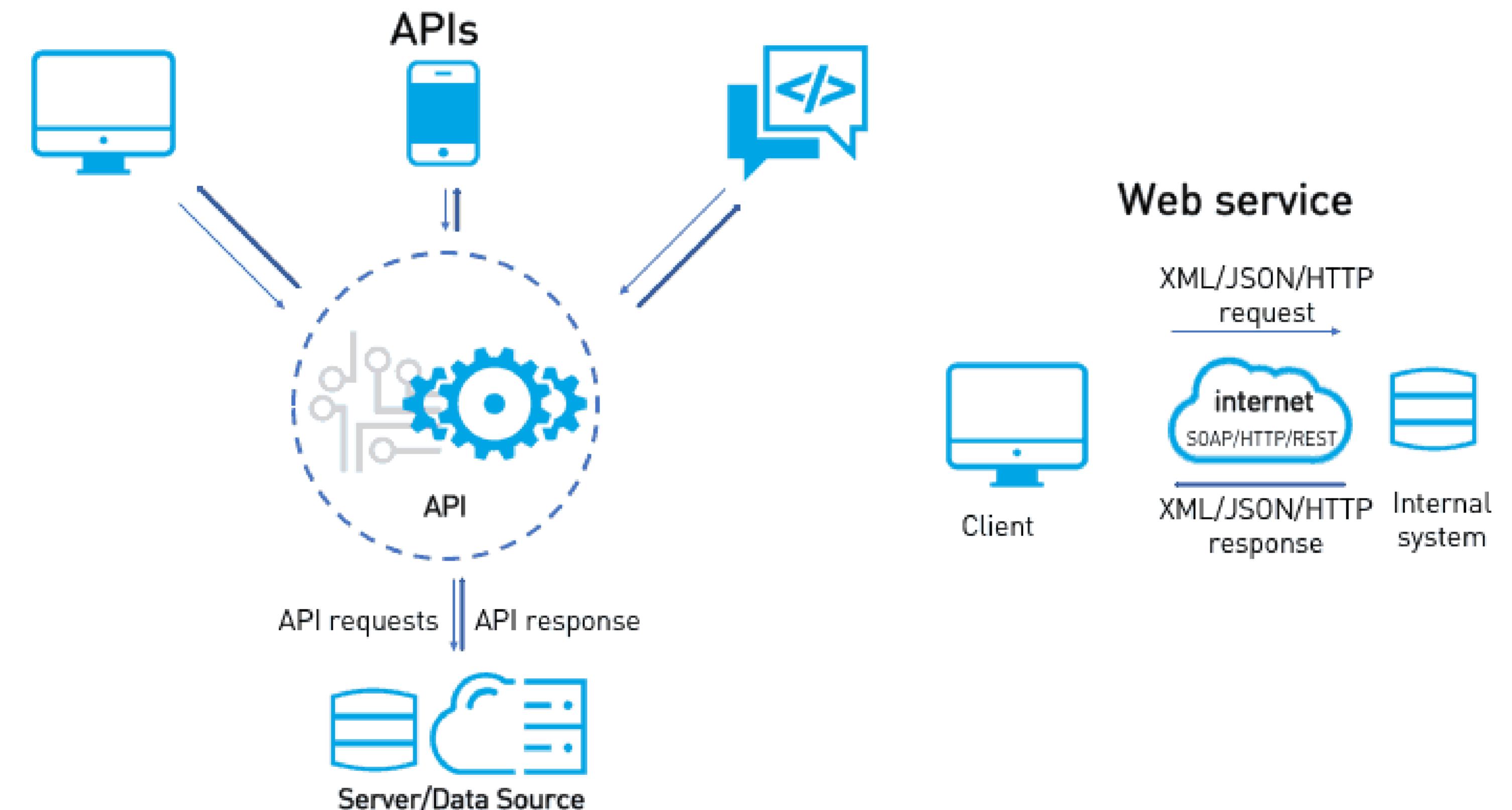
Web Servislere karşı Web API

Neden Web API bir web servisi değildir?

Web Servisler ve Web API

- "Bütün kaplanlar kedilerdir, ancak bütün kediler kaplan değildir. "
- Bütün web servisleri web API'dır, ancak bütün web API'ler web servisi değildir.
- Web API'ları ve web servisleri sık sık karıştırılır, ancak web API'ları web servislerinin bir evrimidir.
- Her ikisi de bilgi transferini kolaylaştırır, ancak web API'ları web servislerine göre daha dinamiktir.

Web Servisi ve API



Web Servisler ve Web API

Web Servis

- Tanım olarak, bir web servisi, kendisini Internet üzerinden kullanılabılır hale getiren ve iletişimini genellikle **XML** kodlaması aracılığıyla standartlaşdırın herhangi bir yazılım parçasıdır.

Web API

- Web API, yazılım bileşenlerinin birbirleriyle web protokolü (**HTTP**) aracılığıyla nasıl etkileşimde bulunması gerektiğini belirtir.

Web Servisler ve Web API

Web Servis

- Birçok insan için web servisleri, **SOA** (Hizmet Odaklı Mimari) ile eşanlamlıdır ve genellikle **XML-RPC** ve **SOAP** gibi standartları kullanır.

Web API

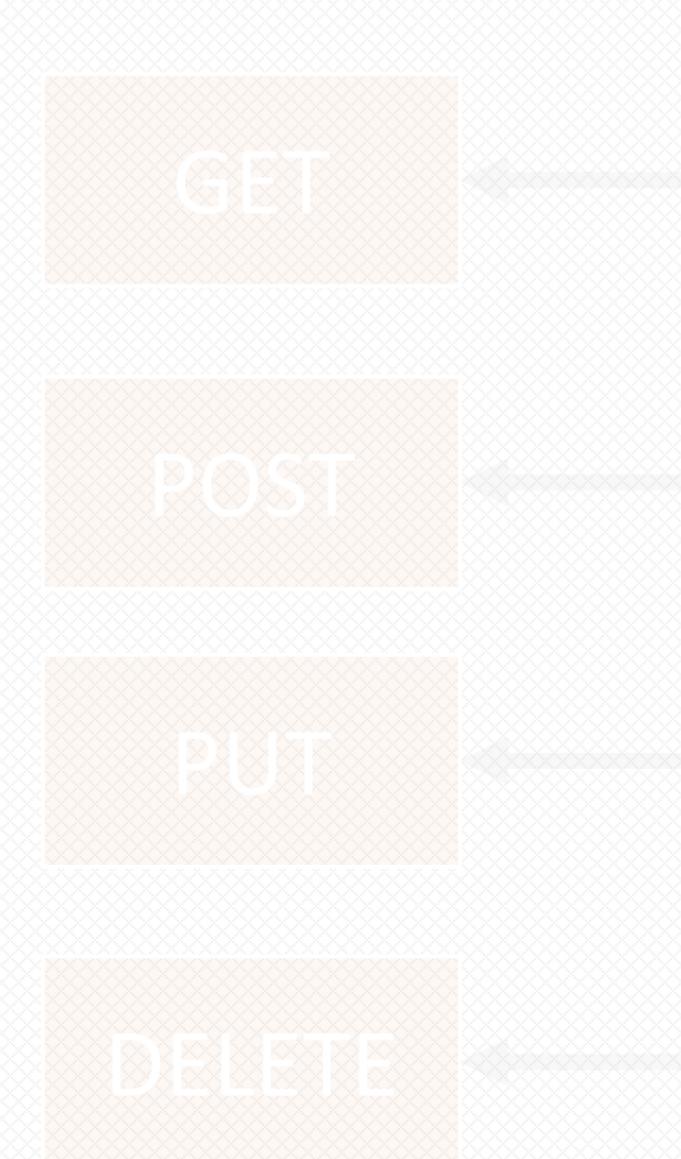
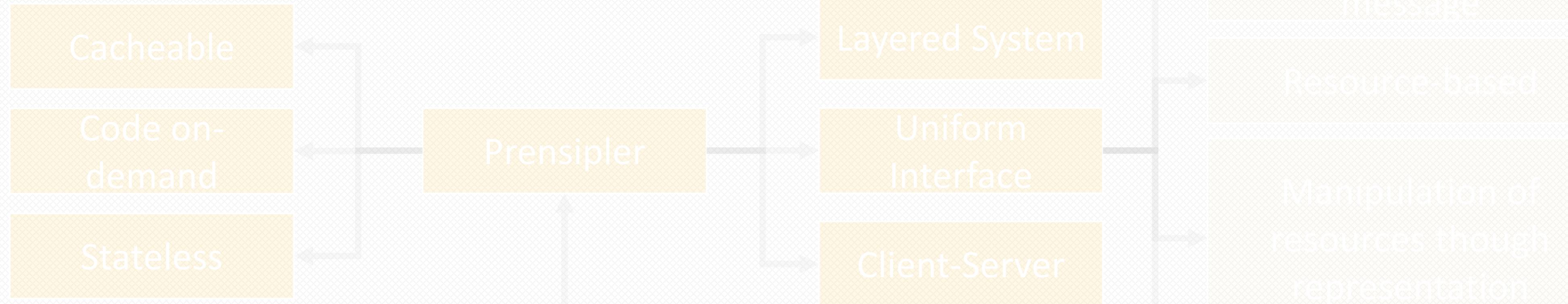
- İstemcinin sunucuda hangi prosedürü çağırması gerektiğini bilmesi gerekmez.

Web API Tasarımı

Web API Design

Web API Tasarımı

- Yük boyutu
- Gecikme
- Ölçekleme
- Yük dengeleme
- Diller arası birlikte çalışma
- Oturum açma
- İzleme
- Günlük
- Payload size
- Latency
- Scalability
- Load Balancing
- Language Interop
- Authentication
- Monitoring
- Logging



REST

Representational State Transfer
Resource-oriented Architecture (ROA)

REST

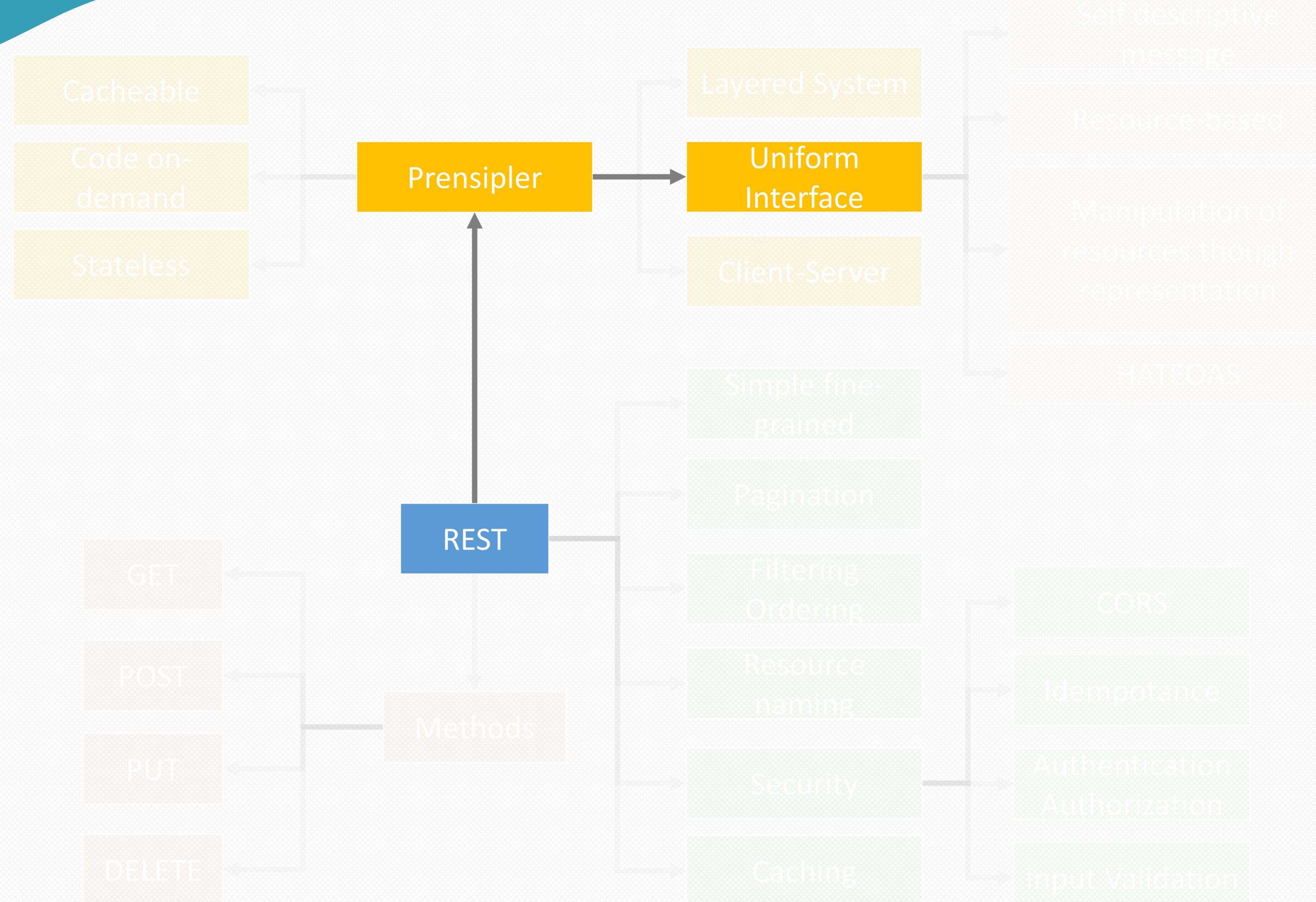
- 2000 yılında [Roy Fielding](#), web hizmetleri tasarıımı için bir mimari yaklaşım olarak REST önerdi.
- REST, hipermedyaya dayalı dağıtılmış sistemler inşa etmek için kullanılan bir mimari stildir.
- REST, herhangi bir temel iletişim protokolüne bağlı olmayan bir yaklaşımıdır. Bununla birlikte, en yaygın REST API uygulamaları genellikle HTTP'yi uygulama protokolü olarak kullanır, ve bu rehber, HTTP için REST API'lerinin tasarıımına odaklanır.

REST

- REST, Temsili Durum Transferi anlamına gelmektedir. Genellikle web hizmetlerinin geliştirilmesinde kullanılan, ağ üzerinden gevşek bir şekilde bağlanmış uygulamalar tasarlamaya yönelik bir mimari tarzıdır.
- REST, istemci ve sunucu ortamlarını birbirinden ayırlır.
- REST, alt düzeyde nasıl uygulanması gerekiğine ilişkin herhangi bir kural koymaz, sadece üst düzey tasarım yönergeleri koyar ve bizi kendi uygulamamızı düşünmeye bırakır.

REST Kısıtlamaları

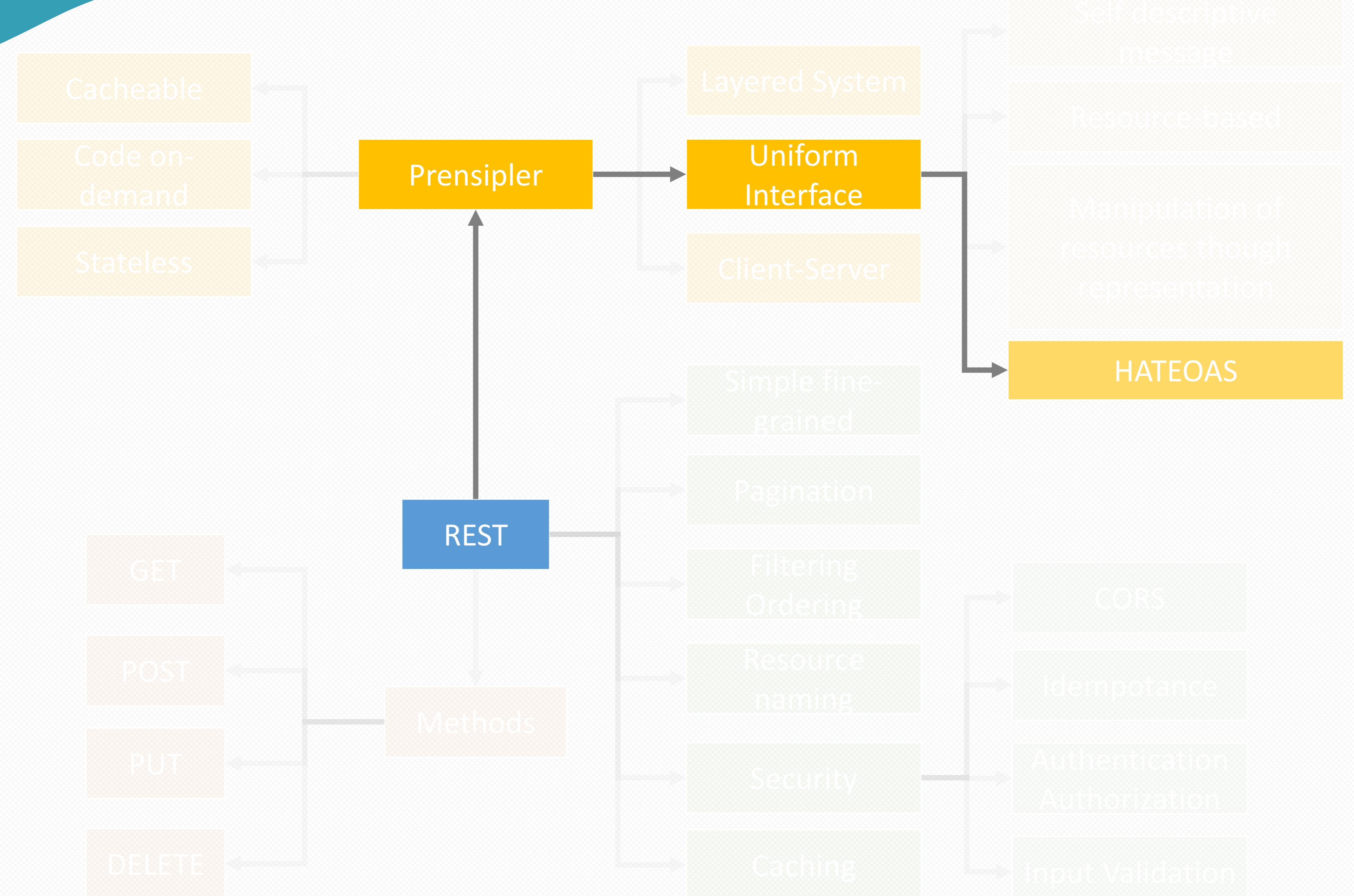
REST Constraints



Tek Tip Arayüz

Uniform Interface

- Bu kısıt, REST API'lerin temel özelliklerinden biridir ve bu kısıt, API'nin arayüzünün basit ve tutarlı olmasını gerektirir.
- Bu kısıtin amacı, farklı istemcilerin aynı API'yi kullanırken aynı temel kurallara uygun bir şekilde etkileşimde bulunabilmesini sağlamaktır.
- Herhangi bir tek kaynak (resource) fazla büyük olmamalı ve temsilinde her şeyi içermemelidir.



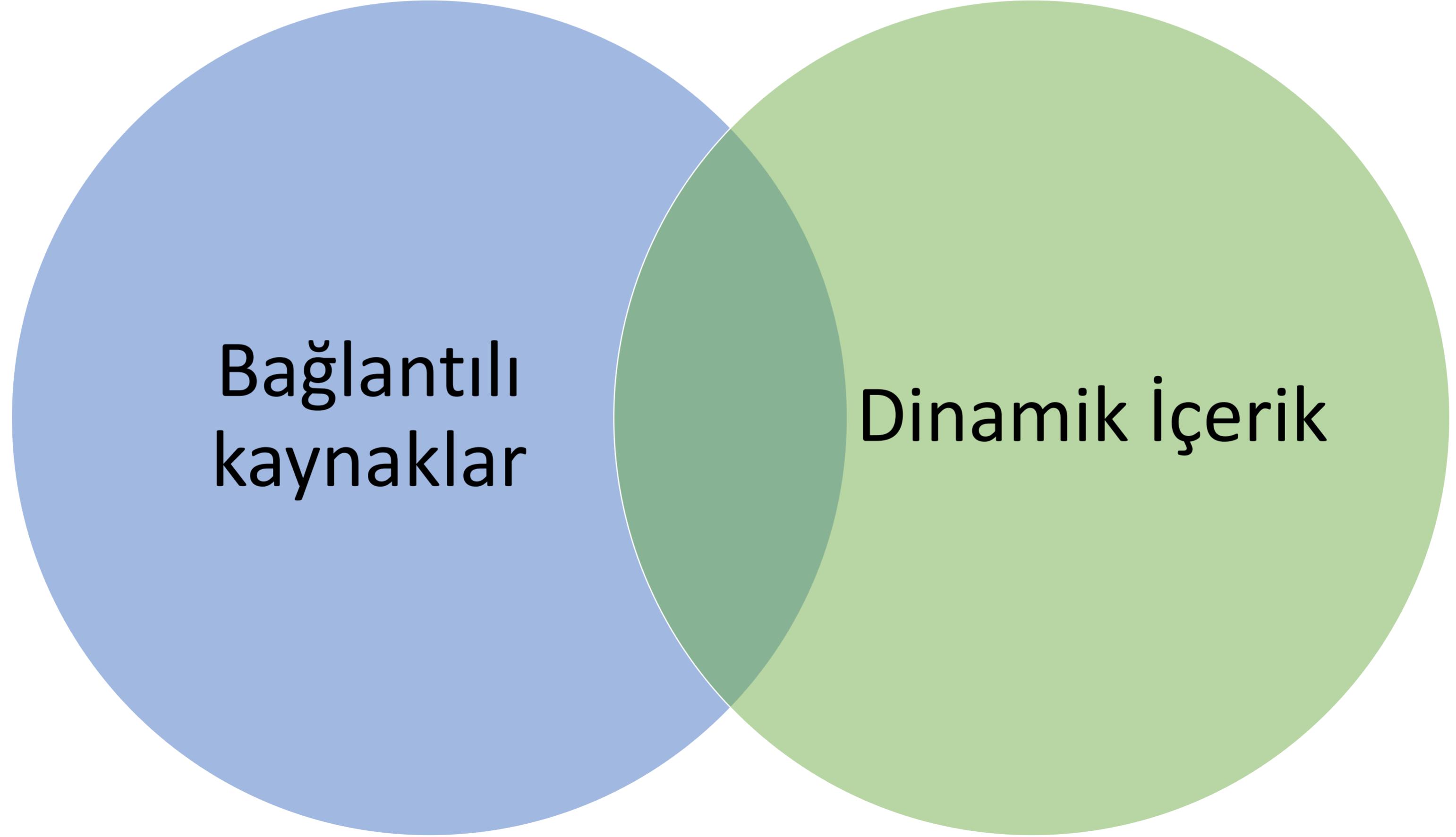
HATEOAS

Uniform Interface

- "HATEOAS" veya "Hypertext as the Engine of Application State," RESTful API'lerin "Uniform Interface" (Tutarlı Arayüz) kısıtının önemli bir bileşenidir.
- Bu kavram, RESTful API'nin temel ilkesinden birini ifade eder. API istemcilerinin, sunucudan alınan yanıtlar aracılığıyla kaynaklar ve bu kaynaklar üzerinde yapılacak işlemler hakkında bilgi edinebilmelidirler.

HATEOAS

Uniform Interface



Bağlantılı
kaynaklar

Dinamik İçerik

HATEOAS

Uniform Interface

- Bir e-ticaret uygulamasının RESTful API'sini düşünün.
- Bir ürün listesi almak için bir GET isteği gönderdiğinizde, sunucu yanıtı aşağıdaki gibi bir JSON belgesi içerebilir.
- Bu yanıt, "HATEOAS" ilkesine uyar.



```
1  {
2      "products": [
3          {
4              "id": 1,
5              "name": "Ürün 1",
6              "price": 19.99,
7              "links": [
8                  {
9                      "rel": "self",
10                     "href": "/products/1"
11                 },
12                 {
13                     "rel": "edit",
14                     "href": "/products/1/edit"
15                 },
16                 {
17                     "rel": "delete",
18                     "href": "/products/1/delete"
19                 }
20             ]
}
```

HATEOAS

Uniform Interface

- Her ürün ögesi, "links" adlı bir bağlantı listesi içerir.
- Bu bağlantılar, bu ürün üzerinde gerçekleştirilebilecek işlemleri temsil eder.
- Örneğin, "edit" bağlantısı bu ürünü düzenlemek için kullanılabilir.

```

1  {
2      "products": [
3          {
4              "id": 1,
5              "name": "Ürün 1",
6              "price": 19.99,
7              "links": [
8                  {
9                      "rel": "self",
10                     "href": "/products/1"
11                 },
12                 {
13                     "rel": "edit",
14                     "href": "/products/1/edit"
15                 },
16                 {
17                     "rel": "delete",
18                     "href": "/products/1/delete"
19                 }
20             ]
}

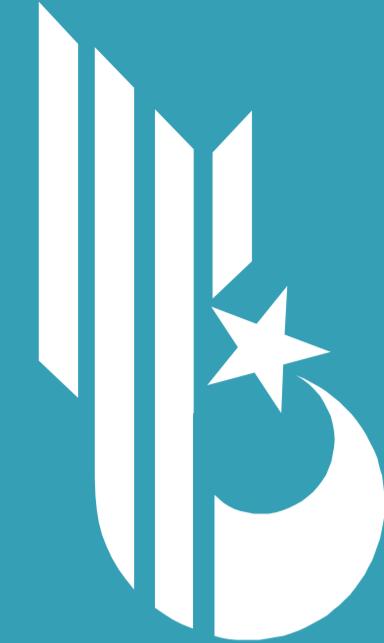
```

HATEOAS

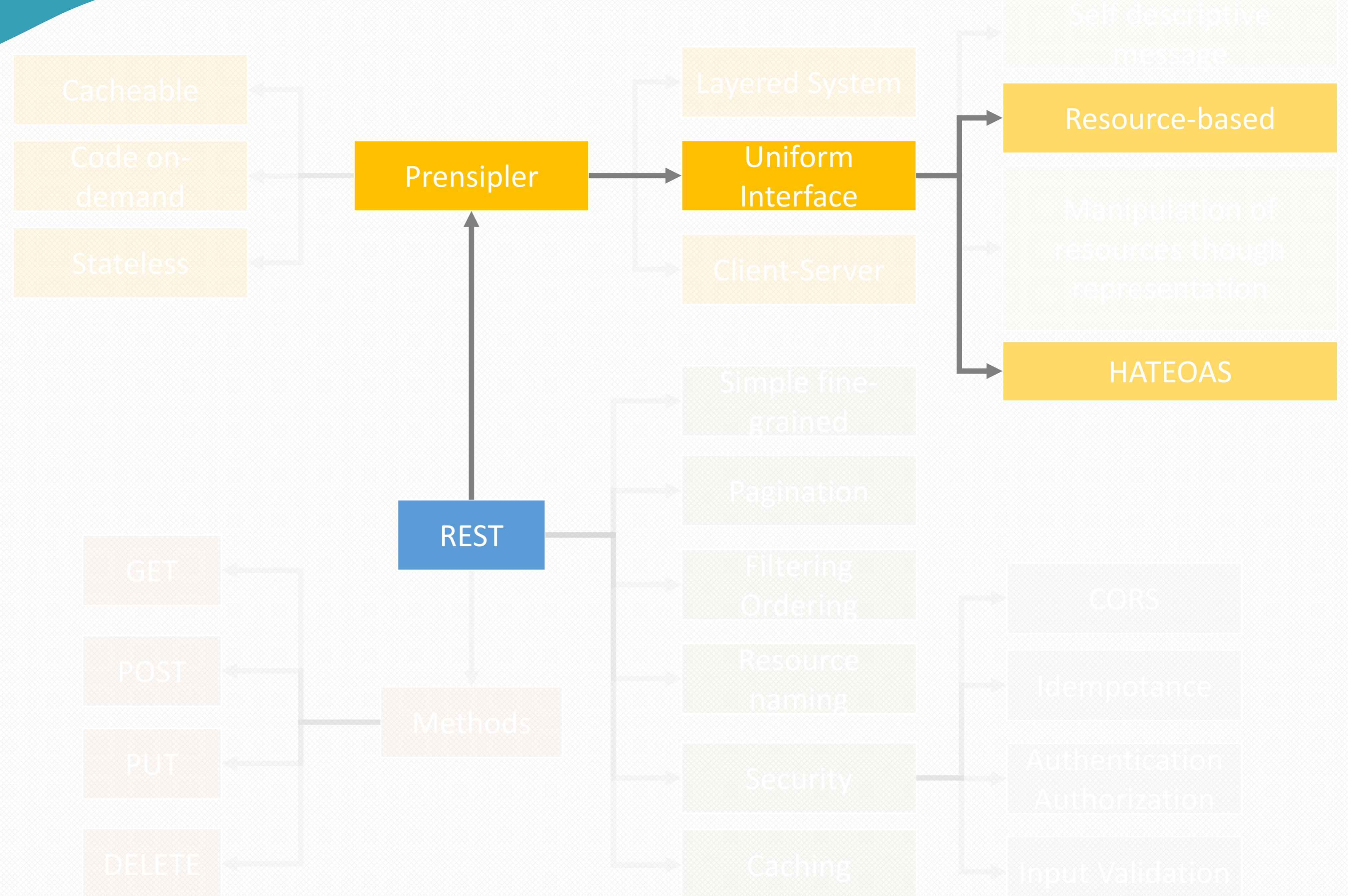
Uniform Interface



```
1  {
2      "orderID":3,
3      "productID":2,
4      "quantity":4,
5      "orderValue":16.60,
6      "links": [
7          {"rel":"product","href":"https://adventure-works.com/customers/3", "action":"GET" },
8          {"rel":"product","href":"https://adventure-works.com/customers/3", "action":"PUT" }
9      ]
10 }
```



HATEOAS, API'nin belirli bir kaynağı nasıl kullanacağını ve nasıl etkileşimde bulunacağını açıklar. Bu da API'nin esnek ve keşfedilebilir olmasını sağlar.



Kaynak Odaklı

Uniform Interface

- Kaynak Odaklı (**Resource-Based**), REST API'nin "Uniform Interface" kısıtının önemli bir yönünü ifade eder.
- Bu kavram, **RESTful API'lerin** kaynaklarını temsil etmek ve bu kaynaklar üzerinde işlem yapmak için kullanılan bir modeli ifade eder.
- Kaynak odaklı bir **REST API**, sunucu tarafından sunulan kaynakların temelini oluşturur ve bu kaynaklara erişim ve etkileşim sağlar.

Kaynak Odaklı

Uniform Interface

GET

POST

PUT

DELETE

Query
String

Route
Data

HTTP Metotları ile etkileşim

Temel URL Yapısı

Her şey kaynaktır

Kaynak Odaklı

Uniform Interface

FİİL	CRUD	İŞLEV	GÜVENLİ	Idempotent
GET	Okuma	Tek veya çoklu kaynak getirme	+	+
POST	Oluşturma	Yeni bir kaynak ekleyin	-	-
PUT	Güncelleme/ Oluşturma	Yeni bir kaynak ekleme veya var olanı güncelleme	-	+
DELETE	Silme	Tek veya birden fazla kaynağı silme	-	+
OPTIONS	Okuma	Bir kaynak üzerinde izin verilen işlemleri listeleme	+	+
HEAD	Okuma	Gövde olmadan yalnızca yanıt üstbilgilerini döndürür	+	+
PATCH	Güncelleme/ Değiştirme	Yalnızca kaynakta sağlanan değişiklikleri güncelleyin	-	-

Kaynak Odaklı

Uniform Interface

Bağlantılı Kaynaklar

Temel URL Yapısı

HTTP Metotları ile
Etkileşim

Her şey kaynaktır

`http://localhost/books/`

`http://localhost/books/ISBN-0011`

`http://localhost/books/ISBN-0011/authors`

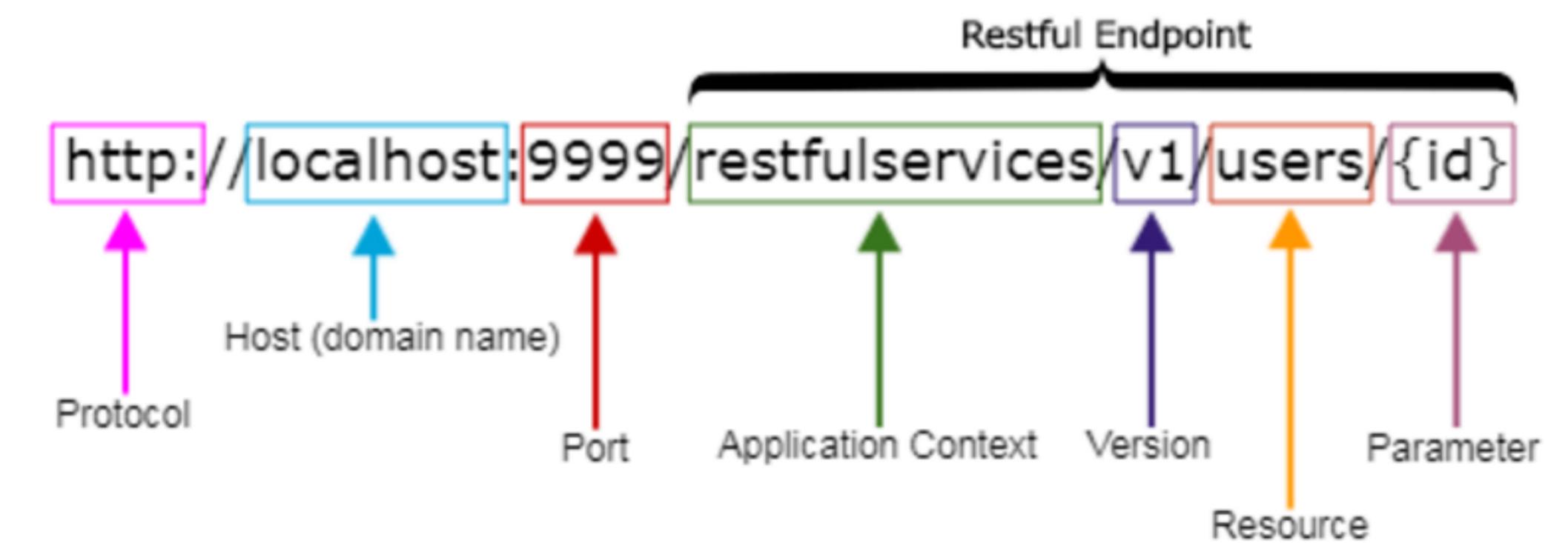
`http://localhost/classes`

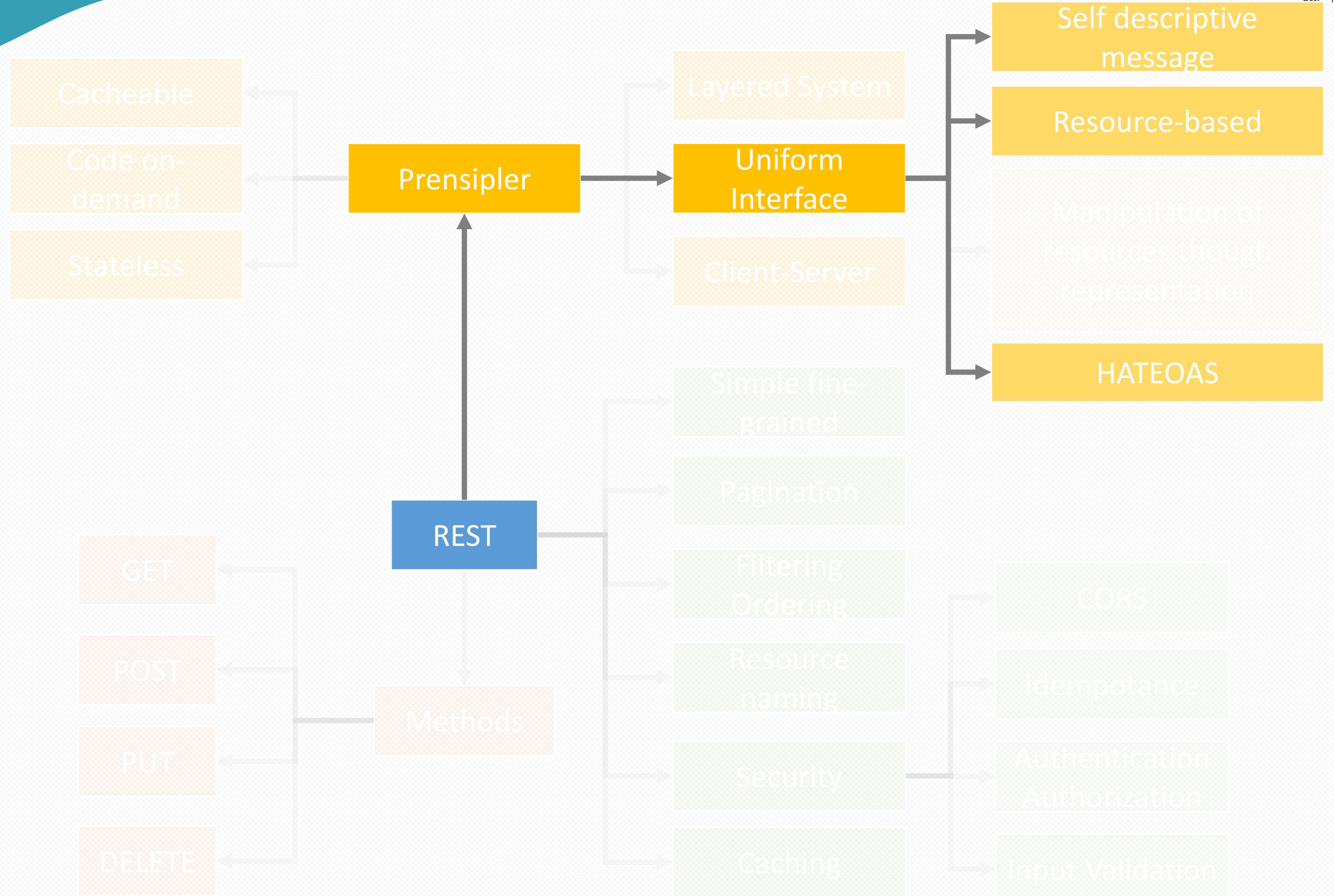
`http://localhost/classes/cs2650`

`http://localhost/classes/cs2650/students`

Kaynak Odaklı

Uniform Interface

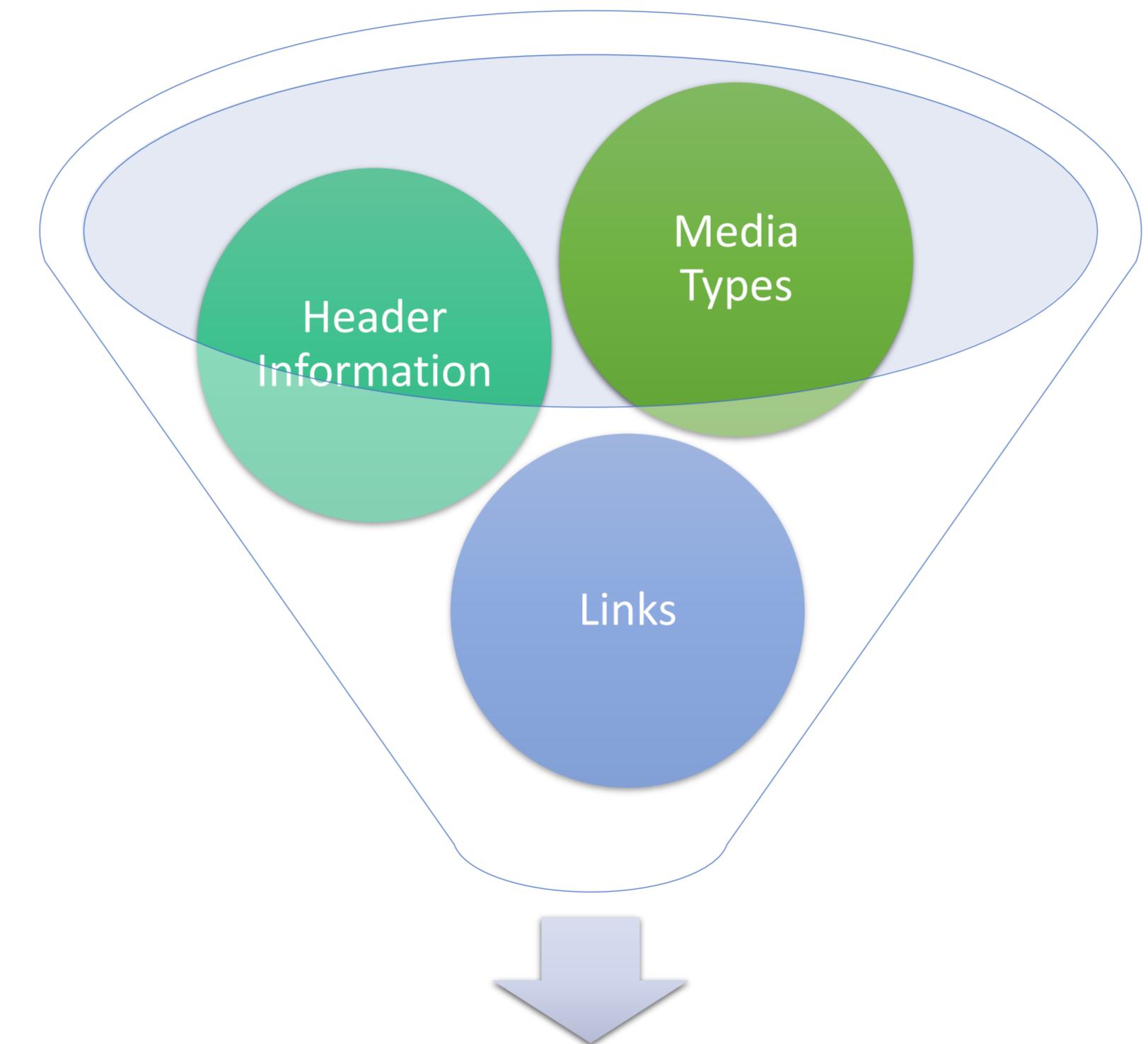




Kendini Açıklayıcı Mesajlar

- "Kendini Açıklayıcı Mesaj" ([Self-Descriptive Message](#)) REST API'nin "[Uniform Interface](#)" kısıtının bir parçasıdır ve temel olarak her iletişimde yer alan mesajların ([HTTP](#) istekleri ve cevapları gibi) kendini açıklayıcı olması gerektiğini belirtir.
- Bu kavram, iletişimdeki tüm bileşenlerin (istemci ve sunucu dahil) bir mesajın içeriğini anlamak için ek bilgilere ihtiyaç duymadan o mesajı yorumlayabilmesi gereği anlamına gelir.

Kendini Açıklayıcı Mesajlar



Self-descriptive

Kendini Açıklayıcı Mesajlar

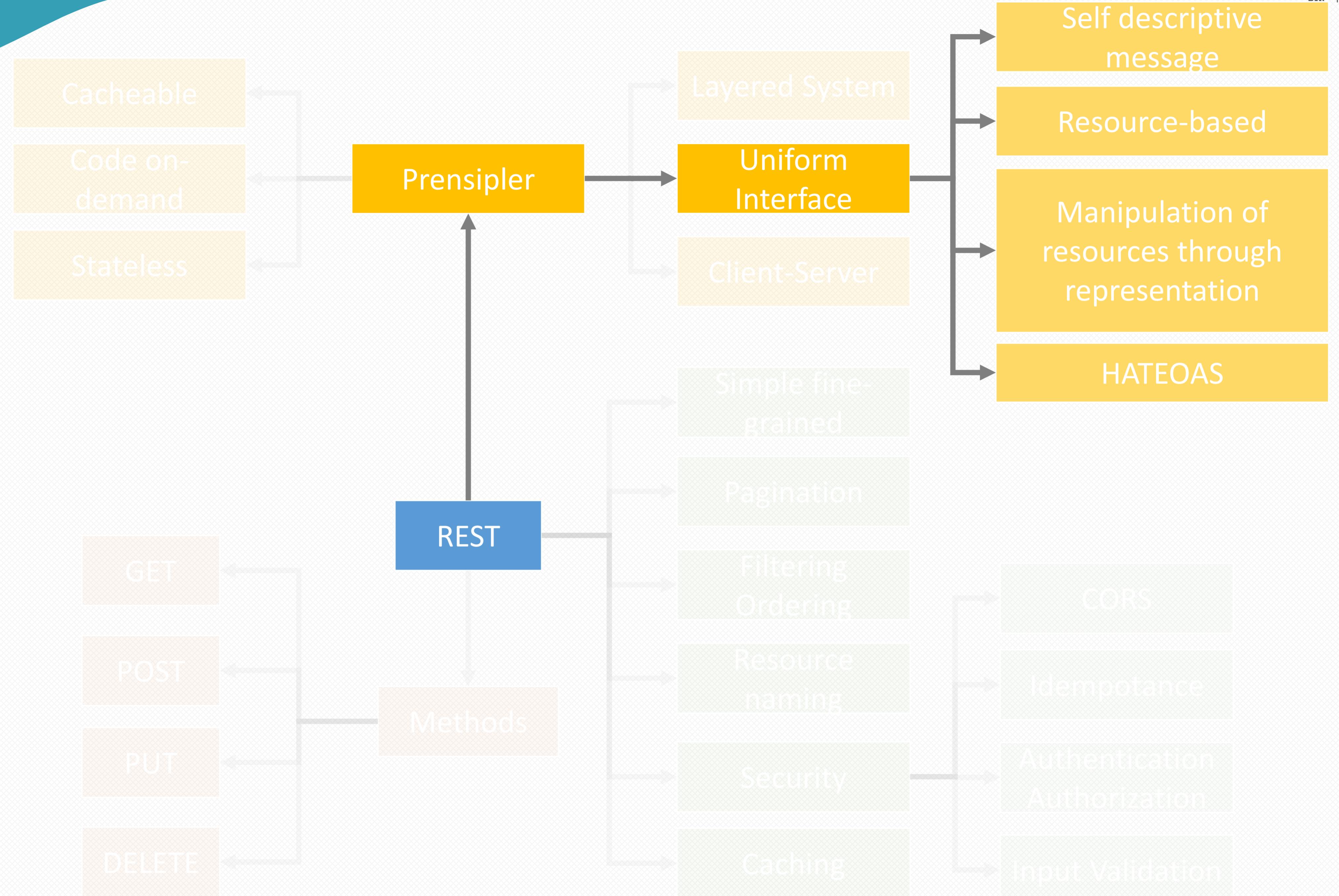
- Mesajlar, içeriklerinin türünü belirten bir medya türü (**media type**) ile etiketlenmelidir.
- Özellikle, **HTTP** başlıkları aracılığıyla "**Content-Type**" başlığı ile ifade edilir.
- Bu, istemcinin ve sunucunun bir mesajın içeriğini anlamak için hangi veri formatının kullanıldığını bilmesini sağlar.
- Örneğin, "**application/json**" medya türü **JSON** veri formatını ifade eder.

Kendini Açıklayıcı Mesajlar

- Mesaj başlıklarları, mesajın içeriği hakkında ek bilgiler içerebilir.
- Örneğin, bir istemciye önbellek kontrolü yapabilmesi için bir kaynağın son değişiklik tarihi ("Last-Modified" başlığı) veya içeriğin sıkıştırıldığı sıkıştırma algoritması ("Content-Encoding" başlığı) hakkında bilgi sağlayabilir.

Kendini Açıklayıcı Mesajlar

- Mesajlar, ilgili kaynaklara erişim için bağlantılar içerebilir.
- Bu bağlantılar, ilişkili kaynaklara veya diğer ilgili kaynaklara nasıl erişileceği konusunda rehberlik eder.
- Bu, **HATEOAS** prensibinin bir parçasıdır.



Temsil ile Manipülasyon

Uniform Interface
Manipulation of resources through representation

- "Manipulation of Resources Through Representation," REST API'nin "Uniform Interface" kısıtının bir parçasıdır ve API'nin temel işlevsellğini açıklar.
- Bu kavram, RESTful bir API'nin istemcilerinin kaynakları temsil eden veri göstergeleri (**representation**) üzerinden bu kaynakları değiştirebilmeleri gerektiğini belirtir.

Temsil ile Manipülasyon

Uniform Interface
Manipulation of resources through representation



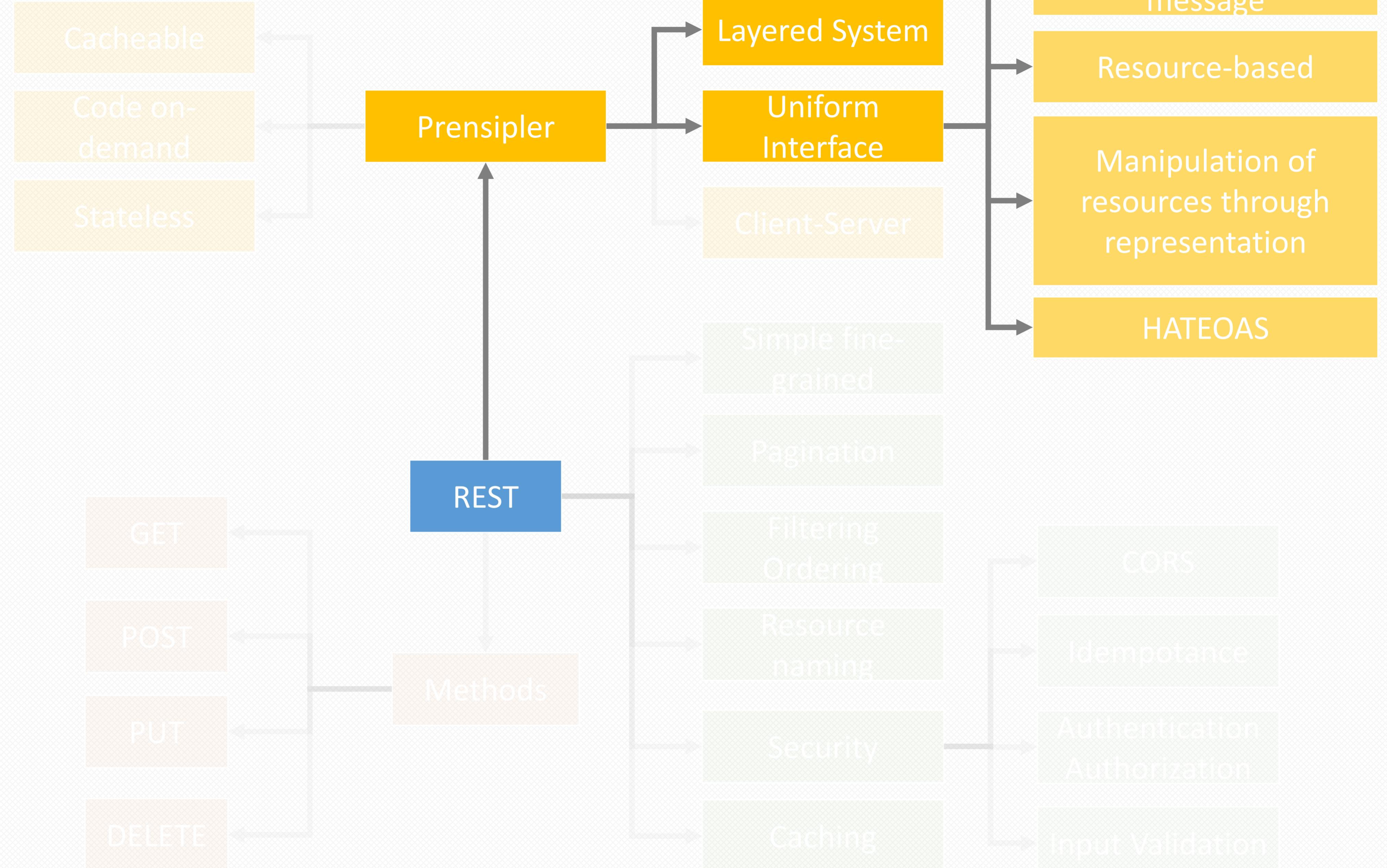
```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <Book>
3   <ID> 1 </ID>
4   <Name> Building REST APIs with Flask </Name>
5   <Author> Kunal Relan </Author>
6   <Publisher> Apress
7   </ Publisher>
8 </Book>
```

Temsil ile Manipülasyon

Uniform Interface
Manipulation of resources through representation



```
1  {
2      "ID": "1",
3      "Name": "Building REST APIs with Flask",
4      "Author": "Kunal Relan",
5      "Publisher": "Apress"
6 }
```



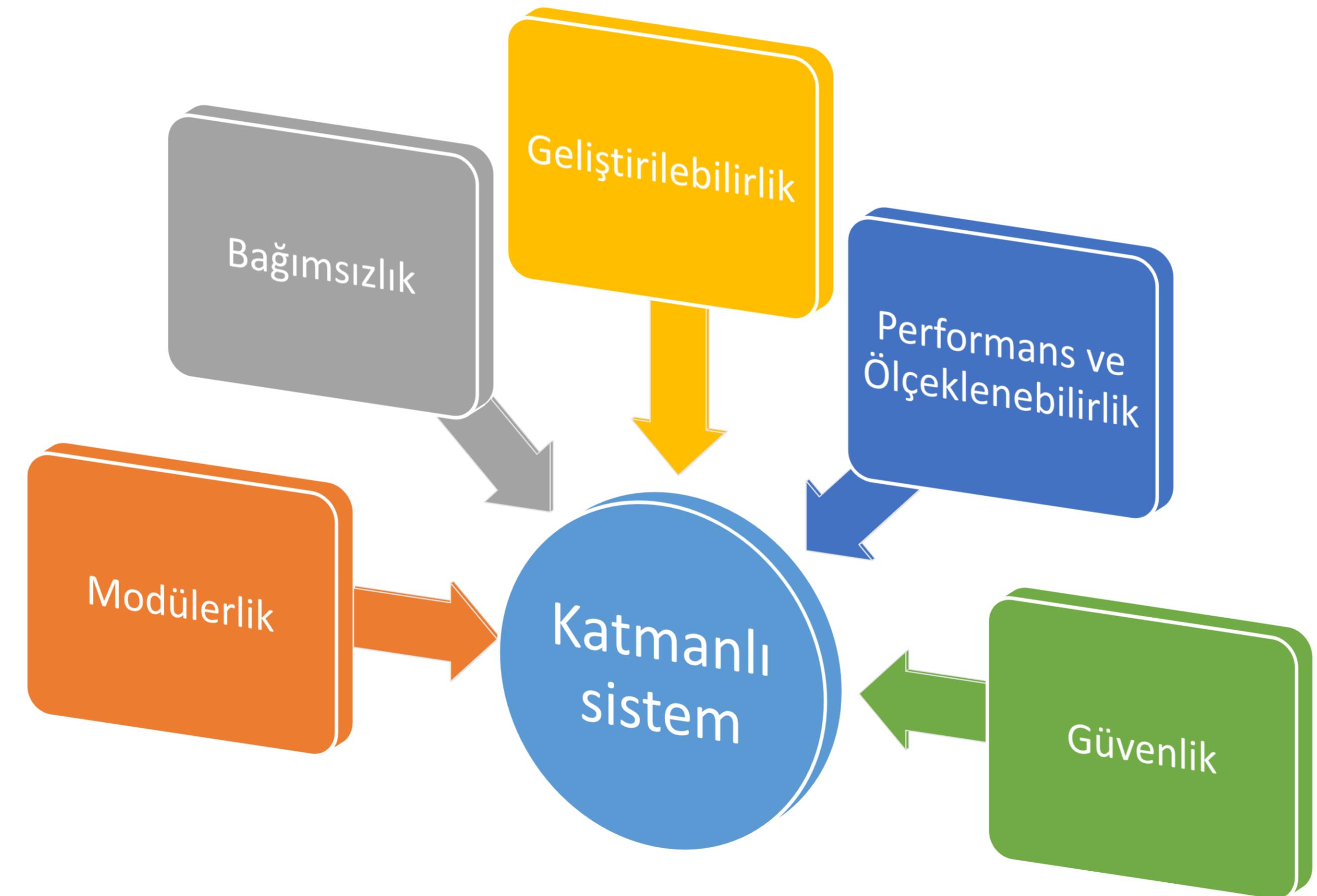
Katmanlı Sistem

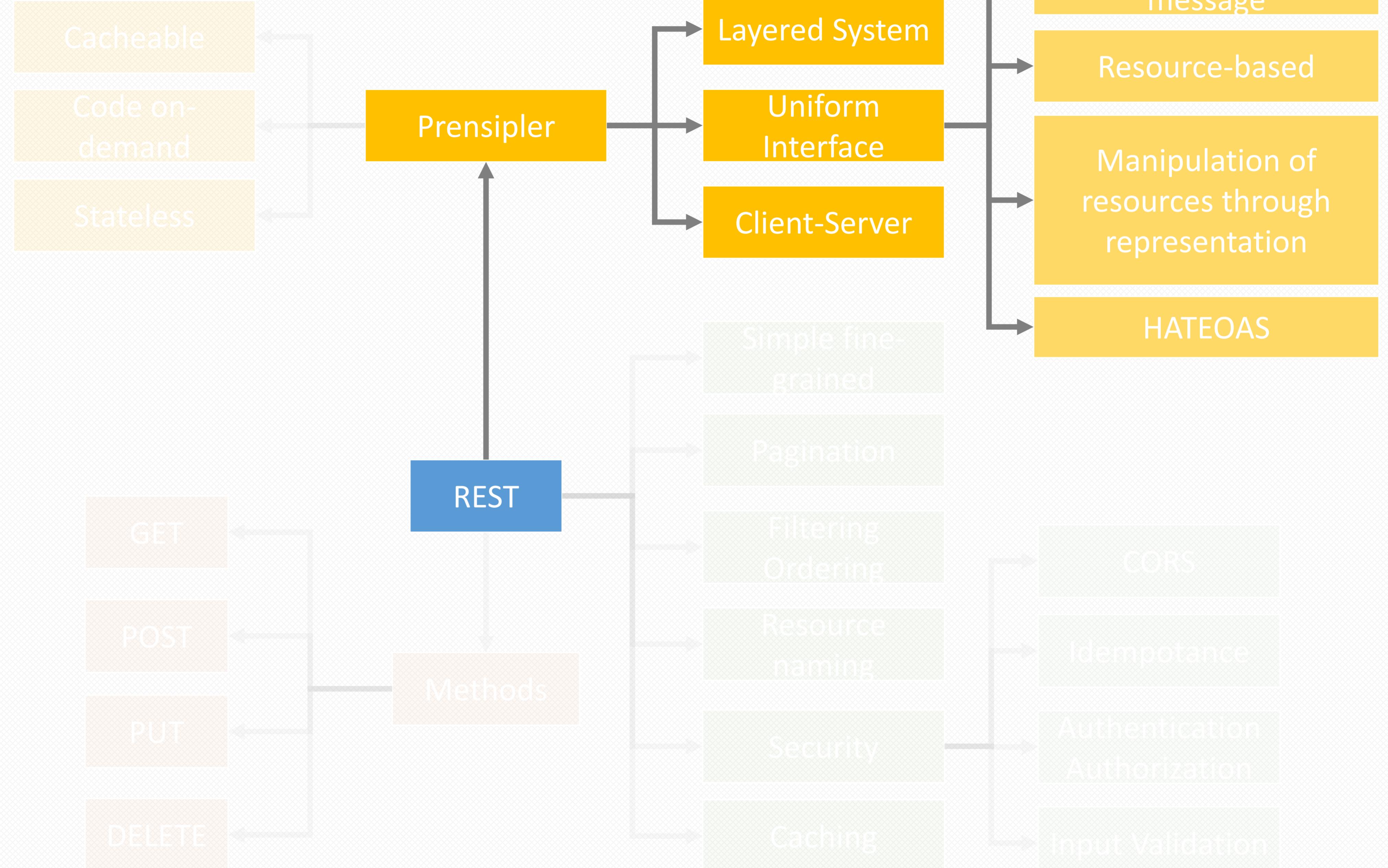
Layered System

- Katmanlı Sistem ([Layered System](#)), RESTful bir [API](#)'nin kısıtları arasında yer alır ve ağırlıklı olarak ağ tabanlı uygulamaların tasarımında önemli bir rol oynar.
- Bu kavram, uygulama mimarilerinin katmanlara ayrılmasını ve bu katmanların farklı görevlere sahip olmasını vurgular.

Katmanlı Sistem

Layered System

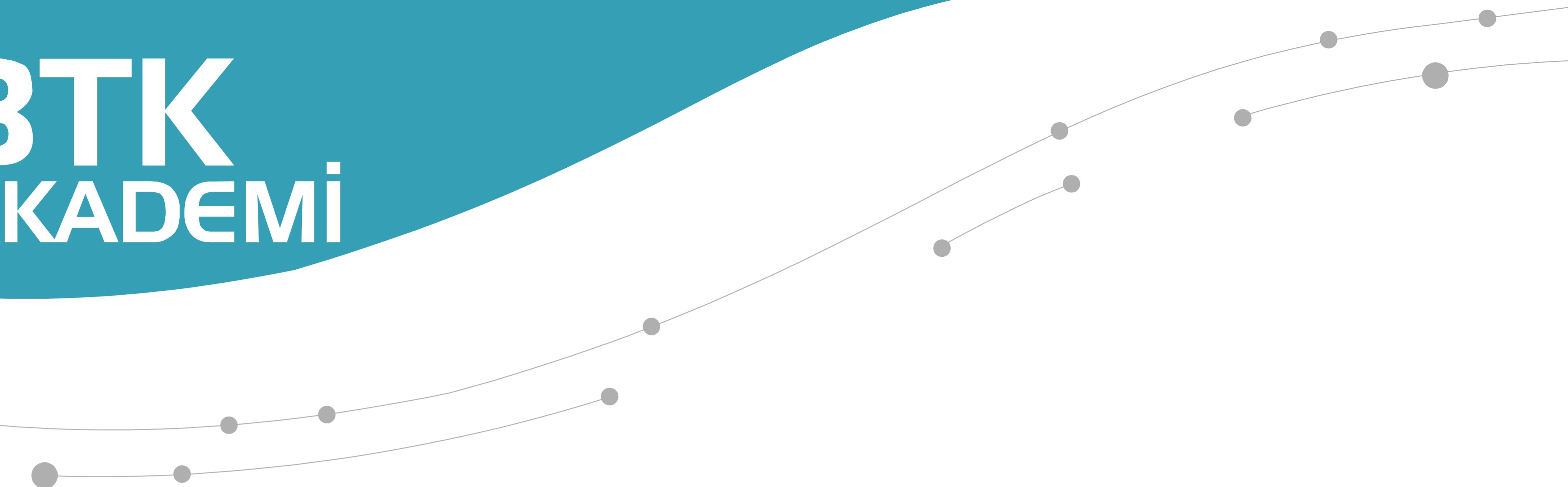
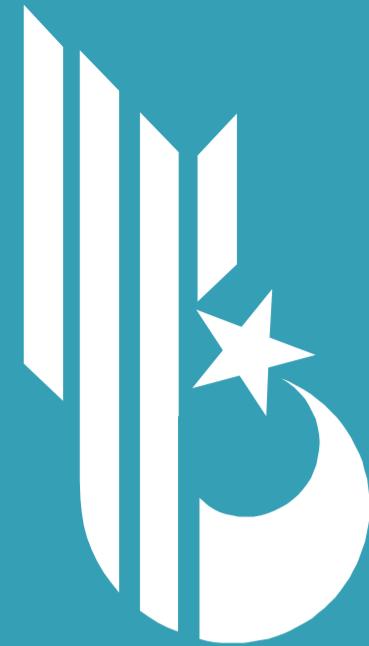




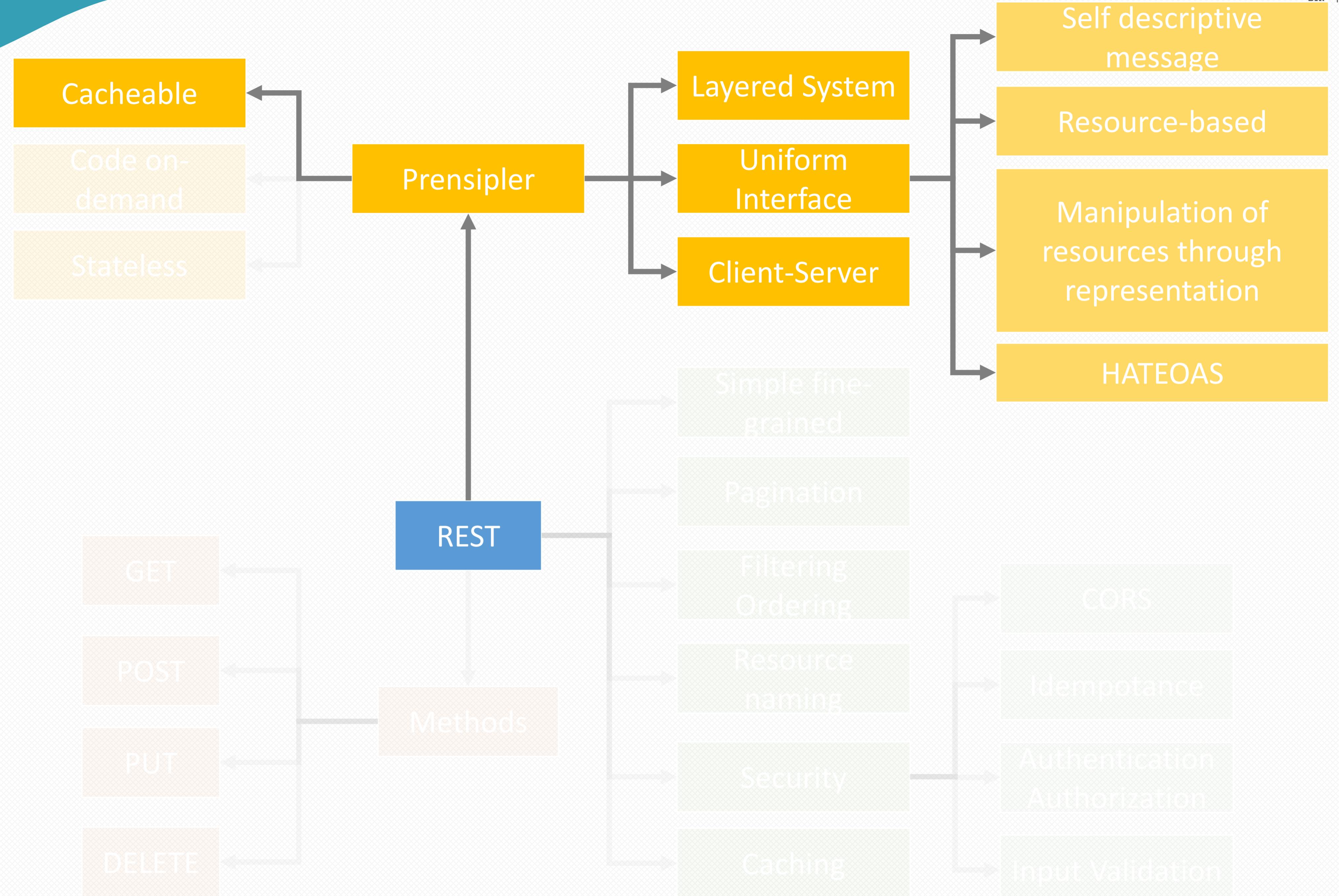
İstemci – Sunucu

Client – Server

- Bu kısıtlama, temelde istemci uygulamalarının ve sunucu uygulamalarının birbirlerine bağımlı olmadan ayrı ayrı geliştirilebilmesi gereği anlamına gelir.
- Bir istemci yalnızca kaynak [URI](#)'larını bilmelidir, başka bir şey değil.
- Günümüzde bu, web geliştirmede standart uygulama olduğundan, özel bir çaba gerektirmez, basit tutulması tavsiye edilir.



*Aralarındaki arabirim değiştirilmediği
surece sunucular ve istemciler ayrı ayrı
değiştirilebilir ve geliştirilebilir.*



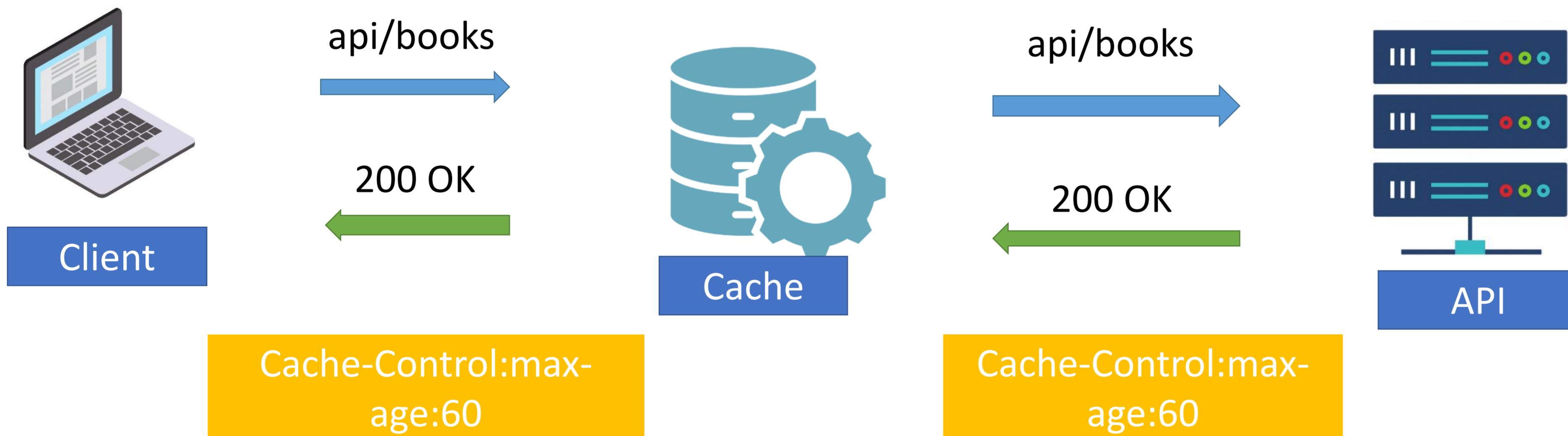
Önbelleğe Alma

Cacheable

- Önbelleğe Alınabilir (**Cacheable**) terimi, RESTful API'lerde bir kaynağın veya yanıtın önbelleğe alınabilir (**cacheable**) olup olmadığını ifade eder.
- Bu kavram, kaynakların veya yanıtların ne kadar süreyle önbellekte saklanabileceğini ve ne zaman güncellenmesi gerektiğini belirler.

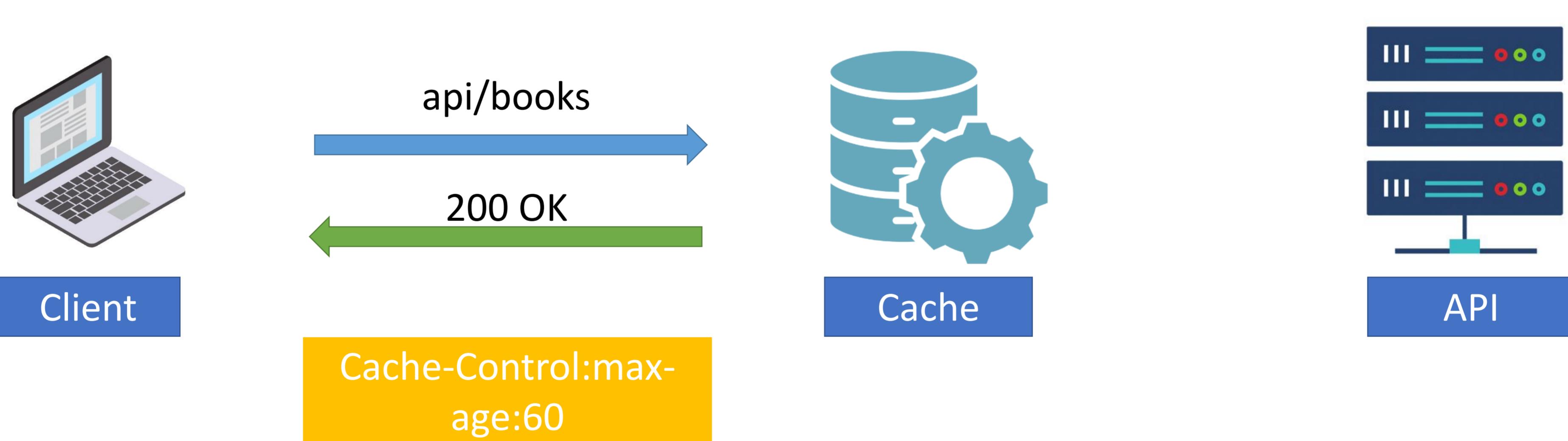
Önbelleğe Alma

Cacheable



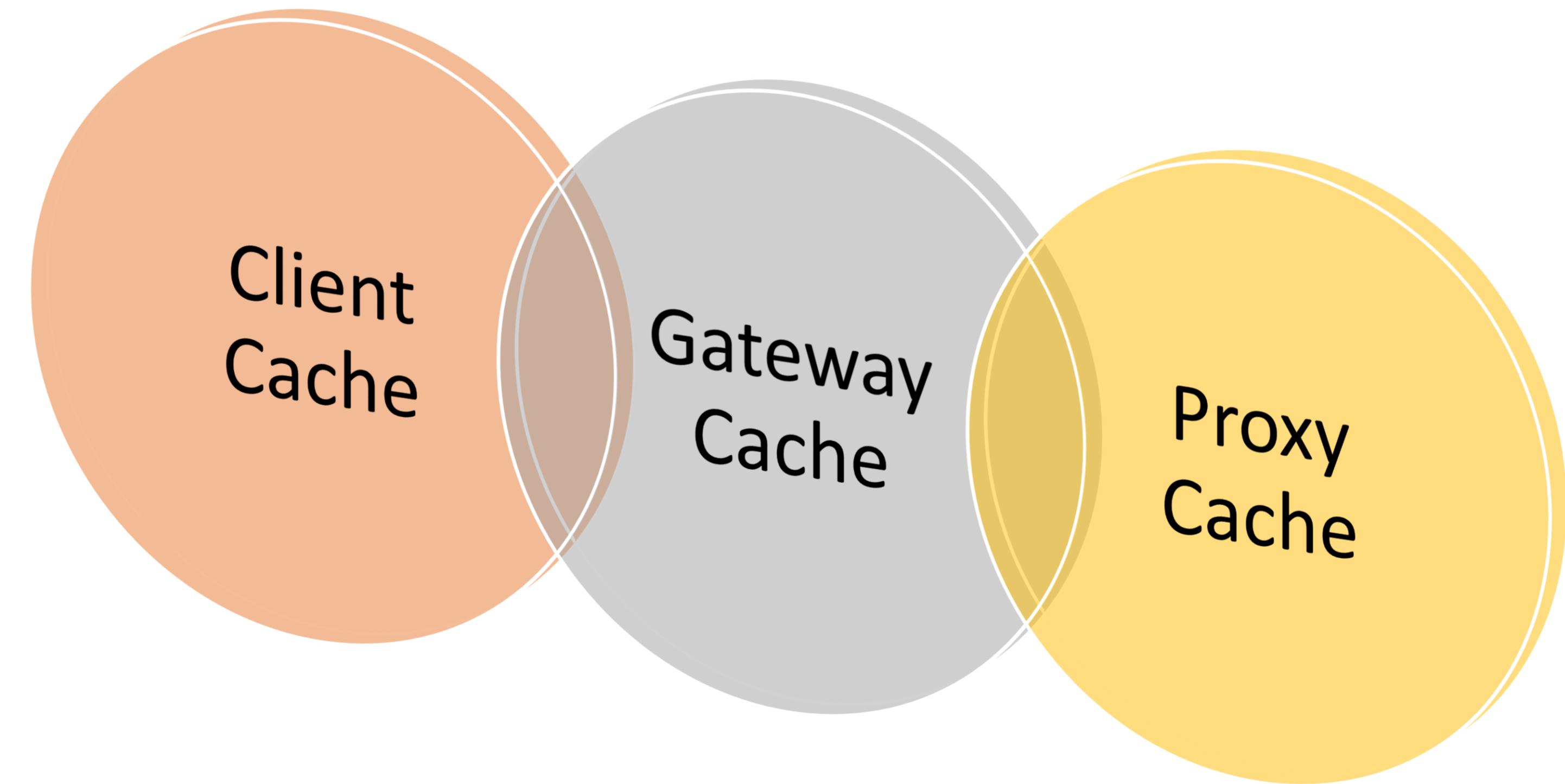
Önbelleğe Alma

Expiration Model



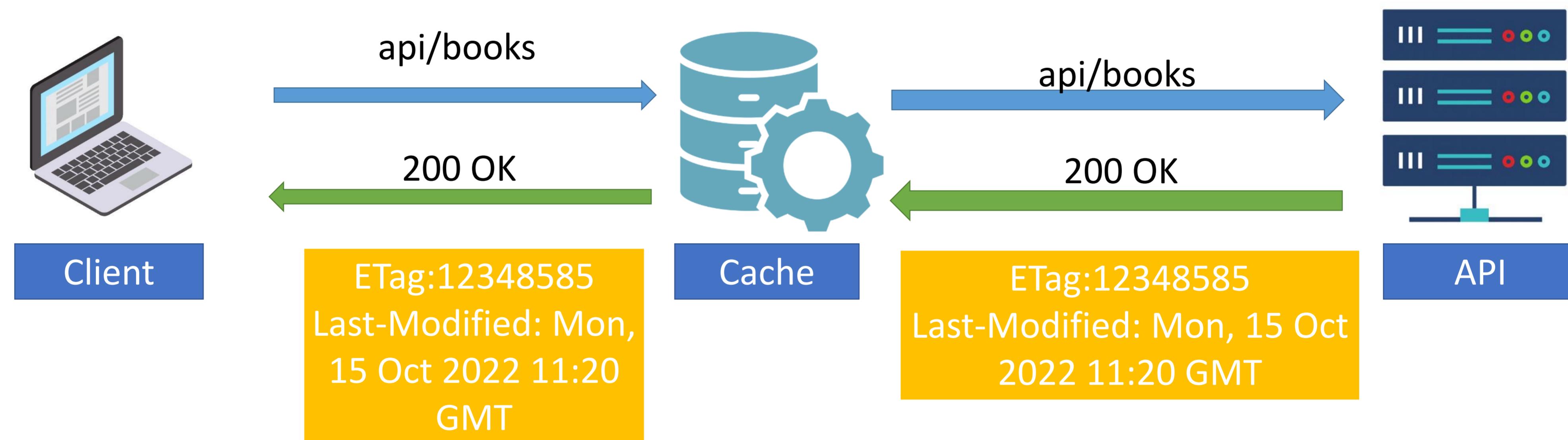
Önbelleğe Alma

Caching



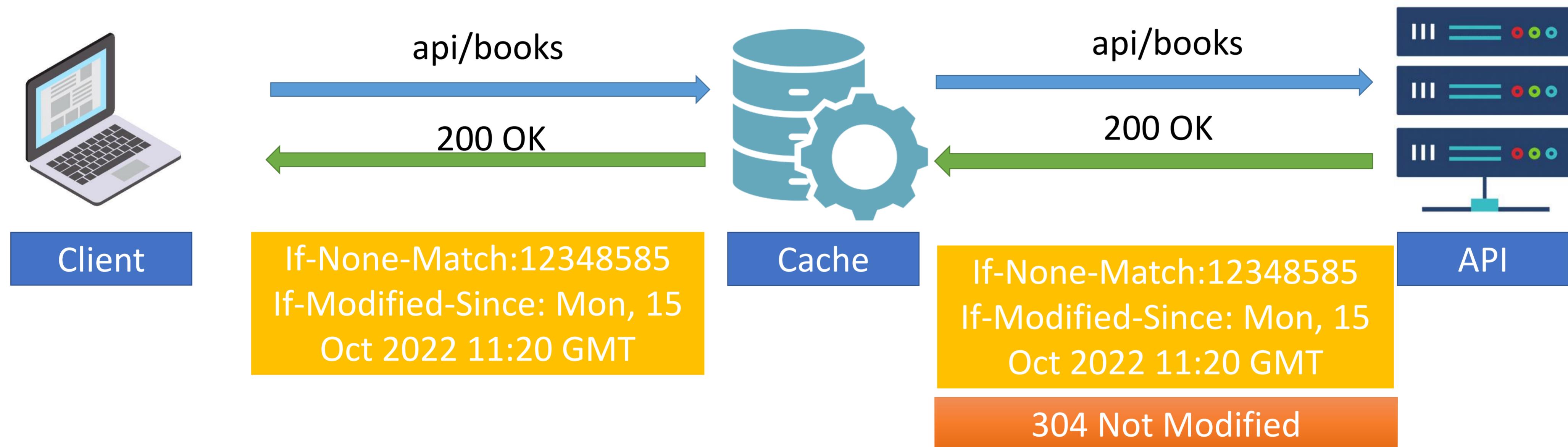
Önbelleğe Alma

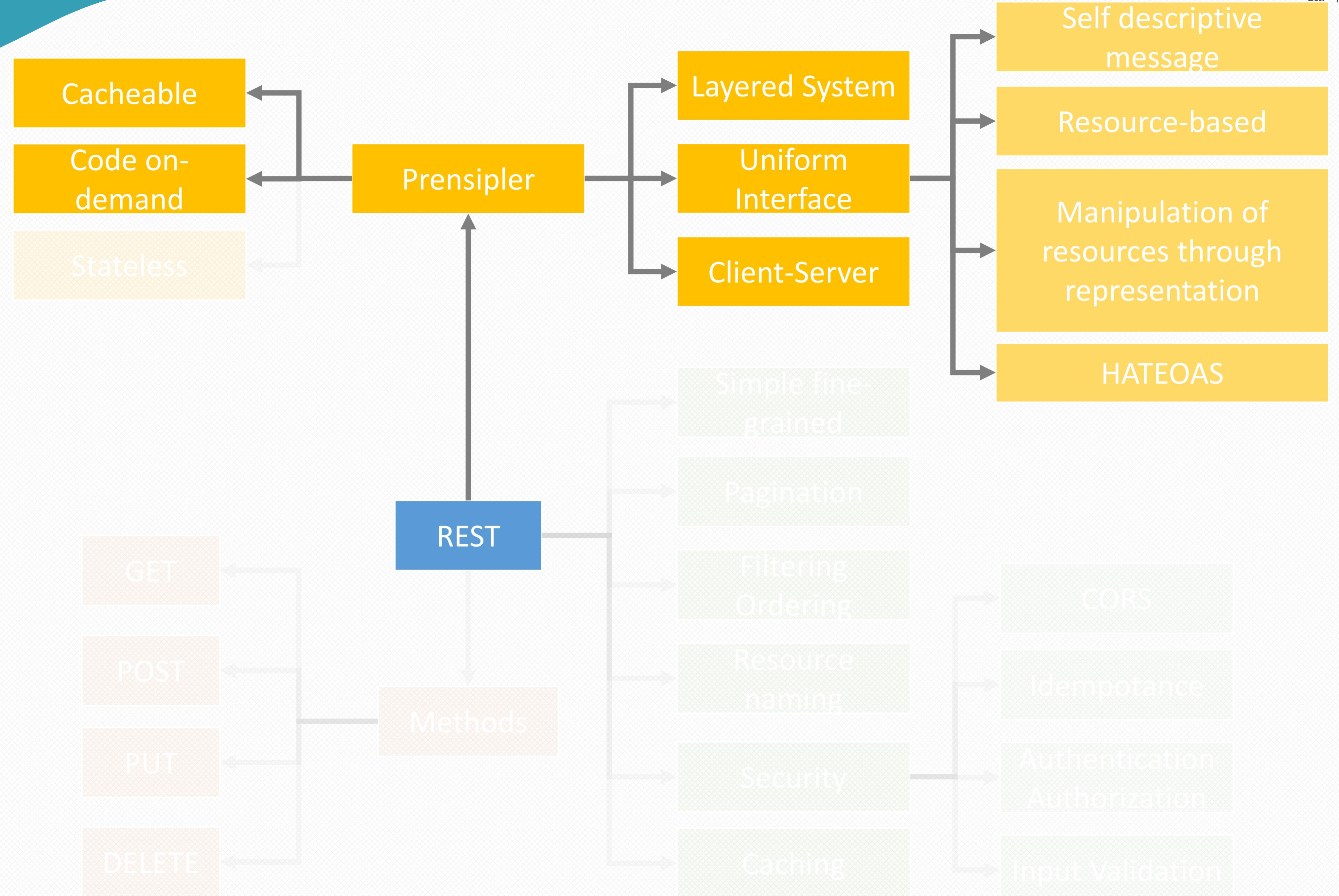
Validation Model



Önbelleğe Alma

Validation Model





Talebe Bağlı Kod

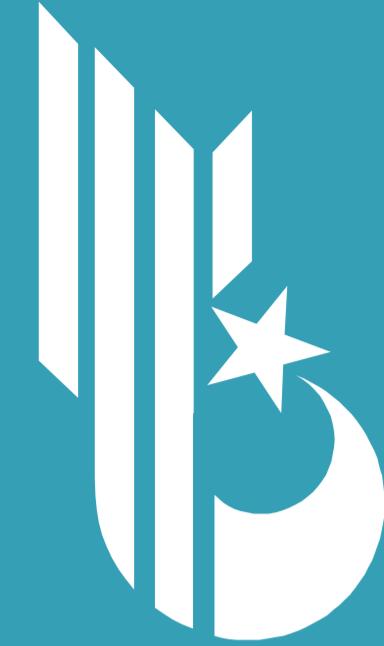
Code on demand

- Bu kısıt isteğe bağlıdır. Çoğu zaman, kaynakların statik temsillerini [XML](#) veya [JSON](#) biçiminde gönderirsiniz.
- Ancak ihtiyaç duyduğunuzda, uygulamanızın bir parçasını desteklemek için yürütülebilir kod döndürme özgürlüğünüz vardır.
- Örneğin, istemciler [API](#)'nizi çağrıarak bir [UI](#) bileşeni oluşturma kodu alabilirler. Buna izin verilir.

Talebe Bağlı Kod

Code on demand

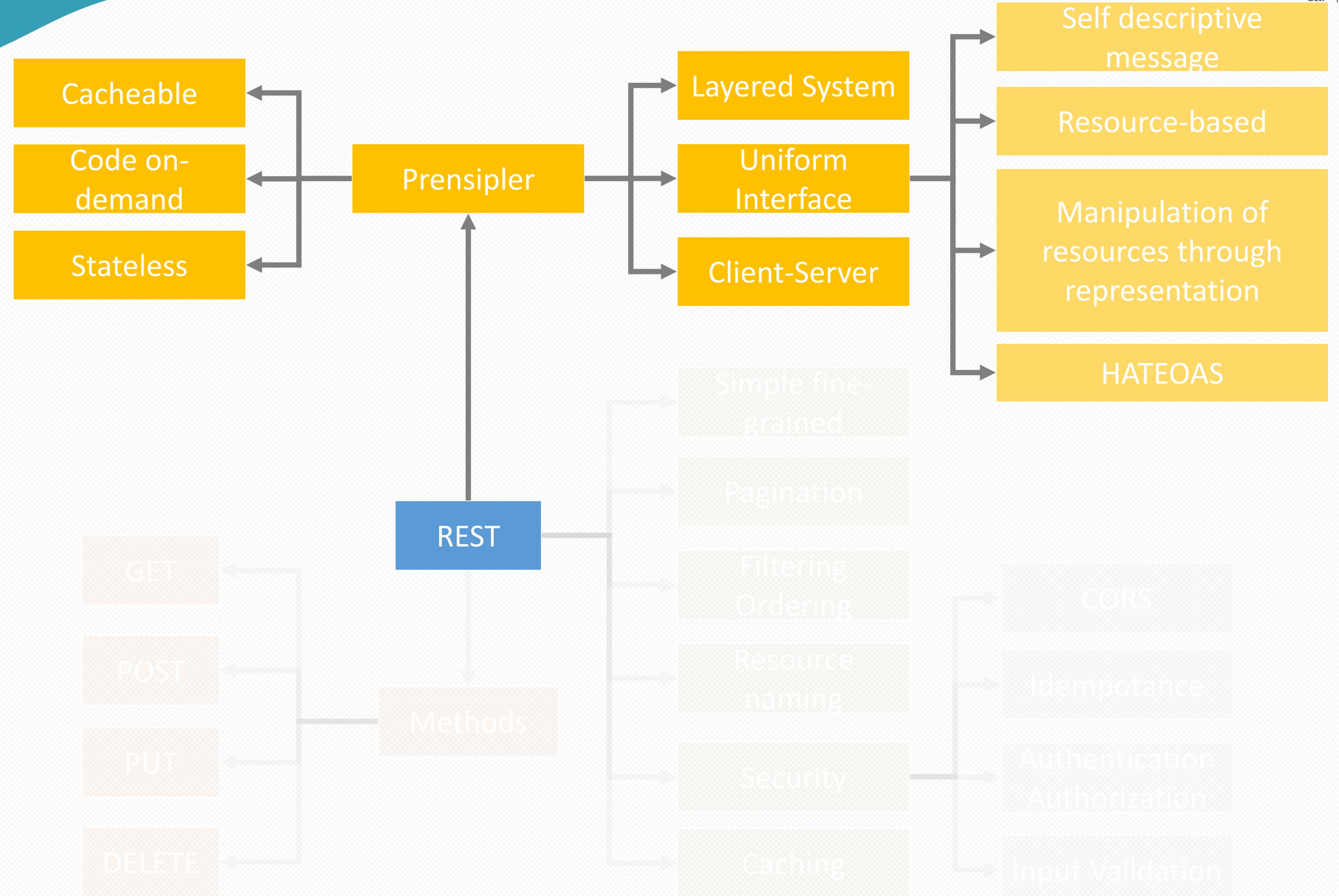
- Tüm kısıtlamalar, gerçekten RESTful bir API oluşturmanıza yardımcı olur ve bunlara uymalısınız.
- Yine de bazen bir veya iki kısıtı ihlal ettiğinizi fark edebilirsiniz. Endişelenmeyin; hala RESTful bir API oluşturuyorsunuz - ancak "gerçekten RESTful" değil.



BTK AKADEMİ

Tüm kısıtlamaların web ile yakından ilişkili olduğunu unutmayın.

RESTful API'ları kullanarak web hizmetlerinize web sayfalarına yaptığınız şeyi yapabilirsiniz.



Durumsuz

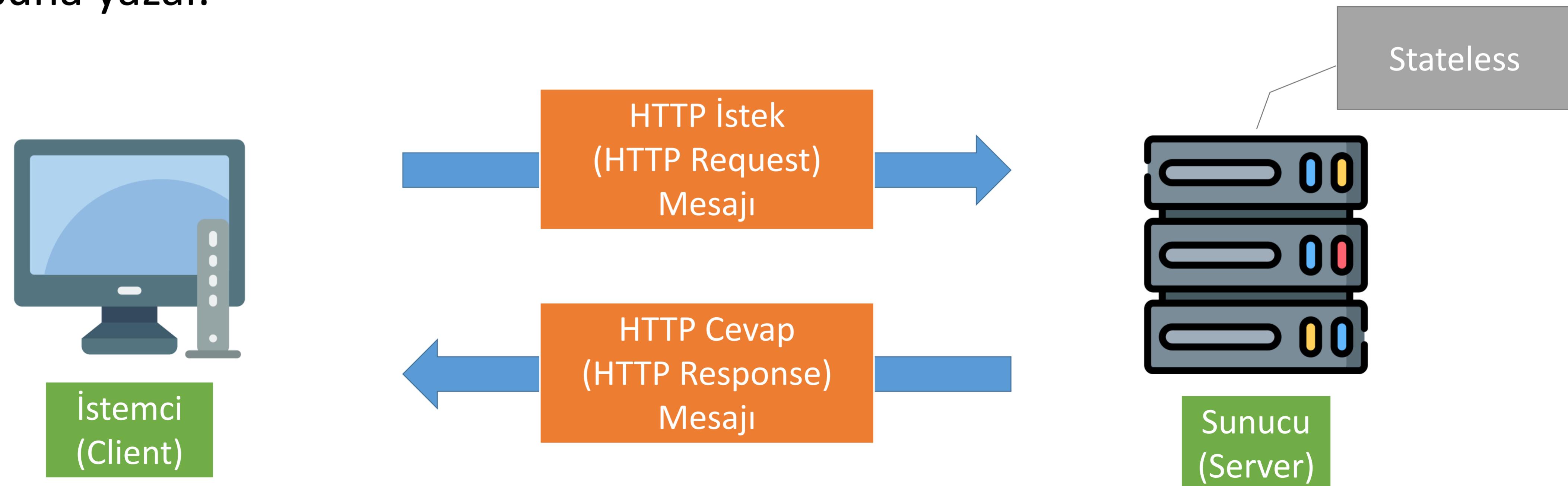
Stateless

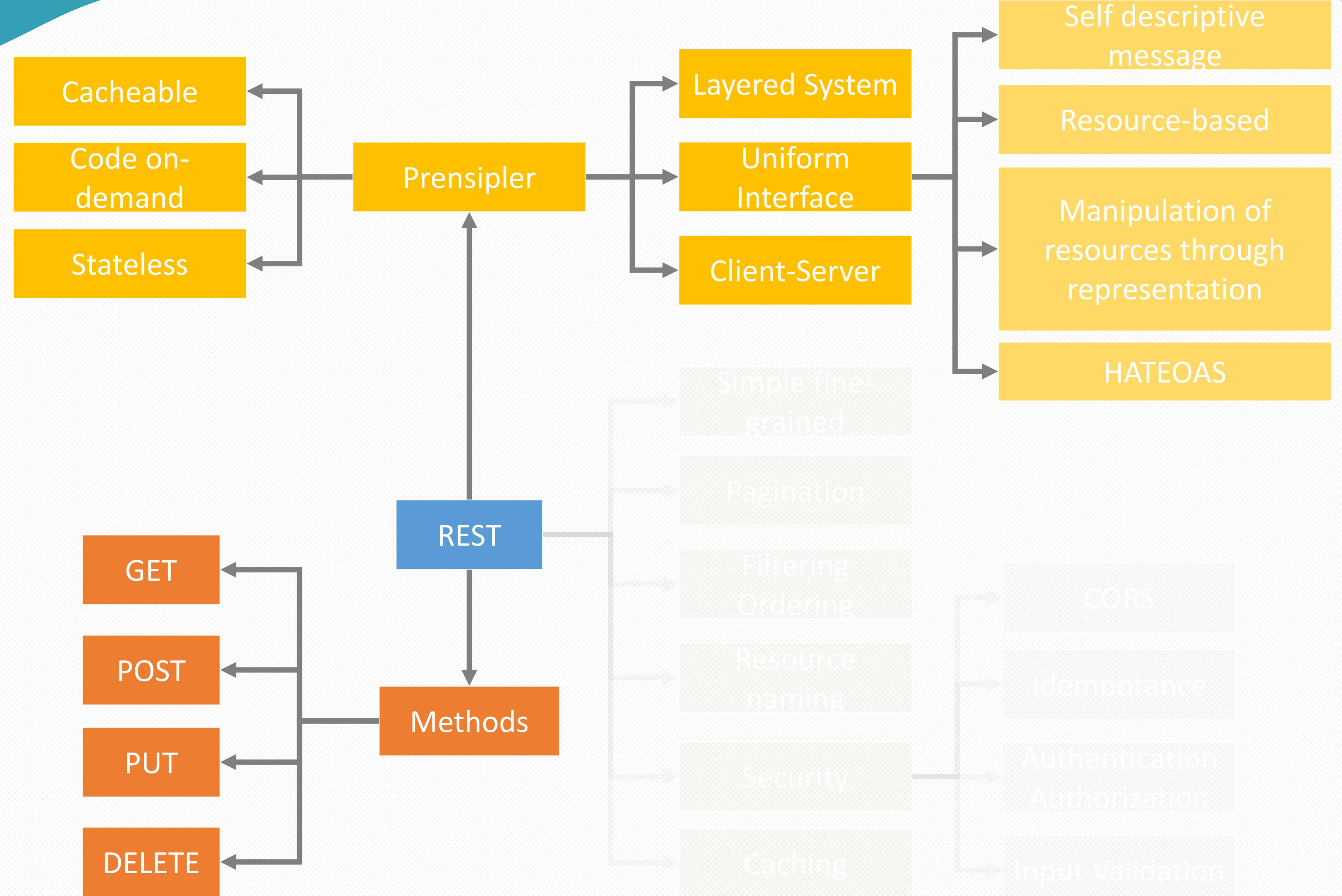
- REST API'leri durumsuz bir istek modeli kullanır.
- HTTP istekleri bağımsız olmalıdır ve herhangi bir sırada gerçekleşebilir, bu nedenle istekler arasında geçici durum bilgisi tutmak mümkün değildir.
- Bilginin depolandığı tek yer kaynaklarının kendisidir ve her istek atomik bir işlem olmalıdır. Bu kısıtlama, web hizmetlerinin yüksek oranda ölçülebilir olmasını sağlar, çünkü istemciler ve belirli sunucular arasında herhangi bir yakınlık tutmaya gerek yoktur.

Durumsuz

Stateless

- Herhangi bir sunucu, herhangi bir istemciden gelen herhangi bir talebi karşılayabilir.
- Bununla birlikte, diğer faktörler ölçülebilirliği sınırlayabilir. Örneğin, birçok web hizmeti, ölçeklendirilmesi zor olabilecek bir arka uç veri deposuna yazar.





HTTP Metotları

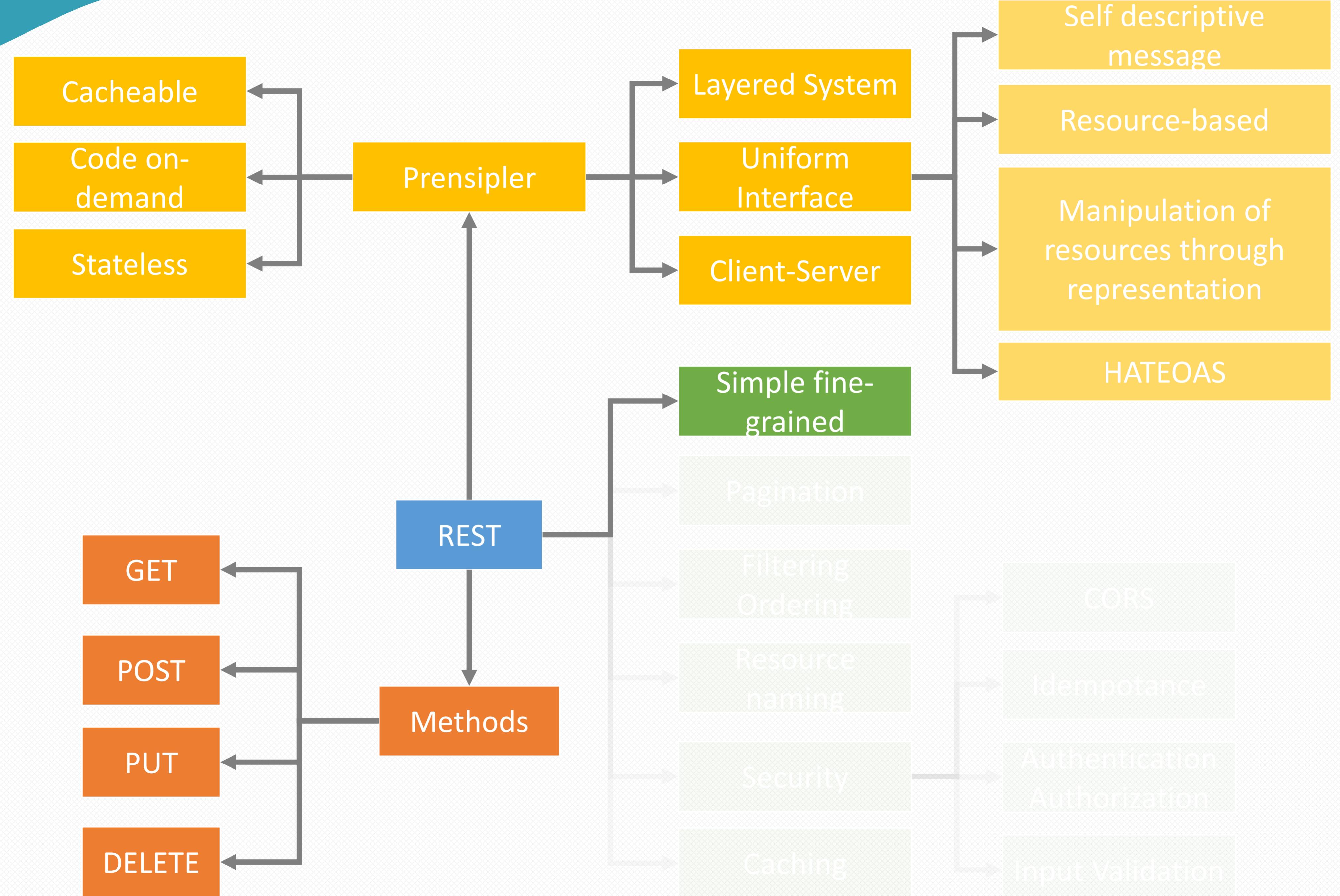
- **GET**, belirtilen **URI**'deki kaynağın bir temsilini alır. Yanıt iletişimiminin gövdesi istenen kaynağın ayrıntılarını içerir.
- **POST**, belirtilen **URI**'de yeni bir kaynak oluşturur. İstek iletişimiminin gövdesi yeni kaynağın ayrıntılarını sağlar. **POST** ayrıca gerçekle kaynak oluşturmayan işlemleri tetiklemek için de kullanılabilir.
- **PUT**, belirtilen **URI**'deki kaynağı oluşturur veya değiştirir. İstek iletişimiminin gövdesi oluşturulacak veya güncellenecek kaynağı belirtir.

HTTP Metotları

- **PATCH**, bir kaynağın kısmi güncellemesini yapar. İstek gövdesi, kaynağa uygulanacak değişiklik kümesini belirtir.
- **DELETE**, belirtilen **URI**'deki kaynağı kaldırır.

HTTP Metotları

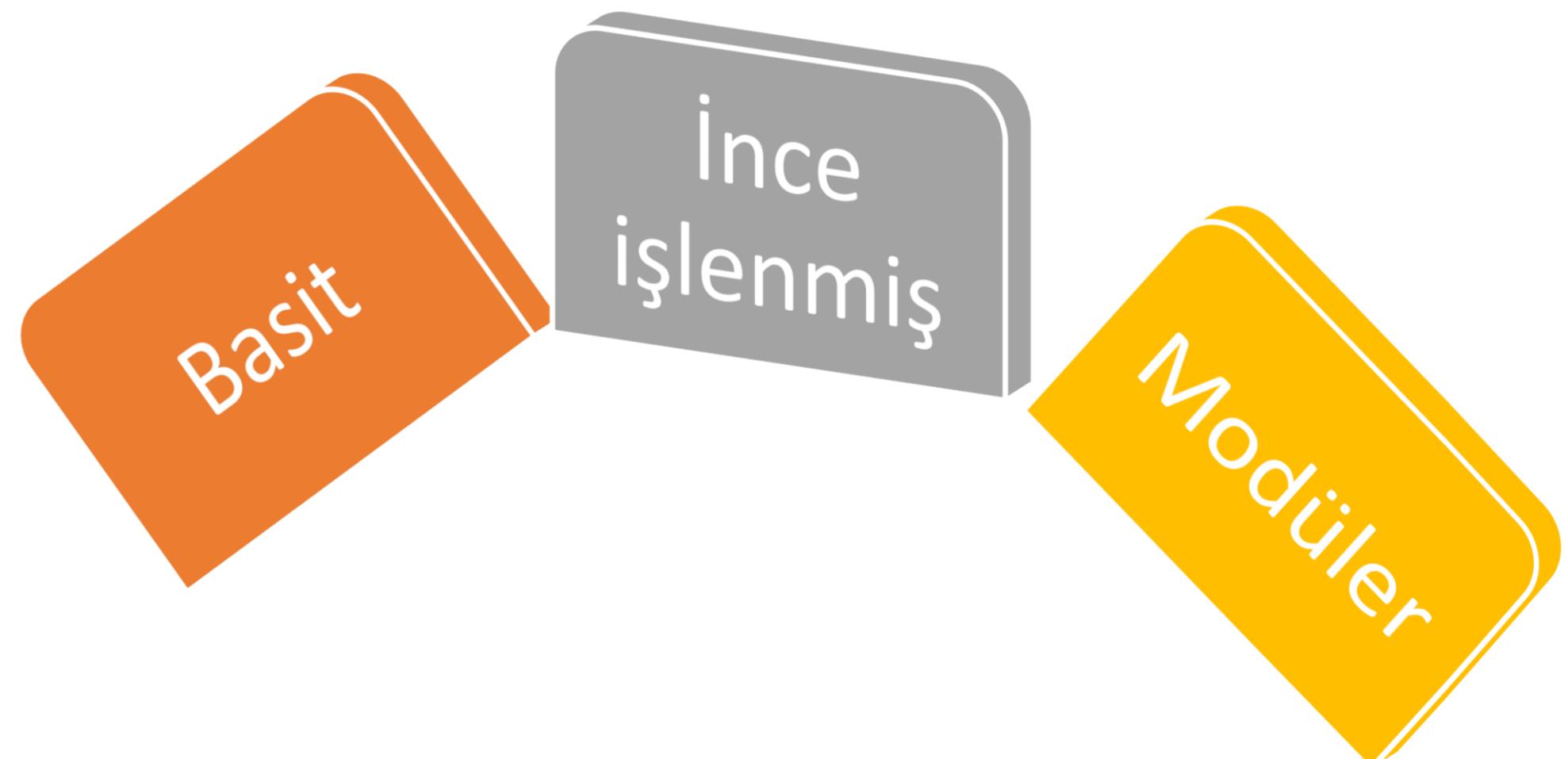
Kaynak	POST	GET	PUT	DELETE
/customers	Yeni müşteri oluşturma	Tüm müşterileri getirme	Müşterilerin toplu olarak güncelleme	Tüm müşterileri silme
/customers/1	Hata	1 numaralı müşterinin bilgilerini getirme	Eğer varsa 1 numaralı müşterilerin bilgilerini güncelleme	1 numaralı müşteriyi silme
/customers/1/orders	1 numaralı müşteri için sipariş oluşturma	1 numaralı müşterinin siparişlerini getirme	1 numaralı müşterinin siparişlerini toplu olarak güncelleme	1 numaralı müşterinin tüm siparişlerini silme



Basit ve İnce İşlenmiş

Simple fine-grained

- Basit ve İnce İşlenmiş ([Simple fine-grained](#)) bir [REST API](#) tasarıımı yaklaşımını ifade eder.
- Bu yaklaşım, [API](#)'de kaynakları ve işlemleri mümkün olduğunda basit ve özgün bir şekilde temsil etmeyi amaçlar.



Basit ve İnce İşlenmiş

Simple fine-grained

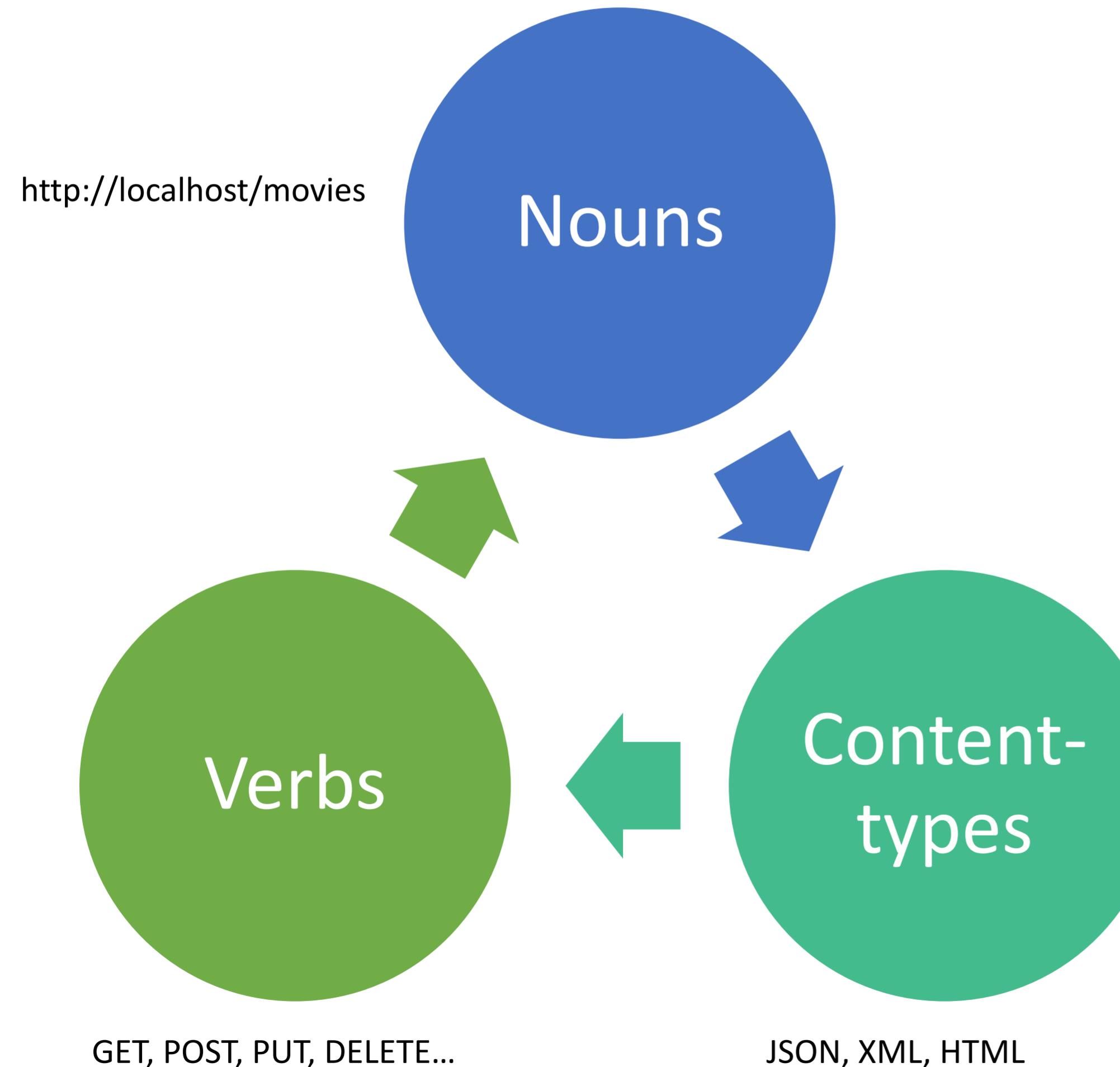
/books: HTTP GET kullanarak tüm kitapları alabilirsiniz.

/books/{bookId}: HTTP GET kullanarak belirli bir kitabın ayrıntılarını alabilirsiniz.

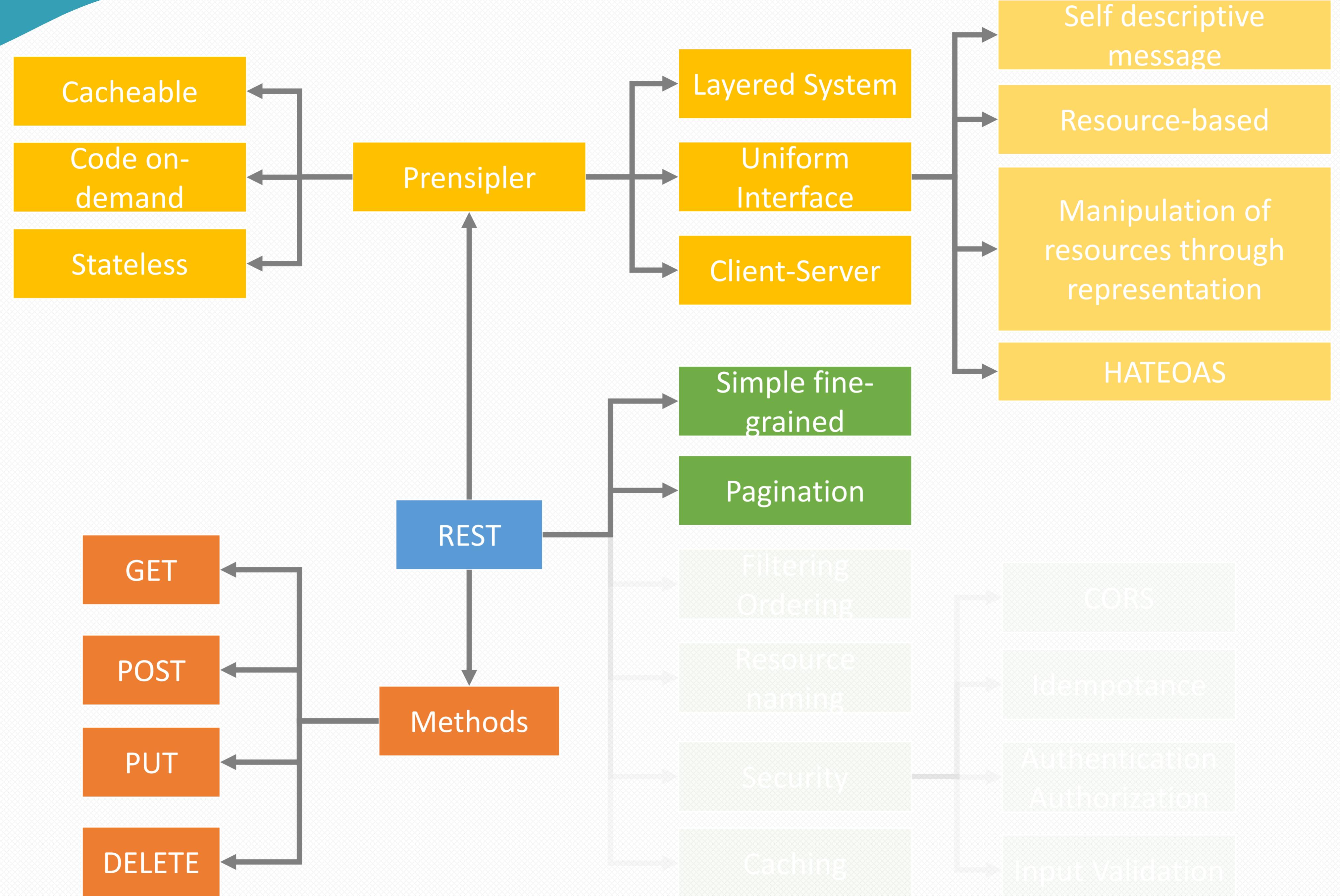
/books/{bookId}/reviews: Belirli bir kitabın incelemelerini alabilirsiniz.

Basit ve İnce İşlenmiş

Simple fine-grained



GET	/movies
GET	/movies/{movield}
POST	/movies
PUT	/movies/{movield}
DELETE	/movies/{movield}



Sayfalama

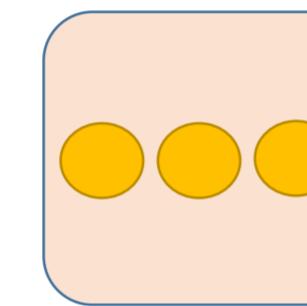
Pagination

- Sayfalama ([Pagination](#)), bir [REST API](#) tasarımının önemli bir özelliğidir ve büyük veri kümelerini daha yönetilebilir parçalara bölmeyi sağlar.
- Bu özellik, kullanıcıların büyük veri kaynaklarına daha etkili bir şekilde erişmelerini ve sorgulamalarını kolaylaştırır.
- Birçok [RESTful API](#), kullanıcılarla tüm verileri tek bir istekte sunmak yerine sayfa sayfa veya belirli bir miktarda veri sunarak çalışır. Bu, büyük veri kümelerini daha küçük parçalara böler ve her birini ayrı bir sayfa olarak sunar. Kullanıcılar sayfalar arasında gezinebilir ve sorgularını daha iyi kontrol edebilirler.

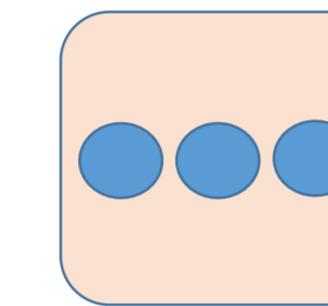
Sayfalama

Pagination

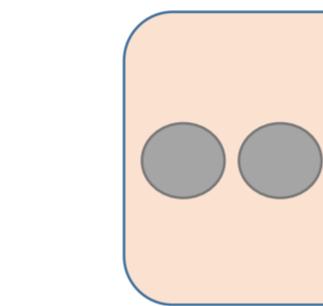
Page = 1



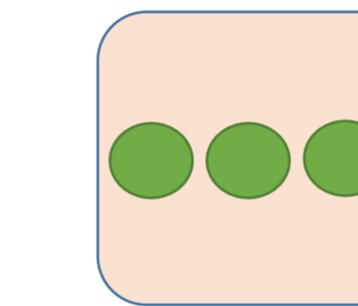
Page = 2



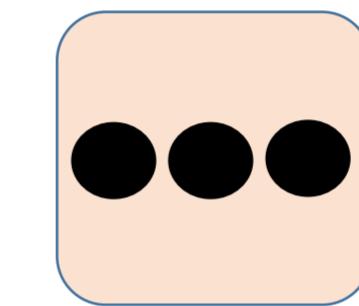
Page = 3



Page = 4



Page = 5



PageSize = 3

İlk Sayfa:

/books?page=1&limit=3

İkinci Sayfa:

books?page=2&limit=3

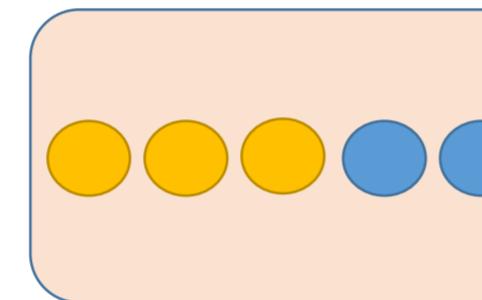
Üçüncü Sayfa:

/books?page=3&limit=3

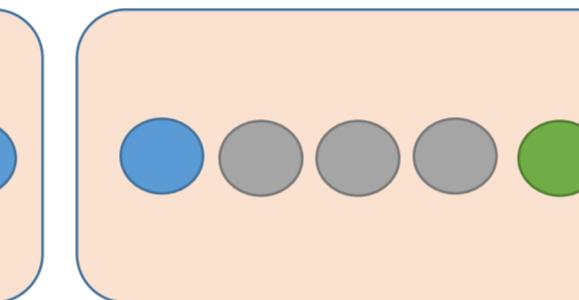
Sayfalama

Pagination

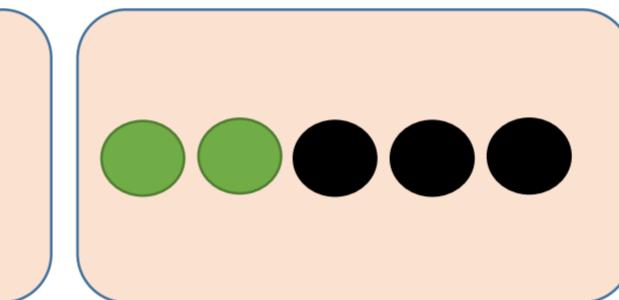
Page = 1



Page = 2



Page = 3



PageSize = 5

İlk Sayfa:

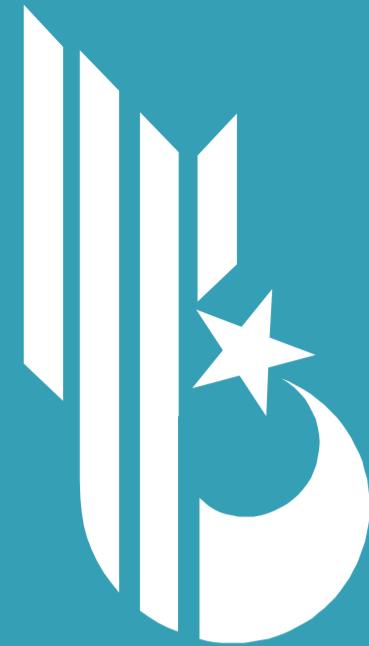
/books?page=1&limit=5

İkinci Sayfa:

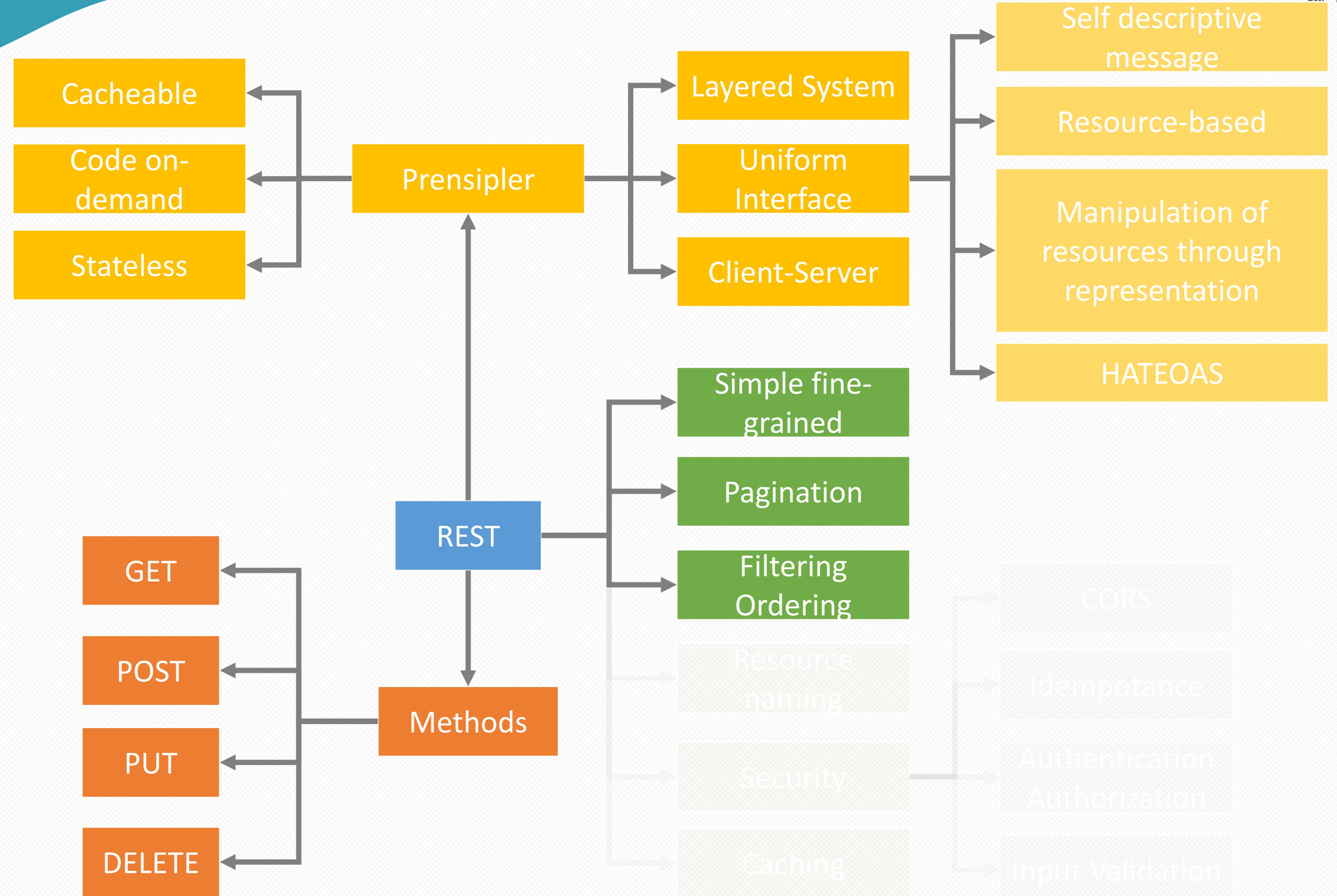
/books?page=2&limit=5

Üçüncü Sayfa:

/books?page=3&limit=5



Pagination, büyük veri kümelerini daha yönetilebilir ve kullanıcı dostu hale getirerek REST API'lerin performansını artırmanın yaygın kullanılan bir yoludur.



Filtreleme ve Sıralama

Filtering and Ordering

- Filtreleme ([Filtering](#)), bir REST API tasarımının önemli bir özelliğidir ve kullanıcıların belirli kriterlere uyan verileri sorgulayarakmasına olanak tanır.
- Bu, büyük veri kümelerlerini daraltmanıza ve yalnızca belirli bir şartı karşılayan verileri almanızı yardımcı olur.

Filtreleme ve Sıralama

Filtering and Ordering

Vasıta

- Otomobil (427.158)
- Arazi, SUV & Pickup (78.108)
- Motosiklet (76.234)
- Minivan & Panelvan (83.947)
- Ticari Araçlar (40.665)
- Elektrikli Araçlar (4.790)
- Kiralık Araçlar (10.377)
- Deniz Araçları (6.892)
- Hasarlı Araçlar (3.943)
- Klasik Araçlar (2.114)
- Modifiye Araçlar (52)
- Hava Araçları (27)
- ATV (1.829)
- İTV

Otomobil

- Acura (1)
- Aion (1)
- Alfa Romeo (1.018)
- Anadol (16)
- Aston Martin (47)
- Audi (15.705)
- Bentley (81)
- BMW (26.976)
- Bugatti (1)
- Buick (4)
- Cadillac (50)
- Chery (101)
- Chevrolet (3.802)
- Chrysler (200)

[Bu Kategorideki Tüm İlanlar](#)

Vasıta

Otomobil

Volkswagen

- Arteon (191)
- Beetle (415)
- Bora (1.439)
- EOS (32)
- Golf (12.775)
- ID.3 (3)
- Jetta (8.102)
- Lupo (23)

Adres

Türkiye

İl

İlçe

Fiyat

TL USD EUR GBP

min TL – max TL

Yakıt

Benzin

Benzin & LPG

Dizel

Hybrid

Elektrik

/books?minPrice=20

/books?maxPrice=300

/books?minPrice=20&maxPrice=300

Filtreleme ve Sıralama

Filtering and Ordering



/movies



/movies?year=2020



/movies?year=2021



/movies



/movies?genre=comedy



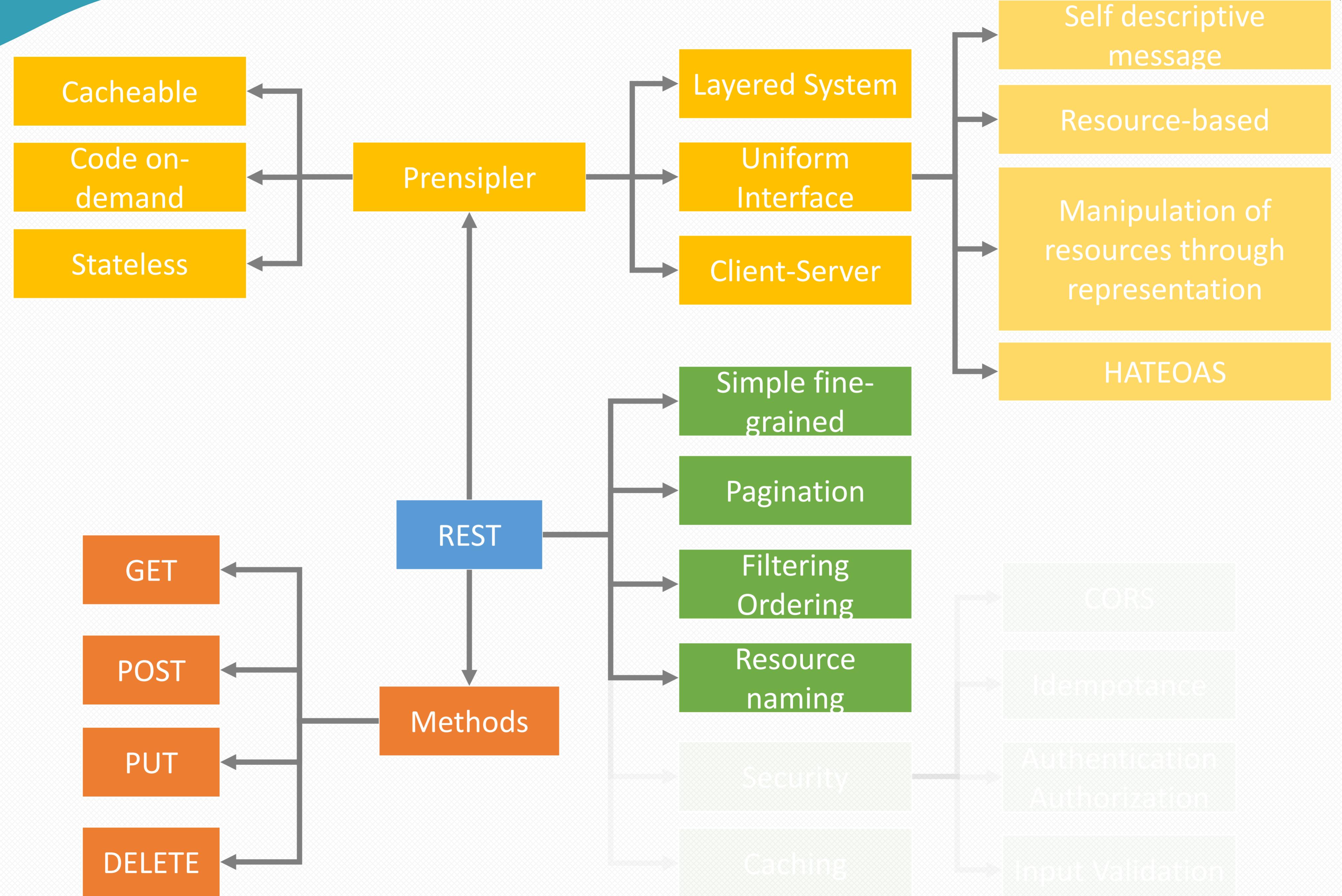
/movies?genre=action

Filtreleme ve Sıralama

Filtering and Ordering

Sorting

- /books?orderBy=title
- /books?orderBy=title desc
- /books?orderBy=price
- /books?orderBy=price desc



Kaynak İsimlendirme

Resource Naming

- Varlıklara Odaklanmak
- Web API'nin sunduğu iş varlıklara (**entities**) odaklanılır. Örneğin, bir e-ticaret sisteminde, ana varlıklar müşteriler (**customers**) ve siparişler (**orders**) olabilir.

İyi tanımlanmış bir örnek:

- <https://adventure-works.com/orders>

Kötü tanımlanmış bir örnek:

- <https://adventure-works.com/create-order>

Kaynak İsimlendirme

Resource Naming

- Bir kaynak tek bir fiziksel öğeye dayanmak zorunda değildir.
- Örneğin, bir sipariş kaynağı, içsel olarak bir ilişkisel veri tabanındaki birkaç tablo olarak uygulanabilir, ancak istemciye tek bir varlık olarak sunulabilir.
- API'ler oluştururken, sadece bir veritabanının iç yapısını yansıtan API'ler oluşturmaktan kaçının. REST'in amacı varlıklarını ve bir uygulamanın bu varlıklar üzerinde gerçekleştirebileceği işlemleri modellemektir. *İstemci içsel uygulamaya maruz bırakılmamalıdır.*

Kaynak İsimlendirme

Resource Naming

- **Varlıklar koleksiyonlar ile gruplandırılabilir**
- Varlıklar genellikle koleksiyonlara (siparişler, müşteriler) gruplandırılır. Bir koleksiyon, koleksiyon içindeki öğeden ayrı bir kaynaktır ve kendi **URI'sine** sahip olmalıdır.

`https://adventure-works.com/orders`

`https://adventure-works.com/customers`

Kaynak İsimlendirme

Resource Naming

- **Koleksiyonlar ve öğeler hiyerarşik bir şekilde düzenlenir**
- **URI'lerde tutarlı bir adlandırma kuralı benimseyin.** Genel olarak, koleksiyonları referans alan **URI'ler** için çoğul isimler kullanmak faydalı olur.
- Koleksiyonlar ve öğeler için **URI'leri** bir hiyerarşi içinde düzenlemek iyi bir uygulamadır.

/customers

/customers/23

/customers/23/orders

Kaynak İsimlendirme

Resource Naming

- Kaynaklar arasındaki ilişkilerin nasıl temsil edileceği dikkate alınmalıdır
- Farklı kaynaklar arasındaki ilişkiler ve bu ilişkilerin nasıl temsil edileceği iyi tasarlanmalıdır.
- Karmaşıklık arttıkça; bakım zorlaşacaktır.

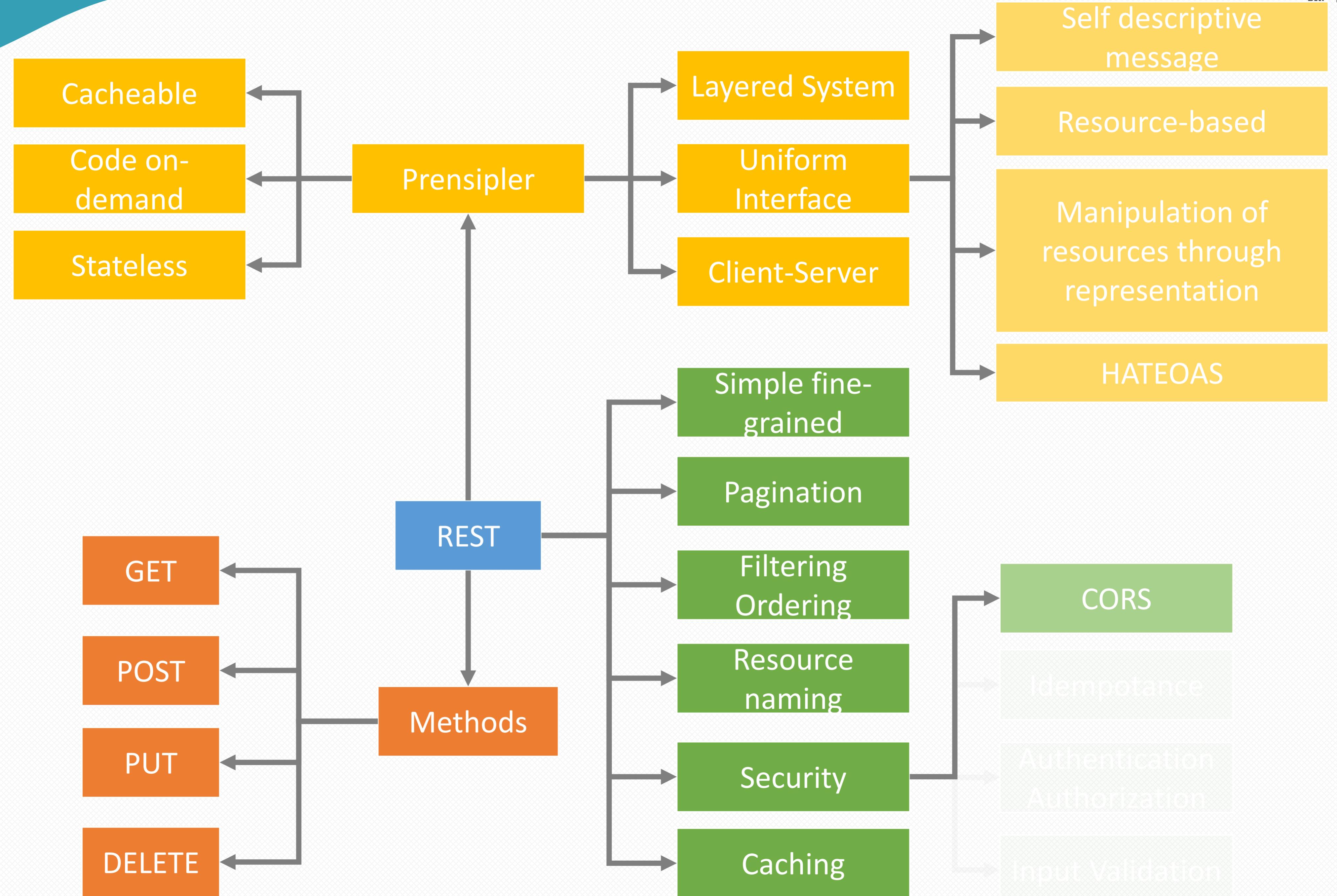


```
/customers/5/orders
```

Kaynak İsimlendirme

Resource Naming

- **Web sunucusundaki yükü dikkat alın**
- Başka bir faktör, tüm web isteklerinin web sunucusuna bir yük getirmesidir. Daha fazla istek, daha büyük bir yük demektir.
- Bu nedenle, çok sayıda küçük kaynağı açıklayan "konuşkan giriş/çıkış ([Chat I/O](#))" ve Fazladan Getirme ([Extraneous Fetching antipattern](#)) web API'lerinden kaçınmaya çalışın.



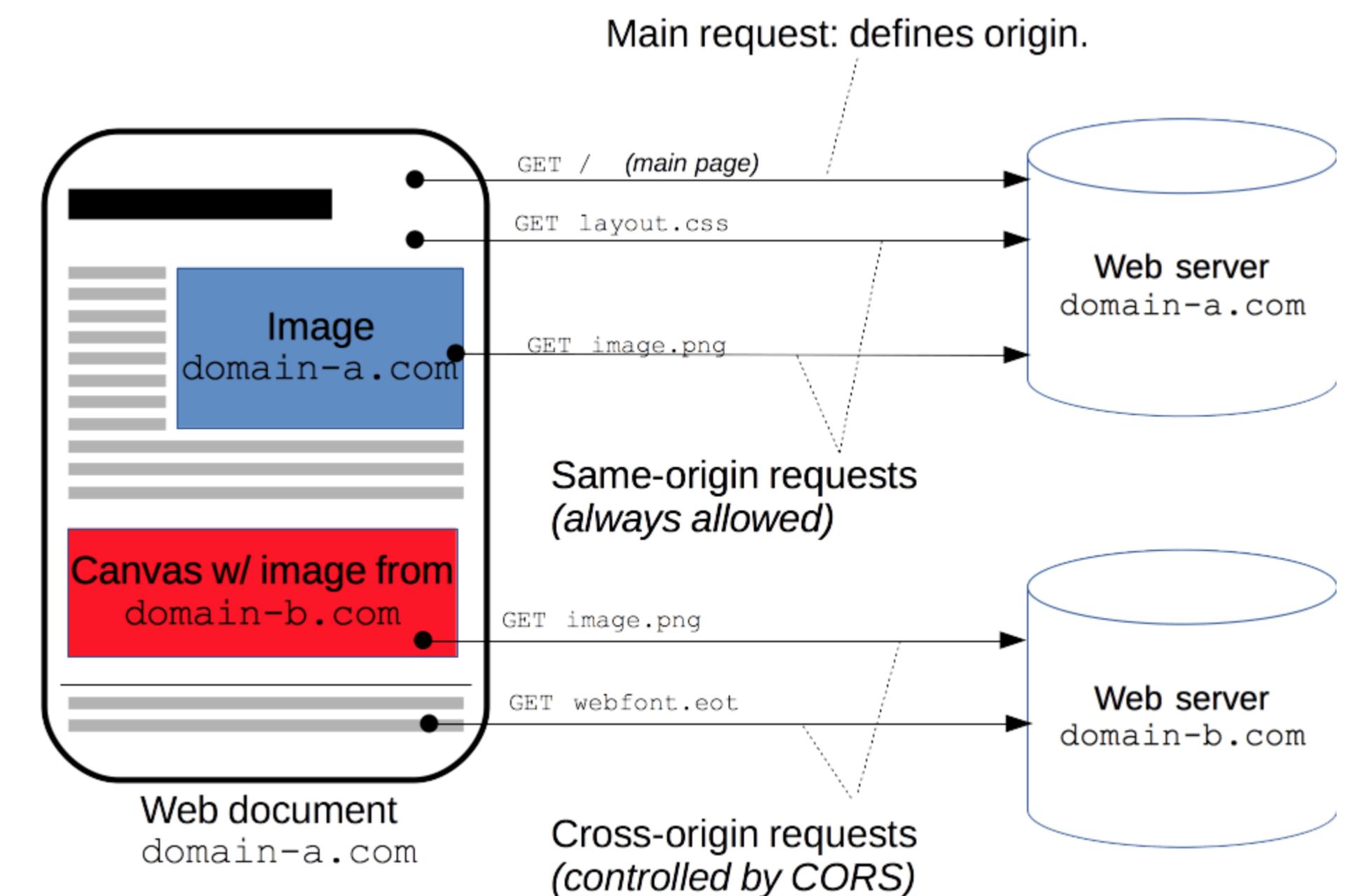
Güvenlik

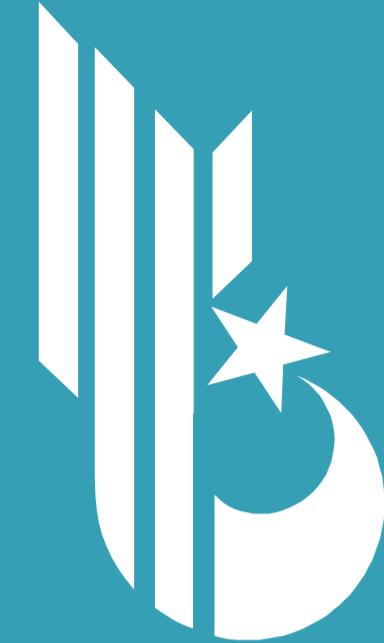
CORS

- "Cross-Origin Resource Sharing (CORS)" kelimelerinin baş harflerinden oluşan bir kısaltmadır ve web uygulamalarının güvenli bir şekilde farklı kaynaklardan (origin) veri almasına ve göndermesine izin veren bir güvenlik önlemidir.
- Temel olarak, CORS, bir web uygulamasının bir kaynaktan (örneğin bir alan adı veya IP adresi) diğer bir kaynağına veri talep etmesine veya veri göndermesine izin veren bir mekanizmadır.

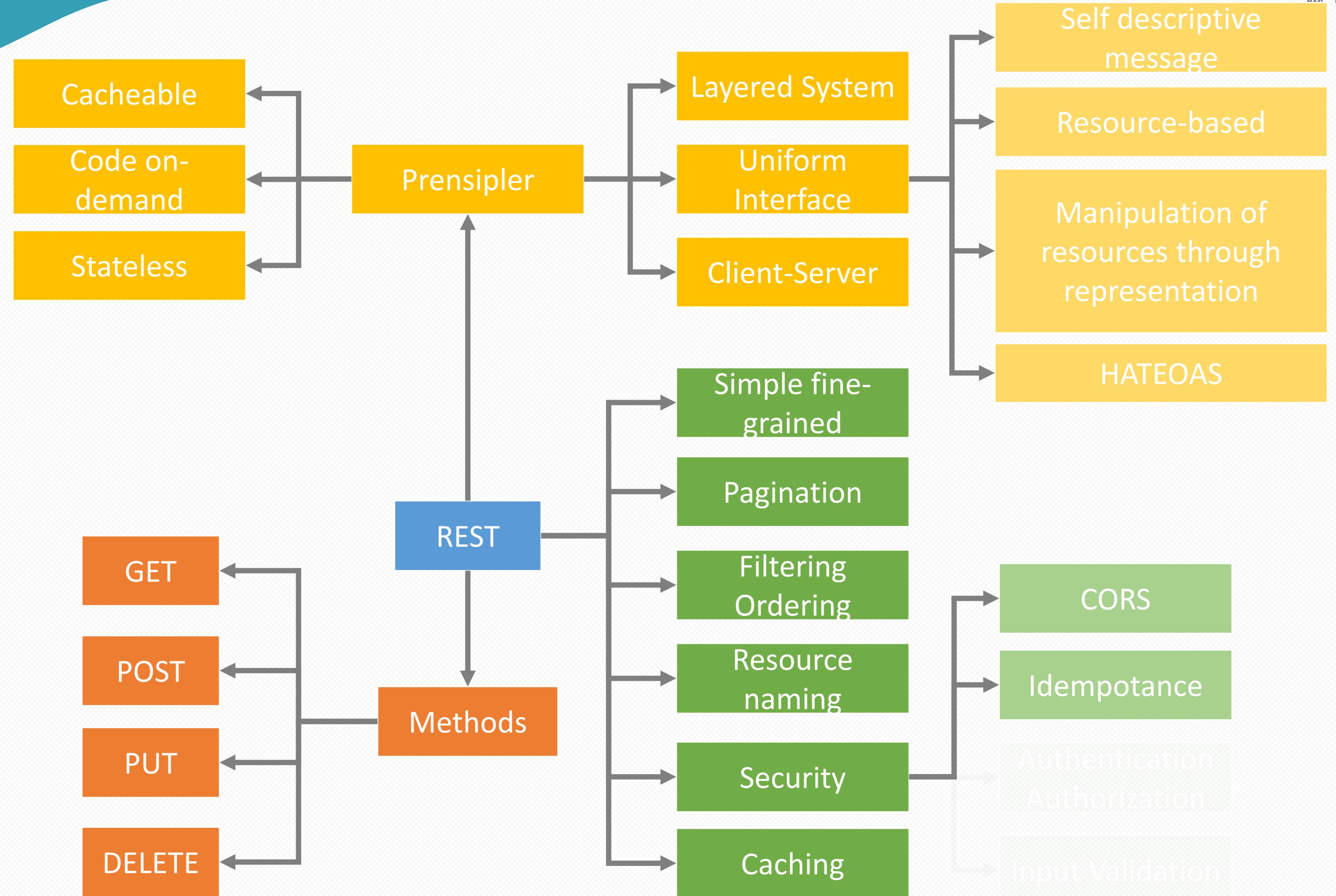
Güvenlik

CORS





CORS, web uygulamalarının daha güvenli ve daha verimli bir şekilde çalışmasına yardımcı olurken, aynı zamanda kötü niyetli taleplere karşı da koruma sağlar. Bu nedenle, web geliştiricilerinin CORS kurallarını doğru bir şekilde yapılandırması ve güncel tutması önemlidir.

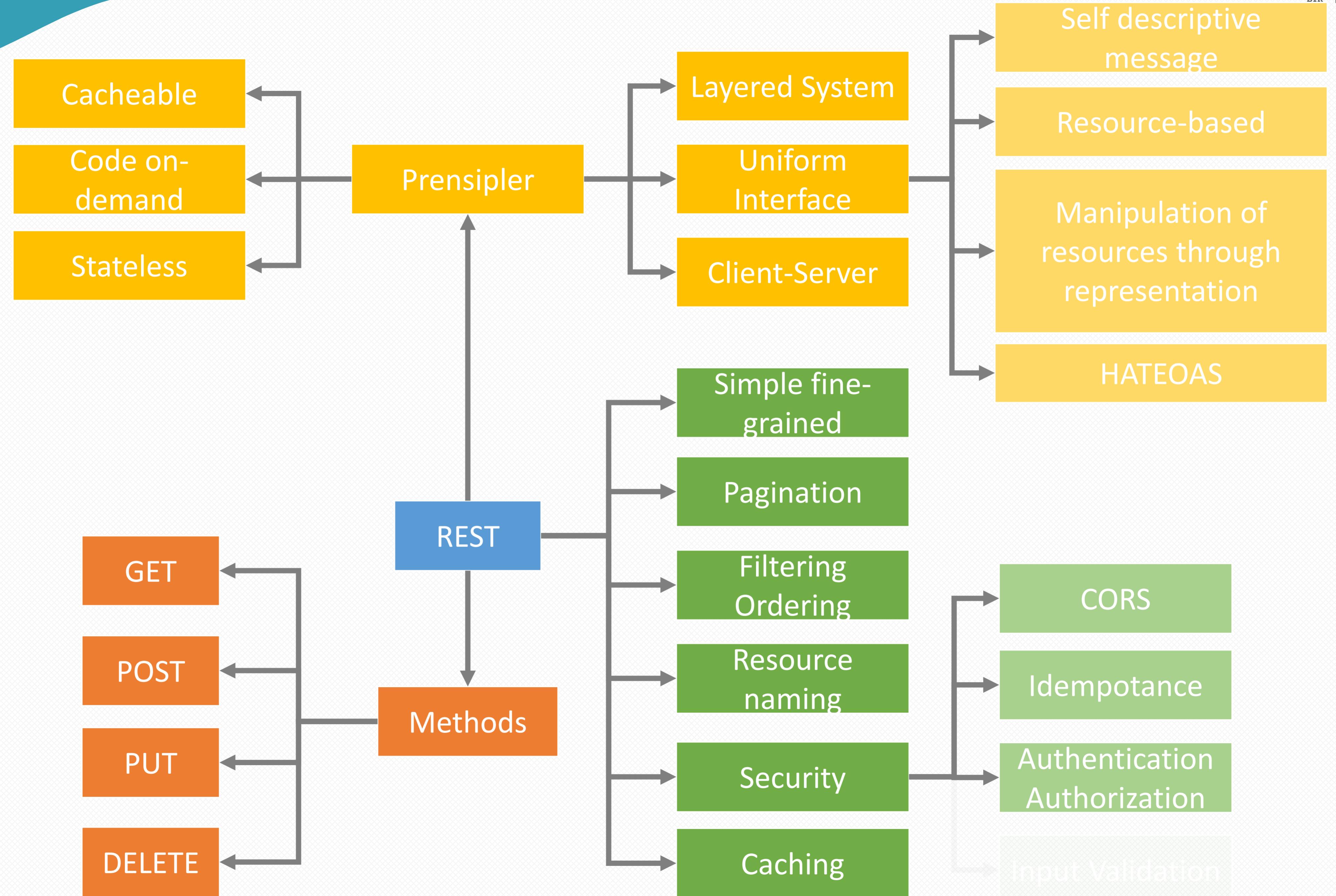


Idempotance

- **Idempotance REST API** güvenliği ile ilgili önemli bir kavramdır ve bir **HTTP** isteği idempotent olup olmadığını belirlemek için kullanılır.
- **Idempotance**, aynı işlemi birden çok kez tekrarladığınızda sonucun aynı kalacağını ifade eder.
- Başka bir deyişle, bir işlem **Idempotance** ise, aynı isteği birden fazla kez göndermek sonucu değiştirmemelidir. Bu, sunucu tarafında veya veri kaynağında istenmeyen etkilerin olmasını önler.

Idempotence

FİİL	CRUD	İŞLEV	GÜVENLİ	İDEMPOTENT
GET	Okuma	Tek veya çoklu kaynak getirme	+	+
POST	Oluşturma	Yeni bir kaynak ekleyin	-	-
PUT	Güncelleme/Oluşturma	Yeni bir kaynak ekleme veya var olanı güncelleme	-	+
DELETE	Silme	Tek veya birden fazla kaynağı silme	-	+
OPTIONS	Okuma	Bir kaynak üzerinde izin verilen işlemleri listeleme	+	+
HEAD	Okuma	Gövde olmadan yalnızca yanıt üstbilgilerini döndürür	+	+



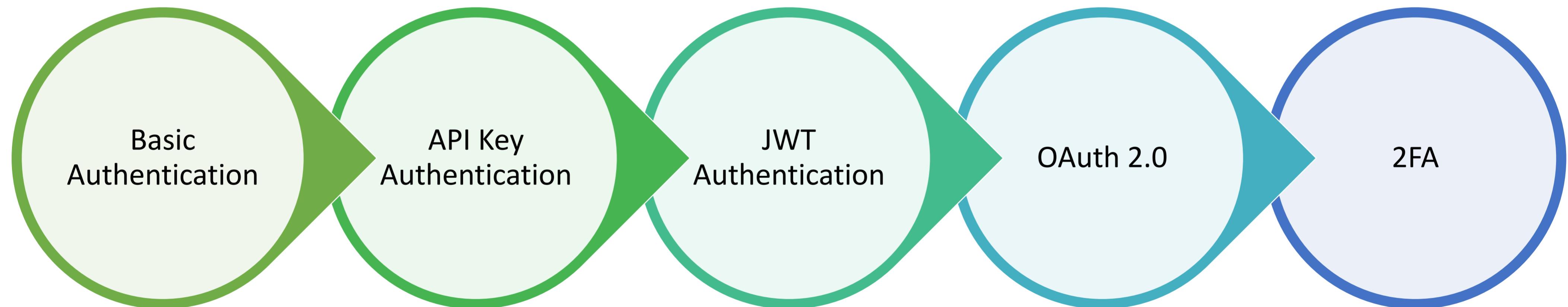
Oturum Açıma ve Yetkilendirme

Authentication and Authorization

- Günümüzün dijital dünyasında, uygulamalar ve servisler arası etkileşim büyük bir önem taşımaktadır.
- Bu etkileşimin güvenli ve düzenli bir şekilde gerçekleştirilmesi, hem kullanıcı verilerinin korunması hem de sistemlerin güvenliği açısından hayatı bir öneme sahiptir.
- RESTful API'lar, bu tür etkileşimlerin temelini oluşturur ve bu API'ları güvenli hale getirmek için oturum açma ve yetkilendirme işlemleri gereklidir.

Oturum Açıma ve Yetkilendirme

Authentication and Authorization



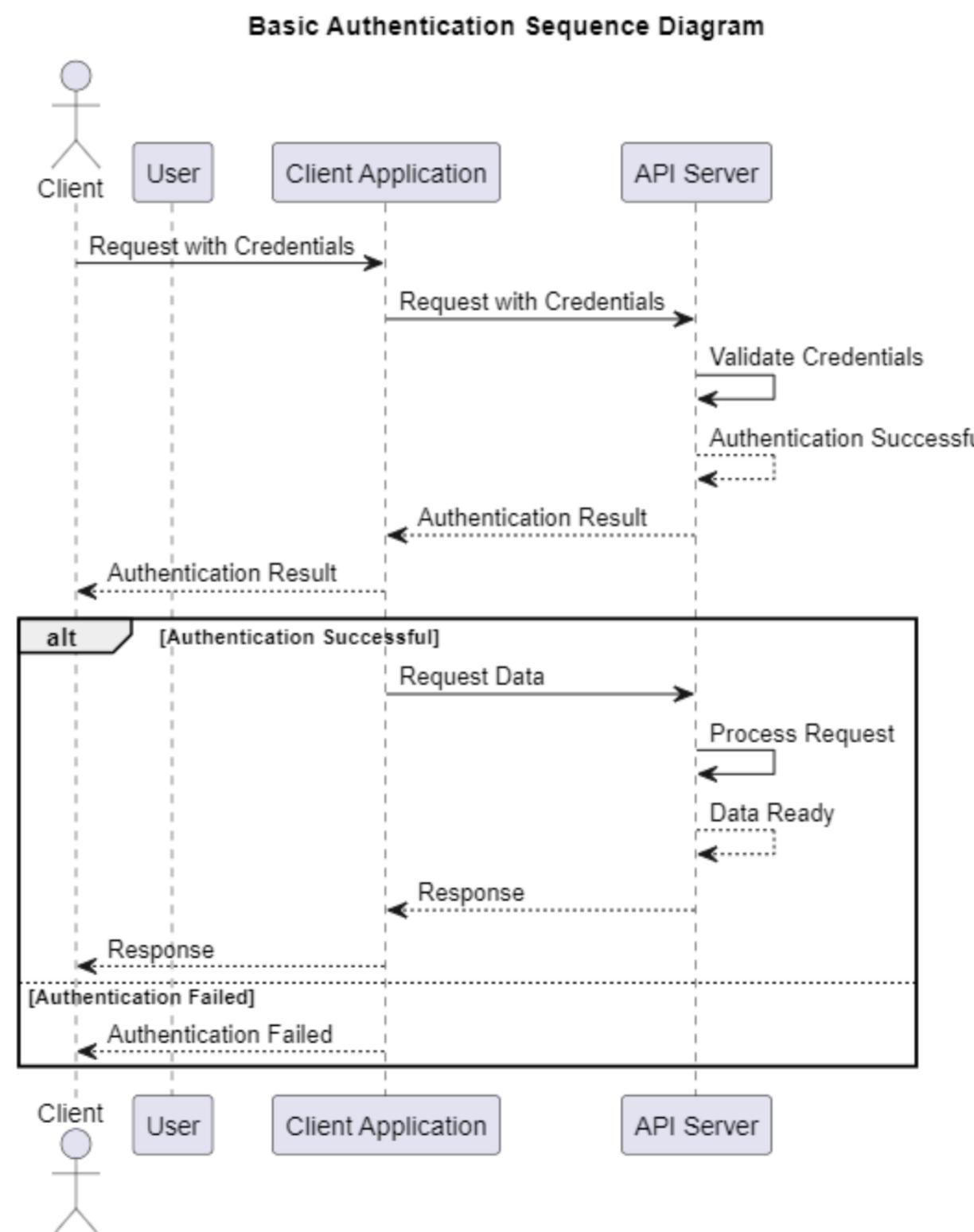
Temel Kimlik Doğrulama

Basic Authentication

- Bu yöntemde, istemci kullanıcı adı ve şifresini birlikte sunar.
- Sunucu, bu kimlik bilgilerini doğrular ve erişim izinleri verir veya reddeder.
- Ancak, bu yöntem güvenli değildir çünkü kimlik bilgileri açık bir şekilde taşınır ve kötü niyetli kişiler tarafından kolayca ele geçirilebilir.

Temel Kimlik Doğrulama

Basic Authentication



API Anahtarı Kullanma

API Key

- API anahtarı ([API key](#)) kullanarak kimlik doğrulama işlemi, bir REST API'ye erişirken kullanıcı kimliğini doğrulamak için kullanılan bir yöntemdir.

[API Anahtarı Oluşturma](#)

[API Anahtarını İstekte Kullanma](#)

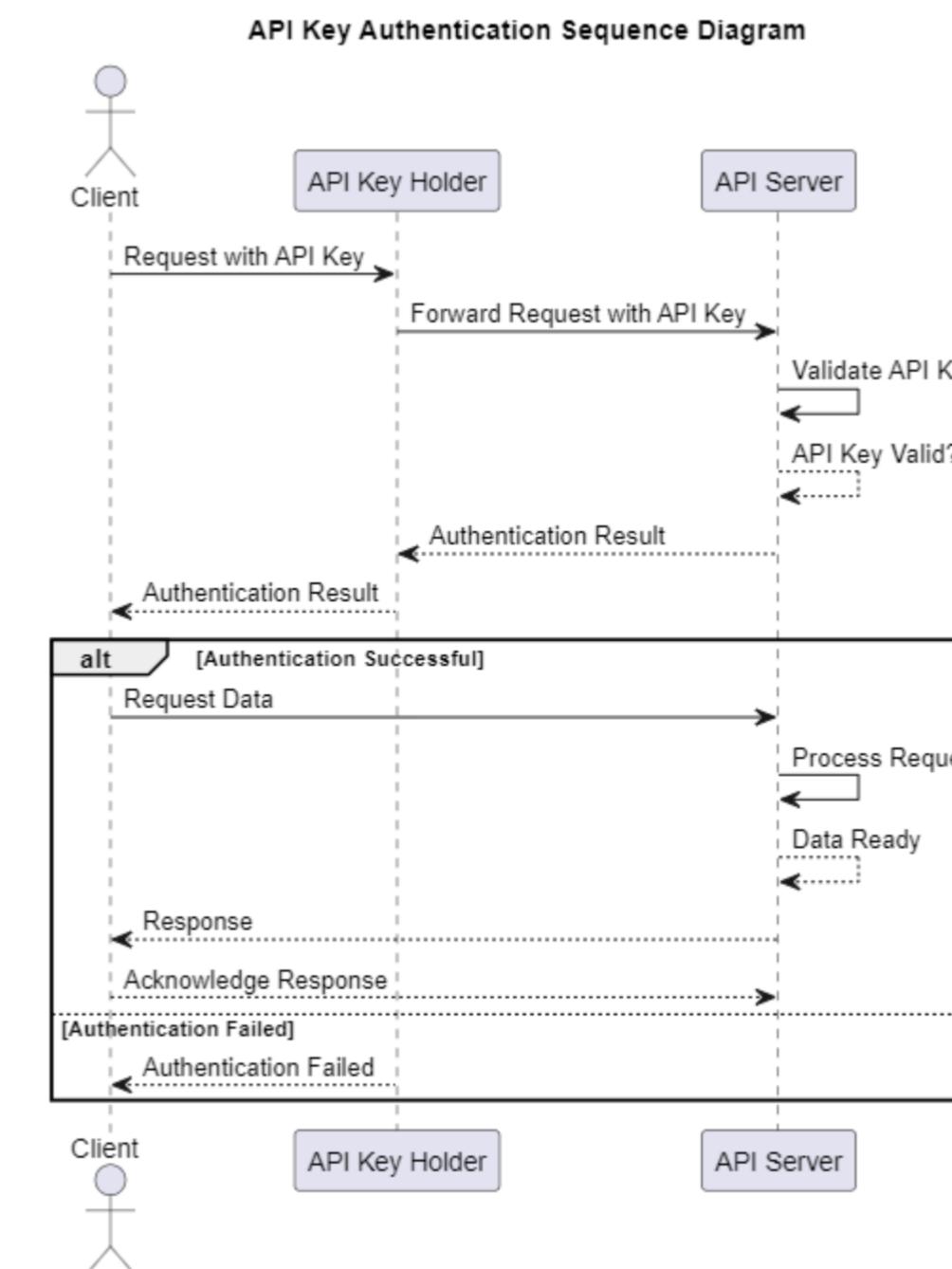
[API Sunucusu Tarafından Doğrulama](#)

[Kimlik Doğrulama Sonucu](#)

[Hata Durumları](#)

API Anahtarı Kullanma

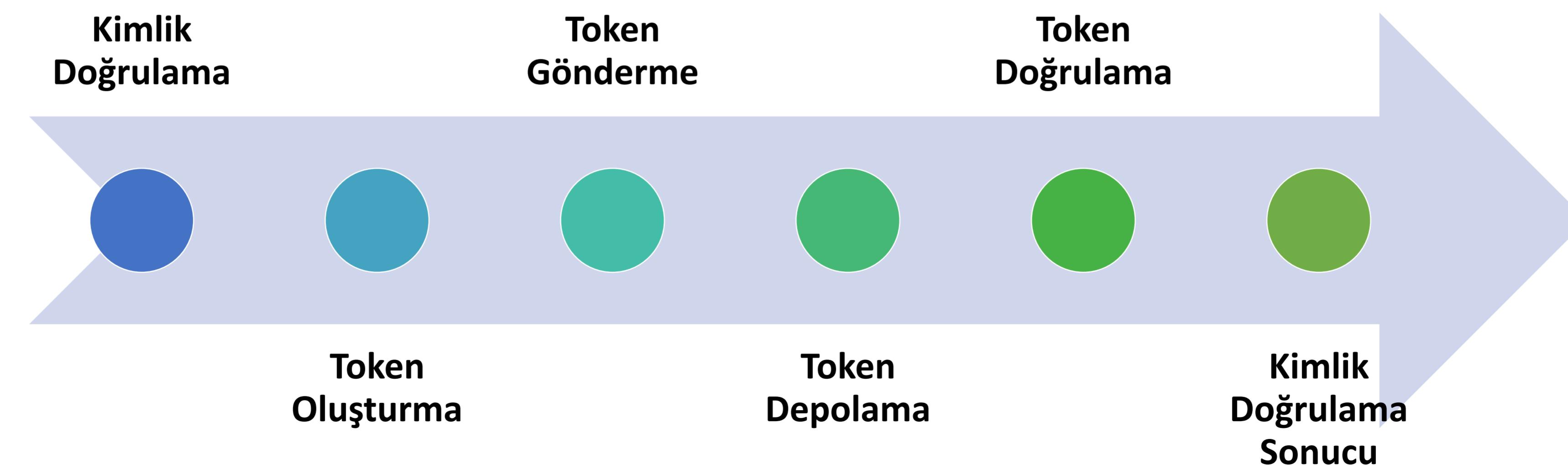
API Key



JWT Tabanlı Kimlik Doğrulama

JWT based Authentication

- **JSON Web Token (JWT)**, API güvenliğini sağlamak için yaygın olarak kullanılan bir yöntemdir. JWT, bilgileri **JSON** formatında kodlayan ve digital olarak imzalayan bir token türüdür.



JWT

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9 . eyJ
zdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ . XbPfbIHM
I6arZ3Y922BhjWgQzWXcXNrz0ogtVhfEd2o

Header	Payload	Signature
<pre>{ "alg": "HS256", "typ": "JWT" }</pre>	<pre>{ "sub": "1234567890", "name": "John Doe", "iat": 1516239022 }</pre>	<pre>HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), superSecretKey) <input checked="" type="checkbox"/> secret base64 encoded</pre>
base64	information about logged-in user	validation

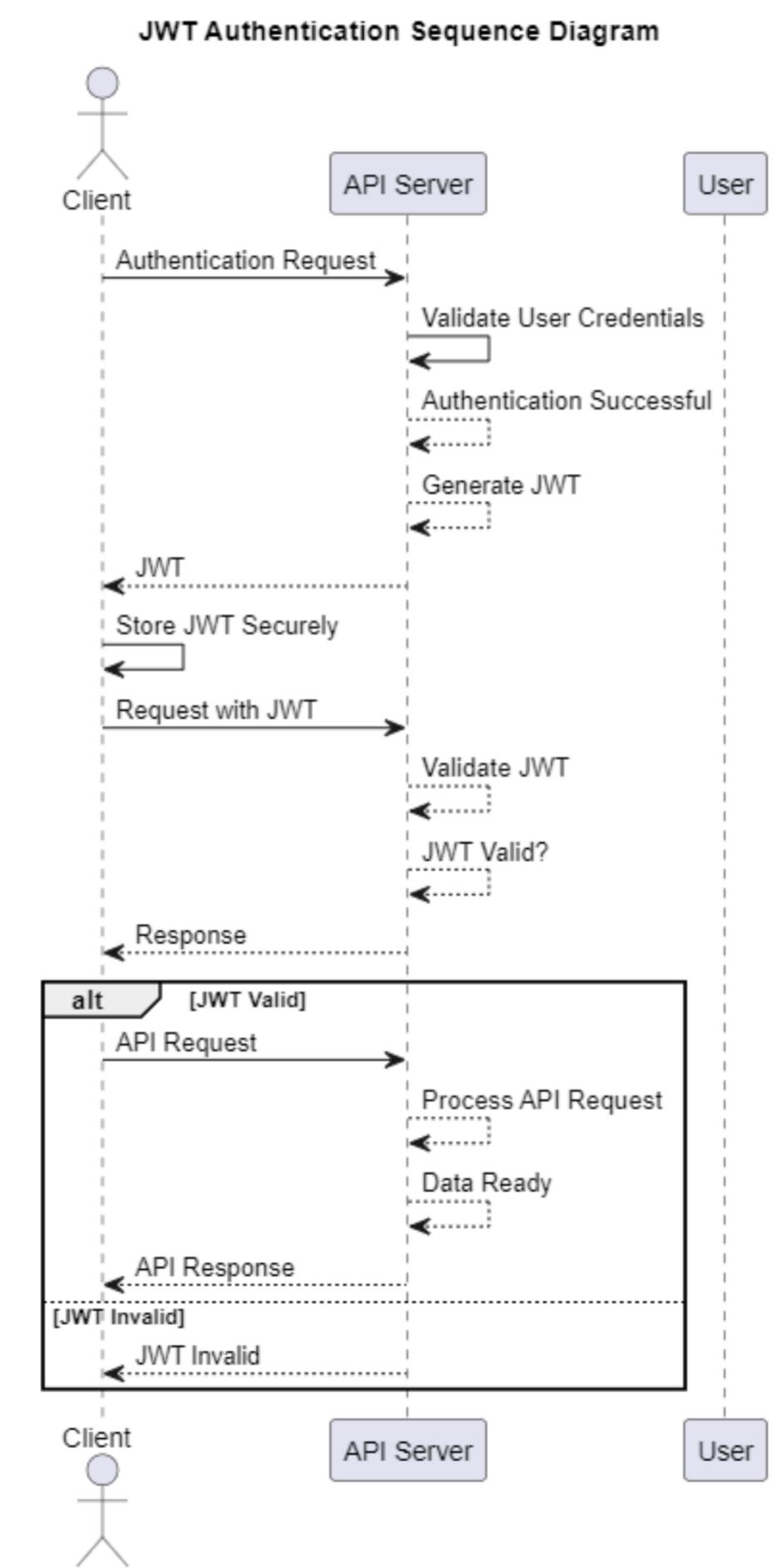
JWT Tabanlı Kimlik Doğrulama

JWT based Authentication

- JWT, genellikle kullanıcı oturumlarını yönetmek, API erişimini kontrol etmek ve tek seferlik erişim belgeleri oluşturmak gibi senaryolarda kullanılır.
- Ayrıca, **JWT'lerin** genellikle süresi dolan (**expire**) bir süresi vardır ve bu süre sona erdiğinde tekrar kimlik doğrulaması gerekebilir.
- Bu amaçla süresi dolan **token'lerin** yenilenmesi için **refresh token** mekanizması da işletilebilir.

JWT Tabanlı Kimlik Doğrulama

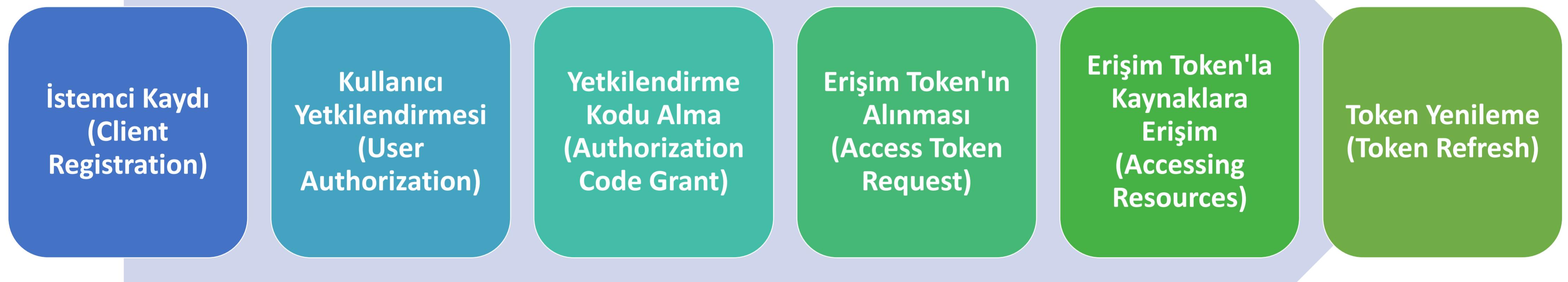
JWT based Authentication



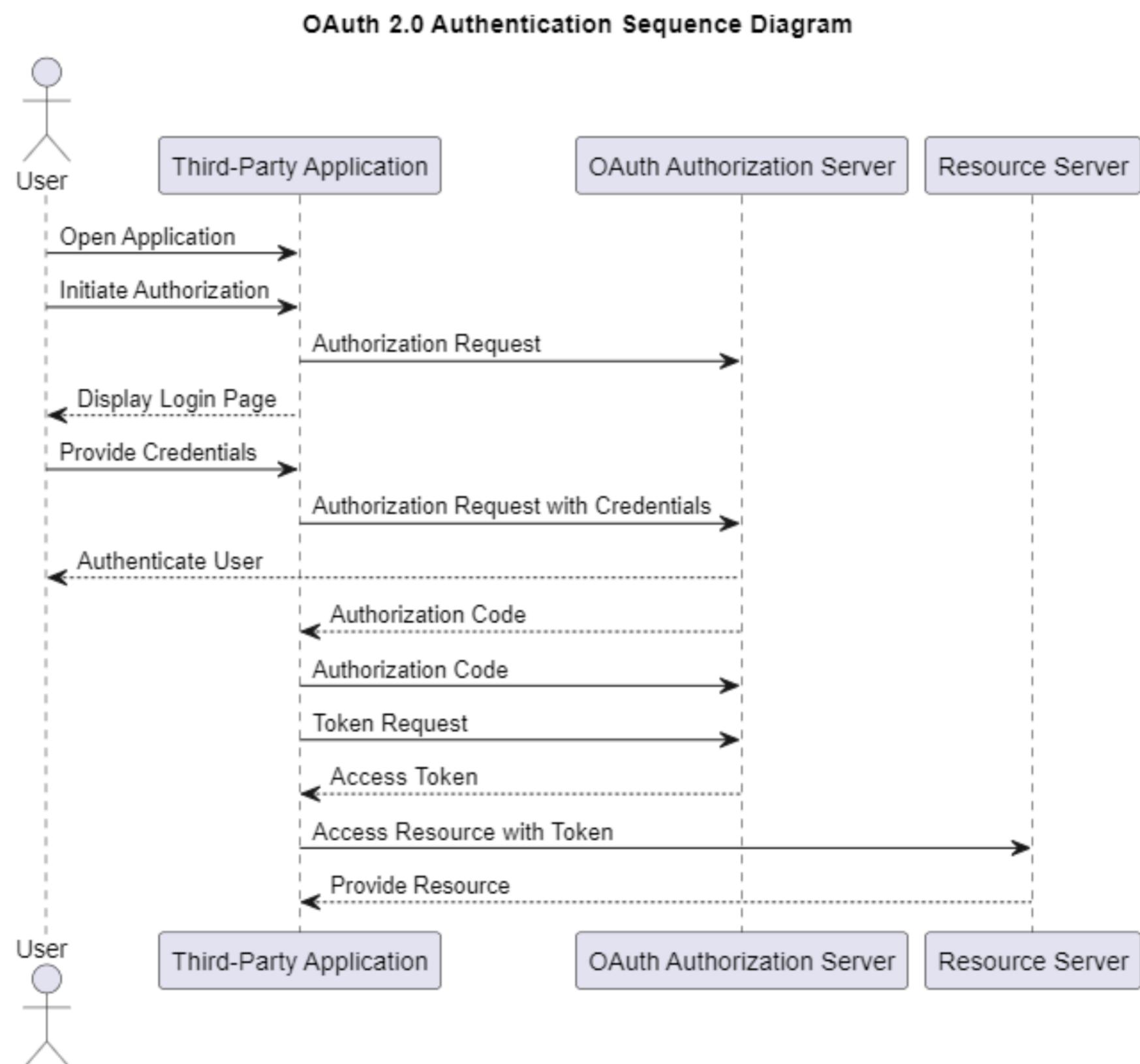
OAuth 2.0

- OAuth 2.0, REST API güvenliğini sağlamak ve yetkilendirme işlemlerini yönetmek için kullanılan bir kimlik doğrulama ve yetkilendirme protokolüdür.
- OAuth 2.0, istemcilerin, bir kullanıcının kaynaklara (örneğin, bir web hizmetine veya bir uygulamaya) erişmesine izin verirken kullanıcı adı ve parola gibi hassas kimlik bilgilerini paylaşmadan bu erişime izin vermesini sağlar.

OAuth 2.0



OAuth 2.0



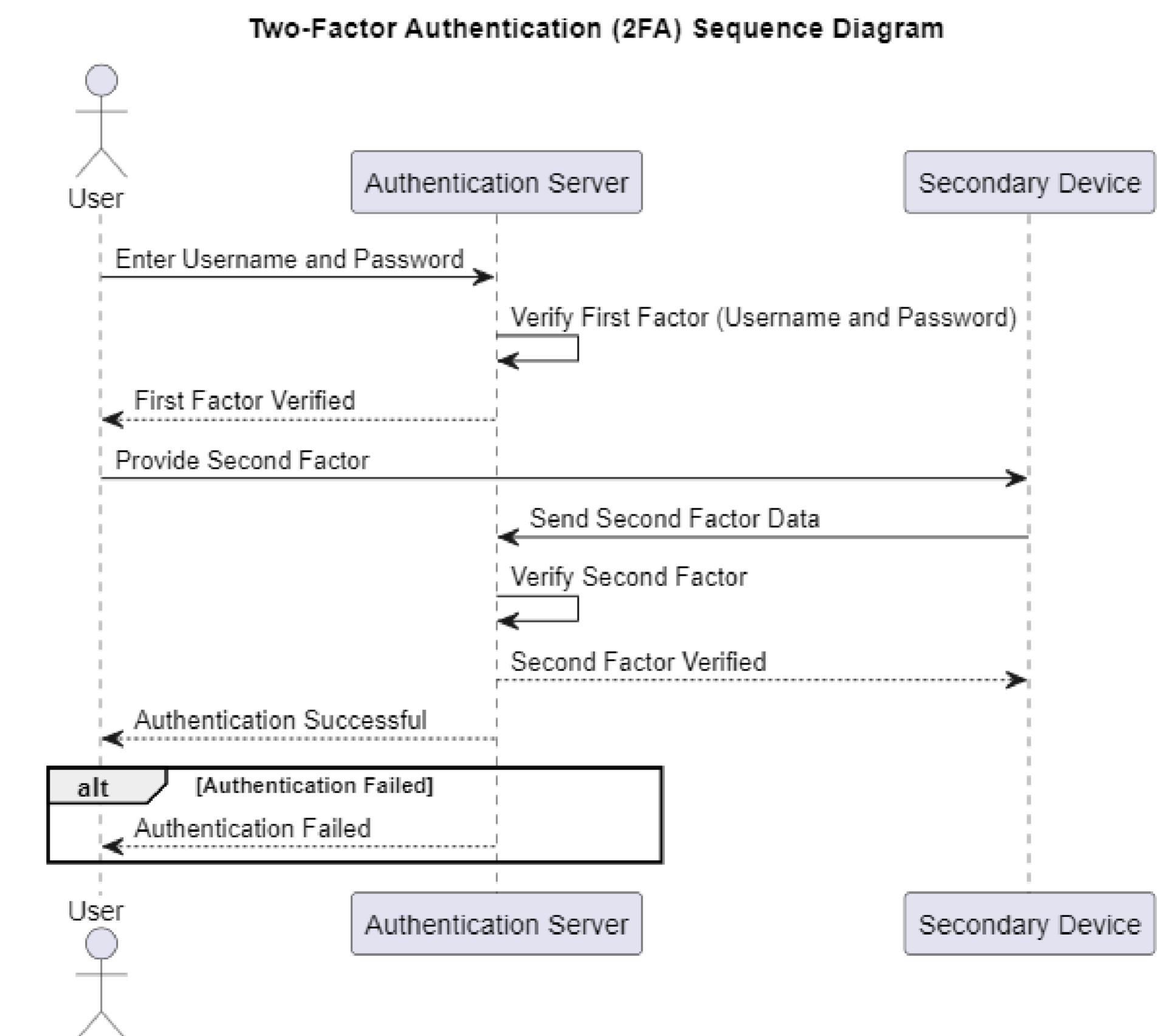
İki Faktörlü Kimlik Doğrulama

Two-Factor Authentication (2FA)

- 2FA, API güvenliğini artırmak için kullanılan bir yöntemdir.
- Bu yaklaşım, kullanıcının sadece bir şifre veya tek bir kimlik doğrulama yöntemi yerine iki farklı doğrulama faktörünü kullanmasını gerektirir. Bu faktörler genellikle şunlardır:
 - **Birincil Faktör (Something You Know)**: Kullanıcının bildiği gizli bir bilgi, genellikle bir şifre veya PIN kodu gibi. Bu faktör, kullanıcı adı ve şifre ile temsil edilir.
 - **İkincil Faktör (Something You Have veya Something You Are)**: Kullanıcının fiziksel bir nesneye sahip olması veya biyometrik kimlik doğrulama gibi fiziksel bir özelliği kullanması. Bu faktör, genellikle bir cep telefonu, bir güvenlik anahtarı veya parmak izi tarayıcısı gibi bir şeydir.

İki Faktörlü Kimlik Doğrulama

Two-Factor Authentication (2FA)



Uygulamalar

- Postman Simpe Book API
- Postman Tests
- .NET Web API Proje Şablonu
- Spring Boot REST API Proje Şablonu
- Node JS Web API Proje Şablonu
- Flask Proje Şablonu