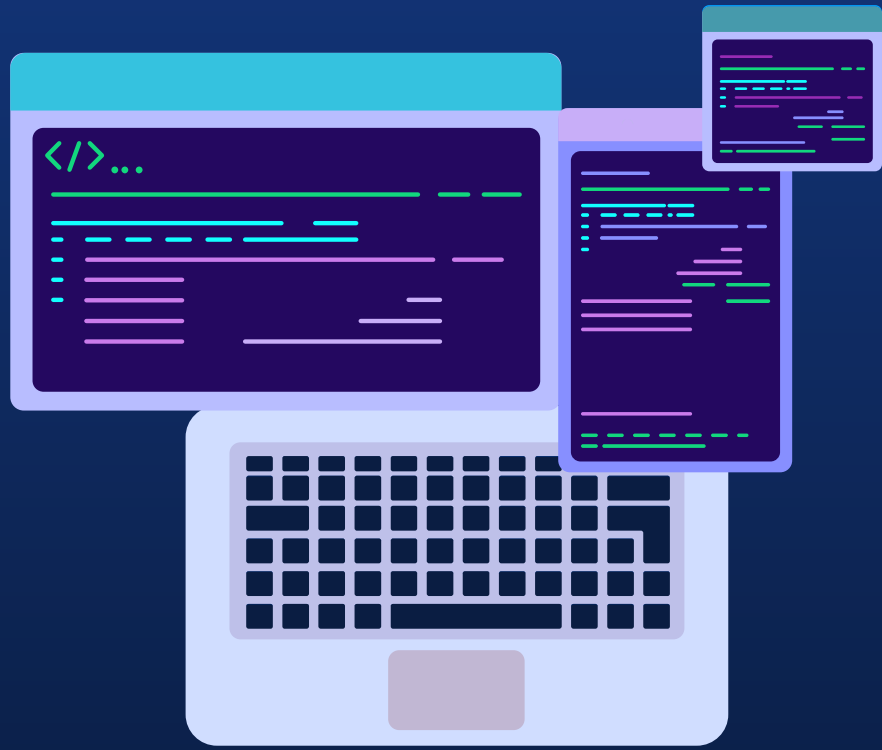
Node.js ve Express ile REST API Geliştirme

Şahin Ersever



ŞAHİN ERSEVER - Full Stack Developer
erseversahin@gmail.com

https://www.instagram.com/sahinersever_

https://www.linkedin.com/in/sahinersever/

https://x.com/sahinersever_

https://github.com/erseversahin

REST API, Representational State Transfer Application Programming Interface

- REST (Representational State Transfer), istemci (client) ve sunucu (server) arasında iletişim kurmayı sağlayan bir mimari stildir.
- REST, genellikle HTTP protokolü üzerinden çalışır ve kaynaklara erişim için standart HTTP yöntemlerini (GET, POST, PUT, DELETE vb.) kullanır.
- İstemci ve sunucu arasında XML ve JSON verilerini taşıyarak uygulamanın haberleşmesini sağlar.
- REST mimarisini kullanan servislere ise **RESTful servis (RESTful API) denir.**

REST İLKELERİ

01

Tek Tip Arayüz (Uniform Interface)

API'nin, kullanıcılar ve istemciler arasında tutarlı bir şekilde iletişim kurmasını sağlayan bir ilkedir.

02

İstemci-Sunucu Özerkliği (Client-Server Independence)

İstemci ve sunucu arasında açık bir ayrım vardır. İkisi de birbirinden bağımsız olarak çalışabilir.

03

Durum Bilgisinin Olmaması (Statelessness)

İstemci-sunucu iletişiminde sunucu, istemciyle ilgili herhangi bir durum bilgisi saklamaz.

04

Önbelleğe Alınabilirlik (Cacheability)

API yanıtları, istemci tarafından önbelleğe alınabilir.

05

Katmanlı Sistem Mimarisi (Layered System Architecture)

REST API, birden fazla katmandan oluşan bir mimaride çalışabilir.

06

İsteğe Bağlı Kod Yapısı (Code on Demand) (Opsiyonel)

İstemciye, yürütülebilir kod (JavaScript gibi) gönderilmesine olanak tanır.

“Makine kodu okur, insan anlamak için anlatanı arar.”

—ŞAHİN ERSEVER 😂

API İSİMLENDİRMESİ

API'nin kolay anlaşılabilir, tutarlı ve doğru kullanılabilir olmasını sağlamak için çok önemlidir.

Kaynaklar için isimlendirme...

KAYNAK(RESOURCE) NEDİR?
Rest API'lerde "**resource**" terimi, genellikle sistemde bulunan her tür veriyi temsil eden nesneleri ifade eder. Bu nesneler, bir kullanıcı, bir şirket veya bir dosya gibi herhangi bir veri olabilir.

1- Kaynaklar isimlendirilirken çoğul isim kullanılır

```
// GET isteği ile tüm kullanıcıları getirme
app.get('/users', (req, res) => {
    res.send([
        { id: 1, name: "John Doe" },
        { id: 2, name: "Jane Smith" },
    ]);
});
```

DOĞRU

```
// GET isteği ile tüm kullanıcıları getirme
app.get('/user', (req, res) => {
    res.send([
        { id: 1, name: "John Doe" },
        { id: 2, name: "Jane Smith" },
    ]);
});
```

YANLIŞ

Kaynaklar için isimlendirme...

2- Tekil Kaynaklar için Benzersiz Kimlik (ID) Kullanımı

```
// GET isteği ile ID'si 1 olan kullanıcıyı getirme
app.get('/users/:id', (req, res) => {
    const userId = req.params.id;
    const user = { id: userId, name: "John Doe" };
    res.send(user);
});
```

DOĞRU

```
// GET isteği ile ID'si 1 olan kullanıcıyı getirme
app.get('/users', (req, res) => {
    const userId = req.query.id;
    const user = { id: userId, name: "John Doe" };
    res.send(user);
});
```

YANLIŞ

HTTP Metodu	Amaç	Açıklama	Örnek URL
GET	Veriyi okumak ve almak için	Sunucudan veri almak için kullanılır. GET istekleri, sunucuda veri değişikliği yapmaz.	`/users` (Tüm kullanıcılar)
POST	Yeni bir kaynak oluşturmak için	Sunucuda yeni bir veri eklemek için kullanılır. Gönderilen veri, sunucuya eklenir ve genellikle yeni bir ID ile geri döner.	`/users` (Yeni kullanıcı)
PUT	Mevcut bir kaynağı tamamen güncellemek için	Var olan bir kaynağın tüm özelliklerini günceller. Kaynağın ID'si gereklidir.	`/users/1` (Kullanıcı güncelle)
PATCH	Mevcut bir kaynağı kısmen güncellemek için	Var olan bir kaynağın sadece belirtilen özelliklerini günceller. PUT'tan farklı olarak, tüm veriyi göndermek zorunlu değildir.	`/users/1` (Kısmi güncelleme)
DELETE	Mevcut bir kaynağı silmek için	Sunucudan bir kaynağı siler. Genellikle silme işlemi başarıyla tamamlandığında `204 No Content` durum kodu döner.	`/users/1` (Kullanıcı sil)
HEAD	Veri olmadan sadece başlık bilgisi almak için	GET isteğine benzer, ancak sunucudan sadece başlık bilgisi alır. Yanıtta gövde (body) bulunmaz.	`/users` (Başlık al)
OPTIONS	Sunucunun desteklediği HTTP metotlarını öğrenmek için	Belirli bir URL için kullanılabilir HTTP metotlarını döner. Özellikle CORS (Cross-Origin Resource Sharing) yapılandırmasında kullanılır.	`/users` (Desteklenen metotlar)
CONNECT	Sunucu ile tünelleme bağlantısı kurmak için	Genellikle HTTPS gibi şifreli bağlantılar için istemci-sunucu arasında tünel oluşturmak için kullanılır.	`/`
TRACE	HTTP isteğinin sunucuda nasıl işlendiğini görmek için	Sunucuya gelen HTTP isteğini döndürür. Genellikle hata ayıklama (debugging) amaçlı kullanılır, ancak güvenlik nedeniyle nadiren etkinleştirilir.	`/users`

HTTP Metodlarına Göre İşlemleri Tanımlayın

```
// Tüm kullanıcıları getir
app.get('/users', (req, res) => {
    res.send([
        { id: 1, name: "John Doe" },
        { id: 2, name: "Jane Smith" },
    ]);
});

// ID'si 1 olan kullanıcıyı getir
app.get('/users/:id', (req, res) => {
    const userId = req.params.id;
    res.send({ id: userId, name: "John Doe" });
});

// Yeni kullanıcı oluştur
app.post('/users', (req, res) => {
    const newUser = req.body; // İstekten gelen kullanıcı verisi
    newUser.id = 3; // Yeni bir ID ata
    res.status(201).send(newUser); // 201 Created
});

// ID'si 1 olan kullanıcıyı güncelle
app.put('/users/:id', (req, res) => {
    const userId = req.params.id;
    const updatedUser = req.body; // Güncellenmiş kullanıcı verisi
    updatedUser.id = userId; // Güncellenen kullanıcının ID'sini koru
    res.send(updatedUser); // Güncellenmiş veriyi döndür
});
```

URL’de Fiil Kullanmaktan Kaçının

```
// Yanlış isimlendirme: GET ile kullanıcıları getir
app.get('/getUsers', (req, res) => {
    res.send([
        { id: 1, name: "John Doe" },
        { id: 2, name: "Jane Smith" },
    ]);
});

// Yanlış isimlendirme: POST ile kullanıcı oluştur
app.post('/createUser', (req, res) => {
    const newUser = req.body;
    newUser.id = 3;
    res.send(newUser);
});

// Yanlış isimlendirme: PUT ile kullanıcı güncelle
app.put('/updateUser', (req, res) => {
    const updatedUser = req.body;
    updatedUser.id = 1;
    res.send(updatedUser);
});

// Yanlış isimlendirme: DELETE ile kullanıcı sil
app.delete('/deleteUser', (req, res) => {
    res.send({ message: "User deleted" });
});
```

/getUsers, /createUser gibi ifadeler, REST’in Uniform Interface ilkesini ihlal eder.

Kaynaklar Arasındaki İlişkiyi URL'de Gösterin

Hiyerarşik yapıyı koruyan bir API tasarımı:

```
const express = require('express');
const app = express();

// Kullanıcının tüm siparişlerini getir
app.get('/users/:userId/orders', (req, res) => {
    const userId = req.params.userId;
    res.send(`ID'si ${userId} olan kullanıcının tüm siparişleri.`);
});

// Belirli bir siparişi getir
app.get('/users/:userId/orders/:orderId', (req, res) => {
    const userId = req.params.userId;
    const orderId = req.params.orderId;
    res.send(`ID'si ${userId} olan kullanıcının ID'si ${orderId} olan siparişi.`);
});

app.listen(3000, () => console.log('Server running on http://localhost:3000'));
```

Kaynaklar Arasındaki İlişkiyi URL'de Gösterin

Hiyerarşik yapıyı kaybettiren bir API tasarımı:

```
const express = require('express');
const app = express();

// Kullanıcının tüm siparişlerini getir
app.get('/orders', (req, res) => {
    const userId = req.query.user_id;
    res.send(`ID'si ${userId} olan kullanıcının tüm siparişleri.`);
});

// Sipariş ve kullanıcı ters hiyerarşi
app.get('/orders/:orderId/users/:userId', (req, res) => {
    const userId = req.params.userId;
    const orderId = req.params.orderId;
    res.send(`ID'si ${orderId} olan sipariş için ID'si ${userId} olan kullanıcı.`);
});

app.listen(3000, () => console.log('Server running on http://localhost:3000'));
```

Küçük Harf Kullanımı

URL'nin okunabilirliğini artırır ve standartlara uygun bir yapı sağlar

```
const express = require('express');
const app = express();

// Kullanıcı profillerini listeleme
app.get('/user-profiles', (req, res) => {
    res.send('Tüm kullanıcı profilleri.');
});

app.listen(3000, () => console.log('Server running on http://localhost:3000'));
```

```
const express = require('express');
const app = express();

// Büyük harf içeren URL
app.get('/UserProfiles', (req, res) => {
    res.send('Tüm kullanıcı profilleri.');
});

app.listen(3000, () => console.log('Server running on http://localhost:3000'));
```

Sorgu Parametreleri ile Filtreleme

URL: /users?role=admin

```
const express = require('express');
const app = express();
const users = [
    { id: 1, name: "John Doe", role: "admin" },
    { id: 2, name: "Jane Smith", role: "user" },
    { id: 3, name: "Emily Johnson", role: "admin" },
];
// Filtreleme: Belirli bir role sahip kullanıcıları getir
app.get('/users', (req, res) => {
    const role = req.query.role;
    if (role) {
        const filteredUsers = users.filter(user => user.role === role);
        res.send(filteredUsers);
    } else {
        res.send(users);
    }
});
app.listen(3000, () => console.log('Server running on
http://localhost:3000'));
```

ÇIKTI:

```
[
    { "id": 1, "name": "John Doe", "role": "admin" },
    { "id": 3, "name": "Emily Johnson", "role":
"admin" }
]
```

Sorgu Parametreleri ile Filtreleme

URL: /products?sort=price&order=asc

```
const products = [
    { id: 1, name: "Laptop", price: 1000 },
    { id: 2, name: "Phone", price: 500 },
    { id: 3, name: "Tablet", price: 700 },
];
// Sıralama: Ürünleri belirli bir özelliğe göre sırala
app.get('/products', (req, res) => {
    const sort = req.query.sort;
    const order = req.query.order;
    if (sort) {
        const sortedProducts = [...products].sort((a, b) => {
            if (order === "desc") {
                return b[sort] - a[sort];
            }
            return a[sort] - b[sort];
        });
        res.send(sortedProducts);
    } else {
        res.send(products);
    }
});
```

ÇIKTI:

```
[
    { "id": 2, "name": "Phone", "price": 500 },
    { "id": 3, "name": "Tablet", "price": 700 },
    { "id": 1, "name": "Laptop", "price": 1000 }
]
```

Sorgu Parametreleri ile Sayfalama

URL: /users?page=2&limit=1

```
// Sayfalama: Kullanıcıları belirli bir sayfaya göre getir
app.get('/users', (req, res) => {
    const page = parseInt(req.query.page) || 1;
    const limit = parseInt(req.query.limit) || 10;
    const startIndex = (page - 1) * limit;
    const endIndex = startIndex + limit;
    const paginatedUsers = users.slice(startIndex, endIndex);
    res.send(paginatedUsers);
});
```

ÇIKTI:

```
[
    { "id": 1, "name": "John Doe", "role": "admin" },
    { "id": 2, "name": "Jane Smith", "role": "user" }
]
```

Belirli Alanları Seçme

URL: /users?fields=id,name

```
// Belirli alanları seçme
app.get('/users', (req, res) => {
    const fields = req.query.fields?.split(',') || null;
    if (fields) {
        const filteredUsers = users.map(user => {
            const filteredUser = {};
            fields.forEach(field => {
                if (user[field] !== undefined) {
                    filteredUser[field] = user[field];
                }
            });
            return filteredUser;
        });
        res.send(filteredUsers);
    } else {
        res.send(users);
    }
});
```

ÇIKTI:

```
[
    { "id": 1, "name": "John Doe" },
    { "id": 2, "name": "Jane Smith" },
    { "id": 3, "name": "Emily Johnson" }
]
```

VERSİYONLAMA

API’nin farklı sürümleri arasında tutarlılığı sağlamak için versiyonlama kullanın.

Versiyonlama Yöntemleri

URL Üzerinden Versiyonlama

```
const express = require('express');
const app = express();
// V1 API: Kullanıcıları listele
app.get('/v1/users', (req, res) => {
    res.send('Version 1: Kullanıcı listesi');
});
// V2 API: Kullanıcıları listele (yeni özellikler eklenmiş)
app.get('/v2/users', (req, res) => {
    res.send('Version 2: Kullanıcı listesi (yeni özelliklerle)');
});
app.listen(3000, () => console.log('Server running on http://localhost:3000'));
```

Versiyonlama Yöntemleri

Header Üzerinden Versiyonlama

```
app.get('/users', (req, res) => {
    const version = req.headers['accept'];
    if (version.includes('vnd.myapi.v1')) {
        res.send('Version 1: Kullanıcı listesi');
    } else if (version.includes('vnd.myapi.v2')) {
        res.send('Version 2: Kullanıcı listesi (yeni özelliklerle)');
    } else {
        res.status(400).send('API versiyonu belirtilmedi.');
    }
});
```

Versiyonlama Yöntemleri

Query Parametre Üzerinden Versiyonlama (Önerilmez)

```
app.get('/users', (req, res) => {
    const version = req.query.version;
    if (version === '1') {
        res.send('Version 1: Kullanıcı listesi');
    } else if (version === '2') {
        res.send('Version 2: Kullanıcı listesi (yeni özelliklerle)');
    } else {
        res.status(400).send('API versiyonu belirtilmedi.');
    }
});
```

HTTP HEADERS

HTTP başlıklarının (headers) kullanımı, istemci ve sunucu arasındaki iletişimi netleştirir.

HTTP Headers

İçerik Türü Belirtme (Content-Type)

```
POST /data HTTP/1.1
Content-Type: application/json
{
  "name": "John Doe",
  "email": "john.doe@example.com"
}
```

```
app.use(express.json());
// JSON formatında veri gönderme ve kabul etme
app.post('/data', (req, res) => {
    if (req.headers['content-type'] === 'application/json') {
        const data = req.body;
        res.send({ message: 'JSON verisi alındı', data });
    } else {
        res.status(400).send({ error: 'Desteklenmeyen içerik türü' });
    }
});
```

```
{
  "message": "JSON verisi alındı",
  "data": {
    "name": "John Doe",
    "email": "john.doe@example.com"
  }
}
```

HTTP Headers

Dil Belirtme (Accept-Language)

```
GET /greeting HTTP/1.1
Accept-Language: tr-TR
```

```
app.get('/greeting', (req, res) => {
    const lang = req.headers['accept-language'];
    if (lang === 'tr-TR') {
        res.send({ message: 'Merhaba Dünya!' });
    } else if (lang === 'en-US') {
        res.send({ message: 'Hello World!' });
    } else {
        res.send({ message: 'Language not supported' });
    }
});
```

```
{
  "message": "Merhaba Dünya!"
}
```

HTTP Headers

Yetkilendirme ve Güvenlik (Authorization)

```
GET /secure-data HTTP/1.1
Authorization: Bearer valid-token
```

```
app.get('/secure-data', (req, res) => {
    const authHeader = req.headers['authorization'];
    if (authHeader && authHeader.startsWith('Bearer ')) {
        const token = authHeader.split(' ')[1];
        if (token === 'valid-token') {
            res.send({ data: 'Güvenli veriye erişim sağlandı' });
        } else {
            res.status(401).send({ error: 'Geçersiz token' });
        }
    } else {
        res.status(403).send({ error: 'Yetkilendirme başlığı eksik' });
    }
});
```

```
{
  "data": "Güvenli veriye erişim sağlandı"
}
```

HTTP Headers

Önbellekleme (Cache-Control)

```
GET /no-cache HTTP/1.1
```

```
app.get('/no-cache', (req, res) => {
    res.set('Cache-Control', 'no-cache');
    res.send({ data: 'Bu veri önbelleğe alınmamalı' });
});
```

```
Cache-Control: no-cache
```

```
{
  "data": "Bu veri önbelleğe alınmamalı"
}
```

HTTP Headers

Kabul Edilebilir Formatlar (Accept)

```
GET /response-format HTTP/1.1
Accept: application/json
```

```
app.get('/response-format', (req, res) => {
    const accept = req.headers['accept'];
    if (accept === 'application/json') {
        res.json({ message: 'JSON formatında yanıt' });
    } else if (accept === 'text/html') {
        res.send('<h1>HTML formatında yanıt</h1>');
    } else {
        res.status(406).send({ error: 'Kabul edilemez format' });
    }
});
```

```
{
  "message": "JSON formatında yanıt"
}
```

HTTP
DURUM KODLARI

Sık Kullanılan HTTP Durum Kodları

Durum Kodu	Kategori	Açıklama
200 OK	Başarılı (2xx)	İstek başarılı bir şekilde işlendi.
201 Created	Başarılı (2xx)	Yeni bir kaynak başarıyla oluşturuldu.
204 No Content	Başarılı (2xx)	Başarılı işlem, ancak döndürülecek veri yok.
301 Moved Permanently	Yönlendirme (3xx)	Kaynak kalıcı olarak başka bir yere taşındı.
302 Found	Yönlendirme (3xx)	Kaynak geçici olarak başka bir yere taşındı.
304 Not Modified	Yönlendirme (3xx)	Kaynak değişmedi; istemci önbelleği kullanabilir.
400 Bad Request	İstemci Hatası (4xx)	Geçersiz istek gönderildi (örneğin eksik veya hatalı parametre).
401 Unauthorized	İstemci Hatası (4xx)	Kimlik doğrulama gerekli veya başarısız.
403 Forbidden	İstemci Hatası (4xx)	İstemcinin bu kaynağa erişim izni yok.
404 Not Found	İstemci Hatası (4xx)	İstenen kaynak bulunamadı.
429 Too Many Requests	İstemci Hatası (4xx)	Çok fazla istek gönderildi; sınır aşıldı.
500 Internal Server Error	Sunucu Hatası (5xx)	Sunucuda beklenmeyen bir hata oluştu.
502 Bad Gateway	Sunucu Hatası (5xx)	Sunucu bir geçit/proxy olarak çalışıyor ve geçersiz bir yanıt aldı.
503 Service Unavailable	Sunucu Hatası (5xx)	Sunucu geçici olarak hizmet veremiyor.
504 Gateway Timeout	Sunucu Hatası (5xx)	Sunucu, bir dış sistemden yanıt beklerken zaman aşımına uğradı.

API DOKÜMANTASYONU

API Dokümantasyonu

1. **Genel Bilgi**
   API'nin amacı ve kullanım alanları.
2. **Authentication (Kimlik Doğrulama)**
   API Key, OAuth, JWT gibi kimlik doğrulama yöntemleri.
3. **Endpoints (Uç Noktalar)**
   URL yapıları ve erişim yolları.
   Örnek: GET /users, POST /orders.
4. **Request ve Response Detayları**
   HTTP metodları: GET, POST, PUT, DELETE.
   Parametre türleri: Query, Path, Body.
   İstek ve yanıt formatları.
5. **HTTP Status Kodları**
   Örnek:
   200 OK: Başarılı.
   404 Not Found: Kaynak bulunamadı.
6. **Örnek Kodlar**
   API'nin kullanımını gösteren kod örnekleri (farklı dillerde).
7. **Sürüm Bilgisi (Versioning)**
   Sürüm bilgileri: /v1/users, /v2/orders.
   Eski sürümlerle uyumluluk.

API Dokümantasyonu

1. **Genel Bilgi**
   API'nin amacı ve kullanım alanları.
2. **Authentication (Kimlik Doğrulama)**
   API Key, OAuth, JWT gibi kimlik doğrulama yöntemleri.
3. **Endpoints (Uç Noktalar)**
   URL yapıları ve erişim yolları.
   Örnek: GET /users, POST /orders.
4. **Request ve Response Detayları**
   HTTP metodları: GET, POST, PUT, DELETE.
   Parametre türleri: Query, Path, Body.
   İstek ve yanıt formatları.
5. **HTTP Status Kodları**
   Örnek:
   200 OK: Başarılı.
   404 Not Found: Kaynak bulunamadı.
6. **Örnek Kodlar**
   API'nin kullanımını gösteren kod örnekleri (farklı dillerde).
7. **Sürüm Bilgisi (Versioning)**
   Sürüm bilgileri: /v1/users, /v2/orders.
   Eski sürümlerle uyumluluk.

TEŞEKKÜRLER