

Softito 4.Dönem Backend Developer Eğitimi 2026



Node.js

Merve Yıldırım – 09.07.2026

1. Özet

1 - Konu/bağlam - Bu çalışmada, JavaScript'in tarayıcı dışında çalıştırılmasını sağlayan Node.js çalışma zamanı ortamı; mimari yapısı, ekosistemi ve endüstriyel uygulamaları açısından incelenmiştir.

2 - Kapsam/yöntem - Çalışmada önce olay döngüsü (event loop), engellemeyen girdi/çıkış (non-blocking I/O) ve tek iş parçacıklı mimari gibi temel kavramlar açıklanmış; ardından npm paket ekosisteminin gelişimi ve bağımlılık riskleri left-pad vakası üzerinden tartışılmıştır.

3 – Vakalar - PayPal, Netflix, LinkedIn ve Walmart'ın Node.js'e geçiş süreçleri, şirketlerin raporladığı performans verileri temel alınarak karşılaştırmalı biçimde analiz edilmiştir.

4 - Bulgu/sonuç - İncelenen vakalar, Node.js'in girdi/çıkış yoğun ve yüksek eşzamanlılık gerektiren sistemlerde belirgin kazanımlar sağladığını, buna karşılık işlemci yoğun görevler için uygun olmadığını göstermektedir.

Anahtar Kelimeler: Node.js, olay döngüsü, asenkron programlama, engellemeyen G/Ç, npm, JavaScript

2. Giriş — JavaScript'in Sunucuya Yolculuğu

JavaScript, 1995 yılında Brendan Eich tarafından Netscape için sadece on gün içinde tasarlandı. Bu tasarım, tarayıcı içinde küçük etkileşimler için yapıldı. Örneğin, form doğrulama ve buton animasyonu gibi işlemler için kullanıldı. Yaklaşık on beş yıl boyunca, JavaScript “oyuncak dil” olarak görülüyordu ve yalnızca tarayıcıda çalışabiliyordu. O zamanlar sunucu tarafında, PHP, Java, Ruby ve Python gibi diller hüküm sürüyordu.

2000'lerin sonunda iki önemli problem ortaya çıktı. Birincisi, C10K problemi olarak biliniyordu. Geleneksel web sunucuları, her gelen bağlantı için ayrı bir thread açıyordu. Ancak,

10.000 eşzamanlı bağlantıya ulaşıldığında, bellek ve işlemci tükeniyor ve sunucu çöküyordu. İkincisi, web artık statik sayfalardan gerçek zamanlı uygulamalara doğru evriliyordu. Örneğin, sohbet, canlı bildirim ve akış gibi uygulamalar ortaya çıktı. Ancak, thread-tabanlı model bu tür uygulamalara uygun değildi.

2009'da Ryan Dahl, JSConf'ta Node.js'i tanıttı. Amacı, Google'ın Chrome için geliştirdiği hızlı V8 JavaScript motorunu tarayıcıdan alıp sunucuda çalıştırmak ve üzerine olay tabanlı, engellemeyen bir girdi/çıkı modeli kurmaktır.

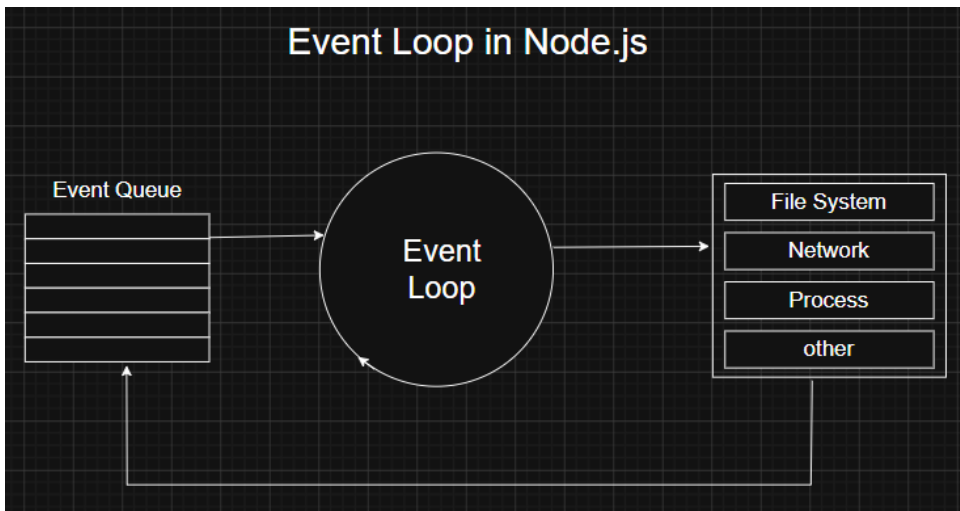
Dahl'ın sunumundaki örnek şuydu: Apache'de dosya okuma işlemi tüm işlemi bekletirken, Node'da program dosya okunurken başka işlere devam edebiliyordu.

Bu durumdan yola çıkarak, Node.js'in sadece bir teknoloji değil, sunucu programlamada bir paradigma değişimi olduğunu kabul edebiliriz. Özetle thread başına istek modelinden olay döngüsü modeline geçiş diyebiliriz.

3. Node.js Nedir ve Nasıl Çalışır?

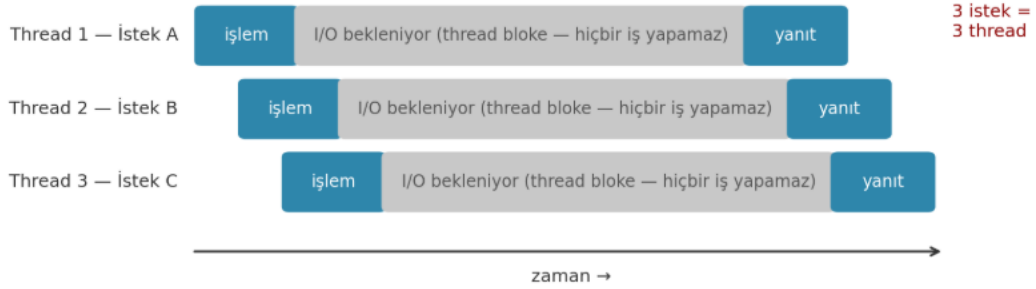
Node.js, bir programlama dili veya framework değildir. Açık kaynaklı bir çalışma ortamıdır ve JavaScript'i tarayıcı dışında çalıştırır. Node.js'nin üç ana bileşeni vardır: V8 motoru, libuv kütüphanesi ve kendi standart kütüphaneleri. V8 motoru, JavaScript kodunu makine koduna çevirir. Libuv kütüphanesi, C ile yazıldı, olay döngüsünü ve asenkron I/O'yu sağlıyor. Node'un kendi standart kütüphaneleri ise dosya sistemi, ağ, kriptoloji gibi işlemleri içerir.

Node.js, ana iş parçacığında tek bir thread çalıştırır. Bu, ilk bakışta bir zayıflık gibi görünse de, sırrı çalışma şeklinde gizli. Geleneksel modelde, her müşteriye bir garson atanır ve garson, mutfak yemeği hazırlayana kadar o masanın başında bekler. Fakat Node.js modelinde, tek garson siparişi alır, mutfaka iletir ve beklemeden diğer masaya geçer. Yemek hazır olunca, mutfak zili çalar ve garson servisi yapar. Burada mutfak, işletim sistemi ve libuv'un thread havuzunu temsil eder. Ağır işler, arka planda paralel yürür, ancak JavaScript kodu tek thread'de akar.

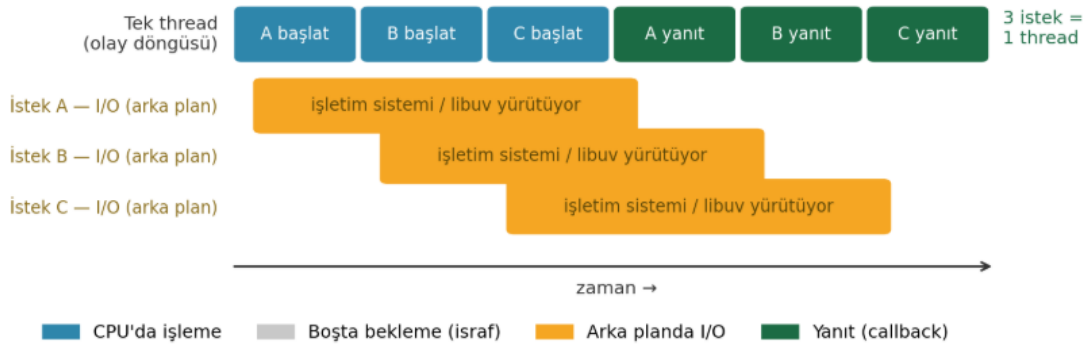


Event loop'un çalışması oldukça basit. Gelen her istek bir olay olarak kuyruğa eklenir. Olay döngüsü sürekli olarak döner: kuyruktan olayı alır, ilgili callback fonksiyonunu çalıştırır, gerektiğinde işleri başka bir sisteme devreder ve bir sonraki olaya geçer. İşlem tamamlandığında sonuç yeni bir olay olarak kuyruğa eklenir. Bu sayede tek bir thread binlerce bağlantıyı aynı anda yönetebilir, çünkü hiçbir bağlantı için boşa bekleme yapılmaz.

(a) Engelleyen (Blocking) Model — istek başına thread



(b) Engellemeyen (Non-blocking) Model — Node.js olay döngüsü, tek thread



Blocking ve non-blocking arasındaki farkı somut bir örnekle açıklamak mümkün. Bir web sayfası isteği geldiğinde, veritabanı sorgusu 100 milisaniye sürerse, blocking modelde thread bu süre boyunca hiçbir şey yapamaz. 1000 eşzamanlı istek için 1000 thread gerekir ve her thread yaklaşık 1MB bellek yeri kullanır. Non-blocking modelde ise tek bir thread sorguyu başlatır ve 100 milisaniye boyunca diğer 999 isteği işlemeye devam eder.

Asenkron kodun gelişimi de önemli bir konudur. İlk yıllarda callback fonksiyonları kullanılıyordu, fakat iç içe geçtiklerinde okunması oldukça zorlaşıyordu. Bu durum literatürde “callback hell” veya “pyramid of doom” olarak bilinir. ES6 ile zicirlenebilen Promise yapısı geldi ve then blokları kullanıma sunuldu. Daha sonra ES2017 ile async/await geldi ve asenkron kod artık senkron kod gibi okunabilir hale geldi. Bu evrimi anlatarak dilin olgunlaştığını söyleyebiliriz.

4. Ekosistem — npm ve Framework'ler

Node'un başarısının yarısı teknolojisinden değil ekosisteminden gelir. npm (Node Package Manager) 2010'da çıktı ve bugün 2 milyondan fazla paketle dünyanın en büyük yazılım deposu oldu. Mantığıysa, her proje bir package.json dosyasıyla bağımlılıklarını bildirir, npm install komutu hepsini indirir. Bir geliştirici kimlik doğrulama, tarih işleme, PDF üretme gibi işleri sıfırdan yazmak yerine hazır paketleri birleştirir. Bu, geliştirme hızını oldukça artırdı.

Framework katmanına gelince: Node'un çekirdeği düşük seviyeli; pratikte herkes bir framework kullanır. Express (2010) minimalist yapısıyla fiili standart oldu — birkaç satırla route tanımlarsın. NestJS kurumsal projeler için Angular benzeri katmanlı mimari sunuyor. Fastify performans odaklı. Ayrıca Socket.io (gerçek zamanlı iletişim) ve Electron'a (masaüstü uygulamalar), VS Code, Discord, Slack bunlarla yazıldı) bir cümleyle değinmek Node'un kapsamını gösterir.



Madalyonun öteki yüzü ise bağımlılık riski. Burada Mart 2016'da yaşanan left-pad vakasından örnek verirsek, Azer Koçulu adında bir geliştirici, npm ile yaşadığı bir isim anlaşmazlığı sonrası tüm paketlerini depodan sildi. Bunlardan biri, bir metnin soluna boşluk ekleyen sadece 11 satırlık "left-pad" paketi idi. Ama binlerce büyük proje (Babel, React dahil) zincirleme olarak bu pakete bağımlıydı — silinince dünya genelinde derleme sistemleri çöktü. Vaka iki şey öğretir: ekosistemin gücü aynı zamanda kırılganlığıdır, ve modern yazılım devasa bir bağımlılık zinciridir.

5. Gerçek Dünya Vakaları

PayPal 2013'te Java tabanlı bir mimariden Node'a geçti. Frontend JavaScript, backend Java — iki ayrı ekip, iki ayrı dil. Tek dile indirgeyince işler değişti. Uygulama neredeyse iki kat hızlı geliştirildi, daha az kod satırıyla yazıldı. Sayfa yanıt süresi %35 iyileşti. Saniyede işlenen istek sayısı iki katına çıktı. Asıl mesele teknik değil, organizasyoneldi. Tek dil, tek ekip.

Netflix dünyanın en büyük akış platformu. Kullanıcı arayüzü katmanını Java'dan Node'a taşıdılar. En çarpıcı rakam: uygulama başlangıç süresi 40+ dakikadan 1 dakikanın altına düştü. Geliştiricilerin günde onlarca kez yaptığı derle-test et döngüsü tamamen değişti. A/B testleri de hızlandı.

LinkedIn 2011'de mobil backend'leri Ruby on Rails üzerindeydi. Node'a geçince aynı trafiği 30 sunucu yerine 3 sunucuyla karşıladılar. Bazı senaryolarda 20 kata varan performans artışı raporladılar. Bu, Node'un I/O-yoğun, çok sayıda küçük istek alan mobil API senaryosuna mükemmel uyduğunun kanıtı.

Walmart 2013-2014'te Black Friday gibi yılın en yüksek trafikli gününde mobil trafiğini Node üzerinden yürüttü. Sistem sorunsuz çalıştı. CPU kullanımı %1 civarında kaldı. Node'un ani trafik patlamalarına dayanıklılığı burada net görülüyor.

Ortak payda? Hepsi I/O-yoğun, yüksek eşzamanlılık gerektiren, gerçek zamanlıya yakın sistemler. Ağır hesaplama yapan yok.

6. Ne Zaman Kullanmalı, Ne Zaman Kullanmamalı?

Güçlü olduğu alanlar net. Gerçek zamanlı uygulamalar - sohbet, canlı bildirim, oyun sunucuları. WebSocket ile bunlar doğal olarak uyumlu. RESTful API'ler ve mikroservisler de aynı şekilde ; hafif, hızlı başlıyor, konteyner çağına tam uygun. Akış uygulamaları, yani veriyi parça parça işleme. Tek sayfa uygulamaların backend'i. IoT — çok sayıda cihazdan gelen küçük mesajlar. Ortak nokta? Hepsi I/O-bound yükler.

Zayıf olduğu alanlar da bir o kadar belirgin. CPU-bound işler: video kodlama, görüntü işleme, ağır matematiksel hesaplar, makine öğrenimi eğitimi. Sorun şu: tek thread'de uzun süren bir hesaplama olay döngüsünü bloke ediyor. O hesap bitene kadar sunucu hiçbir isteğe yanıt veremiyor. Restoran metaforuna dönersek: garson mutfağa girip kendisi 2 saat yemek pişirmeye kalkarsa tüm restoran servis alamaz.

Dengeli bakarsak, Node bu zaafı tamamen çözemese de hafifletti. Worker Threads (Node 10.5+ ile geldi, ağır hesabı ayrı bir thread'e taşıma imkânı) ve cluster modülü (çok çekirdekli işlemcilerde birden fazla Node süreci çalıştırma).

7. Sonuç ve Gelecek

Özetle: Node.js, JavaScript'i tam yığın bir dile dönüştürdü ve olay-tabanlı mimariyi ana akıma taşıdı. Bugün elbette olgun bir teknoloji, ama artık yalnız değil. Bizzat Ryan Dahl, 2018'de "Node.js hakkında pişman olduğum 10 şey" başlıklı konuşmasında Node'un tasarım hatalarını sıraladı ve **Deno**'yu duyurdu (güvenlik odaklı, TypeScript yerleşik). 2022'de çıkan **Bun** ise hız iddiasıyla geldi (JavaScriptCore motoru, çok daha hızlı paket yükleme). Bu rakiplerin varlığını Node'un zayıflığı olarak değil, kurduğu soyut çerçevenin zaferi olarak yorumlayabiliriz: hepsi Node'un açtığı yolda yürüyor ve rekabet Node'un kendisini de geliştirmesine yol açtı.

8. Kaynakça

Node.js. (t.y.). *The Node.js event loop, timers, and process.nextTick()*.

<https://nodejs.org/en/learn/asynchronous-work/event-loop-timers-and-nexttick>

Node.js. (t.y.). *Overview of blocking vs non-blocking*.

<https://nodejs.org/en/learn/asynchronous-work/overview-of-blocking-vs-non-blocking>

libuv. (t.y.). *libuv: Cross-platform asynchronous I/O*. <https://libuv.org>

V8. (t.y.). *What is V8?* <https://v8.dev>

Kegel, D. (t.y.). *The C10K problem*. <http://www.kegel.com/c10k.html>

Dahl, R. (2009). *Node.js* JSConf EU 2009, Berlin. "Ryan Dahl: Original Node.js presentation"

Dahl, R. (2018). *10 things I regret about Node.js* JSConf EU 2018, Berlin.

Harrell, J. (2013, 22 Kasım). *Node.js at PayPal*. PayPal Technology Blog.

<https://medium.com/paypal-tech/node-js-at-paypal-4e2d1d08ce4f>

Harrell, J. (2013). Release the Kraken: A story of Node.js in the enterprise *ACM Queue*, Node Summit 2013. https://queue.acm.org/detail_video.cfm?id=2602531

Liu, A. (2014). JavaScript and the Netflix user interface. *ACM Queue*, 12(10).

<https://queue.acm.org/detail.cfm?id=2677720>

Node at LinkedIn: The pursuit of thinner, lighter, faster (2013). *ACM Queue*, 11(12).

<https://queue.acm.org/detail.cfm?id=2567673>

O'Dell, J. (2011). *Exclusive: How LinkedIn used Node.js and HTML5 to build a better, faster app*. VentureBeat. <https://venturebeat.com/dev/linkedin-node/>

Hoff, T. (2012). *LinkedIn moved from Rails to Node: 27 servers cut and up to 20x faster*. High Scalability. <https://highscalability.com/linkedin-moved-from-rails-to-node-27-servers-cut-and-up-to-2/>

Hammer, E. (2013). Notes from the battlefield: Node.js at Walmart . *ACM Queue*, Node Summit 2013. https://queue.acm.org/detail_video.cfm?id=2601380

Tal, L. (2019). *npm passes the 1 millionth package milestone! What can we learn?* Snyk. <https://snyk.io/blog/npm-passes-the-1-millionth-package-milestone-what-can-we-learn/>

npm, Inc. (2020, Nisan). *So long, and thanks for all the packages!* The npm Blog. <https://blog.npmjs.org/post/615388323067854848/so-long-and-thanks-for-all-the-packages.html>

Wikipedia. *npm*. <https://en.wikipedia.org/wiki/Npm>

Futurice. (2014). *NPM registry in numbers*. <https://www.futurice.com/blog/npm-registry-in-numbers>