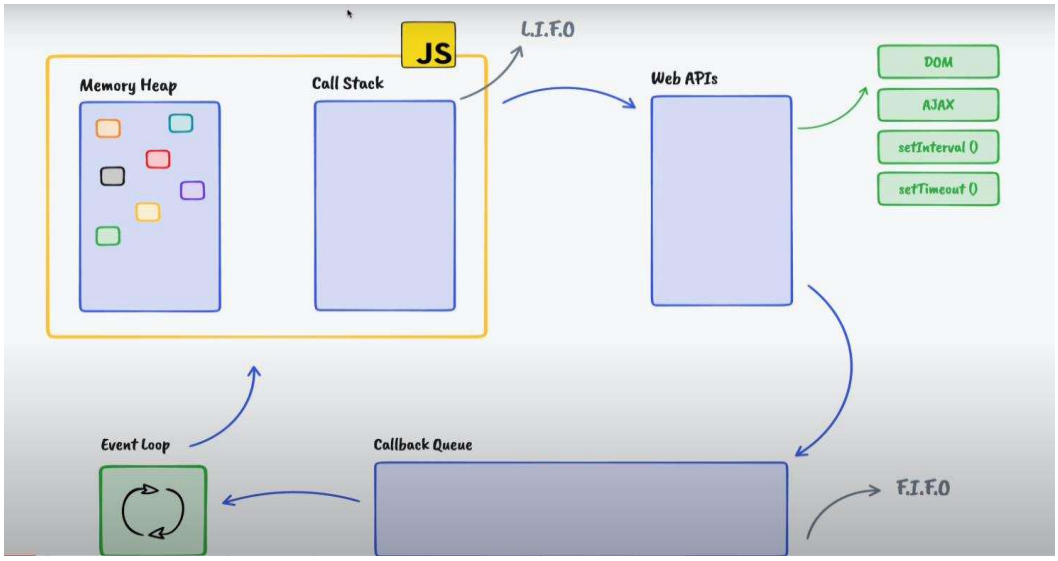
EVENT LOOP

Javascript **single thread** bir dildir yani aynı anda tek bir işlem yapabilir. Bu durum bazı programlarımızda sorun yaşamamıza sebep olabilir örneğin yazdığımız kodda uzun süren bir işlem gerçekleşiyor, bu işlem bitene kadar diğer işlemler durdurulacaktır. Bu şekildeki bir bekletme günümüz için kabul edilebilir bir durum değildir.

Event loop tam da burada devreye giriyor. Event loop, JavaScript'in single-threaded yapısının dezavantajlarıyla başa çıkabilmesi için tasarlanmış bir mekanizmadır. Bu mekanizma, asenkron programlama modelini destekleyerek uzun süren işlemlerin diğer işlemleri bloke etmeden çalışmasını sağlar. Daha akılda kalıcı olması açısından günlük hayattan bir örnek vermek gerekirse, bir kahve dükkanından kahve almak için sıraya girdiğinizi düşünün. Barista, ilk gelen müşterinin siparişini alıp kahvesini teslim ettikten sonra diğer müşteriye geçmek yerine siparişi başka bir çalışana ileterek kahvenin hazırlanmasını sağlar, bu şekilde diğer müşteriyi bekletmeden sipariş almaya devam eder.

Peki bunu nasıl yapıyor ? Event loop, callback fonksiyonları, Promises, ve async/await gibi asenkron yapılarla birleşerek çalışır. Event loop'un çalışma prensibi :



JavaScript kodu çalıştığında, her fonksiyon çağrısı Call stack'e eklenir. Call stack, LIFO (last in, first out) şeklinde çalışan bir yapıdır. İşlem sırası gelen kodlar çalışır ve silinir. Ancak, eğer bir fonksiyon uzun süren bir işlemi temsil ediyorsa, bu fonksiyon Web API'a yönlendirilir. Bu sayede diğer işlemler bekletilmez ve programın akışı kesintisiz devam eder.

Web API, tarayıcı tarafından sağlanan ve asenkron işlemleri gerçekleştirmeye olanak sağlayan bir yapıdır. Uzun süren işlemler Web API'da tamamlandığında, ilgili callback fonksiyonları callback kuyruğuna eklenir. Bu adım, asenkron işlemlerin tamamlanmasını bekler.

Callback queue, FIFO (first in, first out) şeklinde çalışan bir yapıdır. Bu kuyruk, callback fonksiyonlarının sırayla beklediği bir yapıdır.

Event loop, sürekli olarak çalışan bir döngüdür. Her döngüde, call stack'i kontrol eder. Eğer call stack boşsa ve callback kuyruğunda işlem bekleniyorsa, callback fonksiyonları sırayla call stack'e eklenir ve çalıştırılır. Bu sayede asenkron işlemler, diğer işlemleri bloke etmeden program içinde sırayla işlenir.

Sonuç olarak, JavaScript'in single-threaded yapısının getirdiği zorlukları aşmak için event loop, asenkron programlama modelini destekleyen kritik bir mekanizmadır. Uzun süren işlemleri diğer işlemleri bekletmeden yöneterek, kullanıcı deneyimini daha hızlı ve etkileşimli hale getirir.

RABBİTMQ

Rabbitmq, open source (açık kaynaklı), AMQP protokolünü kullanan bir mesaj kuyruklama sistemidir. Mesaj kuyruklama sistemi, uygulamalar arasında veri iletişimini düzenleyen sistemdir. Üreticiler mesajları kuyruğa gönderir, tüketiciler ise bu mesajları kuyruktan alıp işler ve mesajın durumunu geri bildirir. Bu işlemler asenkron olarak gerçekleşir.

Neden buna ihtiyaç duyduğumuz konusuna gelirsek bu sistemler, dayanıklılık, iş akışı düzenleme, yük dengeleme ve esnek iletişim gibi avantajlar sunarak büyük ve karmaşık sistemlerin etkili yönetimini sağlar.

Kullanım alanları :

- Yoğun E-mail gönderilen senaryolar,
- Yoğun bir şekilde veri işlemesinin bulunduğu alanlar,
- Yoğun işlem hacminin bulunduğu yerler,
- Veri kaybetmemenin önemli olduğu yerlerdir.

Çalışma prensibi şu şekildedir :

Publisher mesajı channel'lar aracılığıyla rabbitmq ya iletir. Sonra exchange bileşeni ile bir veya birden fazla kuyruğa yerleştirilir. Bu yerleştirilme işlemi için birden fazla exchange type bulunmaktadır. Kuyruklara yerleştirilme işleminden sonra consumer ile kuyruktan mesajlar çekilir, işlenir ve rabbitmq ya mesajın durumu ile ilgili geri bildirimde bulunur. Işlem başarılı olması durumunda rabbitmq'da bulunun kuyruktaki mesaj silinir, eğer başarılı değilse mesaj kuruğa yeniden yerleşitirilir ve işleme tekrar alınır.