1. **Callback Hell** asenkron çalışmaların çok fazla olduğu birbiri ardından veya beraber çalışma ihtiyacı olduğu durumda kodunuz giderek iç içe girmiş şekilde Callback yığınlarının olduğu bir yapıya dönüşüyor. Buna Callback Hell deniyor. Bu durumu aşağıdaki örnektede görebilirsiniz Kodun giderek okunamaz hale geldiğini görebilirsiniz.

const waitThenCall = (waitTime, callback) => {  
 setTimeout(() => { callback() }, waitTime \* 1000)  
}waitThenCall(1, **() => {**  
 console.log("After1 Seconds")  
 waitThenCall(2, **() => {**console.log("After2 Seconds")   
 waitThenCall(3, **() => {**console.log("After3 Seconds")   
 waitThenCall(4, **() => {**console.log("After4 Seconds")  
 waitThenCall(5, **() => {**console.log("After5 Sec")})  
 })  
 })  
 })  
})

Callback Hell callback sayısının çok olduğu iç içe olduğu yönetilemediği durumlarda karşımıza çıkar bunun için geliştirilmiş farklı farklı kütüphaneler ve tasarım örüntüleri bulunmaktadır.

Callback Hell’den kaçınmak için Async/Await yapısı kullanılabilir.

2) NodeJs Single Thread ve asenkron yapıda çalışır. Asenkron çalıştığı için Api isteklerinde veya databaseden veri çekerken gerçekleşebilecek yoğun trafikte asenkron yapılar kullanılarak uzun sürebilecek bir işlem bitmeden diğer işleme geçilmesi engellenir. Bu şekilde aynı anda gelen istekler de yönetilebilir.

3) 

