

**LIBRARY MANAGEMENT PROJECT**

Kütüphane Yönetim Uygulamamız, kütüphanelerin ve kitap meraklılarının ihtiyaçlarını karşılamak için tasarlanmış kapsamlı bir çözümdür. Bu güçlü uygulama, kitaplar, yazarlar, ve yayınevleri gibi önemli verileri kolayca yönetmek, üyelerin kitap alım ve iade işlemlerini izlemek, aynı zamanda kütüphane işletmecilerine zaman ve kaynak tasarrufu sağlamak için geliştirilmiştir.

**ANA ÖZELLİKLER:**

**Kitap Yönetimi:** Kitapların detaylı kayıtlarını tutabilirsiniz. Her kitap için başlık, yazar, yayınevi, yayın tarihi gibi önemli bilgileri kolayca ekleyebilirsiniz. Bu sayede kütüphanenizin koleksiyonunu düzenlemek ve güncellemek çok daha kolay olur.

**Üye Takibi:** Üyelerinizi kaydedebilir ve her bir üyenin hangi kitapları aldığını ve iade ettiğini izleyebilirsiniz. Bu, üyelerin hesaplarını düzenli ve güncel tutmanıza yardımcı olur.

**Otomatik Bildirimler:** Üyeler kitap aldıklarında ve iade tarihlerini geçirdiklerinde otomatik bildirimler gönderilir. Bu, üyelerin kitapları hatırlamasına ve gecikmeleri önlemelerine yardımcı olur.

**Cezaların Otomatik Hesaplanması:** Üyeler kitapları zamanında iade etmediklerinde veya hasarlı getirdiklerinde ceza miktarları otomatik olarak hesaplanır. Bu sayede kütüphane işletmecileri manuel hesaplamalarla uğraşmak zorunda kalmaz.

**Raporlar ve İstatistikler:** Kütüphane verilerinizi analiz etmek için kullanışlı raporlar ve istatistikler sunar. Bu sayede kütüphane performansınızı izleyebilir ve iyileştirmeler yapabilirsiniz.

**Faydalar:**

Kütüphane işleyişini verimli hale getirir.

Üyelerin kitap alım ve iade işlemlerini basit ve sorunsuz hale getirir.

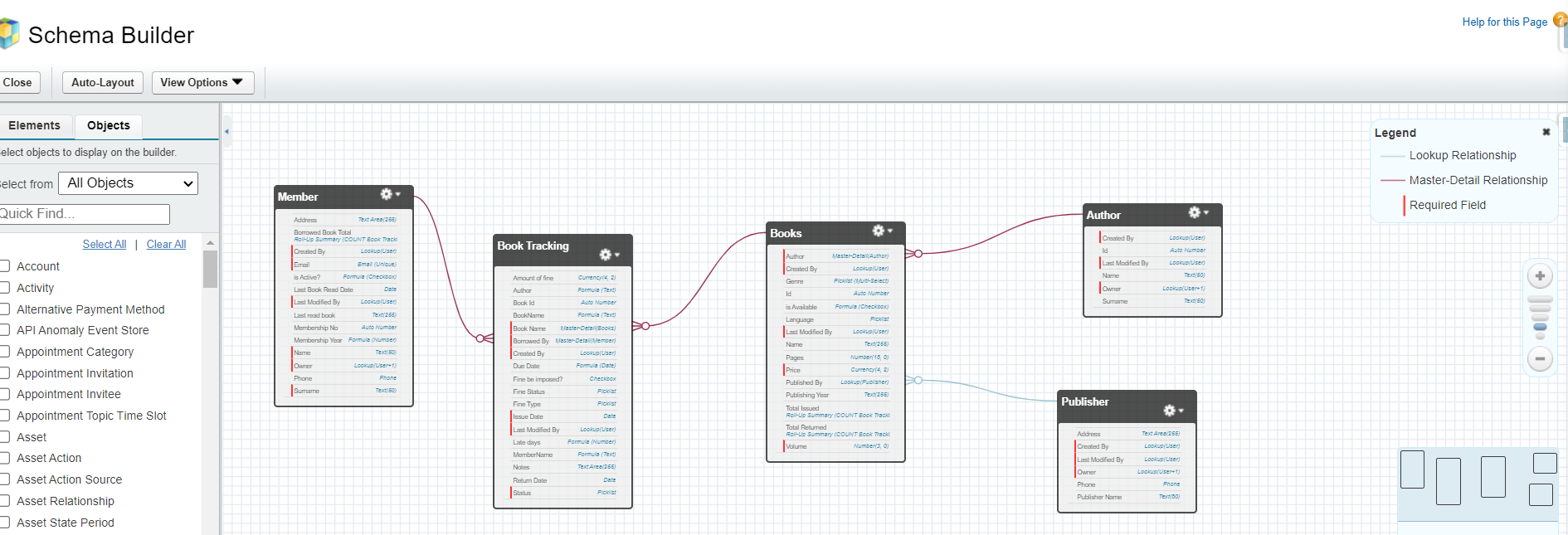
Üyelere hatırlatmalar ve bilgilendirme sağlayarak gecikmeleri azaltır.

Cezaların otomatik hesaplanması ile gelir kaybını önler.

Verileri düzenli ve güncel tutarak kütüphane yönetimini kolaylaştırır.

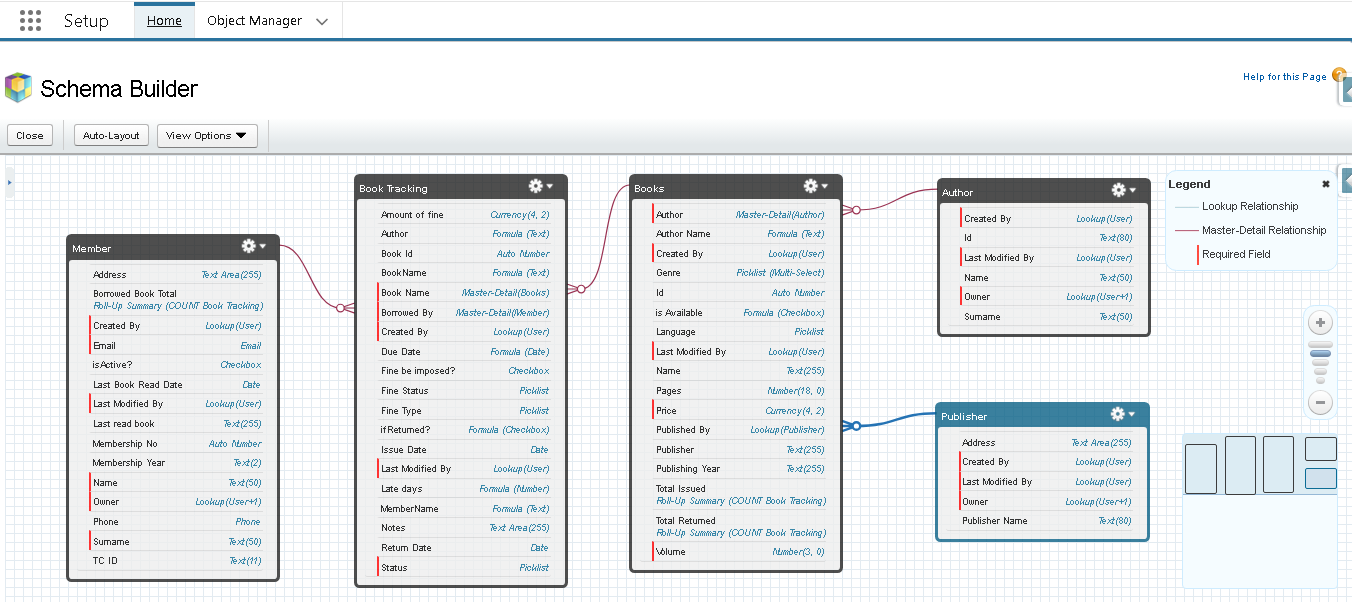
Kütüphane Yönetim Uygulamamız, kütüphane işletmecilerine ve üyelere daha iyi bir deneyim sunmak için tasarlanmıştır. Kitapları, yazarları, ve yayınevlerini yönetmek artık daha kolay ve verimli. Bu uygulama ile kütüphaneniz daha düzenli ve etkili bir şekilde işleyecek ve üyeleriniz daha memnun olacaklar.

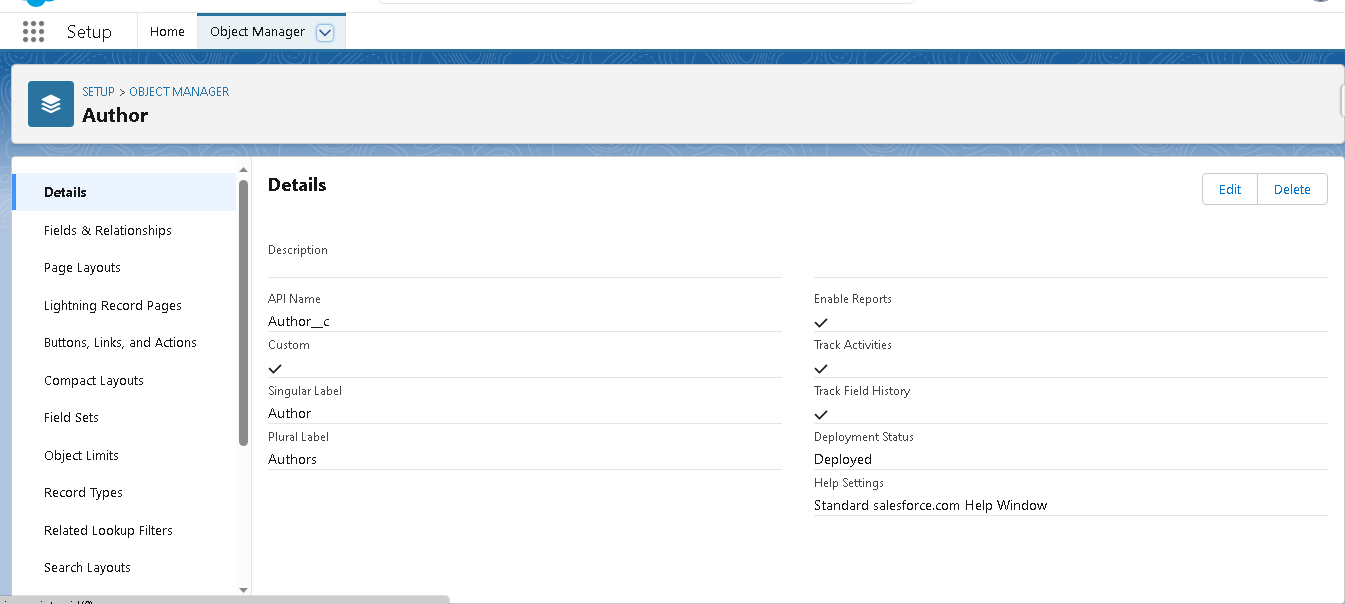
**DATA MODELLEMESİ**

****

**OBJELER, FİELDLAR ve AMAÇLARI**

**AUTHOR\_\_C CUSTOM OBJECT**





Author\_\_c Custom Object, kütüphanede yer alan kitapların yazarlarının bilgilerinin tutulduğu objecttir. Bu objenin amacı, kütüphanedeki kitapların hangi yazara ait olduğu süreçlerini izlemek ve otomatikleştirmek için bir çözüm sunmaktadır.

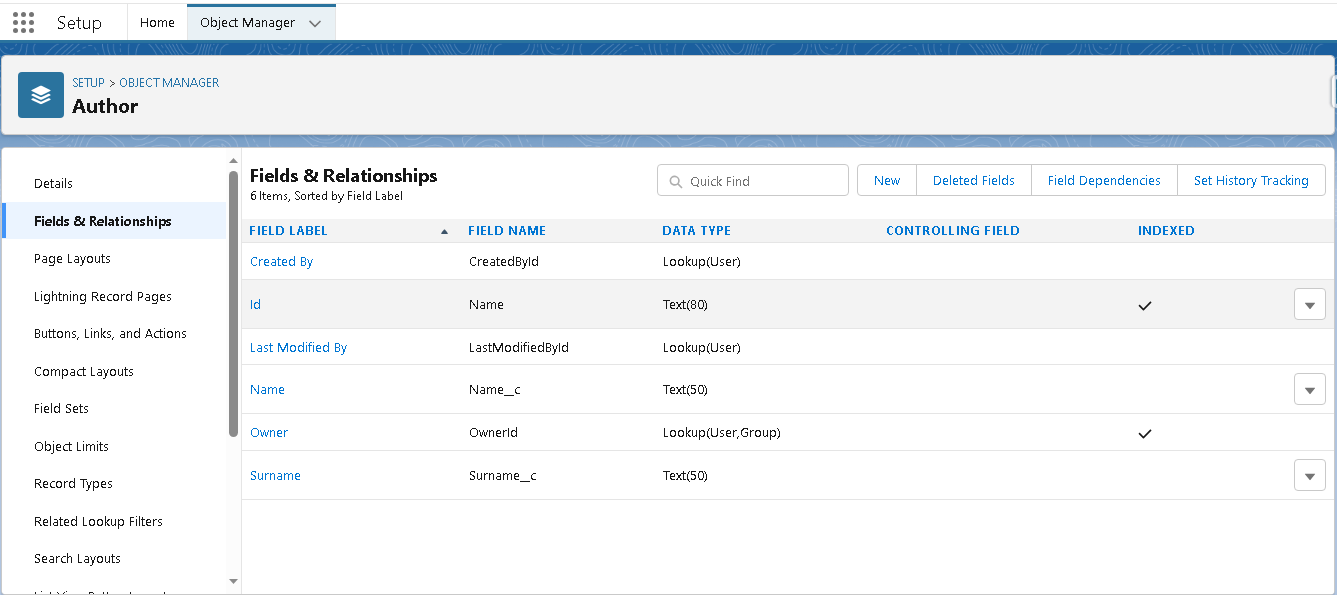
Author\_\_c Custom Object Amaçları ve İşlevleri:

1. Yazar bilgilerini saklama: Kütüphanede bulunan yazarların ad, soyad ve id’leri gibi bilgilerini tutar.
2. Yazar bazlı kitap takip bilgilerini izleme: Kütüphanedeki hangi kitabın hangi yazara ait olduğunu, bir yazarın kaç kitabı olduğu bilgileri bu objede saklanır.

**İlgili Nesneler (Related Objects)**

1. User Object (Lookup ilişkisi ile): "CreatedBy" alanı, "User" object ile Lookup ilişkisi kurar. Bu ilişki, Author kaydını oluşturan Salesforce kullanıcısını belirtir. Aynı şekilde "Last Modified By" alanı da Author kaydını son olarak kimin değiştirdiğini gösterir.

**AUTHOR\_\_C OBJECT FIELDS**

****

Aşağıda Member objecti içinde bulunan fieldlar, bunların data typleri ve ne işe yaradıkları bulunmaktadır:

1. Created By(CreatedById)

* Data Type: Lookup(User)
* Bu alan, "User" object ile ilişki kurarak Author kaydını oluşturan kullanıcıyı gösterir.

1. Id (Name)

* Data Type: Text (80)
* Bu alan Author objesinin Id bilgilerini saklar.

1. Last Modified By (LastModifiedById)

* Data Type: Lookup(User)
* Bu alan otomatik olarak oluşur ve Author kaydını son olarak kimin değiştirdiğini gösterir.

1. Name (Name\_\_c)

* Data Type: Text(50)
* Bu alan, Author objectinin adını gösterir.

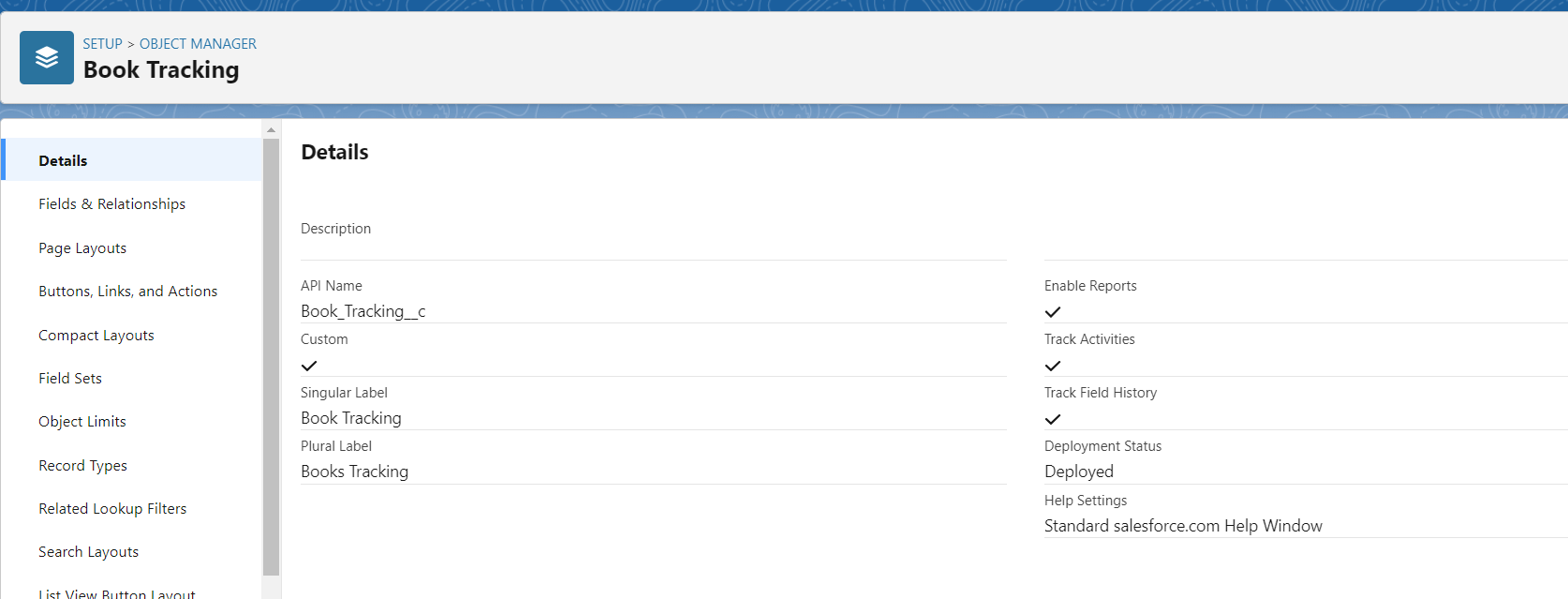
1. Owner (OwnerId)

* Data Type: Lookup(User,Group)
* Bu alan otomatik olarak oluşur. İlgili kaydın Salesforce içindeki sahibini gösterir.

1. Surname (Surname\_\_c)

* Data Type: Text (50)
* Bu alan, Author objectinin soyadını gösterir.

**BOOK TRACKİNG\_\_C" CUSTOM OBJECT**

****

**Book Tracking\_\_c" Custom Object, kitap ödünç alma ve iade işlemlerini izlemek, cezaları yönetmek ve kitapların durumunu takip etmek amacıyla oluşturulmuş custom bir objecttir. Bu nesnenin amacı, kütüphane veya benzeri bir ortamda kitaplarla ilgili süreçleri otomatikleştirmek ve yönetmek için bir çözüm sağlamaktır.**

**Book Tracking\_\_c Custom object Amaçları ve İşlevleri:**

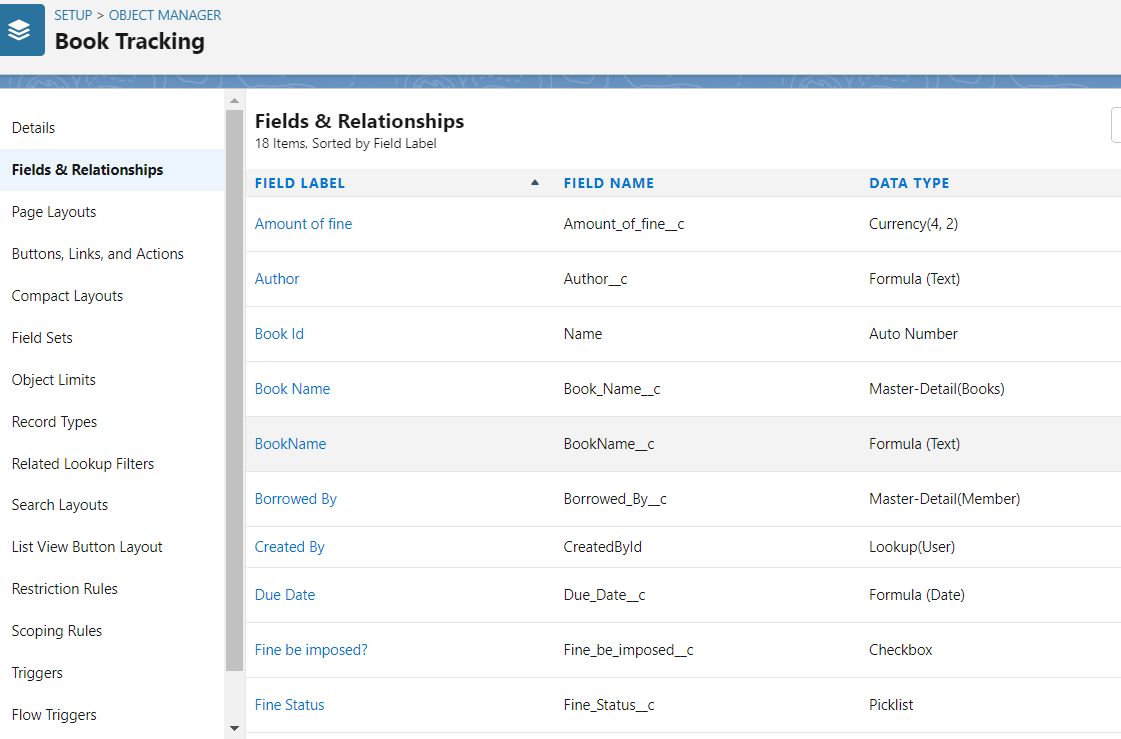
1. **Kitap Ödünç Alma ve İade İşlemleri İzleme: "Book Tracking\_\_c" custom object, ödünç alınan kitapların bilgilerini kaydetme ve bu kitapların iade edilme sürecini izleme yeteneği sağlar. Örneğin, bir üye belirli bir kitabı ne zaman ödünç aldı, ne zaman iade etti gibi bilgileri bu nesne üzerinden takip edebilirsiniz.**
2. **Cezaları ve Geç İade Durumunu Yönetme: Ödünç alınan kitapların geç iade edilmesi durumunda oluşan cezaları yönetmek için bu custom object kullanılabilir. "Amount of fine" alanı, ödünç alınan kitapla ilişkili olarak uygulanan ceza miktarını saklar. "Fine Type" ve "Fine Status" alanları ise ceza türünü ve ceza durumunu belirlemeye yardımcı olur.**
3. **Kitap Durumunu İzleme: "Status" alanı, ödünç alınan kitabın mevcut durumunu belirtir. Örneğin, kitap ödünç alındığında "Issued" olarak işaretlenirken, iade edildiğinde "Returned" olarak güncellenebilir.**

**İlgili Nesneler (Related Objects):**

1. **Books\_\_c Custom Object (Master-Detail ilişkisi ile): "Book Name" alanı, "Books\_\_c" object ile Master-Detail ilişkisi kurar. Bu ilişki, ödünç alınan kitapların hangi kitaplarla ilişkili olduğunu gösterir.**
2. **Member\_\_c custom object (Master-Detail ilişkisi ile): "Borrowed\_By\_\_c" alanı, "Member\_\_c" object ile Master-Detail ilişkisi kurar. Bu ilişki, ödünç alınan kitabı kimin ödünç aldığını gösterir.**
3. **User Object (Lookup ilişkisi ile): "CreatedBy" alanı, "User" object ile Lookup ilişkisi kurar. Bu ilişki, takip kaydını oluşturan kullanıcıyı belirtir. Aynı şekilde "Last Modified By" alanı da takip kaydını son olarak kimin değiştirdiğini gösterir.**

**"Book Tracking\_\_c" Object, kütüphane veya benzeri yerlerde kitap ödünç alma, iade, cezalar ve durum takibi gibi işlemleri kolaylaştırmak için kullanılabilir. Bu nesne ve ilişkili alanlar, kitap takibini daha düzenli ve otomatik hale getirirken, ilgili nesneler sayesinde değişiklikleri ve ilişkili bilgileri daha iyi anlamamıza yardımcı olur.**

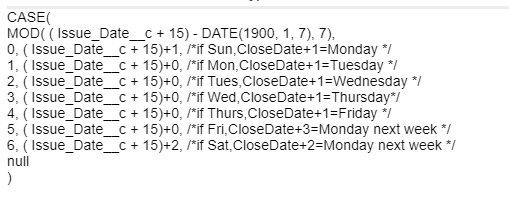
**BOOK TRACKİNG\_\_C OBJECT FİELDS**

****

****

**Bu custom object,** ödünç alınan kitapların takibini yapmak için kullanılır. Aşağıda, bu custom objectyle ilişkili olan alanlar ve bu alanların veri tipleri bulunmaktadır:

1. **Amount of fine (Amount\_of\_fine\_\_c)**
   * **Data Type:** Currency(4, 2)
   * Bu alan, ödünç alınan kitapla ilişkili olan ceza miktarını saklar
2. **Book Id (Name)**
   * **Data Type:** Auto Number
   * Bu alan, her kitap için otomatik olarak oluşturulan bir tanımlama numarasını temsil eder.
3. **Book Name (Book\_Name\_\_c)**
   * **Data Type:** Master-Detail(Books)
   * Bu alan, "Books" object ile ilişki kurarak bu takip kaydının hangi kitapla ilişkili olduğunu gösterir.
4. **Borrowed By (Borrowed\_By\_\_c)**
   * **Data Type:** Master-Detail(Member)
   * Bu alan, "Member" object ile ilişki kurarak kitabı kimin ödünç aldığını gösterir.
5. **Created By (CreatedById)**
   * **Data Type: Lookup(User)**
   * Bu alan, "User" object ile ilişki kurarak takip kaydını oluşturan kullanıcıyı gösterir.
6. **Due Date (Due\_Date\_\_c)**
   * **Data Type:** Formula (Date)
   * MOD( ( Issue\_Date\_\_c + 15) - DATE(1900, 1, 7), 7): Bu bölüm, "Issue\_Date\_\_c" alanına 15 gün eklenip 1900 yılından başlayarak geçen gün sayısı ile bölümlerini hesaplar. Bu işlem, haftanın hangi gününe denk geldiğini belirlemek amacıyla yapılır.
   * 0: Eğer mod sonucu 0 ise (yani Pazar), "Issue\_Date\_\_c" alanına 15 gün eklenerek bir sonraki pazara denk gelecek şekilde tarih hesaplanır.
   * 1: Eğer mod sonucu 1 ise (yani Pazartesi), "Issue\_Date\_\_c" alanına 15 gün eklenerek aynı tarihte kalınır.
   * 2, 3, 4, 5: Eğer mod sonucu sırasıyla 2, 3, 4, 5 ise (Salı, Çarşamba, Perşembe, Cuma), "Issue\_Date\_\_c" alanına 15 gün eklenerek aynı tarihte kalınır.
   * 6: Eğer mod sonucu 6 ise (Cumartesi), "Issue\_Date\_\_c" alanına 15 gün eklenerek bir sonraki pazara denk gelecek şekilde tarih hesaplanır.
   * null: Mod sonucu hiçbir yukarıdaki durumla eşleşmezse, yani bu hafta içinde değilse, null döndürülür.
   * **Sonuç olarak,** bu formül, "Issue\_Date\_\_c" alanına 15 gün eklenen tarihin haftanın hangi gününe denk geldiğini kontrol eder ve buna göre ilgili tarihleri hesaplar. Bu, genellikle hafta sonları ve sonraki haftanın başlangıcı gibi durumlar için kullanılır.

****

1. **Fine be imposed? (Fine\_be\_imposed\_\_c)**
   * **Data Type:** Checkbox
   * Bu alan, ödünç alınan kitap için ceza uygulanıp uygulanmayacağını gösteren bir onay kutusudur.
2. **Fine Status (Fine\_Status\_\_c)**
   * **Data Type: Picklist** (Pending, Paid ,Wave off and Canceled)

Bu alandaki seçenekler:

**Pending:** Bu durum, cezanın henüz ödenmediği veya işleme alınmadığı durumu ifade eder. Cezanın beklediği bir aşamayı temsil eder.

**Paid:** Bu durum, cezanın ödendiği ve kapatıldığı anlamına gelir. Üye tarafından ceza ödendiğinde bu durum seçilebilir**.**

**Wave off:** Bu durum, cezanın iptal edildiği veya kaldırıldığı anlamına gelir. Özel bir durumda, ceza normalden farklı bir nedenle iptal edildiğinde bu seçenek kullanılabilir.

**Canceled:** Bu durum, cezanın iptal edildiği veya geçersiz kılındığı durumu temsil eder. Örneğin, ödünç alan üye tarafından haksız yere uygulanan bir ceza iptal edildiğinde bu seçenek kullanılabilir.

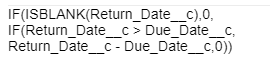
1. **Fine Type (Fine\_Type\_\_c)**
   * **Data Type:** Picklist (Late fee , Lost Book and Damage Book)
   * Bu alan, ödünç alınan kitaplarla ilişkilendirilen cezanın türünü ifade eder. Yani, kitap ödünç alma sürecinde hangi tür cezanın uygulandığını belirtir
   * **Bu alandaki seçenekler:**

**Late fee:** Bu seçenek, ödünç alınan kitabın geç iade edilmesi nedeniyle uygulanan ceza türünü belirtir. Yani, üye ödünç aldığı kitabı belirlenen süre içinde iade etmezse uygulanan ceza türüdür.

**Lost Book**: Bu seçenek, ödünç alınan kitabın kaybedilmesi veya iade edilememesi durumunda uygulanan ceza türünü belirtir. Kitap kaybolduğunda veya geri verilemediğinde bu tür bir ceza uygulanabilir.

**Damage Book:** Bu seçenek, ödünç alınan kitabın hasar görmesi durumunda uygulanan ceza türünü ifade eder. Kitapın hasar gördüğü veya kullanılamaz hale geldiği durumda bu tür bir ceza uygulanabilir.

1. **Issue Date (Issue\_Date\_\_c)**
   * **Data Type:** Date
   * Bu alan, kitabın ödünç verildiği tarihi saklar**.**
2. **Last Modified By (LastModifiedById)**
   * **Data Type:** Lookup(User)
   * Bu alan, takip kaydını son olarak kimin değiştirdiğini gösterir**.**
3. **Late days (Late\_days\_\_c)**
   * **Data Type:** Formula (Number)
   * Bu alan ;kitabın iade tarihi ("Return\_Date\_\_c") ve beklenen teslim tarihi ("Due\_Date\_\_c") arasındaki farkı hesaplar. Böylalikle kitabın geç iade edilip edilmediği ve ne kadar süreyle geç iade edildiği number olarak döndürülür.

****

1. **Notes (Notes\_\_c)**
   * **Data Type:** Text Area(255)
   * Bu alan, takip kaydına not veya yorum eklemek için kullanılır.
2. **Return Date (Return\_Date\_\_c)**
   * **Data Type:** Date
   * Bu alan, kitabın iade edildiği tarihi saklar.
3. **Status (Status\_\_c)**
   * **Data Type:** Picklist (Issued , Return and Re-Issued)
   * Bu alan, ödünç alınan kitabın mevcut durumunu ifade eder. Kitabın hangi aşamada olduğunu ve ne tür işlemler yapıldığını göstermek amacıyla kullanılır.
   * Bu alandaki seçenekler:

**Issued:** Bu durum, kitabın ödünç alındığı ve üyenin üzerinde olduğu anlamına gelir. Yani, bir üye tarafından kitap ödünç alındığında, durum "Issued" olarak işaretlenebilir.

**Return:** Bu durum, kitabın iade edildiği ve ödünç alınabilir duruma geldiği anlamına gelir. Üye kitabı geri verdiğinde veya iade edildiğinde, durum "Return" olarak güncellenebilir.

**Re-Issued:** Bu durum, kitabın tekrar ödünç alındığı anlamına gelir. Örneğin, bir üye kitabı iade ettikten sonra tekrar aynı kitabı ödünç alırsa, durum "Re-Issued" olarak güncellenebilir.

1. **Author (**Author\_\_c**)**
   * **Data Type:** Formula (Text)
   * Bu alan, "Book Tracking\_\_c" objectinde "Book\_Name\_\_c" field ile ilişkili "Books\_\_c" object ve onun içindeki "Author\_\_c" alanına ilişkili "Author" object içindeki field alanları Name\_\_c ve Surname\_\_c birleştiren bir ifade oluşturuyor

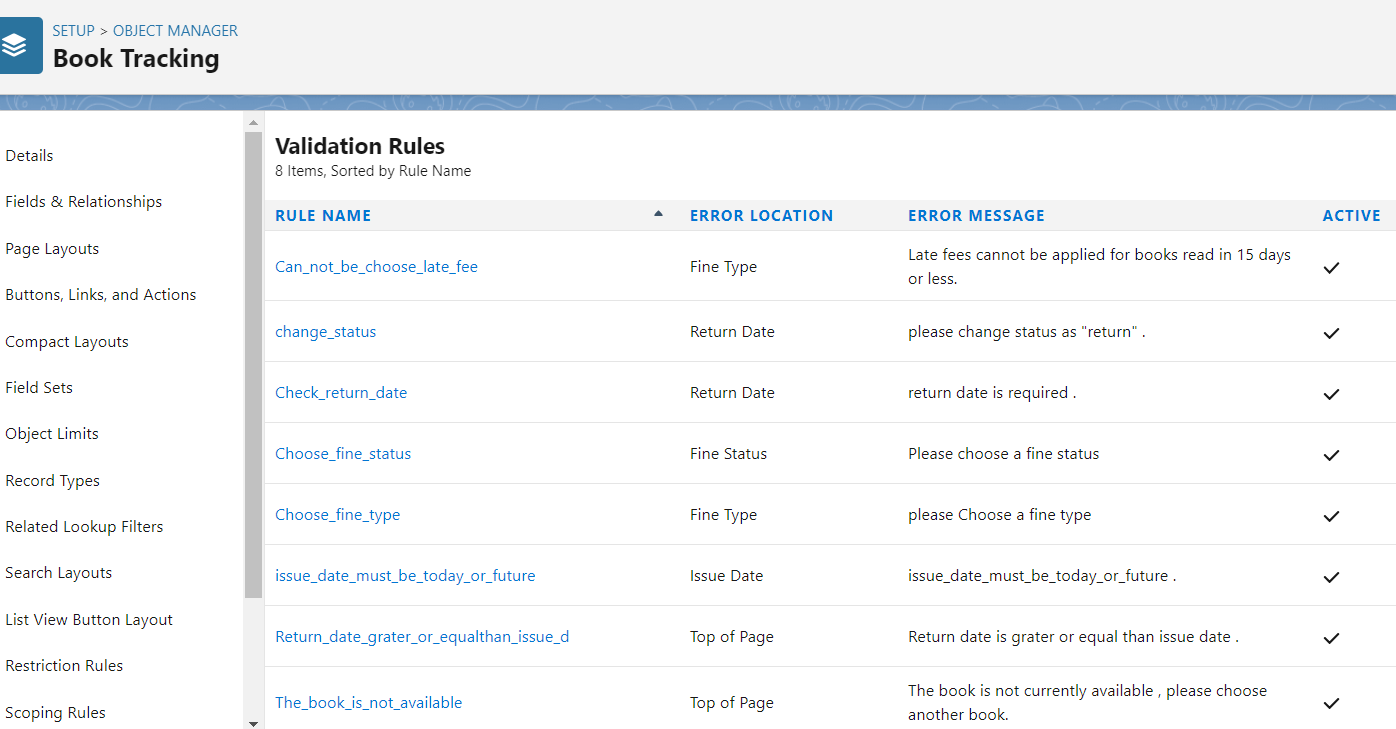
****

1. **MemberName (MemberName \_\_c)**
   * **Data Type:** Formula(Text)
   * Bu alan, "Book Tracking\_\_c" objectindeki "Borrowed\_By\_\_c" field ile ilişkili "Member" object Name\_\_c ve Surname\_\_c birleştiren bir ifade oluşturur.



**BOOK TRACKİNG VALİDATİON RULES**

**Validation Rules, "Book Tracking\_\_c" object kullanımını daha güvenli ve tutarlı hale getirir. İlgili alanlara girilen verileri kontrol ederek yanlış veya eksik bilgilerin önüne geçilmesini sağlar.**

****

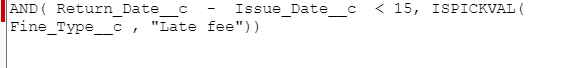
**CAN\_NOT\_BE\_CHOOSE\_LATE\_FEE VALİDATİON RULE**

* **Kural: Late fees cannot be applied for books read in 15 days or less.**

Bu kural, iki koşulu kontrol eder:

**Return\_Date\_\_c - Issue\_Date\_\_c < 15**: Bu kısım, kitabın ödünç alındığı tarih ile iade edildiği tarih arasındaki farkın 15 günden az olup olmadığını kontrol eder. Yani, kitabın ödünç alındıktan sonraki iade tarihi arasındaki gün sayısı 15'ten azsa, bu koşul doğru olarak değerlendirilir.

**ISPICKVAL(Fine\_Type\_\_c, "Late fee"):** Bu kısım, "Fine\_Type\_\_c" alanının "Late fee" (Geç cezası) olarak seçildiğini kontrol eder. Yani, ceza türü "Late fee" olarak seçildiyse, bu koşul doğru olarak değerlendirilir.

**Eğer her iki koşul da doğru olarak değerlendirilirse,** yani kitap 15 günden az bir süre içinde iade edildiyse ve ceza türü "Late fee" olarak seçildiyse, bu doğrulama kuralı devreye girer ve bir hata mesajı gösterir. Bu, kitabın erken iade edildiği durumda geç iade cezası uygulanmamasını sağlar.****

**CHANGE\_STATUS VALİDATİON RULE**

* **Kural: Please change status as "return".**

Bu kural, iki koşulu kontrol eder:

**NOT(ISNULL(Return\_Date\_\_c)):** Bu kısım, "Return\_Date\_\_c" alanının boş olmadığını (null olmadığını) kontrol eder. Yani, bir kitap iade edildiğinde, "Return\_Date\_\_c" alanına bir tarih girilir ve bu koşul doğru olarak değerlendirilir.

**NOT ISPICKVAL(Status\_\_c, "return"):** Bu kısım, "Status\_\_c" alanının "return" (İade) olarak seçilmediğini kontrol eder. Yani, kitap henüz iade edilmediğinde, "Status\_\_c" alanına "return" seçeneği atanmamışsa, bu koşul doğru olarak değerlendirilir.

**Eğer her iki koşul da doğru olarak değerlendirilirse,** yani bir kitap iade edilmişse ve iade durumu "return" olarak işaretlenmemişse, bu doğrulama kuralı devreye girer ve bir hata mesajı gösterir. Bu, bir kitap iade edildiğinde durumun "return" olarak güncellenmesini zorlar ve iade işlemi gerektiği gibi yapılmamışsa uyarı verir.

****

**CHECK\_RETURN\_DATE VALİDATİON RULE**

* **Kural: Return date is required.**

Bu kural, iki koşulu kontrol eder:

**ISPICKVAL(Status\_\_c, “Return”):** Bu kısım, “Status\_\_c” alanının “Return” (İade) olarak seçildiğini kontrol eder. Yani, bir kitap iade edildiğinde, “Status\_\_c” alanına “Return” seçeneği atanır ve bu koşul doğru olarak değerlendirilir.

**ISBLANK(Return\_Date\_\_c):** Bu kısım, “Return\_Date\_\_c” alanının boş (null) olup olmadığını kontrol eder. Yani, bir kitap iade edildiğinde, “Return\_Date\_\_c” alanına bir tarih girilir ve bu koşul yanlış (false) olarak değerlendirilir.

**Eğer her iki koşul da doğru olarak değerlendirilirse,** yani kitap iade edilmişse ve iade tarihi (“Return\_Date\_\_c”) alanına bir tarih girilmemişse, bu doğrulama kuralı devreye girer ve bir hata mesajı gösterir. Bu, bir kitapın iade işlemi yapıldığında iade tarihini girmeyi zorlar ve eksik veya hatalı iade işlemlerini önler.

****

**CHOOSE\_FİNE\_STATUS VALİDATİON RULE**

* **Kural: Please choose a fine status.**

**Bu kural, birkaç koşulu iç içe geçmiş bir şekilde kontrol eder:**

**Fine\_be\_imposed\_\_c:** Bu kısım, "Fine\_be\_imposed\_\_c" alanının doğru (true) olup olmadığını kontrol eder. Yani, bir cezanın uygulanıp uygulanmayacağını belirten bir alan varsa ve bu alan doğru ise, bu koşul doğru olarak değerlendirilir**.**

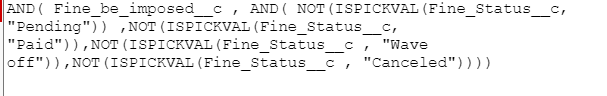
**NOT(ISPICKVAL(Fine\_Status\_\_c, "Pending**")): Bu kısım, "Fine\_Status\_\_c" alanının "Pending" (Beklemede) seçeneği olarak işaretlenmediğini kontrol eder. Yani, bir ceza durumu "Pending" olarak seçilmediğinde bu koşul doğru olarak değerlendirilir.

**NOT(ISPICKVAL(Fine\_Status\_\_c, "Paid")):** Bu kısım, "Fine\_Status\_\_c" alanının "Paid" (Ödendi) seçeneği olarak işaretlenmediğini kontrol eder. Yani, bir ceza durumu "Paid" olarak seçilmediğinde bu koşul doğru olarak değerlendirilir.

**NOT(ISPICKVAL(Fine\_Status\_\_c, "Wave off")):** Bu kısım, "Fine\_Status\_\_c" alanının "Wave off" (İptal) seçeneği olarak işaretlenmediğini kontrol eder. Yani, bir ceza durumu "Wave off" olarak seçilmediğinde bu koşul doğru olarak değerlendirilir.

**NOT(ISPICKVAL(Fine\_Status\_\_c, "Canceled")):** Bu kısım, "Fine\_Status\_\_c" alanının "Canceled" (İptal Edildi) seçeneği olarak işaretlenmediğini kontrol eder. Yani, bir ceza durumu "Canceled" olarak seçilmediğinde bu koşul doğru olarak değerlendirilir**.**

**Eğer "Fine\_be\_imposed\_\_c" alanı doğru (ceza uygulanması gerekiyorsa) ve hiçbir ceza durumu yukarıdaki seçeneklere uymuyorsa, yani "Pending", "Paid", "Wave off" veya "Canceled" değilse, bu doğrulama kuralı devreye girer ve bir hata mesajı gösterir. Bu, ceza durumu seçildiğinde ceza uygulama durumlarını kontrol eder ve uygun olmayan durumların engellenmesini sağlar..**

****

**CHOOSE\_FİNE\_TYPE VALİDATİON RULE**

* **Kural: Please choose a fine type.**

**Bu kural, birkaç koşulu iç içe geçmiş bir şekilde kontrol eder:**

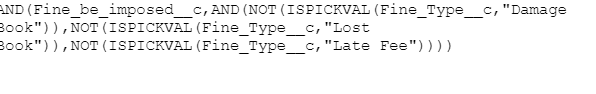
**Fine\_be\_imposed\_\_c:** Bu kısım, "Fine\_be\_imposed\_\_c" alanının doğru (true) olup olmadığını kontrol eder. Yani, bir cezanın uygulanıp uygulanmayacağını belirten bir alan varsa ve bu alan doğru ise, bu koşul doğru olarak değerlendirilir**.**

**NOT(ISPICKVAL(Fine\_Type\_\_c, "Damage Book")):** Bu kısım, "Fine\_Type\_\_c" alanının "Damage Book" (Hasarlı Kitap) seçeneği olarak işaretlenmediğini kontrol eder. Yani, ceza türü "Damage Book" olarak seçilmediğinde bu koşul doğru olarak değerlendirilir**.**

**NOT(ISPICKVAL(Fine\_Type\_\_c, "Lost Book")):** Bu kısım, "Fine\_Type\_\_c" alanının "Lost Book" (Kayıp Kitap) seçeneği olarak işaretlenmediğini kontrol eder. Yani, ceza türü "Lost Book" olarak seçilmediğinde bu koşul doğru olarak değerlendirilir.

**NOT(ISPICKVAL(Fine\_Type\_\_c, "Late Fee")):** Bu kısım, "Fine\_Type\_\_c" alanının "Late Fee" (Geç Cezası) seçeneği olarak işaretlenmediğini kontrol eder. Yani, ceza türü "Late Fee" olarak seçilmediğinde bu koşul doğru olarak değerlendirilir.

**Eğer "Fine\_be\_imposed\_\_c" alanı doğru (ceza uygulanması gerekiyorsa) ve hiçbir ceza türü yukarıdaki seçeneklere uymuyorsa, yani "Damage Book", "Lost Book" veya "Late Fee" değilse,** bu doğrulama kuralı devreye girer ve bir hata mesajı gösterir. Bu, cezanın türünü kontrol eder ve uygun olmayan türlerin engellenmesini sağlar.

****

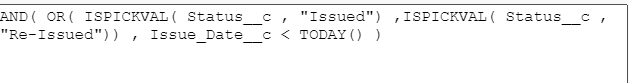
**İSSUE\_DATE\_MUST\_BE\_TODAY\_OR\_FUTURE VALİDATİON RULE**

* **Kural: issue\_date\_must\_be\_today\_or\_future.**

**Bu kural, birkaç koşulu kontrol eder:**

**OR(ISPICKVAL(Status\_\_c, "Issued"), ISPICKVAL(Status\_\_c, "Re-Issued")):** Bu kısım, "Status\_\_c" alanının "Issued" (Verildi) veya "Re-Issued" (Yeniden Verildi) seçeneği olarak işaretlenip işaretlenmediğini kontrol eder. Yani, kitap verilmiş veya yeniden verilmişse, bu koşul doğru olarak değerlendirilir.

**Issue\_Date\_\_c < TODAY():** Bu kısım, "Issue\_Date\_\_c" alanının bugünden önce (geçmiş bir tarih) olup olmadığını kontrol eder. Yani, kitap veriliş tarihi ("Issue\_Date\_\_c") bugünden önce ise, bu koşul doğru olarak değerlendirilir.

**Eğer her iki koşul da doğru olarak değerlendirilirse, yani kitap verilmiş veya yeniden verilmişse ve veriliş tarihi bugünden önceyse,** bu doğrulama kuralı devreye girer ve bir hata mesajı gösterir. Bu, geçmiş tarihte kitap veriliş durumlarını engellemek veya düzeltmeleri sağlamak için kullanılır.****

**RETURN\_DATE\_GRATER\_OR\_EQUALTHAN\_İSSUE\_D VALİDATİON RULE**

* **Kural: Return date is greater or equal than issue date.**

**Bu formül, iki tarihi karşılaştırarak bir koşulu kontrol eder:**

**Return\_Date\_\_c:** Bu, iade tarihinin alanını ifade eder**.**

**Issue\_Date\_\_c:** Bu, veriliş tarihinin alanını ifade eder.

**Eğer iade tarihi ("Return\_Date\_\_c") veriliş tarihinden ("Issue\_Date\_\_c") önceyse, yani iade tarihi kitabın verildiği tarihten daha önce bir tarihe denk geliyorsa, bu koşul doğru olarak değerlendirilir.**

**Bu tür bir doğrulama kuralı**, kitapların veriliş tarihinden önce iade edilmesini engellemek için kullanılır. Çünkü mantıklı bir senaryo olarak, bir kitabın iade tarihi, veriliş tarihinden sonra veya en azından veriliş tarihiyle aynı olmalıdır. Bu kural, bu tür durumların yanlışlıkla veya hatalı veri girişi sonucu yaşanmasını engellemek amacıyla kullanılır**.**

****

**THE\_BOOK\_İS\_NOT\_AVAİLABLE VALİDATİON RULE**

* **Kural: The book is not currently available, please choose another book.**

**Bu formül, birkaç koşulu kontrol eder:**

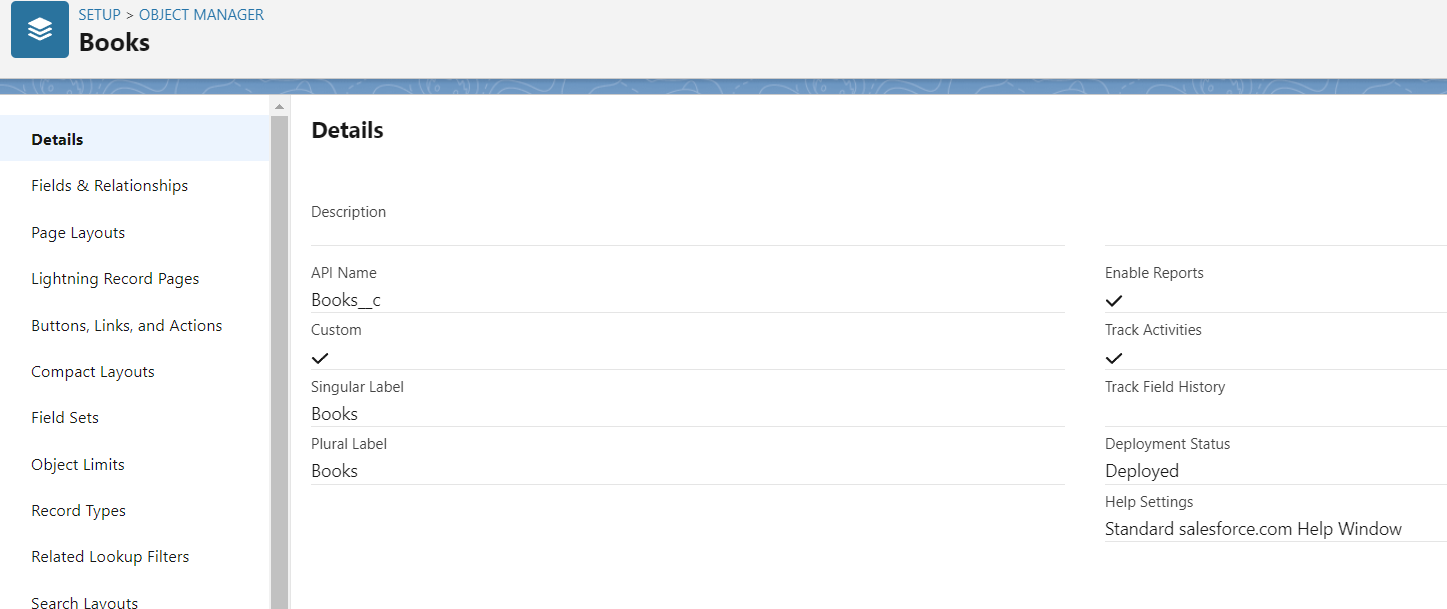
**NOT(Book\_Name\_\_r.is\_Available\_\_c):** Bu kısım, "Book\_Name\_\_r" ilişkili nesnesinin "is\_Available\_\_c" alanının doğru (true) olmadığını kontrol eder. Yani, ilişkilendirilmiş kitap ("Book\_Name\_\_r") mevcut değilse, bu koşul doğru olarak değerlendirilir**.**

**OR(ISPICKVAL(Status\_\_c, "Issued"), ISPICKVAL(Status\_\_c, "Re-Issued**")): Bu kısım, "Status\_\_c" alanının "Issued" (Verildi) veya "Re-Issued" (Yeniden Verildi) seçeneği olarak işaretlenip işaretlenmediğini kontrol eder. Yani, kitap verilmiş veya yeniden verilmişse, bu koşul doğru olarak değerlendirilir.

Eğer her iki koşul da doğru olarak değerlendirilirse, yani ilişkilendirilmiş kitap mevcut değilse ve kitap verilmiş veya yeniden verilmişse, bu doğrulama kuralı devreye girer ve bir hata mesajı gösterir. Bu, mevcut olmayan bir kitabın verilmiş veya yeniden verilmiş olarak işaretlenmesini engellemek amacıyla kullanılır.

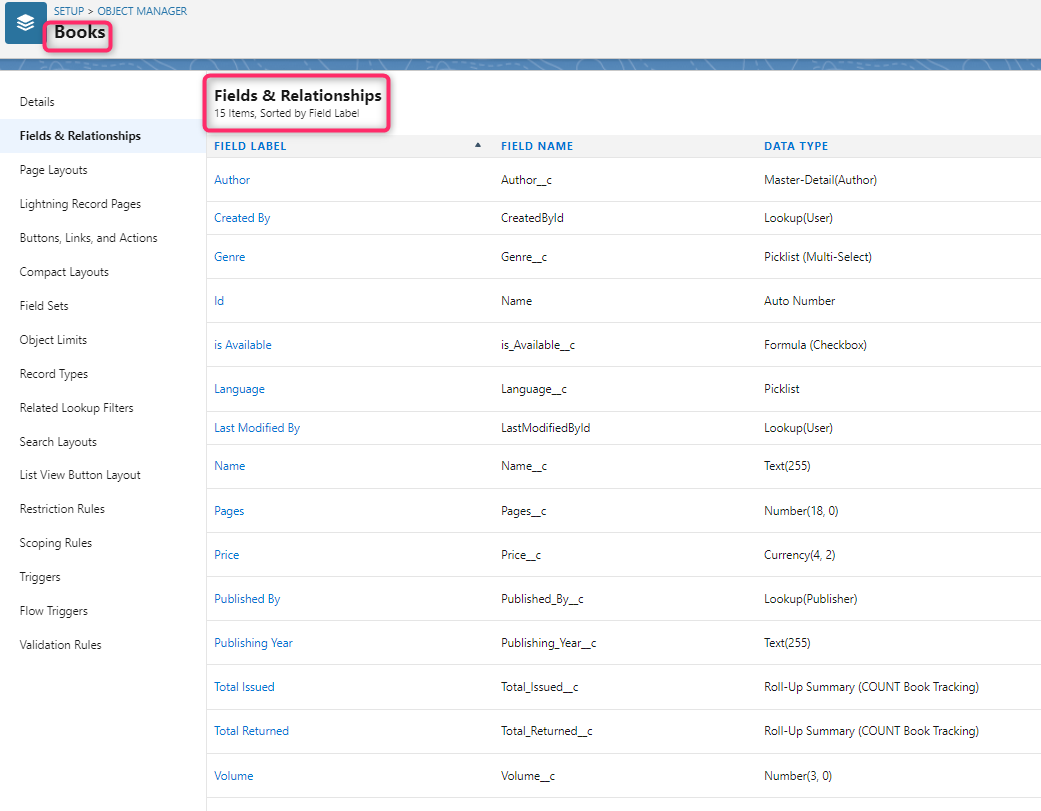
****

**BOOK CUSTOM OBJECT**



Books Custom Object, sistemde (kütüphanede) bulunan kitapların listesini tutmak amacıyla oluşturulmuş custom bir objecttir. Bu objectin amacı, kütüphane veya benzeri bir ortamda kitaplarla ilgili süreçleri otomatikleştirmek ve yönetmek için bir çözüm sağlamaktır.

**BOOK OBJECT FIELDS**

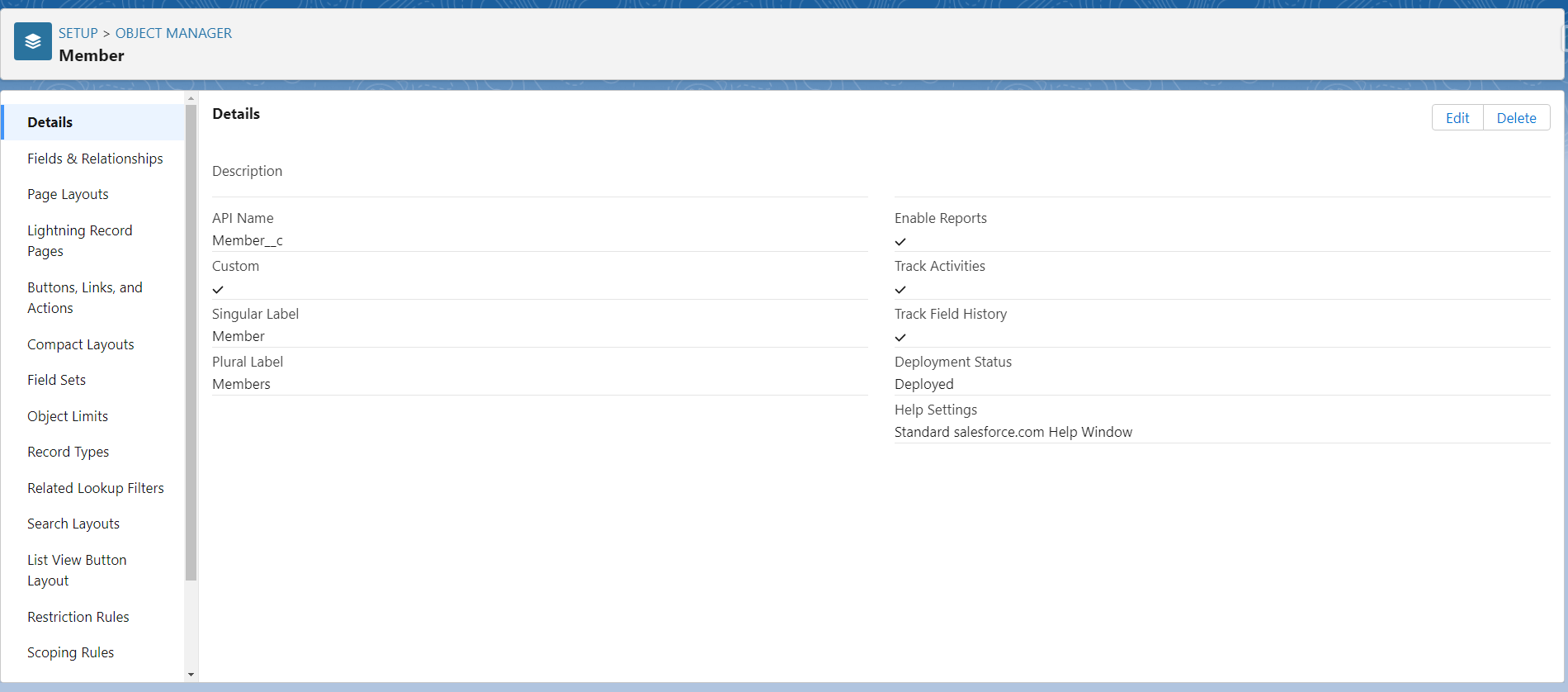




İlgili Nesneler (Related Objects)

1. Book\_Tracking\_\_c Object (Master-Detail ilişkisi ile): “Book\_Tracking\_\_c” objecti içindeki “Book\_Name\_\_c” fieldı Book objecti ile Master\_detail ilişkisi kurar. Burada Book objecti Parent rolündedir. Bu ilişki sayesinde kütüphaneden ödünç alınan kitabın hangi üye tarafından alındığı bilgisi sisteme işlenmiş olur.
2. User Object (Lookup ilişkisi ile): "CreatedBy" alanı, "User" object ile Lookup ilişkisi kurar. Bu ilişki, Member kaydını oluşturan Salesforce kullanıcısını belirtir. Aynı şekilde "Last Modified By" alanı da Member kaydını son olarak kimin değiştirdiğini gösterir.

**MEMBER\_\_C CUSTOM OBJECT**



Member\_\_c Custom Object, kütüphaneye üye olan kişilerin bilgilerinin tutulduğu objecttir.

**Member\_\_c Custom Object Amaçları ve İşlevleri**

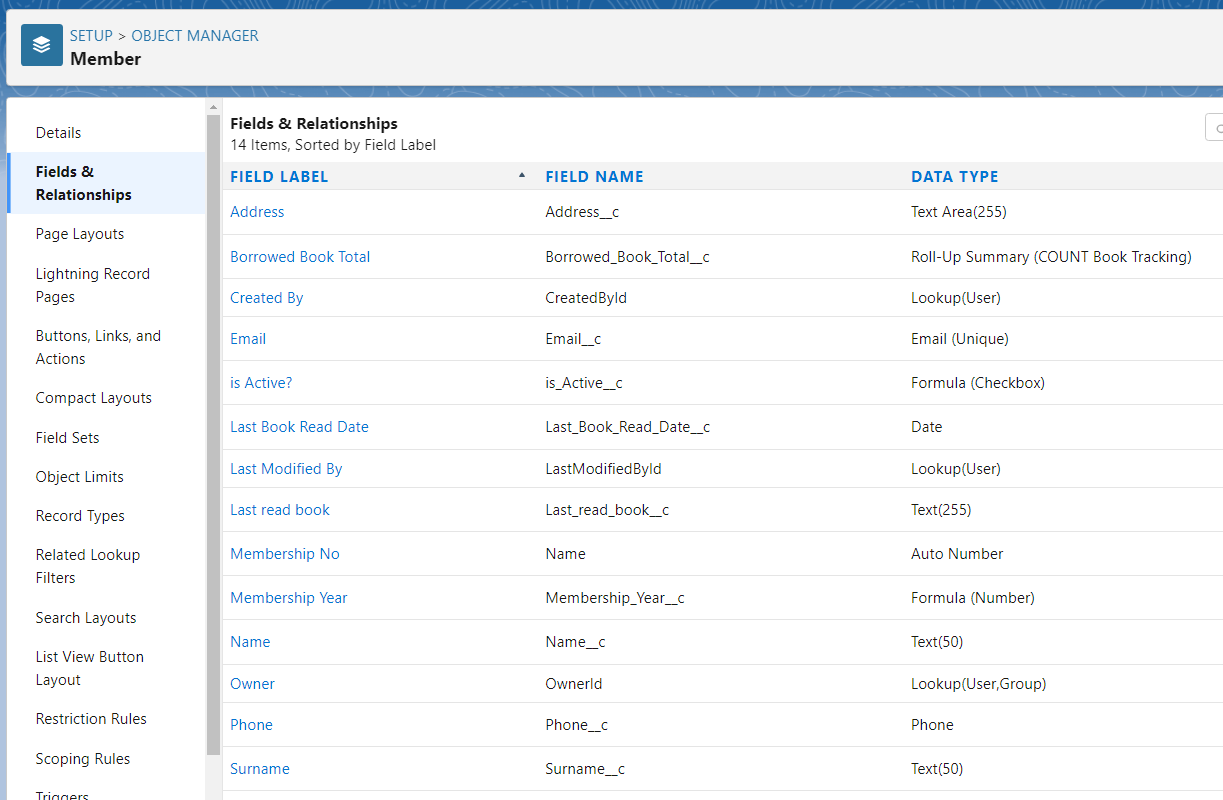
1. Üye bilgilerini saklama: Kütüphaneye üye olan kişilerin ad, soyad, email, telefon numarası, adres, üyelikte geçen yılı ve aktif olup olmadığı gibi bilgilerini tutar.
2. Üye bazlı kitap takip bilgilerini izleme: Herhangi bir üyenin toplam kaç kitap okuduğu, en son hangi kitabı okuduğu ve hangi tarihte okuduğu bilgileri Book Tracking objecti ile kurulan ilişki sayesinde bu objede saklanır.

**İlgili Nesneler (Related Objects)**

1. Book\_Tracking\_\_c Object (Master-Detail ilişkisi ile): “Book\_Tracking\_\_c” objecti içindeki “Borrowed\_By\_\_c” fieldı Member\_\_c objecti ile Master\_detail ilişkisi kurar. Burada Member objecti Parent rolündedir. Bu ilişki sayesinde kütüphaneden ödünç alınan kitabın hangi üye tarafından alındığı bilgisi sisteme işlenmiş olur.

1. User Object (Lookup ilişkisi ile): "CreatedBy" alanı, "User" object ile Lookup ilişkisi kurar. Bu ilişki, Member kaydını oluşturan Salesforce kullanıcısını belirtir. Aynı şekilde "Last Modified By" alanı da Member kaydını son olarak kimin değiştirdiğini gösterir.

MEMBER\_\_C OBJECT FİELDS



Aşağıda Member objecti içinde bulunan fieldlar, bunların data typleri ve ne işe yaradıkları bulunmaktadır:

1. Address (Address\_\_c)

* Data Type: Text Area(255)
* Bu alan Member kaydının adres bilgilerini saklar

1. Borrowed Book Total (Borrowed\_Book\_Total\_\_c)

* Data Type: Roll-Up Summary (COUNT Book tracking)
* Bu alan bir Memberın toplam kaç kitap okuduğunu Book Tracking objecti ile kurulan ilişki sayesinde hesaplayarak bunu saklar.

1. Created By(CreatedById)

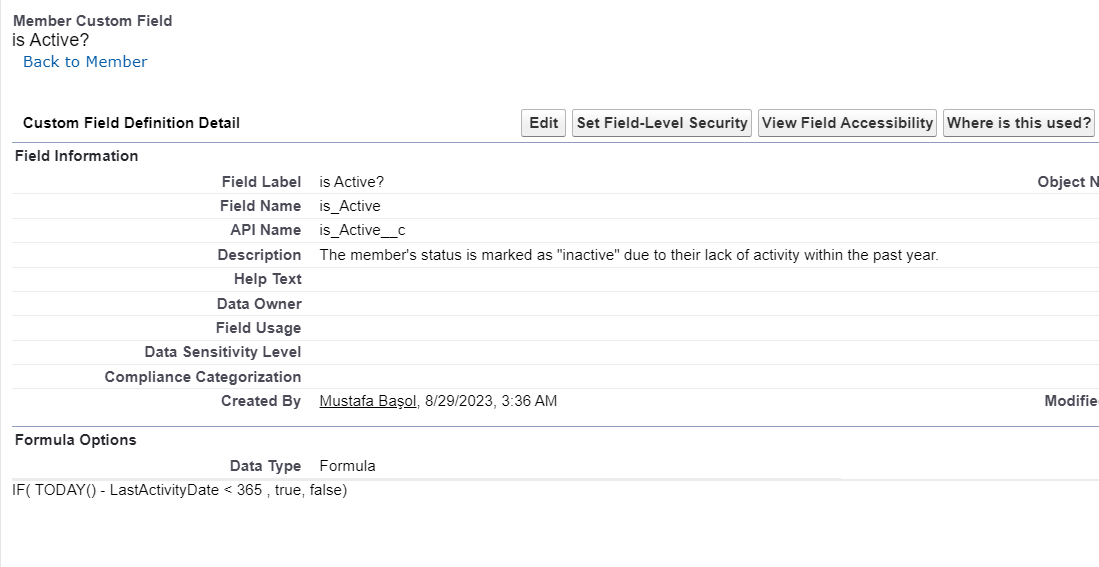
* Data Type: Lookup(User)
* Bu alan, "User" object ile ilişki kurarak Member kaydını oluşturan kullanıcıyı gösterir.

1. Email (Email\_\_c)

* Data Type: Email (Unique)
* Bu alan, Member kaydının email bilgisini saklar. Benzersiz olmalıdır. Member kaydı oluşturulurken doldurulması zorunlu olan bir alandır.

1. İs Active (is\_Active\_\_c)

* Data Type: Formula (Checkbox)
* Bu alan, günümüz tarihinden Memberın son aktif olduğu tarihi çıkartarak bu sürenin 1 yıldan az olması durumunda Memberı aktif olarak işaretlemektedir.



1. Last Book Read Date (Last\_Book\_Read\_Date\_\_c)

* Data Type: Date
* Bu alan bir Apex Trigger ile doldurulmaktadır. Member yeni bir kitap aldığında Apex Trigger tetiklenmekte ve Member’ın okuduğu en son kitabın tarihi bu alana otomatik olarak yazılmaktadır.

1. Last Modified By (LastModifiedById)

* Data Type: Lookup(User)
* Bu alan otomatik olarak oluşur ve Member kaydını son olarak kimin değiştirdiğini gösterir.

1. Last Read Book (Last\_read\_book\_\_c)

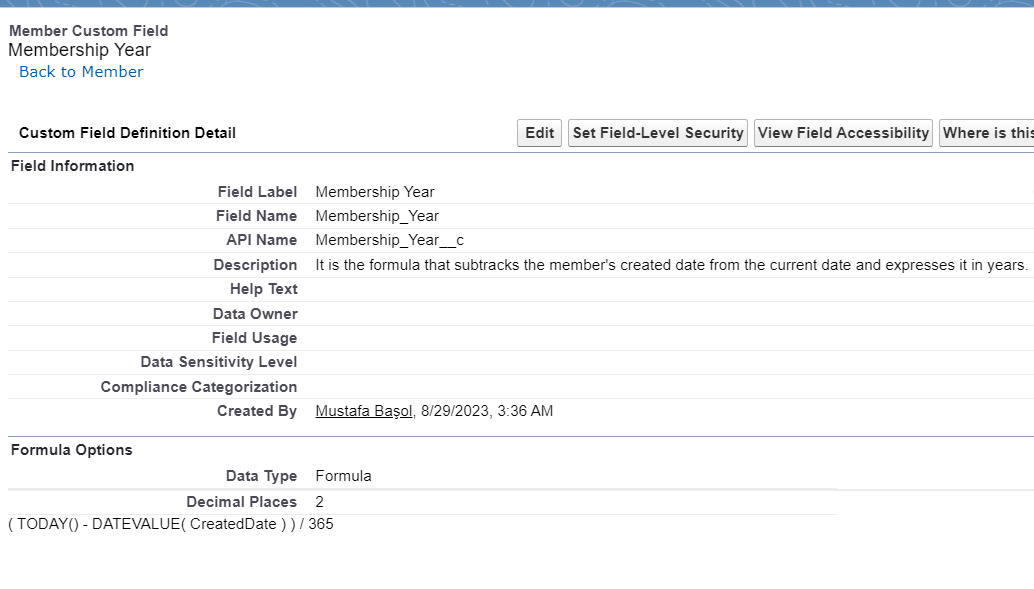
* Data Type: Text(255)
* Bu alan bir Record-Triggered Flow ile doldurulmaktadır. Member yeni bir kitap aldığında Flow tetiklenmekte ve Member’ın okuduğu en son kitap bu alana otomatik olarak yazılmaktadır.

1. Membership No (Name)

* Data Type: Auto Number
* Bu alan, her Member için otomatik olarak oluşturulan bir tanımlama numarasını temsil eder.

1. Membership Year (Membership\_Year\_\_c)

* Data Type: Formula (Number)
* Bu alan Member kaydının kaç yıldır üye olduğu bilgisini saklar.



1. Name (Name\_\_c)

* Data Type: Text (50)
* Bu alan Member kaydının isim bilgisini saklar

1. Owner (OwnerId)

* Data Type: Lookup(User,Group)
* Bu alan otomatik olarak oluşur. İlgili kaydın Salesforce içindeki sahibini gösterir.

1. Phone (Phone\_\_c)

* Data Type: Phone
* Bu alan Member kaydının telefon bilgisini saklar.

1. Surname (Surname\_\_c)

* Data Type: Text(50)
* Bu alan Member kaydının soyad bilgisini saklar.

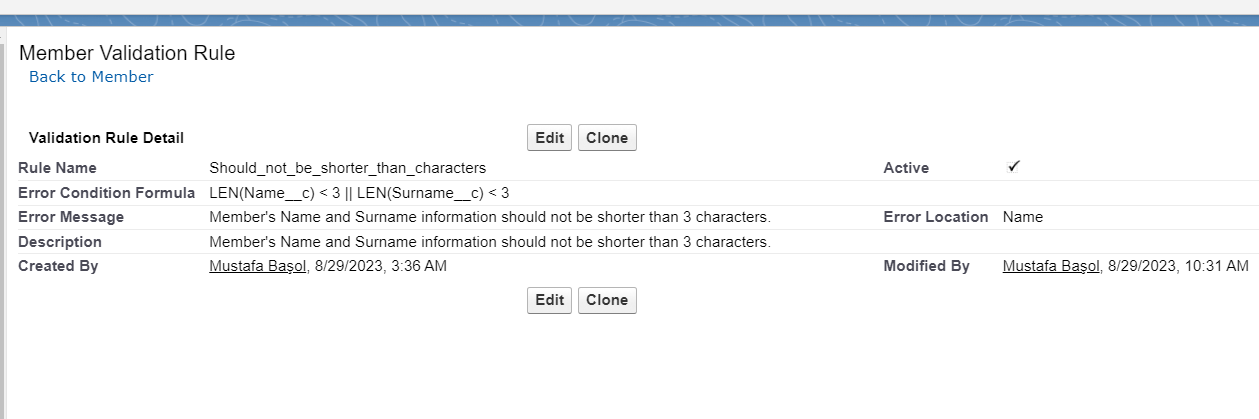
**MEMBER VALİDATİON RULE**

Validation Rules, "Member" object kullanımını daha güvenli ve tutarlı hale getirir. İlgili alanlara girilen verileri kontrol ederek yanlış veya eksik bilgilerin önüne geçilmesini sağlar.

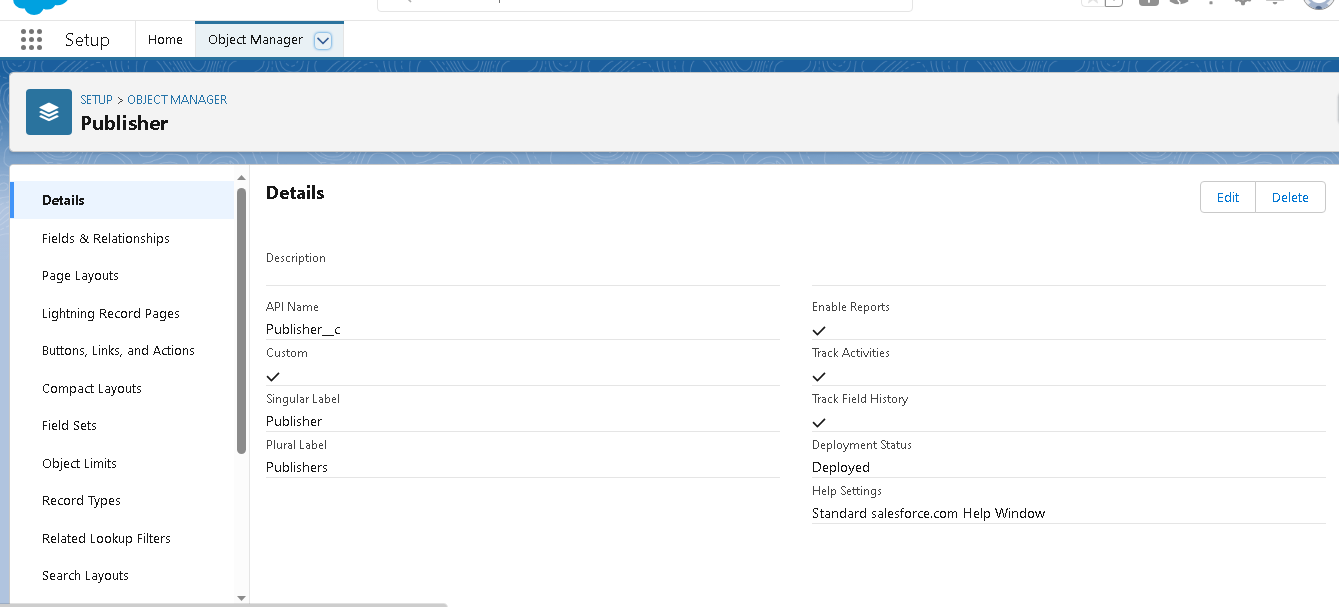
SHOULD\_NOT\_BE\_SHORTER\_THAN\_CHARACTERS VALİDATİON RULE

* Kural: Member's Name and Surname information should not be shorter than 3 characters.

Bu kural Member adının ve soy adının 3 karakterden daha kısa olmaması koşulunu kontrol eder.



**PUBLISHER\_\_C CUSTOM OBJECT**



Publisher\_\_c Custom Object, kütüphanede yer alan kitapların yayınevi bilgilerinin tutulduğu objecttir. Bu objenin amacı, kütüphanedeki kitapların hangi yayınevine ait olduğu süreçlerini izlemek ve otomatikleştirmek için bir çözüm sunmaktadır.

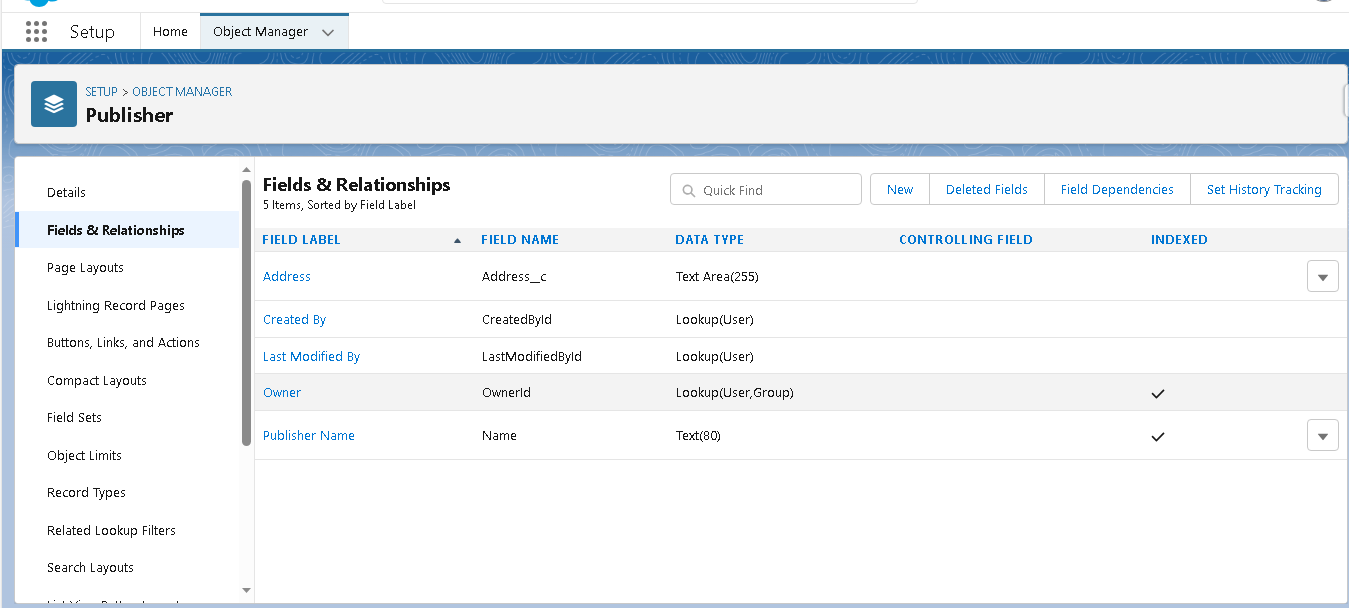
Publisher\_\_c Custom Object Amaçları ve İşlevleri:

1. Yayınevi bilgilerini saklama: Kütüphanede bulunan kitapların yayınevlerinin isim ve adresleri gibi bilgilerini tutar.
2. Yayınevi bazlı kitap takip bilgilerini izleme: Kütüphanedeki hangi kitabın hangi yayınevine ait olduğunu, bir yayınevinin kaç kitabı olduğu ve yayınevinin adres bilgileri bu objede saklanır.

**İlgili Nesneler (Related Objects)**

1. User Object (Lookup ilişkisi ile): "CreatedBy" alanı, "User" object ile Lookup ilişkisi kurar. Bu ilişki, Publisher kaydını oluşturan Salesforce kullanıcısını belirtir. Aynı şekilde "Last Modified By" alanı da Publisher kaydını son olarak kimin değiştirdiğini gösterir.

**PUBLISHER\_\_C OBJECT FIELDS**

****

Aşağıda Publisher objecti içinde bulunan fieldlar, bunların data typleri ve ne işe yaradıkları bulunmaktadır:

1. Address (Address\_\_c)

* Data Type: Text Area(255)
* Bu alan, Publisher objectnin adres bilgilerini gösterir.

1. Created By(CreatedById)

* Data Type: Lookup(User)
* Bu alan, "Publisher" object ile ilişki kurarak Publisher kaydını oluşturan kullanıcıyı gösterir.

1. Last Modified By (LastModifiedById)

* Data Type: Lookup(User)
* Bu alan otomatik olarak oluşur ve Publisher kaydını son olarak kimin değiştirdiğini gösterir.

1. Owner (OwnerId)

* Data Type: Lookup(User,Group)
* Bu alan otomatik olarak oluşur. İlgili kaydın Salesforce içindeki sahibini gösterir.

1. Publisher Name (Name)

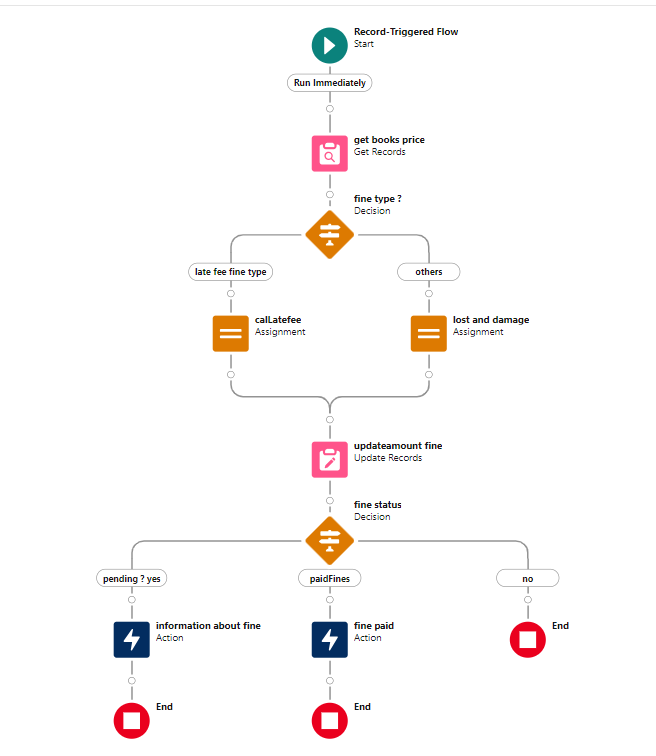
* Data Type: Text (80)
* Bu alan, Publisher objectnin ismini gösterir.

**FLOWLAR**

**1. CALCULATE AMOUNT FİNE AND INFORMATİON MAİL FLOW**

Description:This flow calculates the amount fine based on fine type and sends information email to members.

**Amaç:** Book\_Tracking\_\_c object içerisindeki "Amount\_of\_fine\_\_c" alanı otomatik dolduran bir flow oluşturulacak. "Fine\_be\_imposed\_\_c" alanı işaretlendiğinde tetiklenecek. "Fine\_Type\_\_c" alanına göre farklı hesaplamalar yapılacak ve "Amount\_of\_fine\_\_c" alanına değer atanacak. Sonrasında ilgili üyeye e-posta gönderilecek.

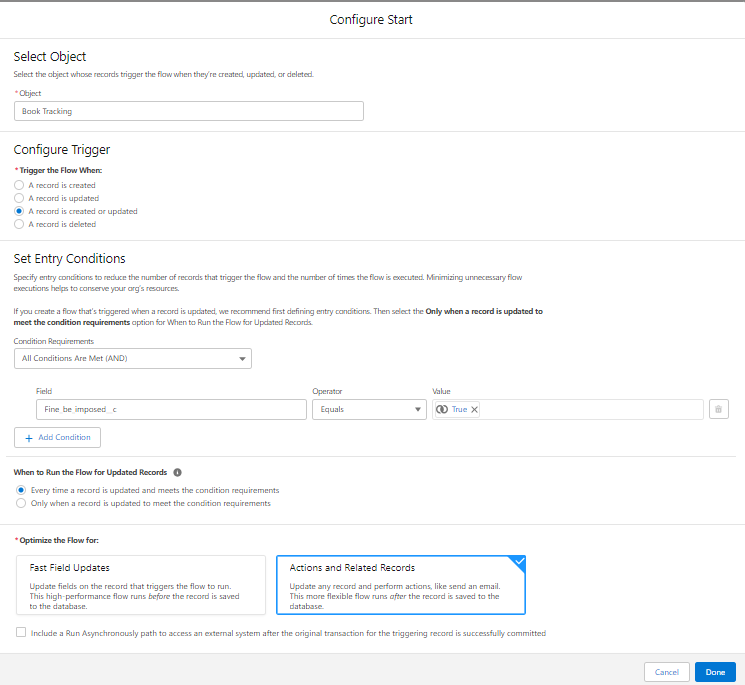


**1**

1-

Bu flow oluşturulurken **‘Record Triggered Flow’ template** kullanıldı.

**2**



2-

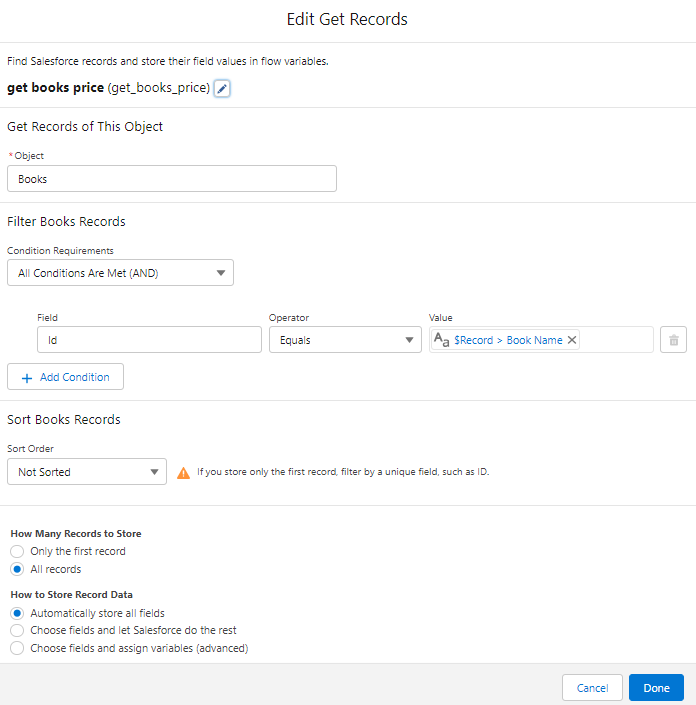
Object **Book\_Tracking\_\_c** seçildi.

Trigger the Flow When **A record is created or updated.**

Set Entry Conditions: **Fine\_be\_imposed\_\_c>equqls>true**

**Every time a record is updated and meets the condition requirements**.

### Optimize the Flow for: Actions and Related Records



**3**

3-

GET RECORDS ELEMENT

Object **Book\_c** seçildi.

### Filter Books Records (AND)

**Id>Equals> {!$Record.Book\_Name\_\_c}**

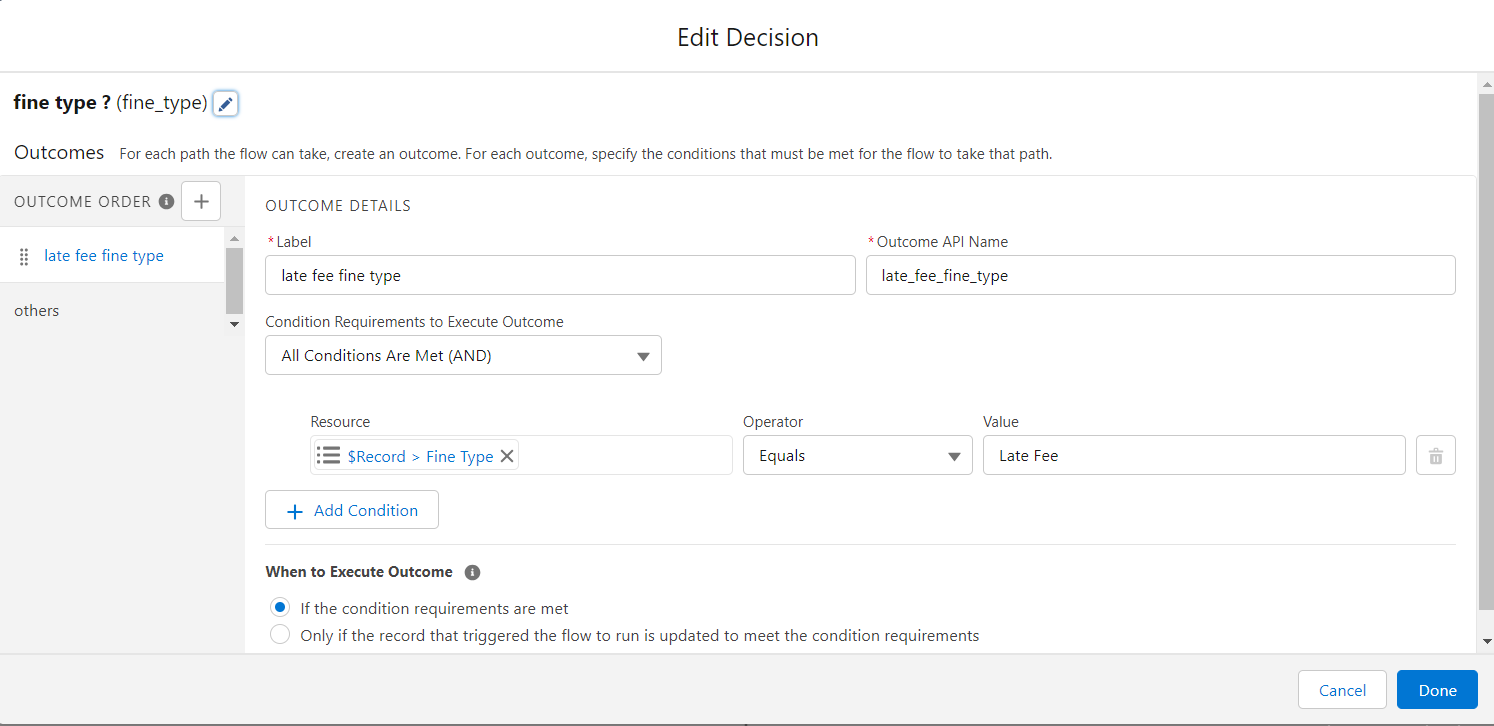
How Many Records to Store: **All Recods**

How to Store Record Data: **Automatically store all fields**

4-

DECİSİON ELEMENT

EĞER Book\_Tracking\_\_c object Fine\_Type\_\_c piclist alanında Late fee seçilirse ‘late fee fine type’ yönünde değilse ‘others’ yönünde devam edecek.

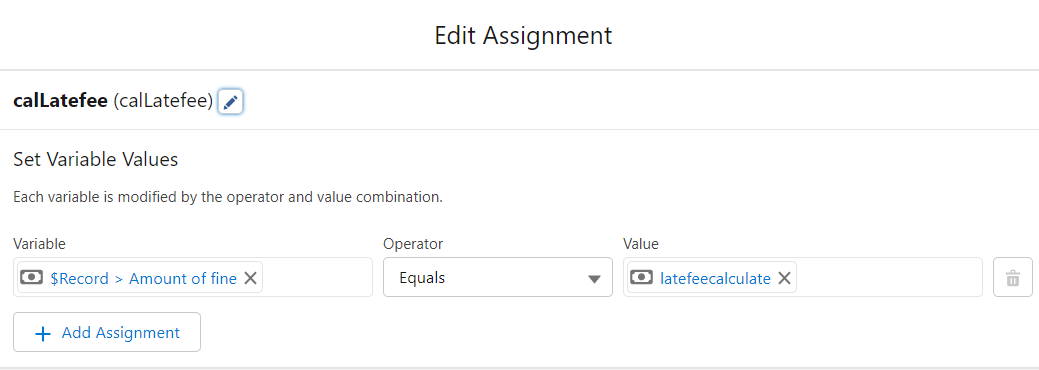


**4**

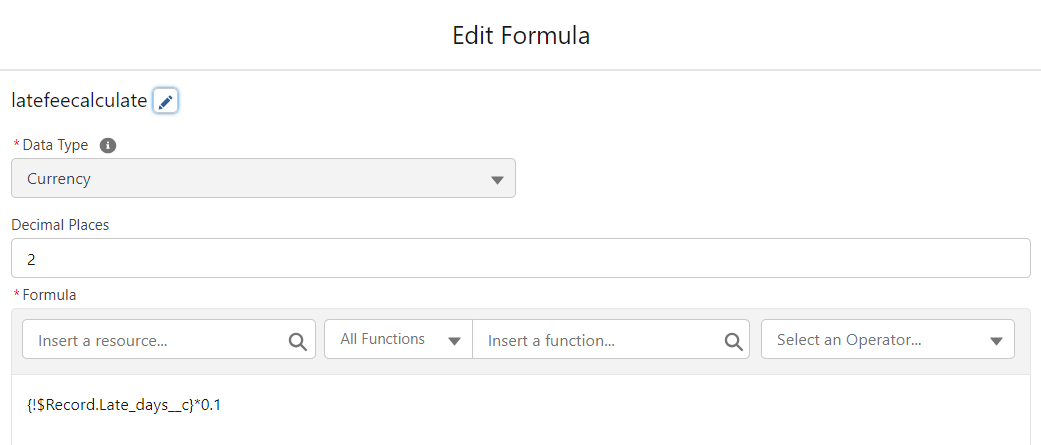
5.1- ASSIGNMENT ELEMENT

‘late fee fine type’ yönünde ise Book\_Tracking\_\_c object Amount of fine alanını Latefeecalculate (5.2) değeri atanacak.

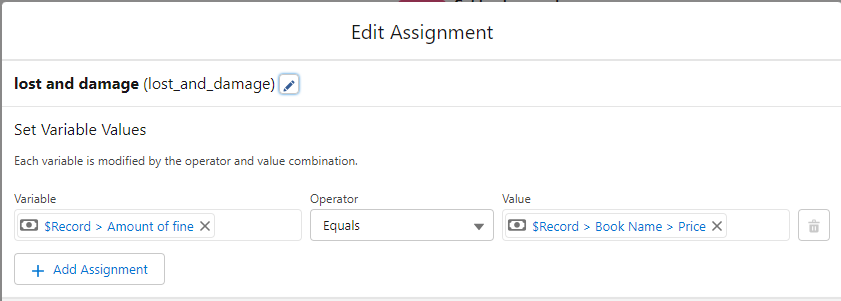
5.2- Latefeecalculate ise Formula (Currency) Resource ve değeri {!$Record.Late\_days\_\_c}\*0.1 dir. Yani Book\_Tracking\_\_c object içinde Late\_days\_\_c (geç kalan gün sayısı) ile 0.1 çarpımıdır.



**5.1**



**5.2**



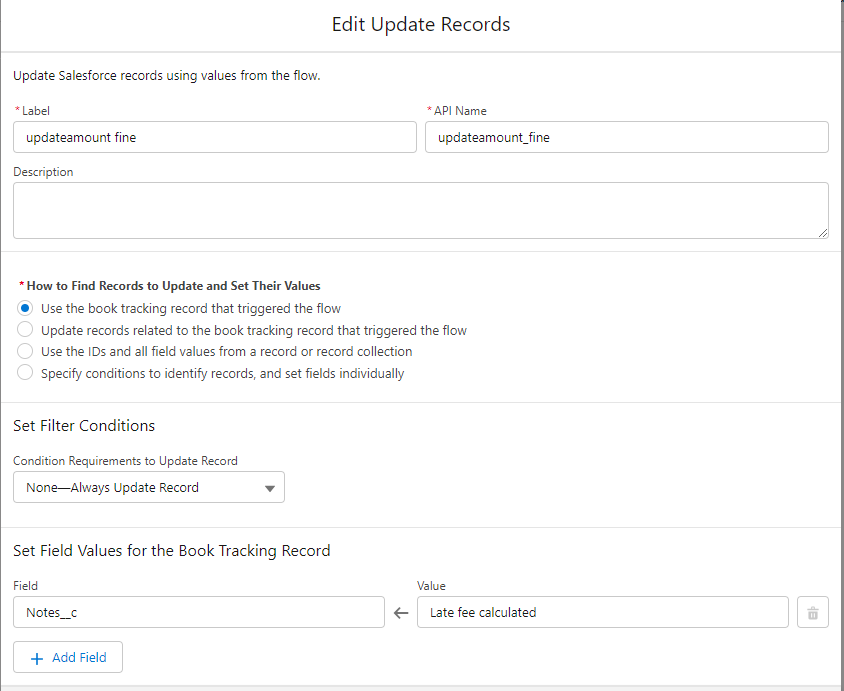
6- ASSIGNMENT ELEMENT

‘others’ yönünde ise yani Fine\_Type\_\_c piclist değeri Lost Book veya Damage Book seçilmiş ise Book\_Tracking\_\_c objectin Amount of fine alanını Book\_Tracking\_\_c objectin ilişkili olduğu Book\_\_c object içerisindeki Price\_\_c değeri atanacak.

**6**

7- UPDATE RECORDS ELEMENT

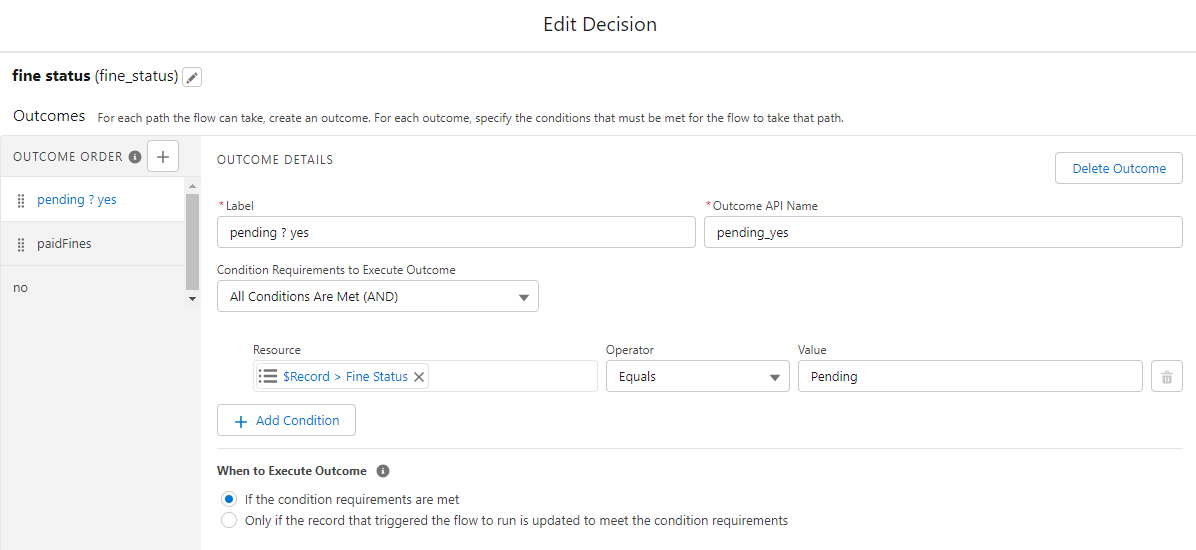
‘late fee fine type’ yönünde ve ‘others’ yönünde Assignment element atanan değerler UpdateRecords element ile Book\_Tracking\_\_c object Amount of fine alanına ilgili değerler atanmış oluyor.



8- DECİSİON ELEMENT

Update edilen kayıtlardan sonra ceza durumlarına göre otomatik mail göndermek için kullanılacak. Book\_Tracking object de Fine\_status\_\_c Piclist alanı Pending veya Paid veya diğerleri seçilmişse ona göre farklı yollardan oluşturulan Email Template ler ilgili üyeye gönderim yapılacak.

**7**



**8**

9- ACTİON ELEMENT

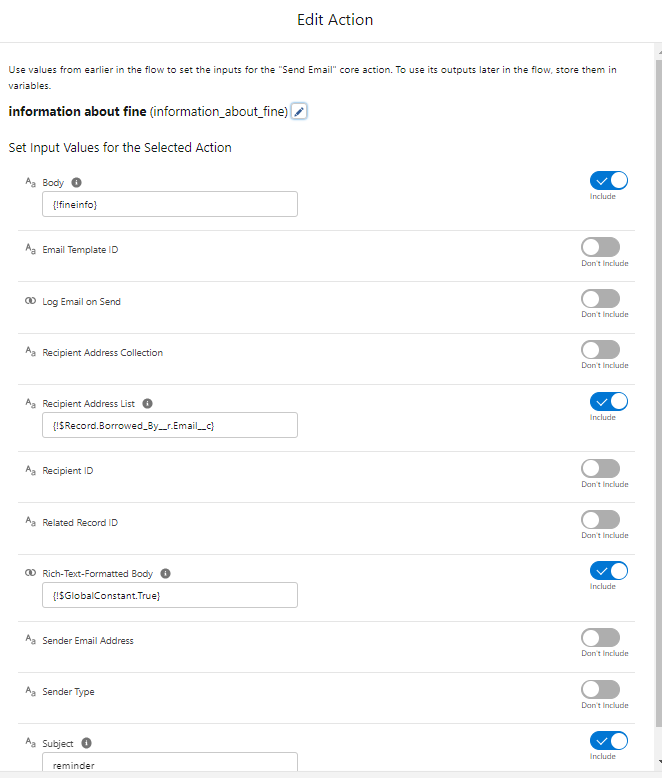
Action element içerisinde Category “Email” , action “Send Email” seçiyoruz ve 9.1 ekranı geliyor.

Body: {!fineinfo} (9.2 Text Template Resource oluşturuyoruz.)

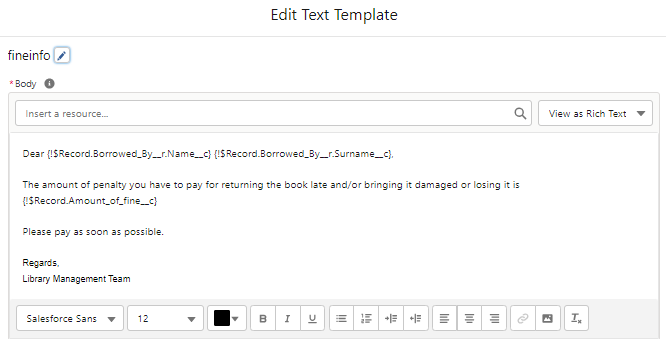
Recipient Address List: {!$Record.Borrowed\_By\_\_r.Email\_\_c}

Rich-Text-Formatted Body: True

Subject: Reminder (your suggested)



**9.1**



**9.2**

10- ACTİON ELEMENT

Action element içerisinde Category “Email” , action “Send Email” seçiyoruz ve 9.1 ekranı geliyor.

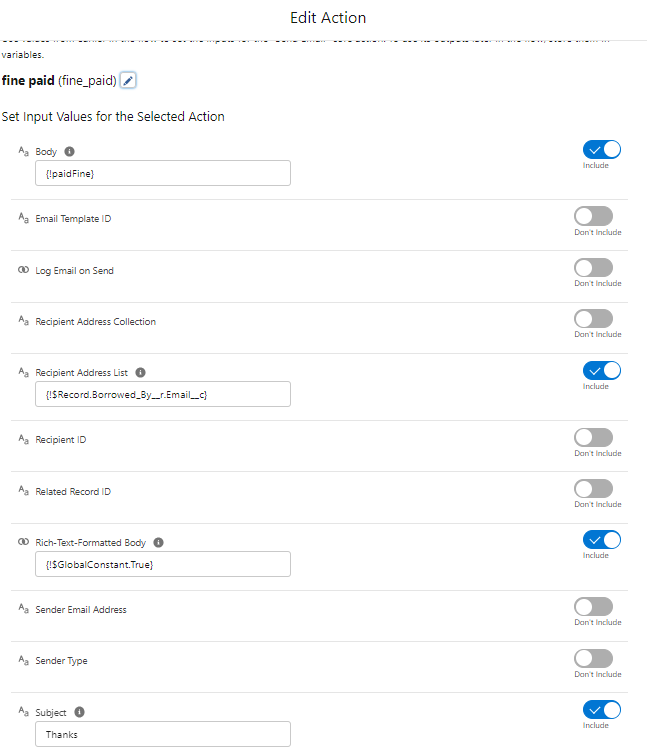
Body: {!paidFine} (10.2 Text Template Resource oluşturuyoruz.)

Recipient Address List: {!$Record.Borrowed\_By\_\_r.Email\_\_c}

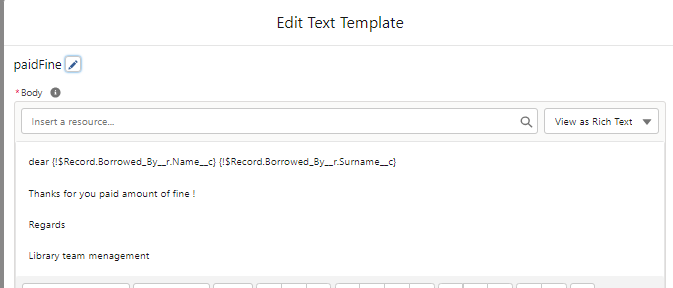
Rich-Text-Formatted Body: True

Subject: Thanks (your suggested)

**10.1**



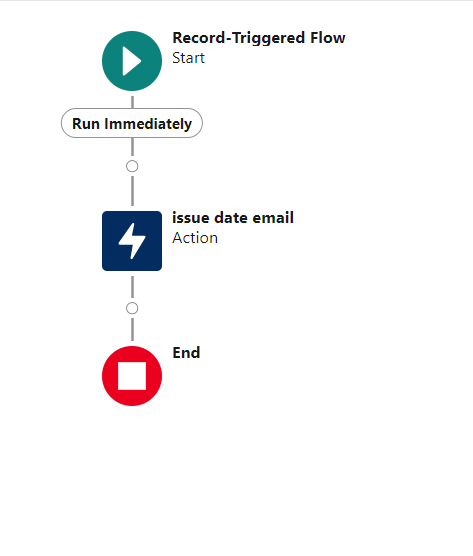
**10.2**



1. **DUE DATE INFORMATİON EMAİL FLOW**

**Description:** This flow is used to remind the member of due date information. The flow works on the issue date and sends an email to members.

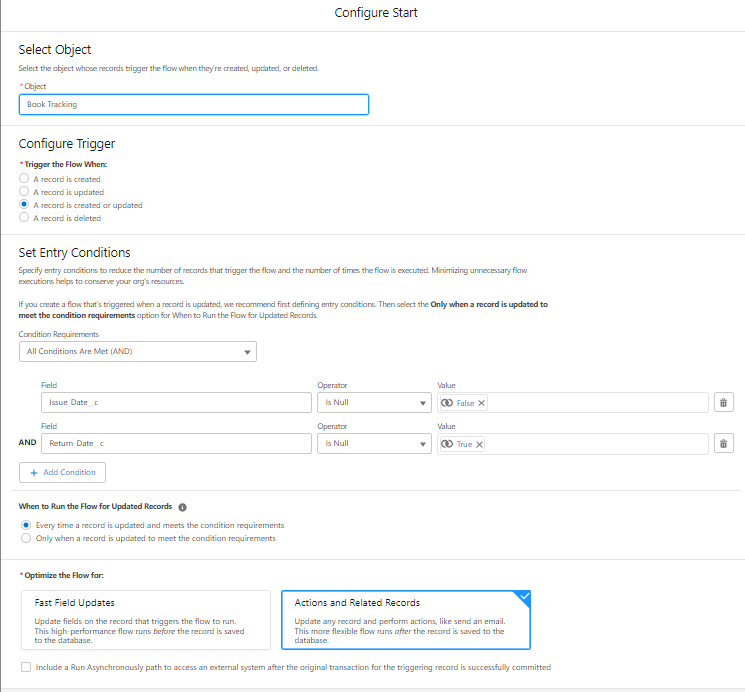
**Amaç:** Book\_Tracking\_\_c object içerisindeki "Amount\_of\_fine\_\_c" alanı otomatik dolduran bir flow oluşturulacak. "Fine\_be\_imposed\_\_c" alanı işaretlendiğinde tetiklenecek. "Fine\_Type\_\_c" alanına göre farklı hesaplamalar yapılacak ve "Amount\_of\_fine\_\_c" alanına değer atanacak. Sonrasında ilgili üyeye e-posta gönderilecek.



1-

Bu flow oluşturulurken **‘Record Triggered Flow’ template** kullanıldı.

**1**



**2**

2-

Object **Book\_Tracking\_\_c** seçildi.

Trigger the Flow When **A record is created or updated.**

Set Entry Conditions: **(AND)**

**Issue\_Date\_\_c>IsNull>false**

**Return\_Date\_\_c>IsNull>true**

**Every time a record is updated and meets the condition requirements**.

### Optimize the Flow for: Actions and Related Records

3- ACTİON ELEMENT

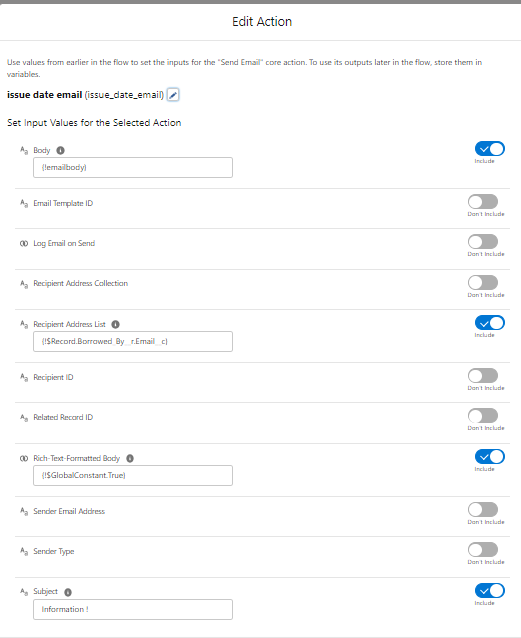
Action element içerisinde Category “Email” , action “Send Email” seçiyoruz ve 3.1 ekranı geliyor.

Body: {!emailbody} (3.2 Text Template Resource oluşturuyoruz.)

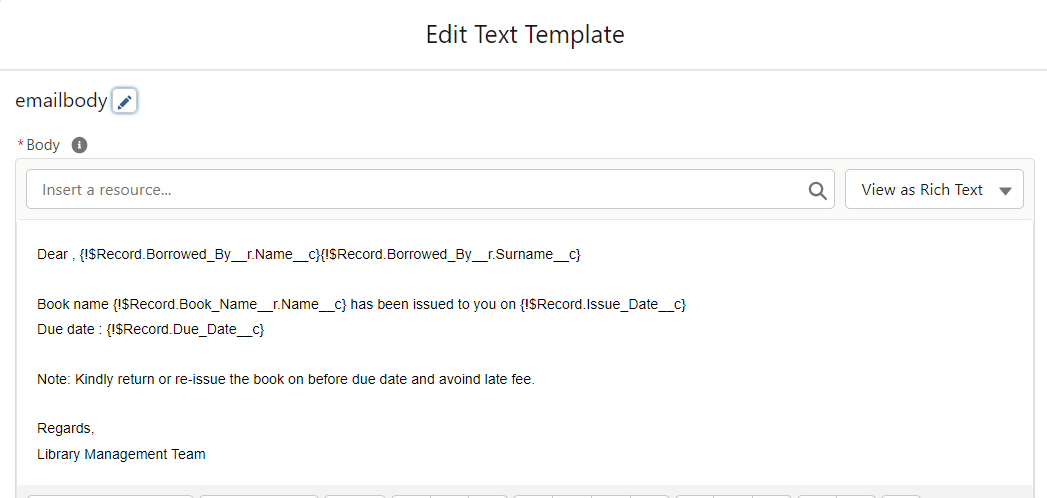
Recipient Address List: {!$Record.Borrowed\_By\_\_r.Email\_\_c}

Rich-Text-Formatted Body: True

Subject: İnformation ! (your suggested)



**3.1**

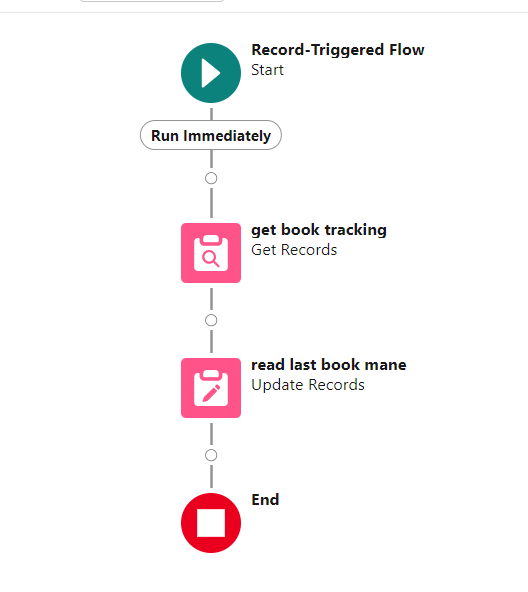


**3.2**

1. **DUE DATE INFORMATİON EMAİL FLOW**

**Description:** This flow is used to remind the member of due date information. The flow works on the issue date and sends an email to members.

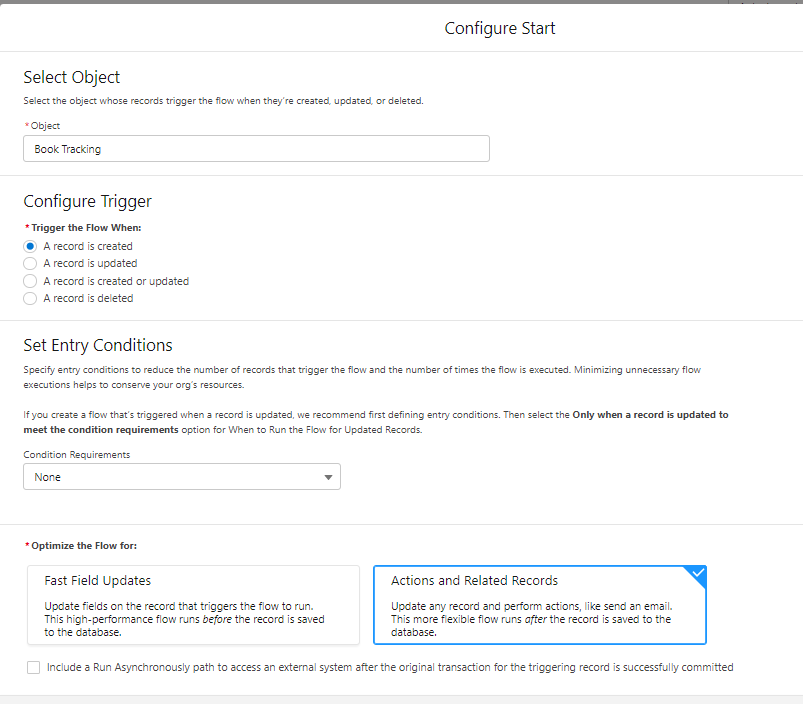
**Amaç:** Book\_Tracking\_\_c object içerisindeki "Amount\_of\_fine\_\_c" alanı otomatik dolduran bir flow oluşturulacak. "Fine\_be\_imposed\_\_c" alanı işaretlendiğinde tetiklenecek. "Fine\_Type\_\_c" alanına göre farklı hesaplamalar yapılacak ve "Amount\_of\_fine\_\_c" alanına değer atanacak. Sonrasında ilgili üyeye e-posta gönderilecek.



1-

Bu flow oluşturulurken **‘Record Triggered Flow’ template** kullanıldı.

**1**



2-

Object **Book\_Tracking\_\_c** seçildi.

Trigger the Flow When **A record is created.**

Set Entry Conditions: **None**.

### Optimize the Flow for: Actions and Related Records

**2**

3-GET RECORDS ELEMENT

Object **Book\_Tracking\_\_c** seçildi.

Filter Books Records (AND)

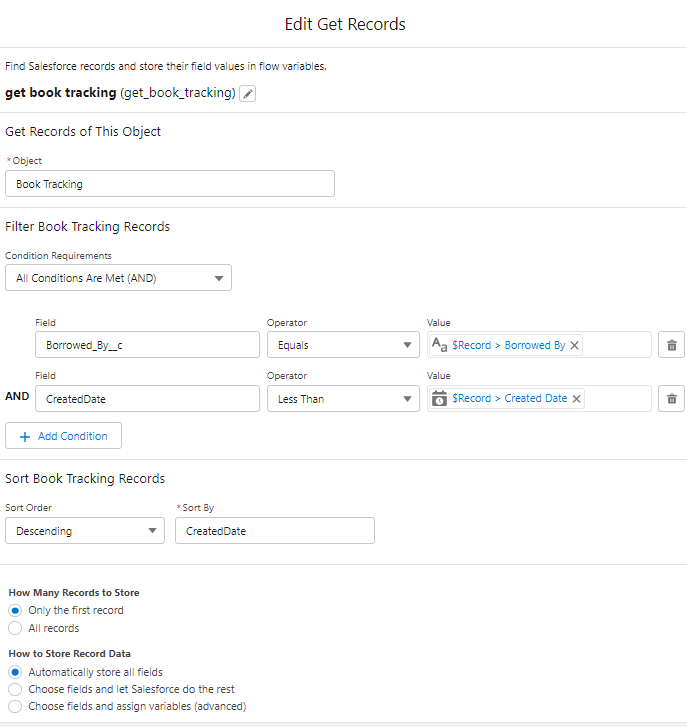
**Barrowed\_By\_\_c>Equals> {!$Record.Barrowed\_By\_\_c }**

**CreatedDate>LessThan>{!$Record. CreatedDate }**

**Sort by: CreatedDate (Descending)**

How Many Records to Store: **Only the first record.**

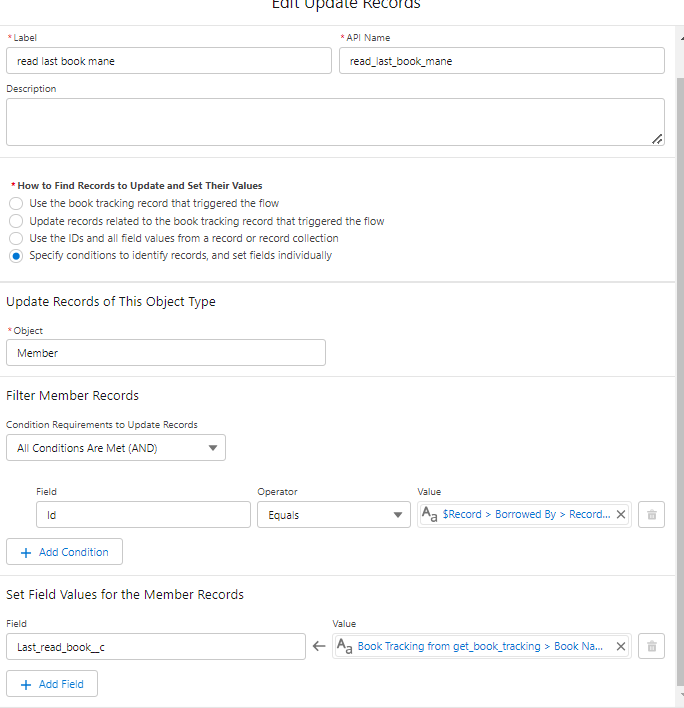
How to Store Record Data: **Automatically store all fields**



**3**

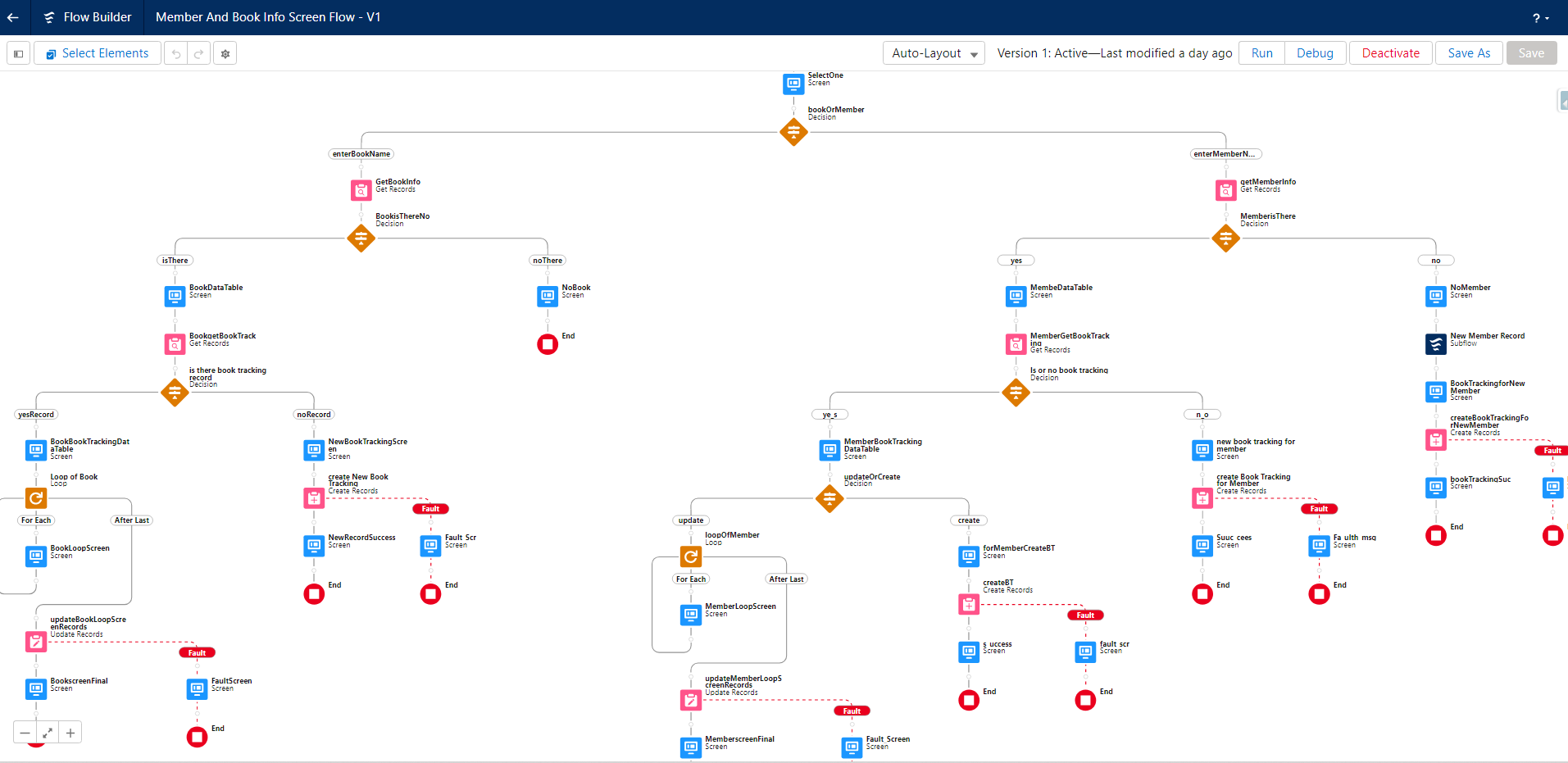
4- UPDATE RECORDS ELEMENT

Get Record element de elde edilen kayıtlar ile üye (Member\_\_c)RecordId sini eşitleyecek ve bookName değeriniMember\_\_c object içindeki Last\_Read\_Book\_c alanına değer olarak atamasını yapacak.



**4**

1. MEMBER AND BOOK INFO SCREEN FLOW



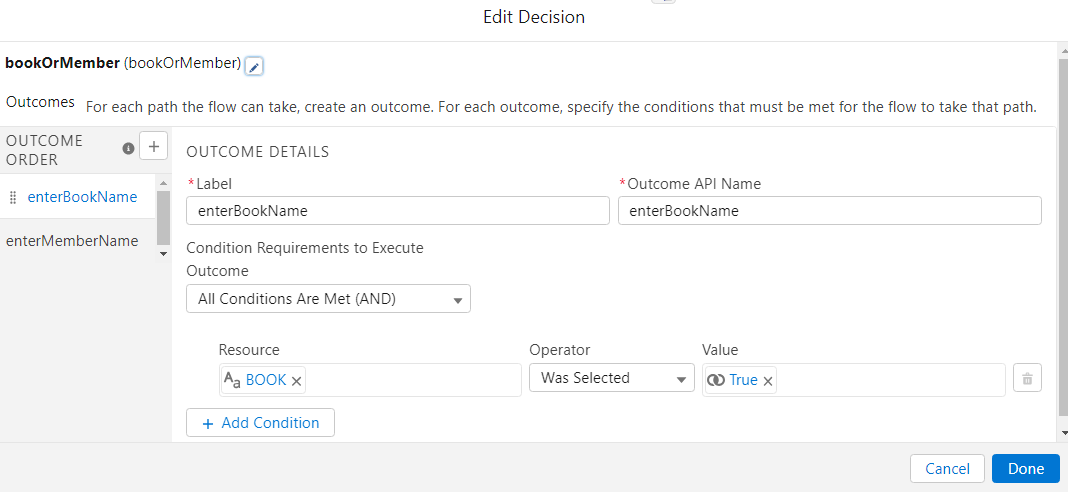
'Kitap' ve 'üye'yi arama seçeneklerinin bulunduğu bir arama akışımız var. Aramayı yaptıktan sonra seçilen seçeneğe göre kitap takip kayıtlarının güncellemelerini veya oluşturulmasını gerçekleştiriyoruz. Bu süreç, kitapların ve üyelerin kayıtlarını etkin bir şekilde yönetmemizi ve saklamamızı sağlar. Bağlama bağlı olarak ya mevcut takip kayıtlarını güncelliyoruz ya da yenilerini oluşturarak doğru ve güncel bilgiler sağlıyoruz. Bu akış, kitaplarla ve üyelerle ilgili durumu ve etkileşimleri takip etme yeteneğimizi geliştirerek daha akıcı ve düzenli bir sisteme katkıda bulunur.

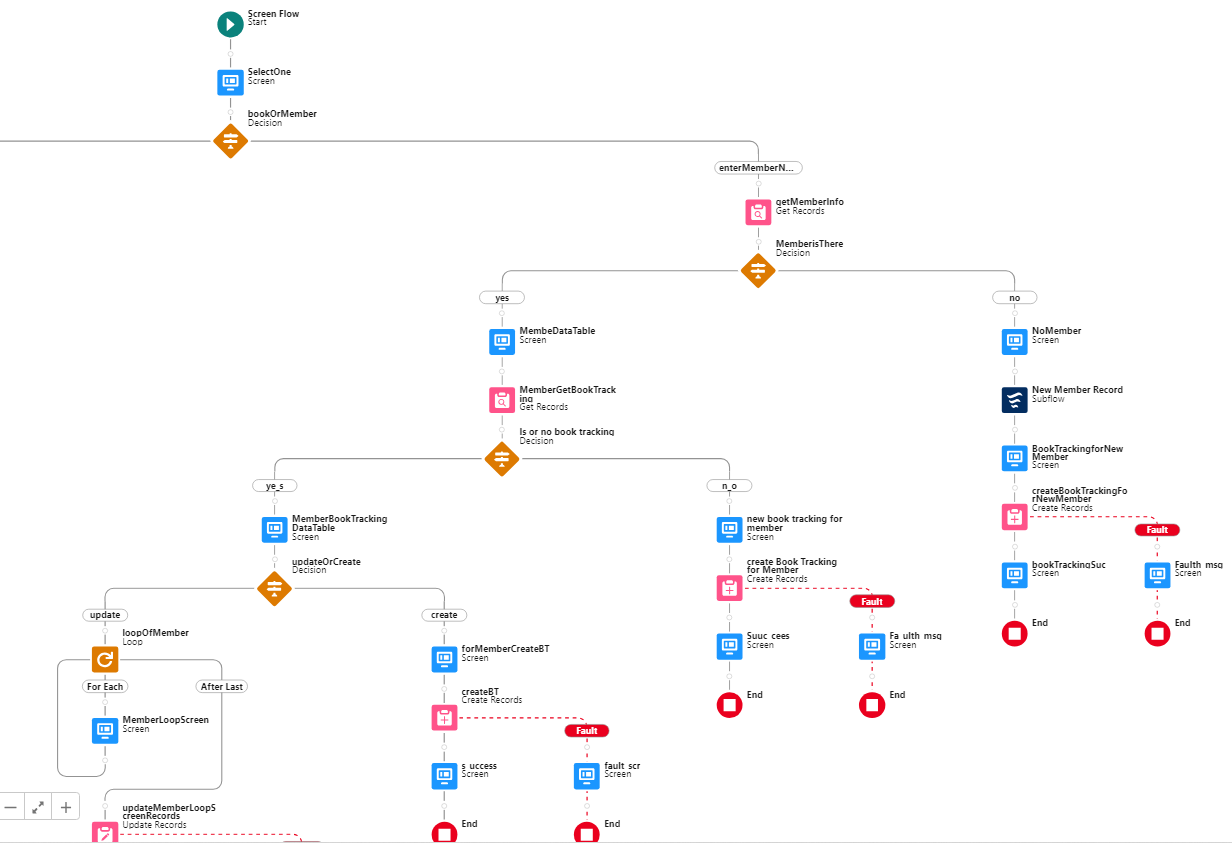
Bu flowdaki elementler ve amaçları şöyledir:

Flowdaki ilk element bir screen elementi. Burada aramak istediğimiz objeyi seçiyoruz. Kitap arıyorsak kitabın ismini, eğer member arıyorsak memberın email adresini yazmamız gerekiyor.

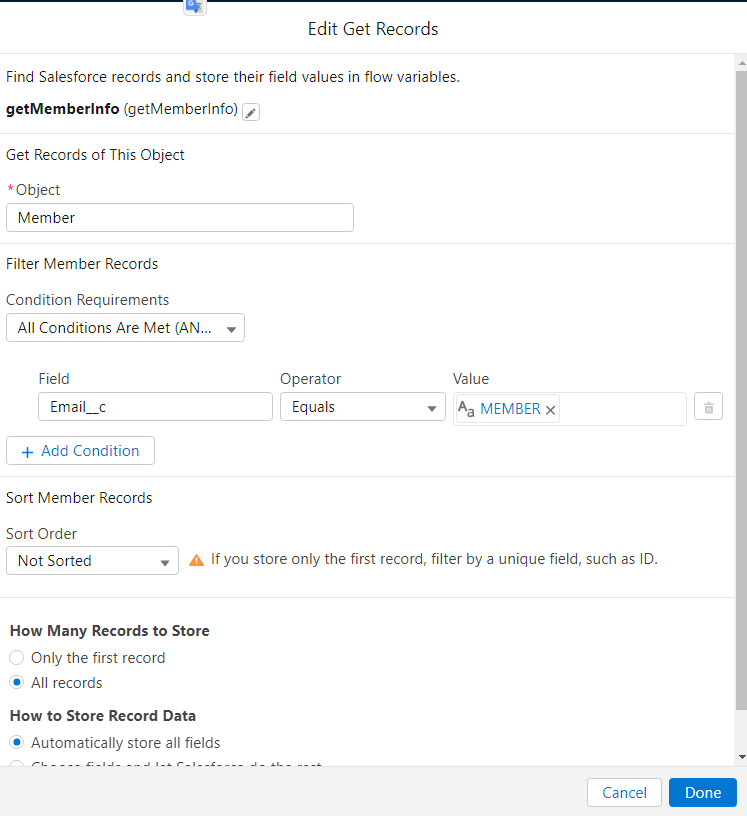


Bundan sonra bir desicion elementi ile Flow hangi yoldan ilerleyeceğine karar veriyor. Member aramayı tercih ettiysek Flow sağ yoldan ilerleyecektir.

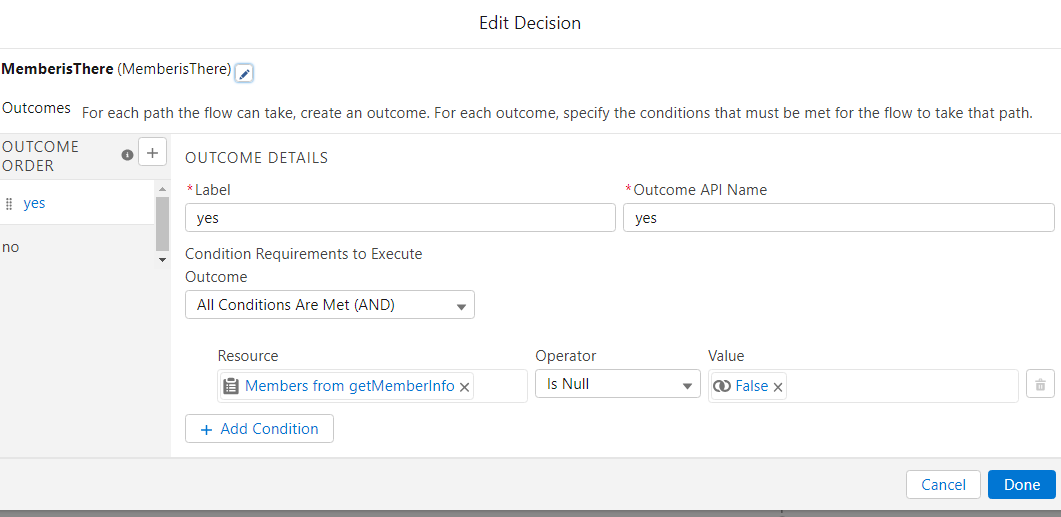


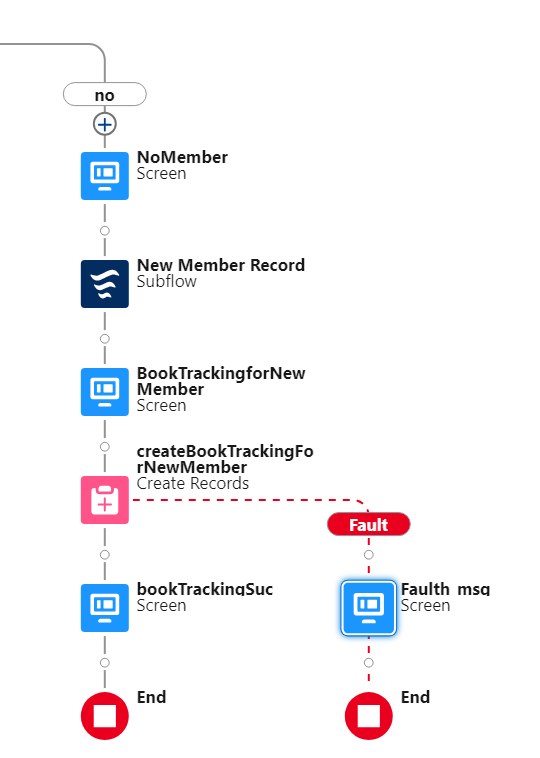


Member aramayı tercih ettiğimizde yazdığımız email, sistemde kayıtlı emailler içinde aranıyor ve bu emaile sahip bir member varsa Get Record elementi bize bu member kaydını getiriyor.

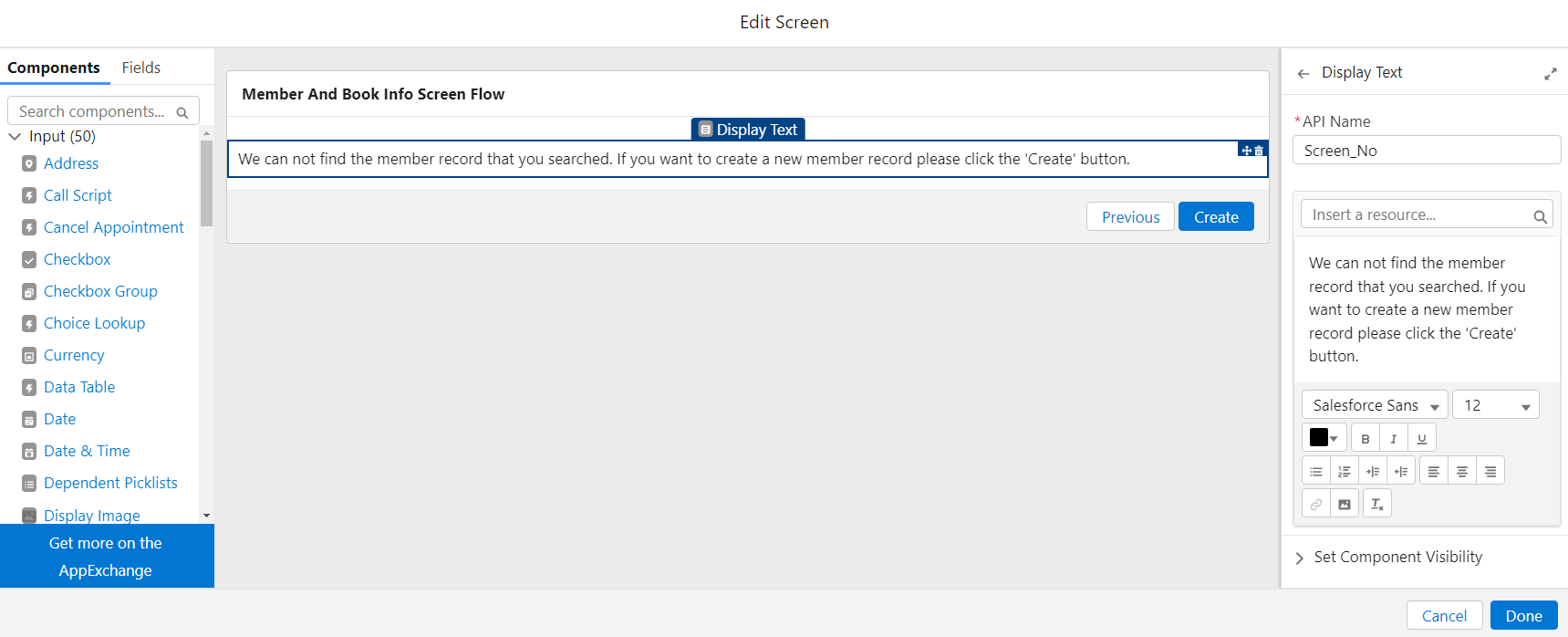


Burada iki durum karşımıza çıkıyor. Sistemde bu emaile sahip bir member varsa Get Record elementi çalışmış ve bu member bilgilerini depolamıştır, eğer böyle bir member yoksa Get Record elementinin içi boştur. Aşağıdaki desicion elementi ile bu iki senaryoya göre karşımıza iki akış yönü çıkarmaktadır.

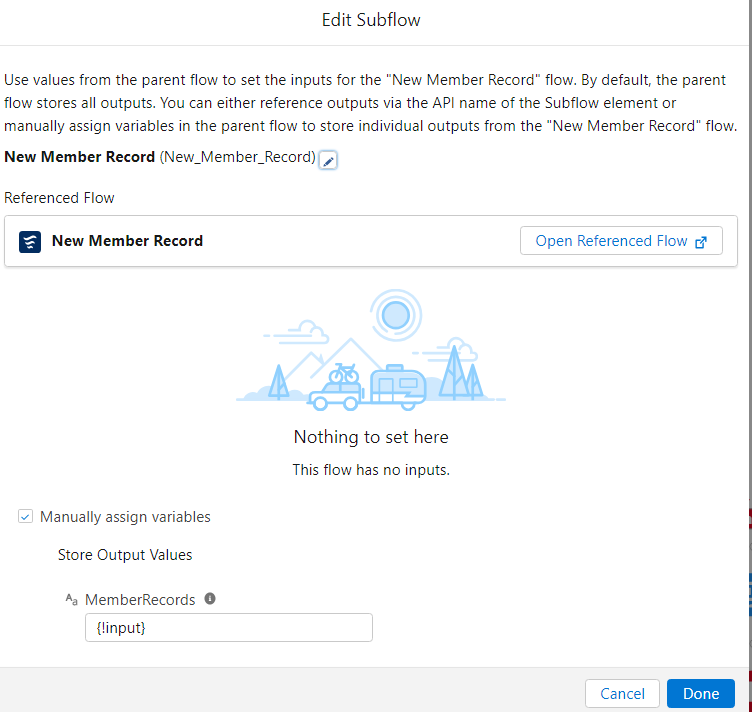




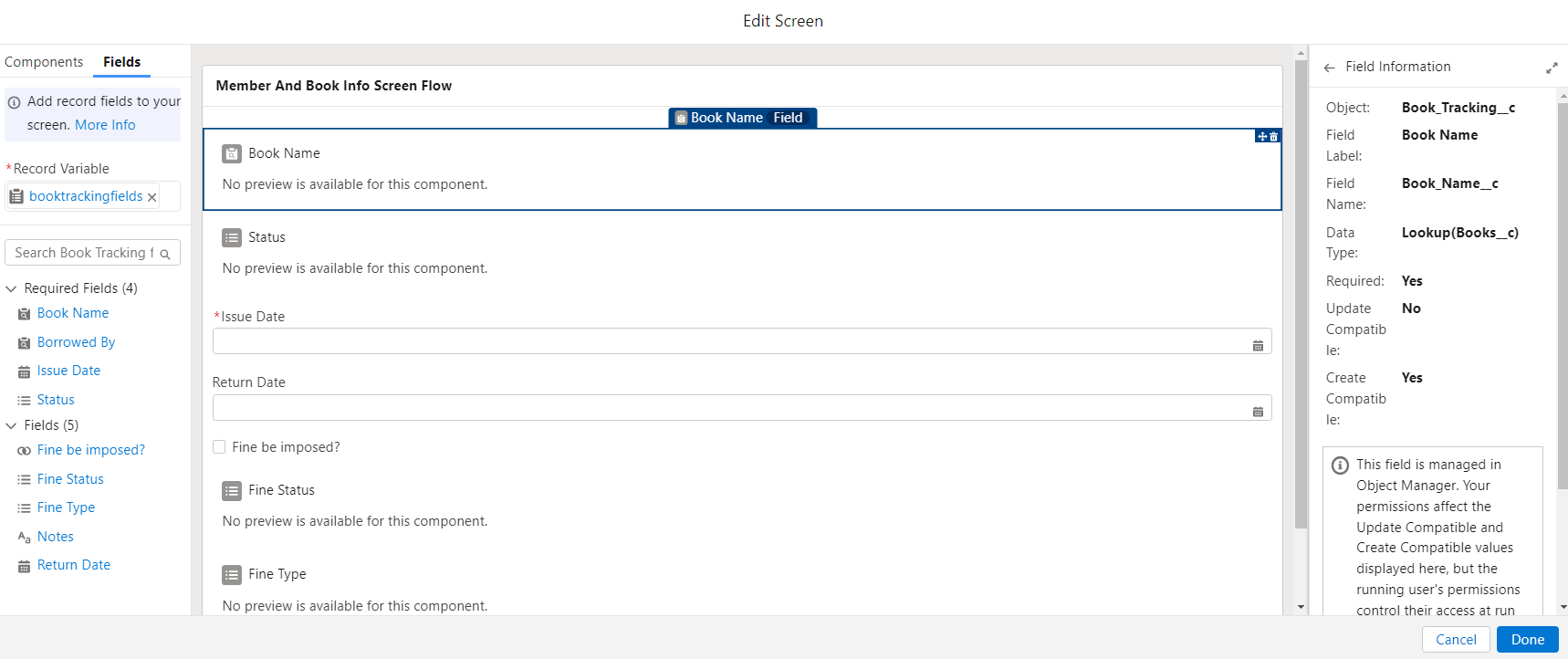
Bu yönlerden önce ilgili member kaydının olmadığı yönü inceleyelim. Bu durumda Flow bize bu emaile sahip bir member olmadığını ve yeni bir kayıt yapmak istiyorsak “Create” butonuna basmamız gerektiğini söyler.



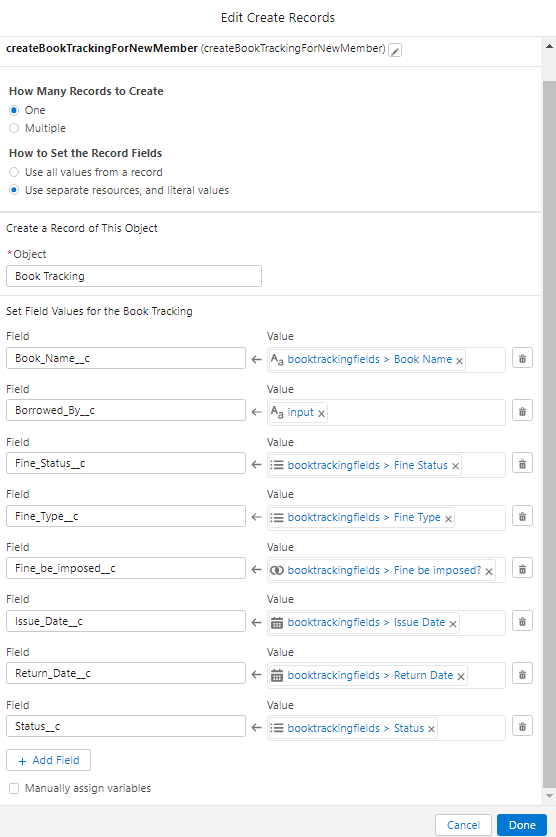
Burada karşımıza “New Member Record” Subflowu çıkar. Bu flowda member kaydı oluşturabilmek için ilgili alanları doldurmamız gerekir. Subflow member kaydını yaptıktan sonra yapılan member kaydının Id’sini “MemberRecords” adlı bir variable’da saklar. Bu Id bilgisini “Member And Book Info Screen Flow” flowundaki “Input” variable’ına aktarırız.



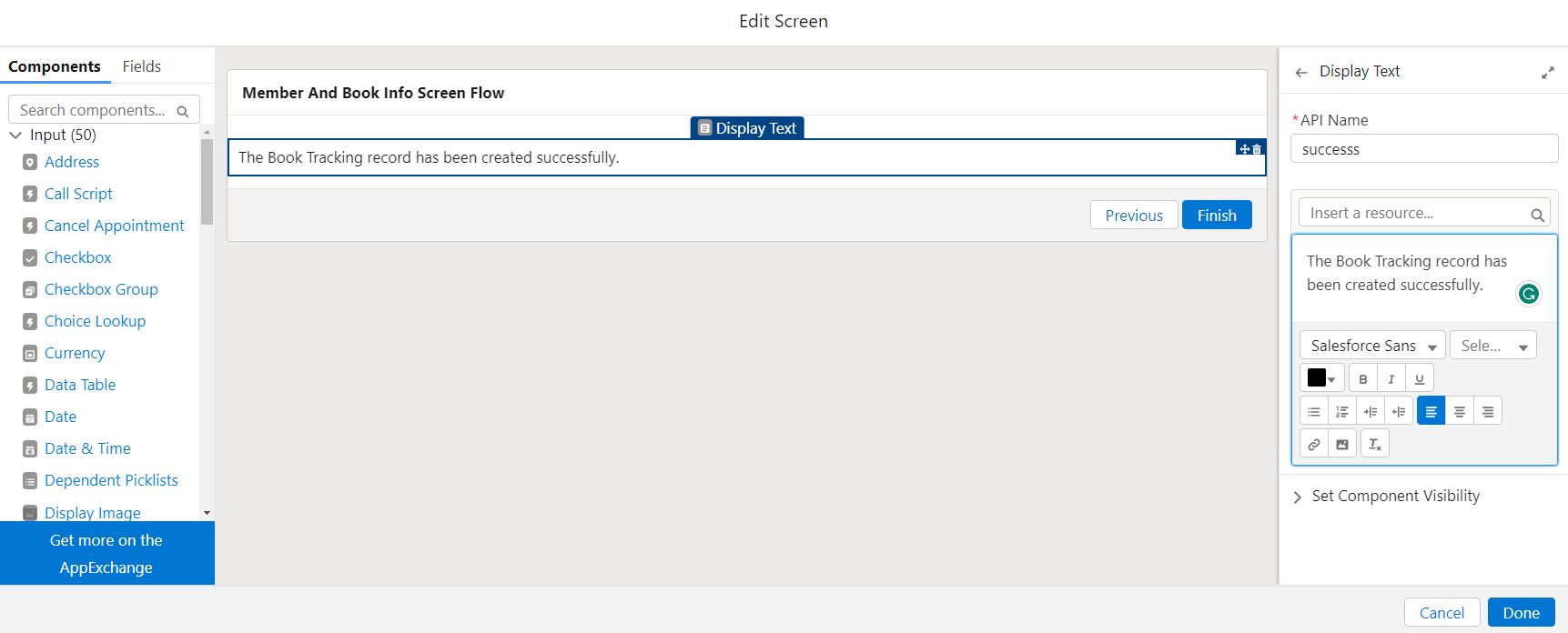
Member kaydını oluşturduktan sonra oluşturduğumuz bu member için Book tracking kaydı oluşturabilmek için karşımıza ilgili alanları doldurabileceğimiz bir Screen Componenti çıkar.



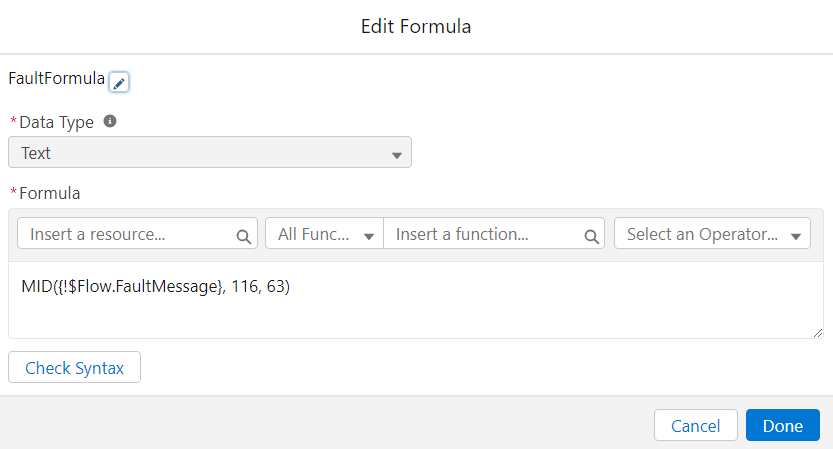
Book Tracking bilgilerini doldurduktan sonra bu bilgileri ve oluşturduğunuz Member kaydının Id’sini “Input” variable’ından alarak bunları Create Record Componentinde Book Tracking objesinin fieldları ile teker teker eşleştiririz.

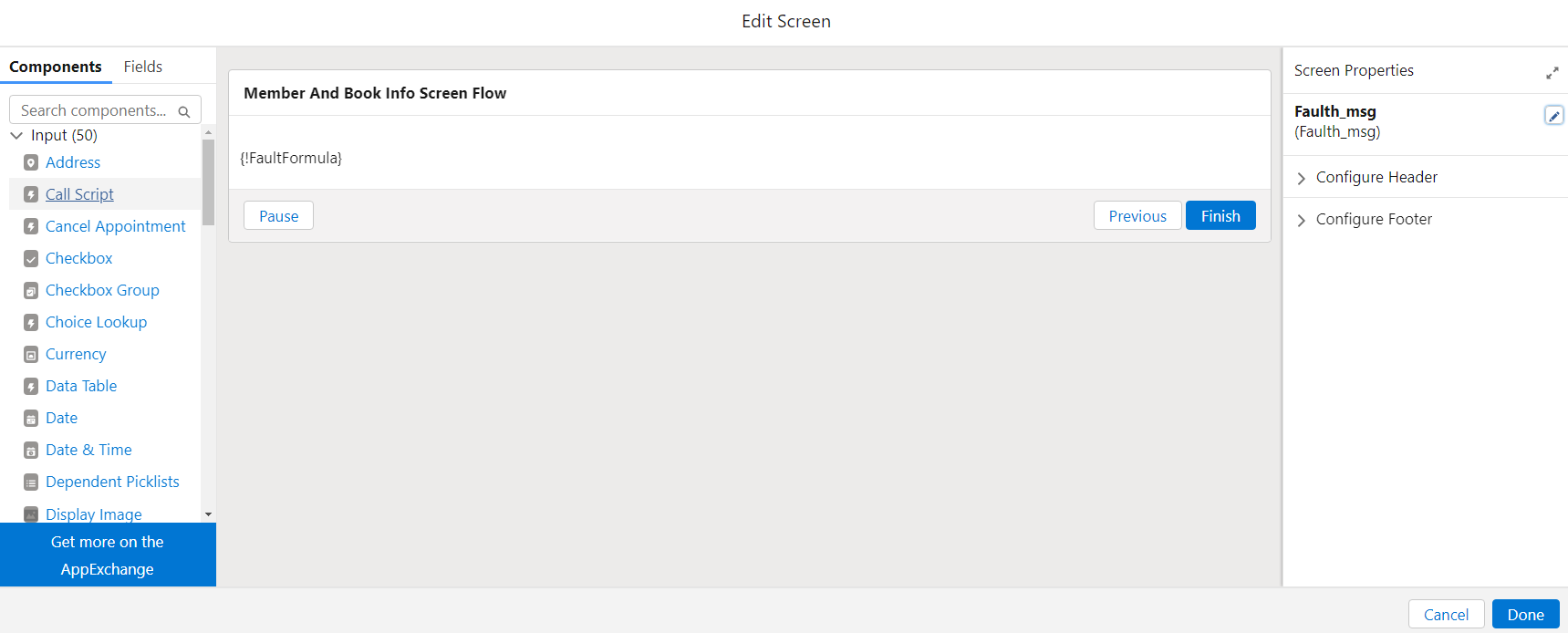


Eğer kayıt işlemi başarı ile gerçekleştiyse karşımıza aşağıdaki başarı mesajını içeren ekran çıkacaktır.



Eğer kayıt işlemi başarısız olmuş ise alınan hata mesajı, “Fault Path” eklenerek burada kullanılan “Screen” elementinde “FaultFormula” adlı bir formül yardımıyla ekrana yazdırılır. Bu formülde hata mesajının sadece ilgili kısmının ekrana yazdırılması amaçlanmıştır. Hata mesajını gören kullanıcı Flow’dan ayrılmadan bir önceki ekrana geri dönerek hatayı düzeltip Flow’u başarı ile sonlandırabilir.

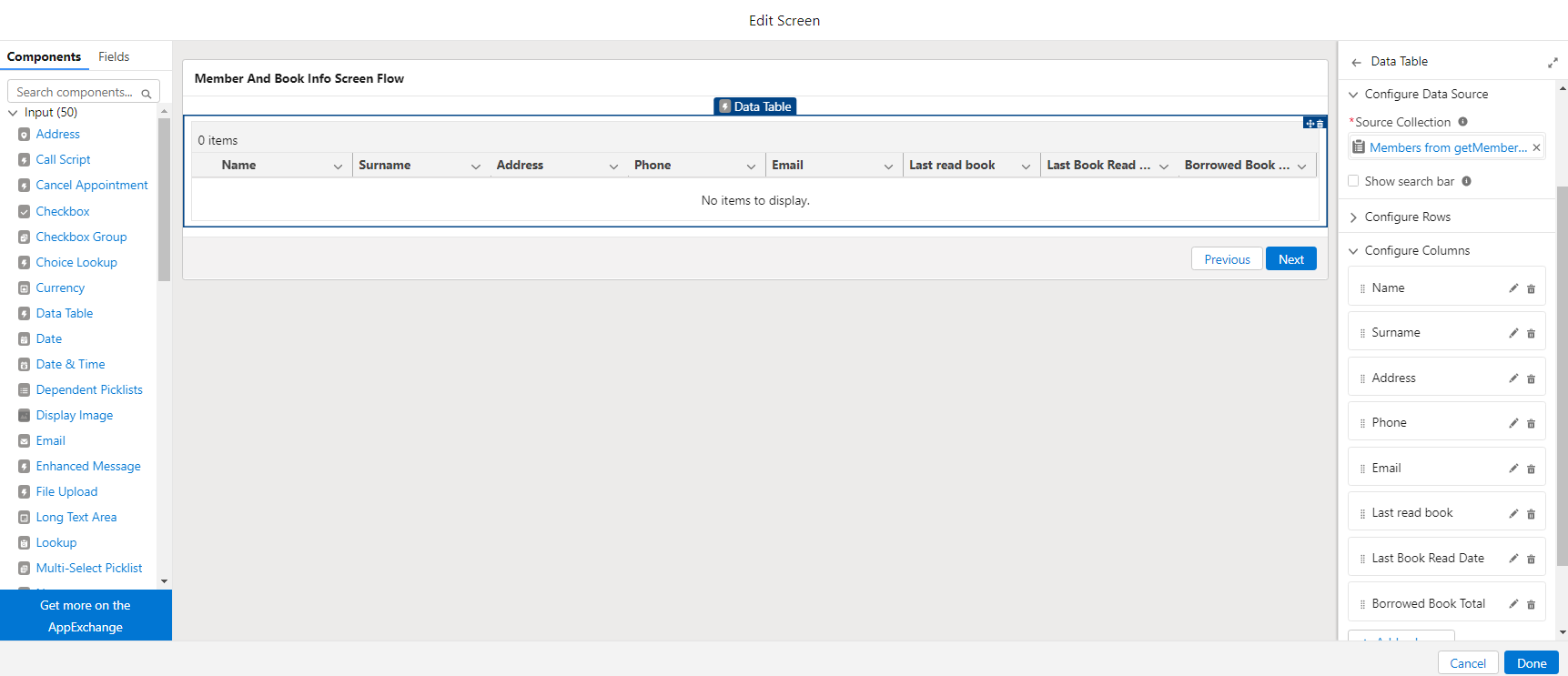




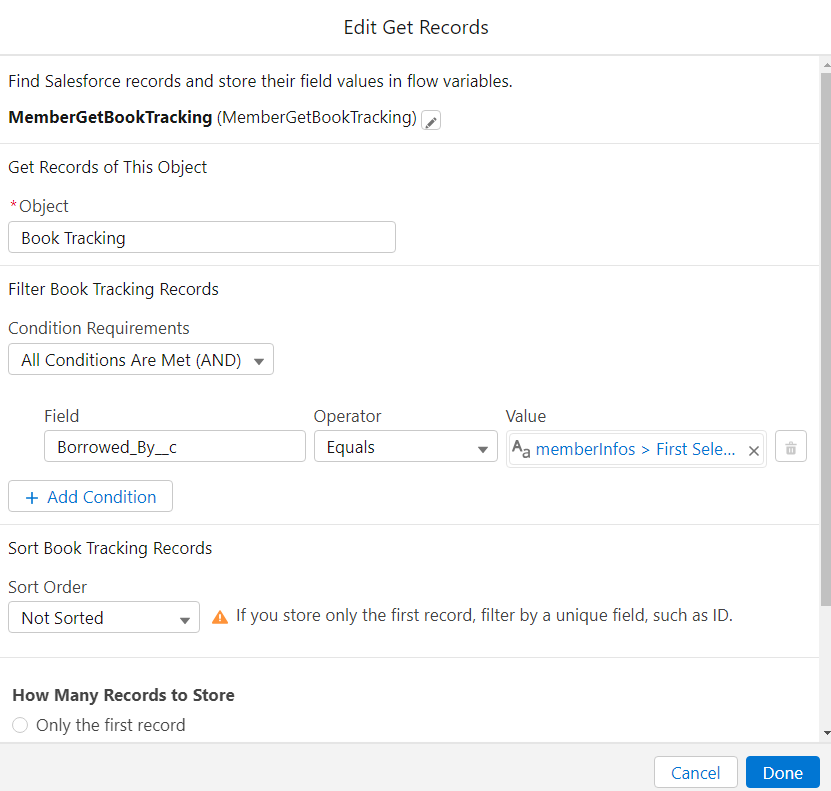
Search kısmında email yazılarak member aratıldıktan sonra bu emaile sahip kullanıcı bulunursa Flow desicion elementinin sol tarafından devam eder.



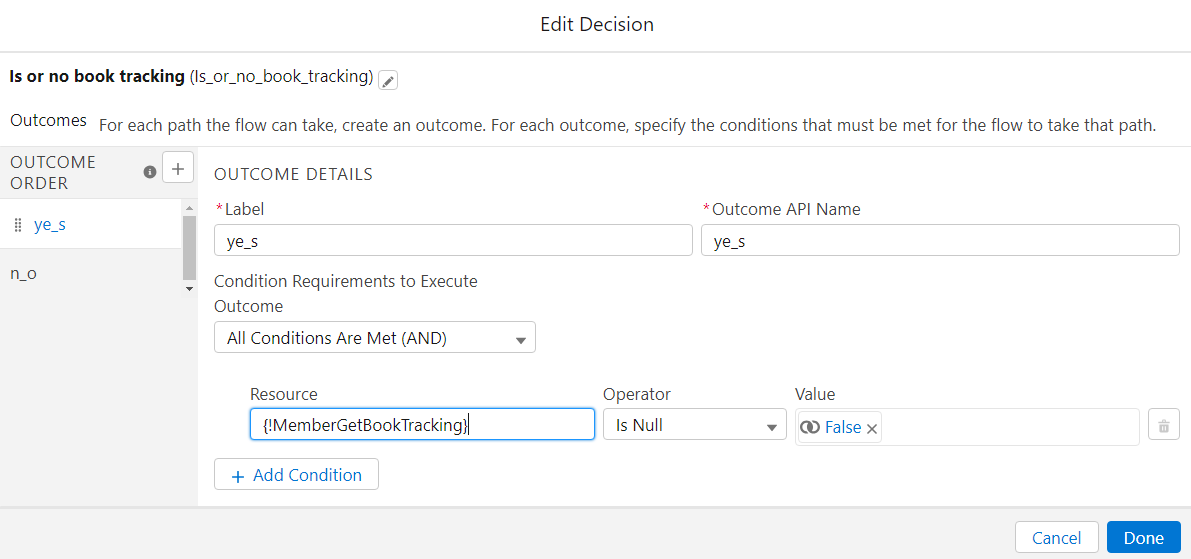
Bulunan member kaydını ekrana screen elementi ile getirir. Burada “Data Table” componenti kullanılarak bulunan member kaydının tüm fieldlarının ekrana getirilmesi amaçlanmıştır. Eğer ekrana getirilen member kaydı aranan member kaydı ile aynı ise bu kayıt seçilerek bir sonraki ekrana geçilmelidir. Eğer ekranda aranılandan farklı bir kayıt görünüyorsa bir önceki ekrana dönülüp girilen email adresi düzeltilmelidir.



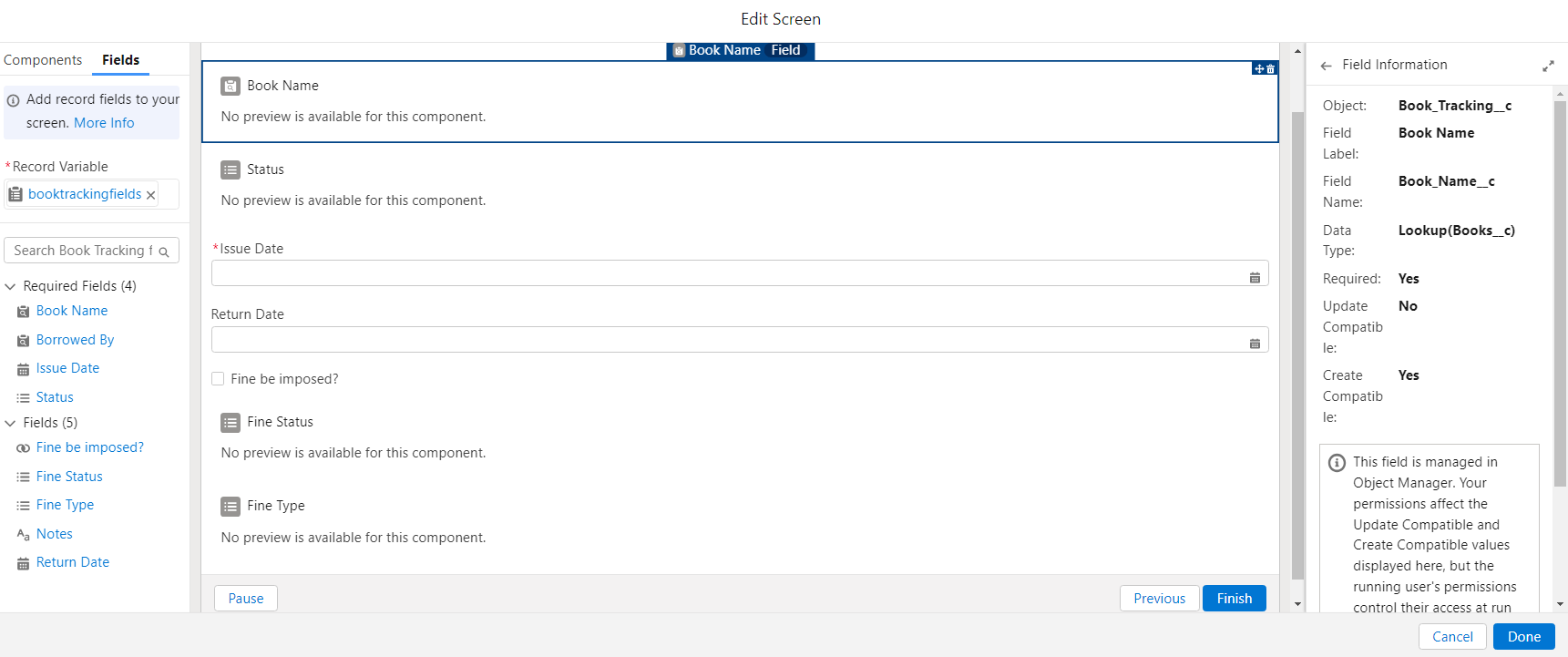
Ekranda görünen member kaydı seçilip next butonuna basıldıktan sonra Flow bize “Get Record” elementi ile bu membera ait Book Tracking kayıtlarını varsa getirmektedir.



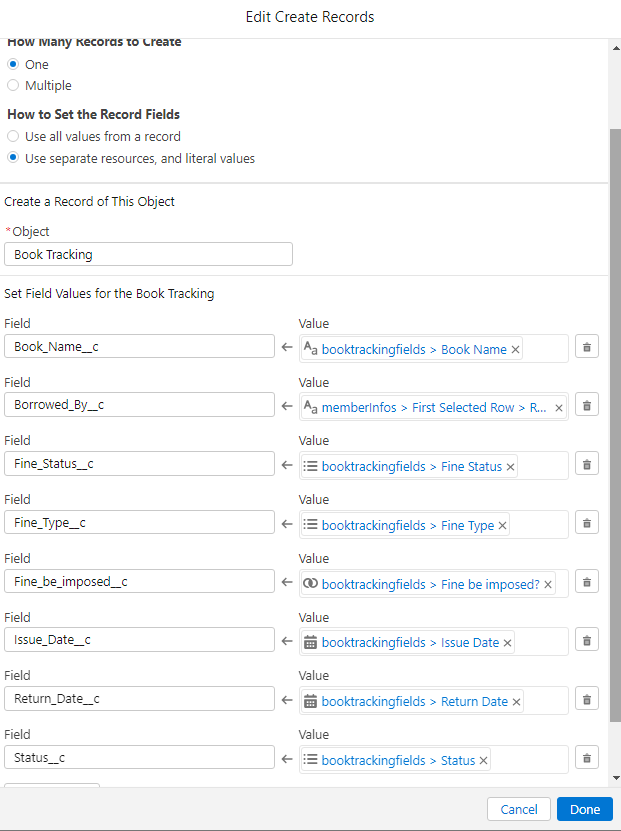
İlgili member kaydına ait Book Tracking kaydının olmama durumu da olabileceğinden burada bu iki durum için farklı akış güzergahı oluşturacak bir “Desicion” elementi karşımıza çıkar.



Eğer Book Tracking kaydı yok ise bunu oluşturabileceğimiz bir “Screen” elementi ile Flow devam eder.



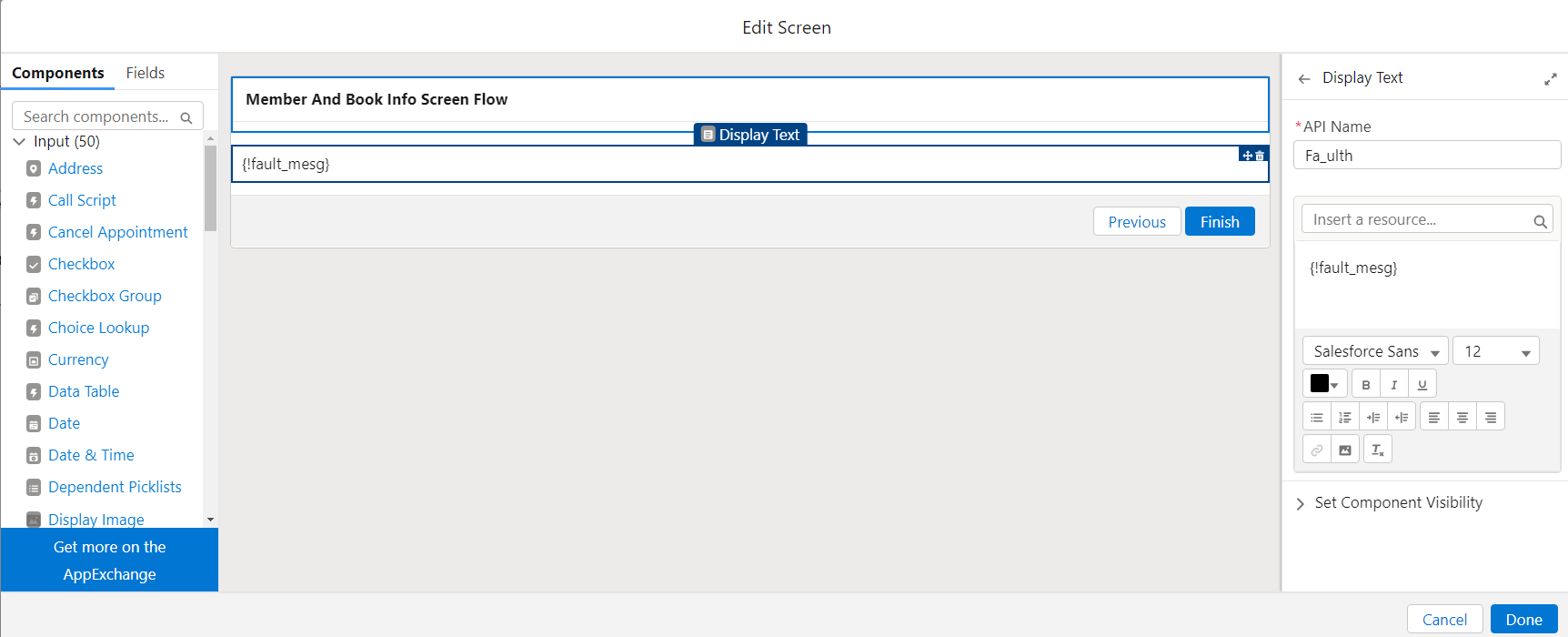
Sonrasında screen elementinden girilen bilgiler ve bir önceki ekranda seçilen member bilgisi “Create Record” elementinde birleştirilerek Book Tracking kaydı oluşturulur.



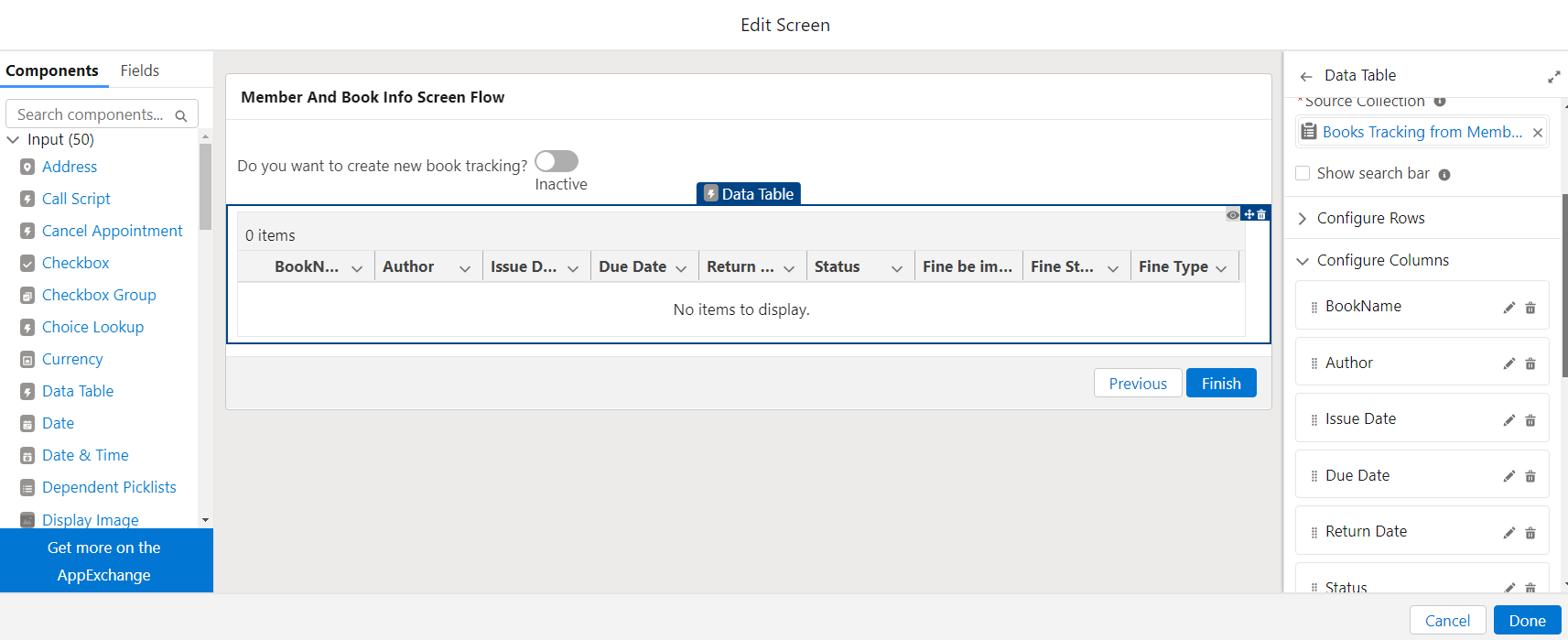
Kayıt başarılı ise bu “Screen” elementi ile kullanıcıya bildirilir.



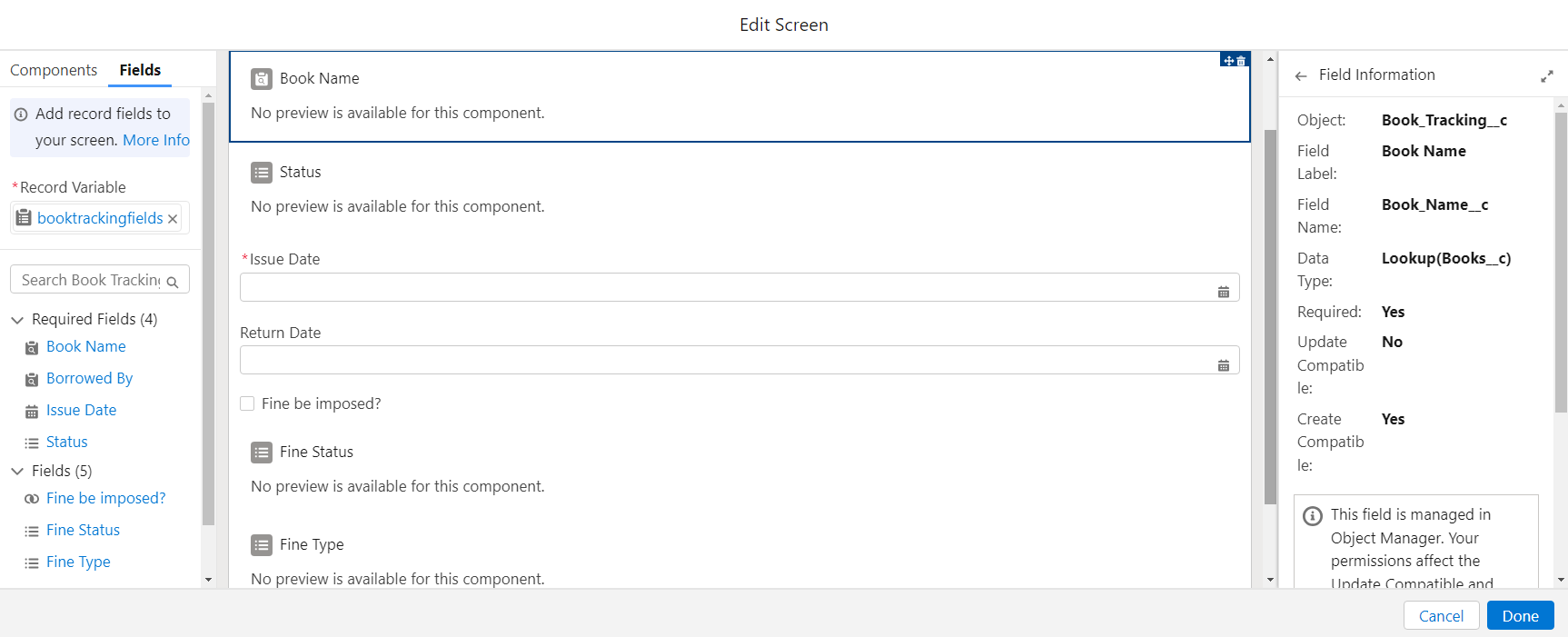
Kayıt başarısız ise “Fault Path” eklenerek burada kullanılan “Screen” elementinde hata mesajı bir formül yardımıyla ekrana yazdırılır. Bu formülde hata mesajının sadece ilgili kısmının ekrana yazdırılması amaçlanmıştır. Hata mesajını gören kullanıcı Flow’dan ayrılmadan bir önceki ekrana geri dönerek hatayı düzeltip Flow’u başarı ile sonlandırabilir.

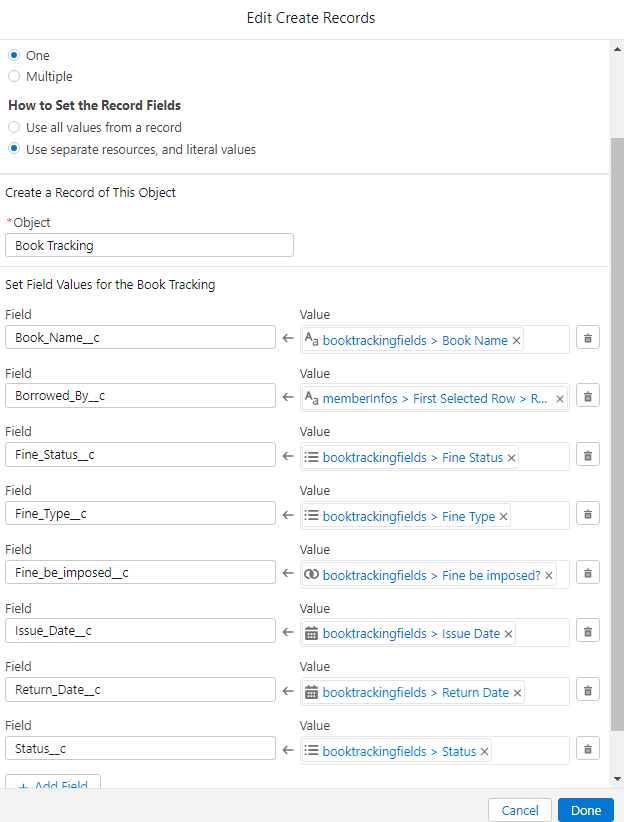


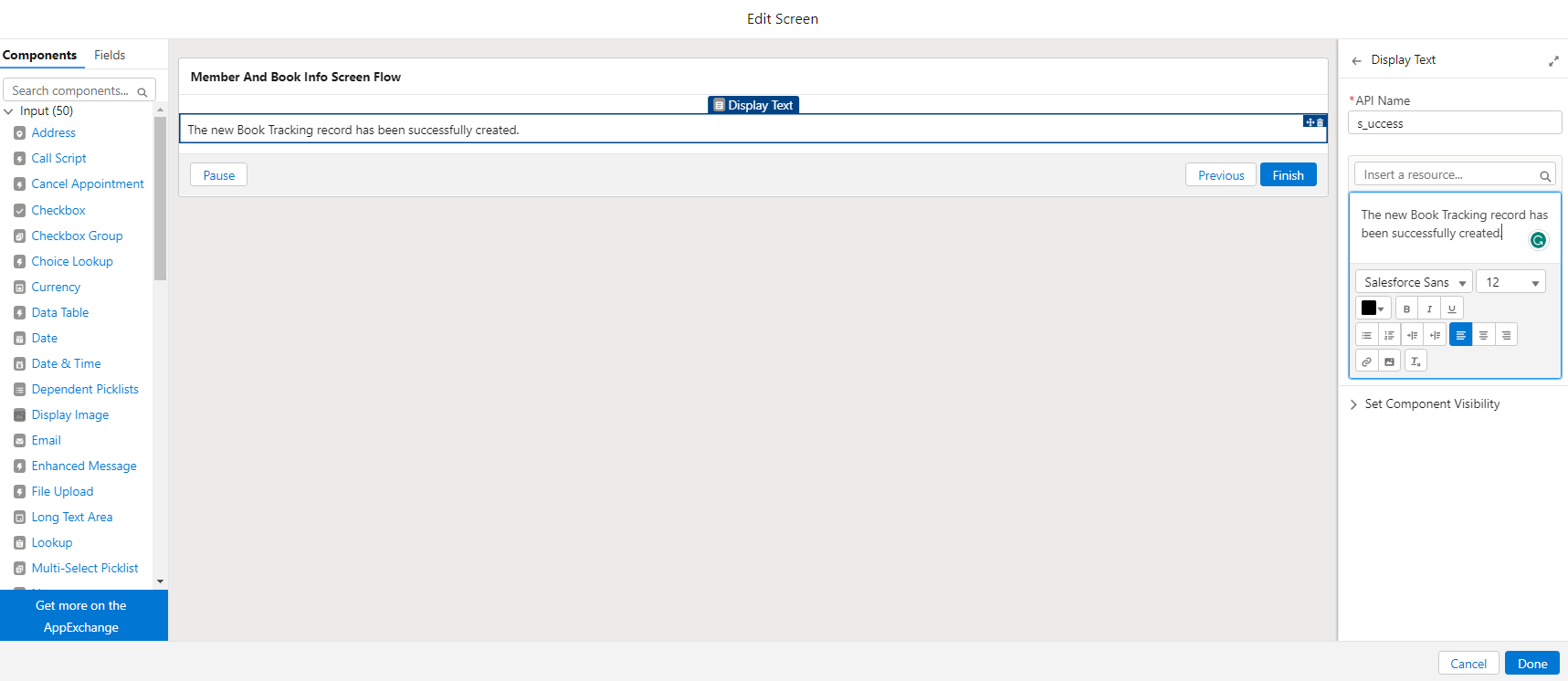
Eğer aratılan membera ait Book Tracking kaydı var ise Flow desicion elementinin sol tarafından devam edecektir. Burada “Screen” elementinin içine konulan “Data Table” komponenti ile bulunan kayıt veya kayıtlara ait ilgili fieldların ekrana yazdırılması amaçlanmıştır. Burada kullanıcı değiştirmek istediği kayıt varsa bu kayda ait satırı seçerek next butonuna basmalıdır. Eğer kullanıcı burada gösterilen mevcut kayıtları düzenlemek yerine yeni bir kayıt oluşturmak istiyorsa buradaki “Toggle” komponentini aktif hale getirmeli ve next butonuna basmalıdır.

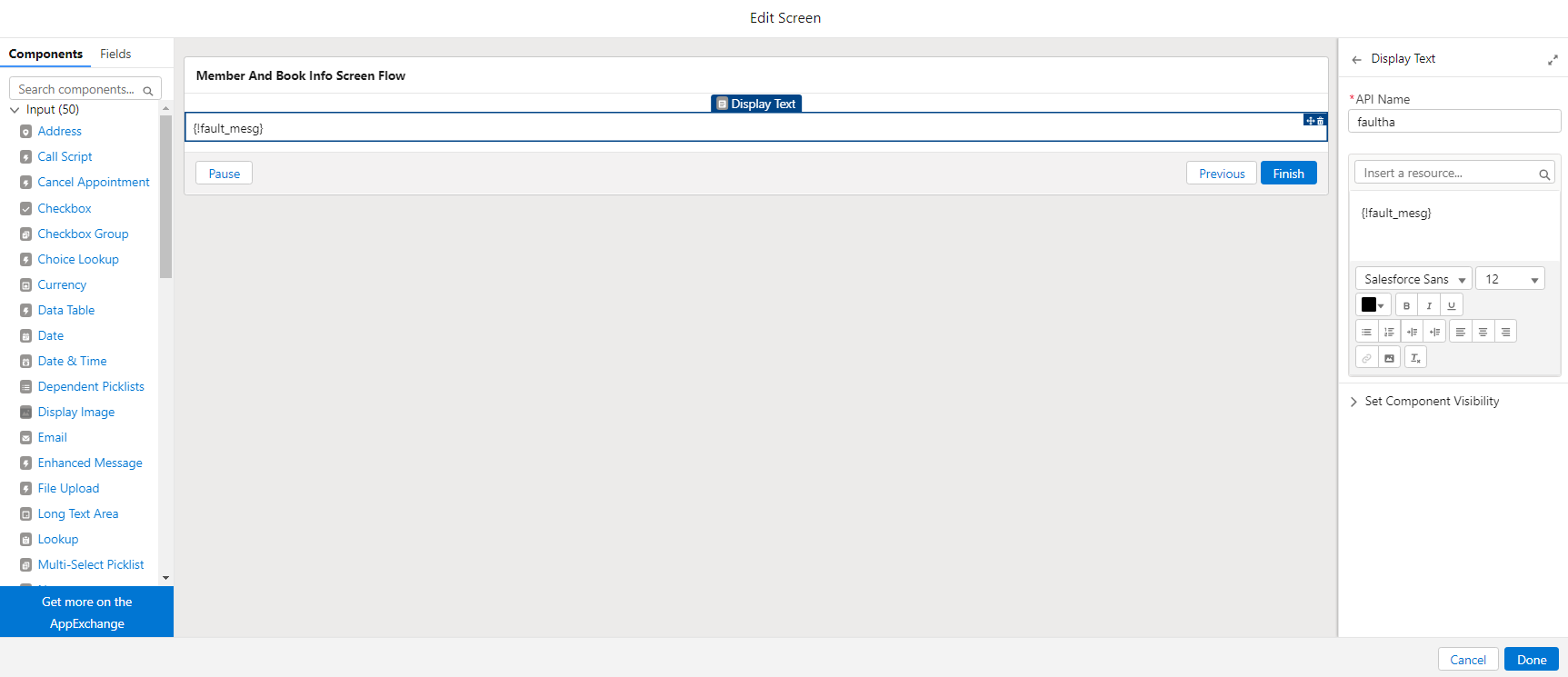


Eğer toggle aktif hale getirildiyse flow sağ yolu takip edecek ve ilgili member için yeni bir Book Tracking kaydı oluşturacaktır. Bunun için ilgili fieldların olduğu “Screen” komponenti karşımıza çıkar. Bundaki sonraki işleyiş daha önce anlatılan işleyişle aynıdır.

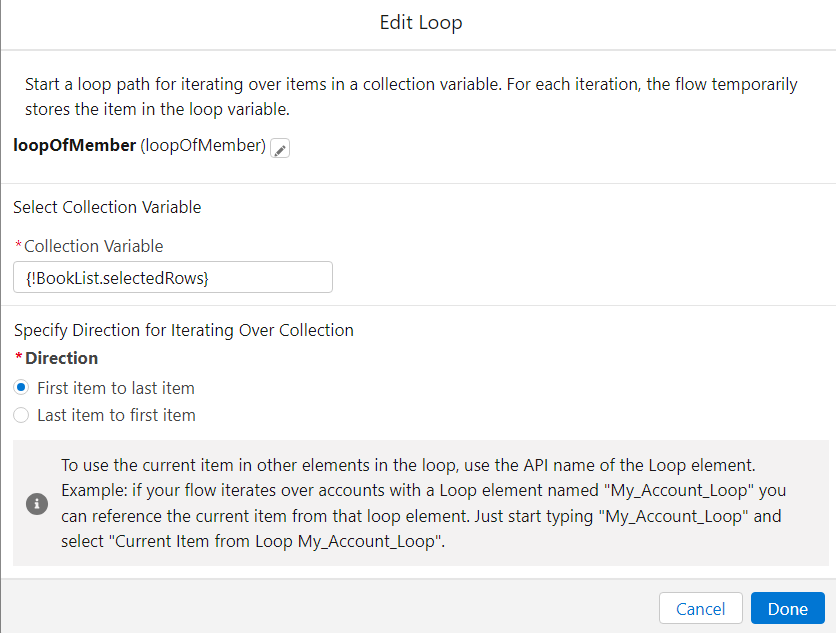




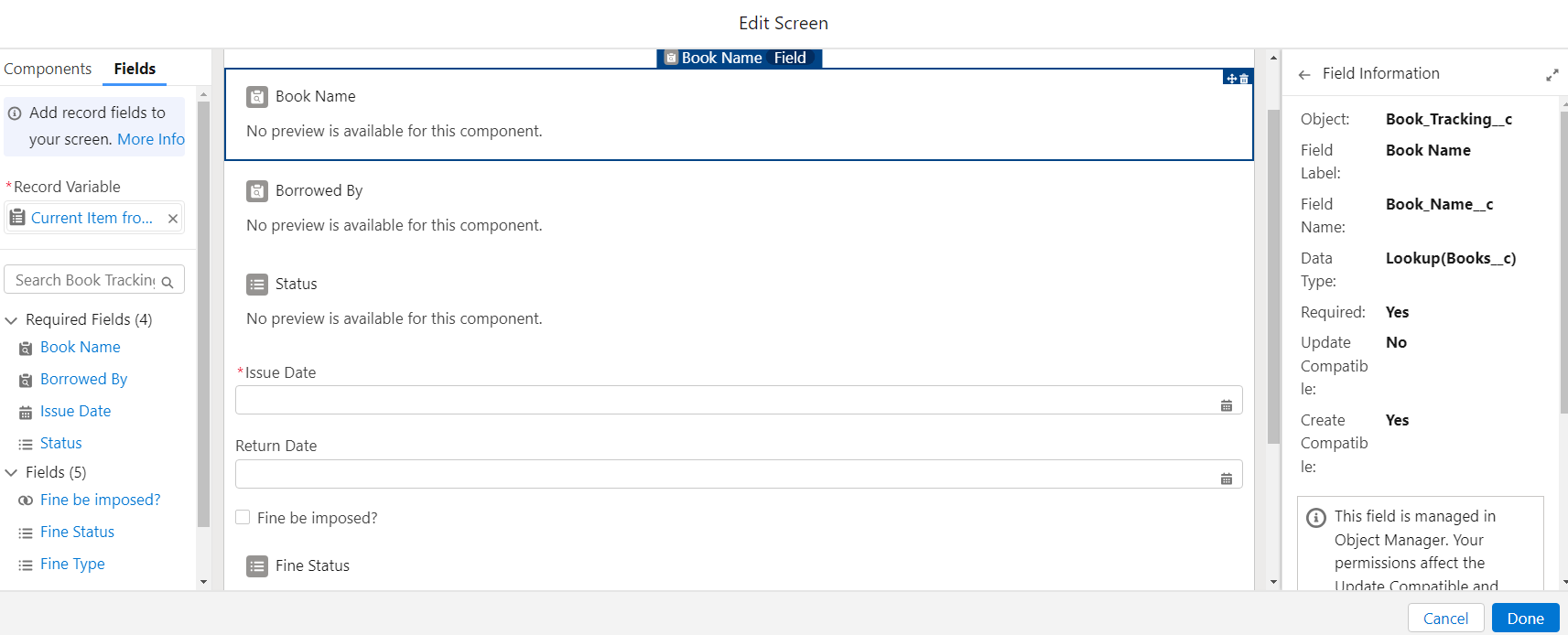




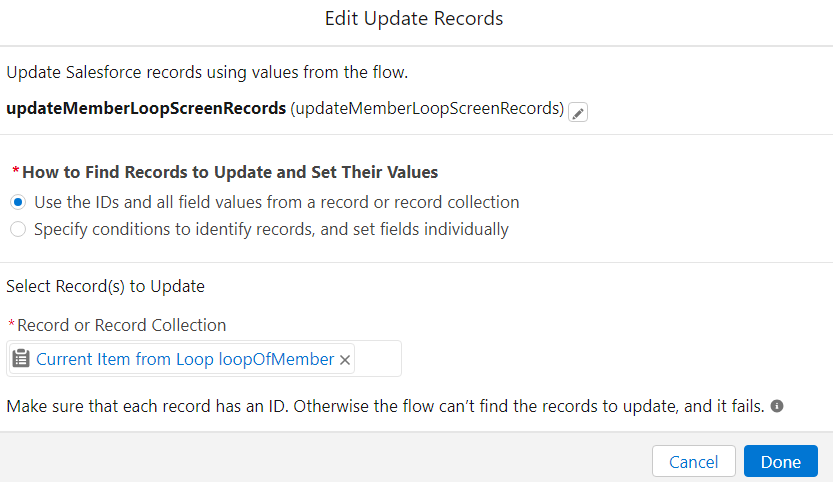
Eğer “Toggle” komponenti aktif hale getirilmeyip tablodaki Book tracking kayıtlarına ait bir satır seçilir ve next butonuna basılırsa Flow diğer yoldan devam eder. Burada karşımıza “Loop” elementi çıkar. Loop elementi ile tabloda birden fazla satır seçilmesi durumunda bunların hepsinin güncellenmesi amaçlanmıştır.



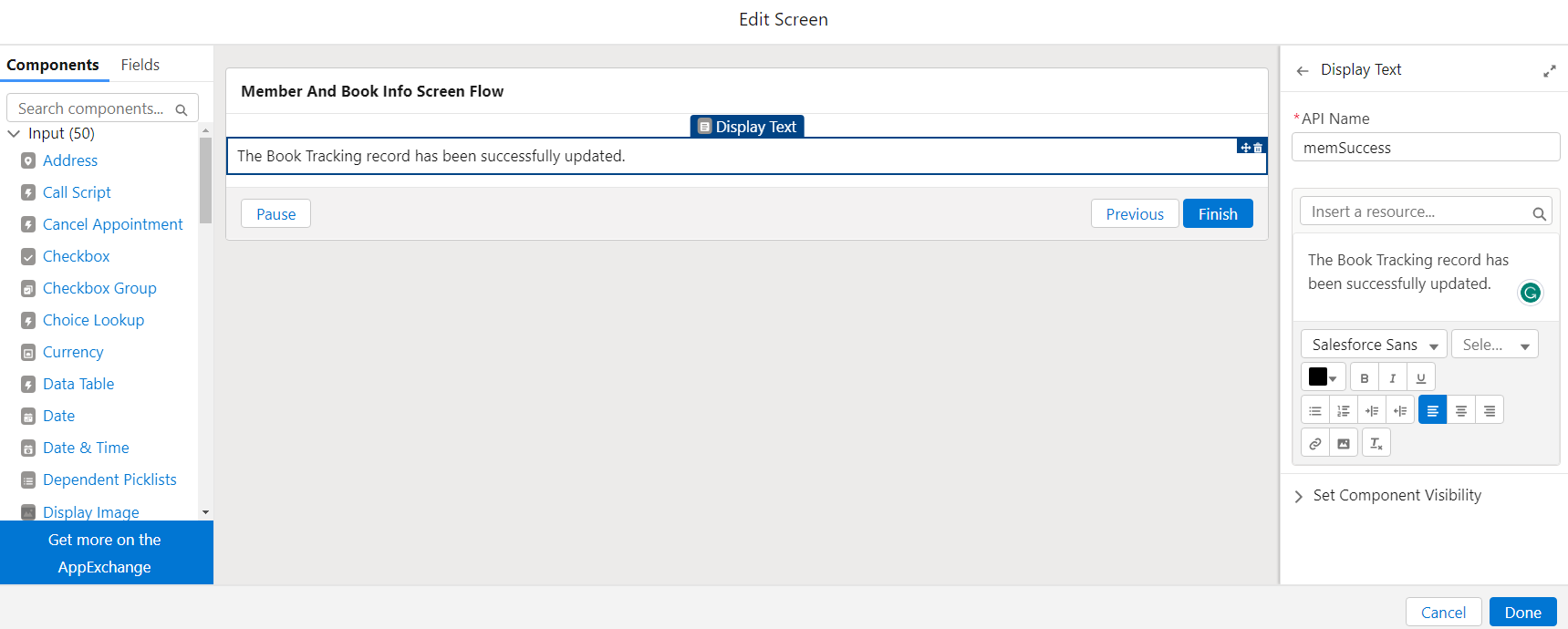
Loop’tan sonra “Screen” elementi ile bir önceki ekranda seçilen Book Tracking kaydına ait bilgiler ekrana getirilir. Kullanıcı burada değiştirmek istediği bilgileri değiştirebilir.



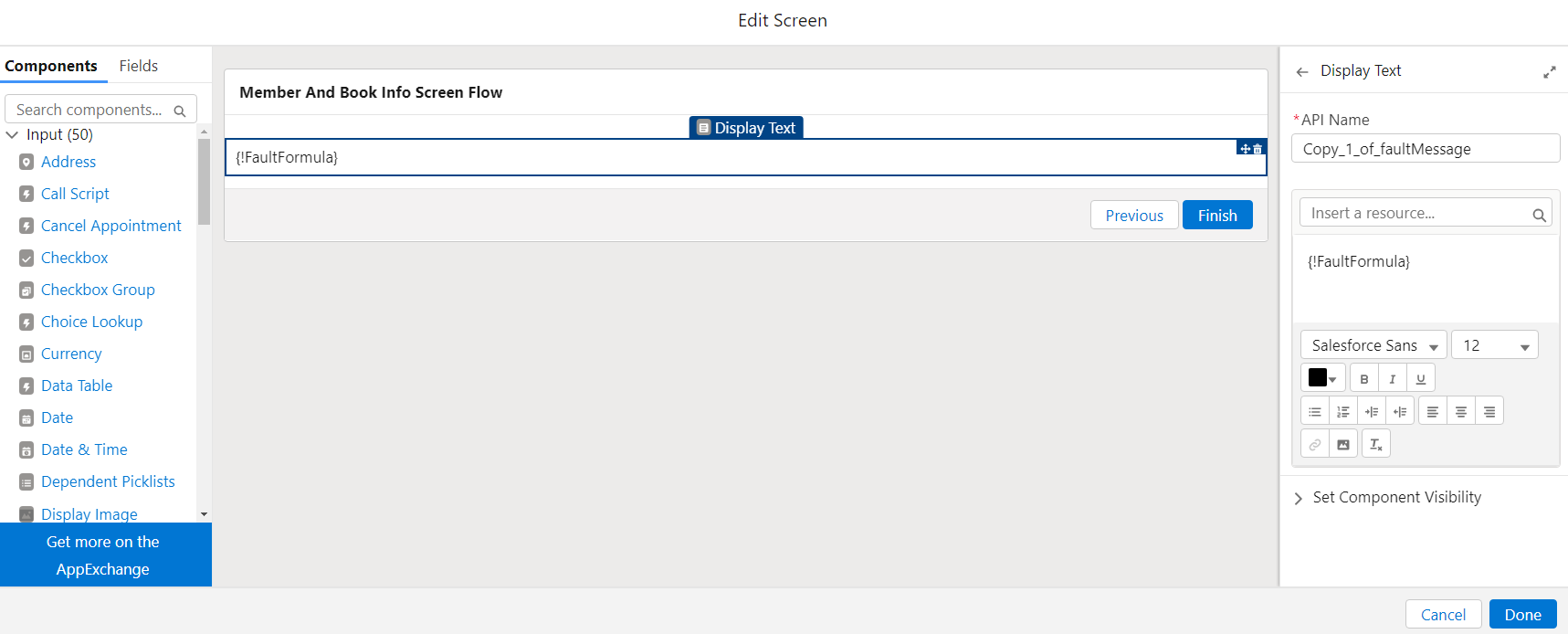
Daha sonra değiştirilen bu bilgiler “Update” elementi yardımıyla sisteme kaydedilir.



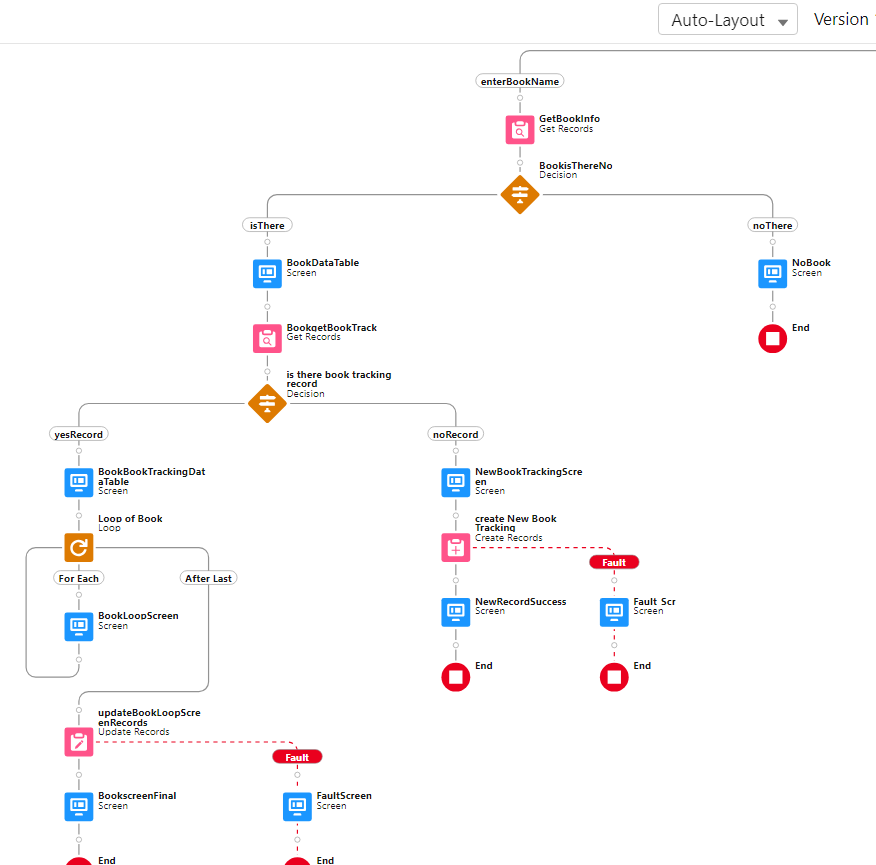
Update başarılı ise aşağıdaki ekranda bu kullanıcıya bildirilir.



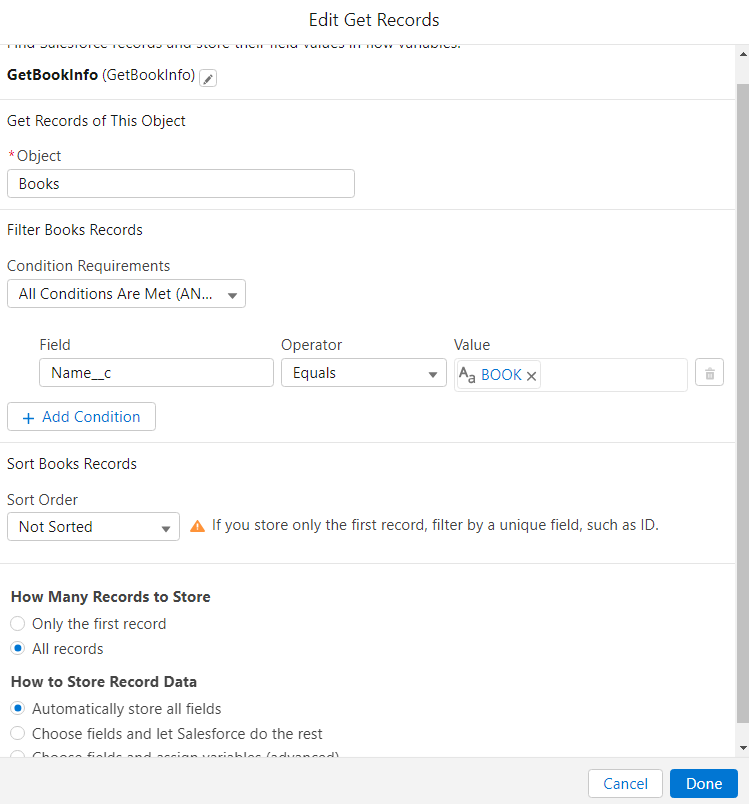
Başarısız ise daha öncekilere benzer şekilde hata ekrana yazıdırılır.



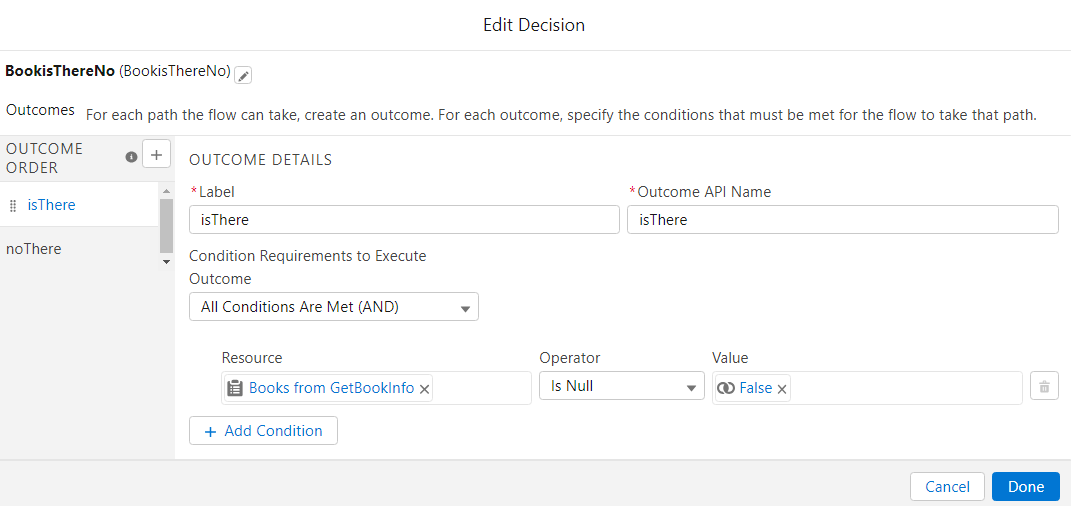
Buradan sonra kullanıcının bir “Book” kaydı arattığı durumu inceleyeceğiz.



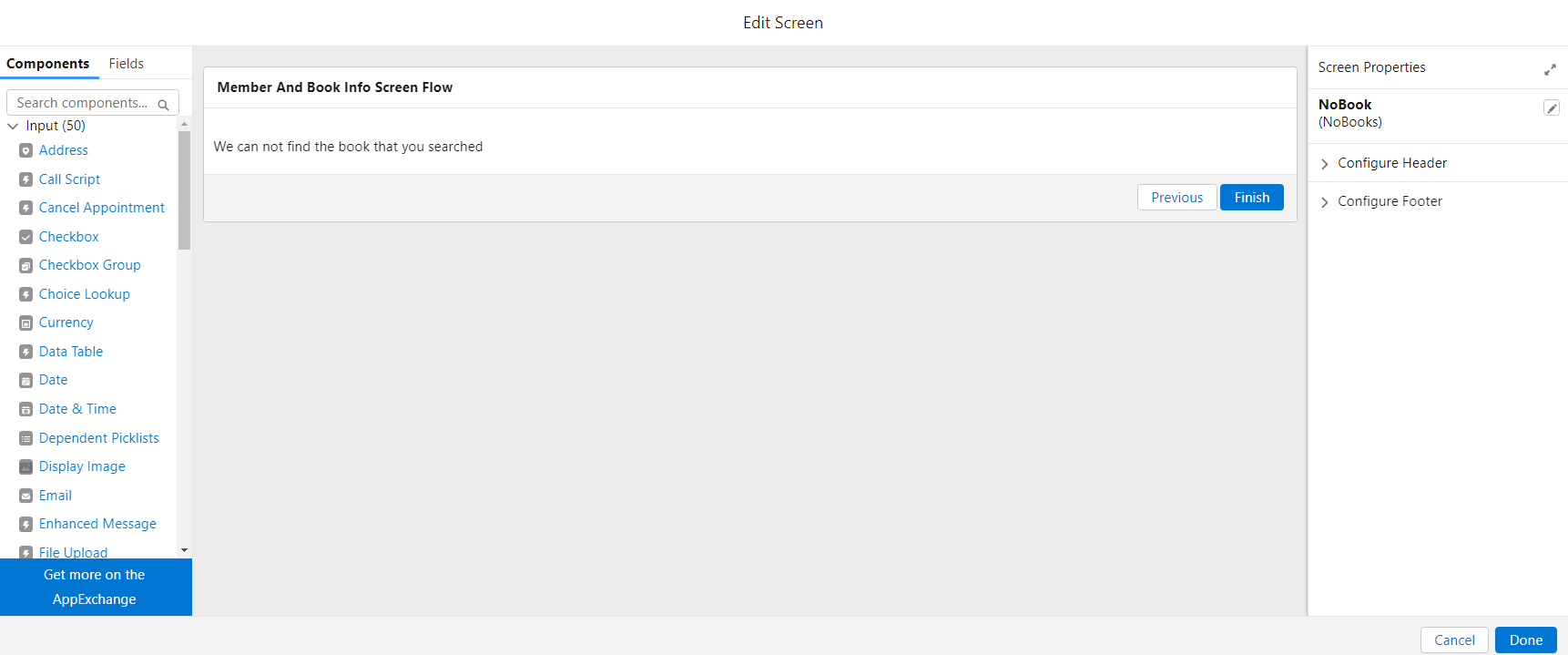
Kullanıcı aradığı kitap ismini yazdıktan sonra varsa bu kitaba ait bilgiler “Get Record” elementi ile Flow’a getirilir.

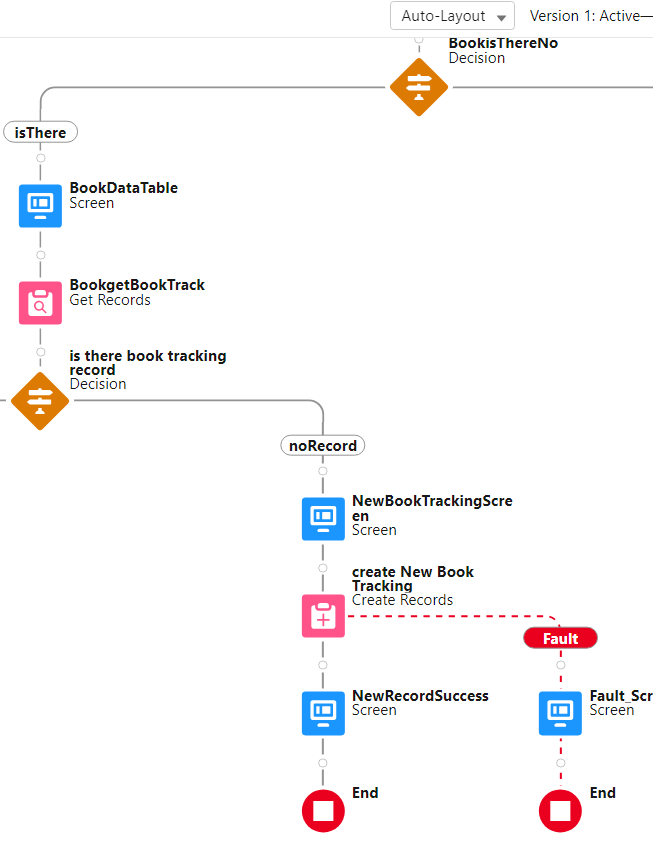


“Desicion” elementi ile burada Flow iki yola ayrılır. “Get Record” elementi ilgili Book kaydını bulamadıysa yani böyle bir kitap sistemde kayıtlı değil ise Flow sağ yoldan devam eder.

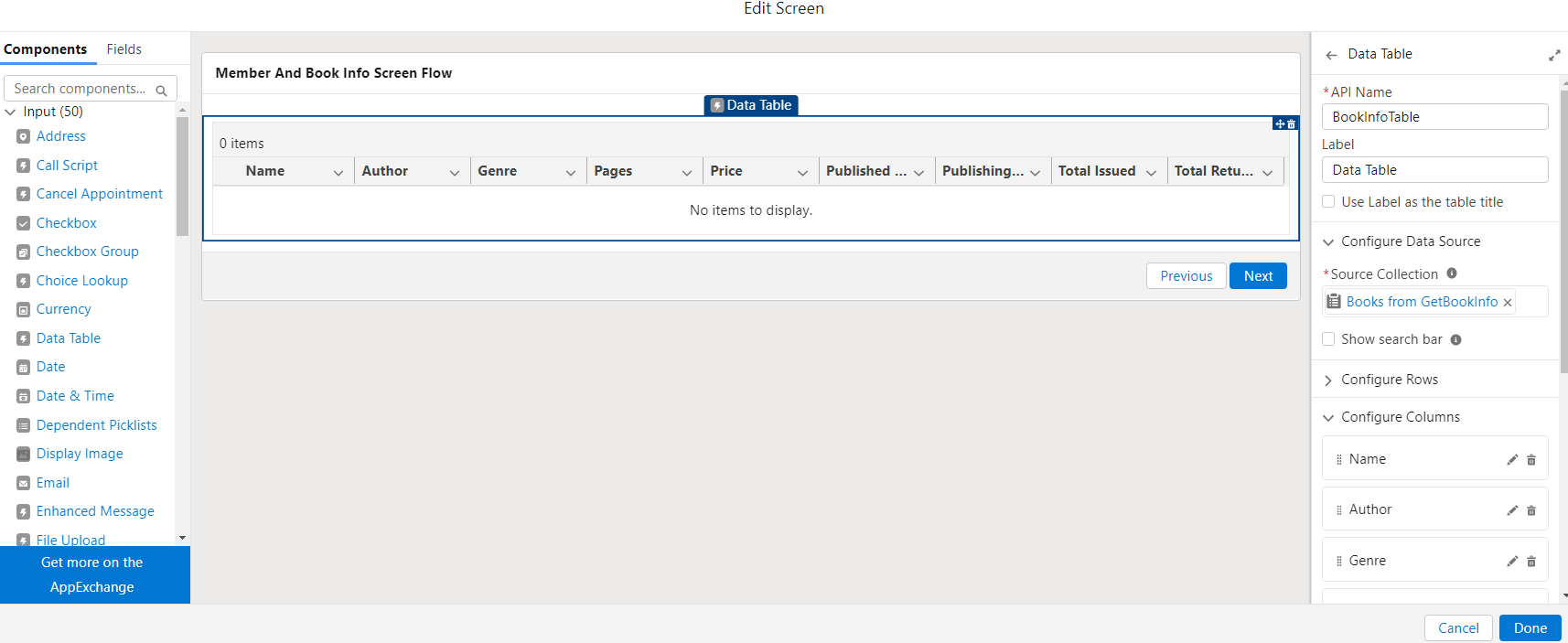


Bu kitabın bulunamadığı “Screen” elementi yardımıyla kullanıcıya bildirilir.

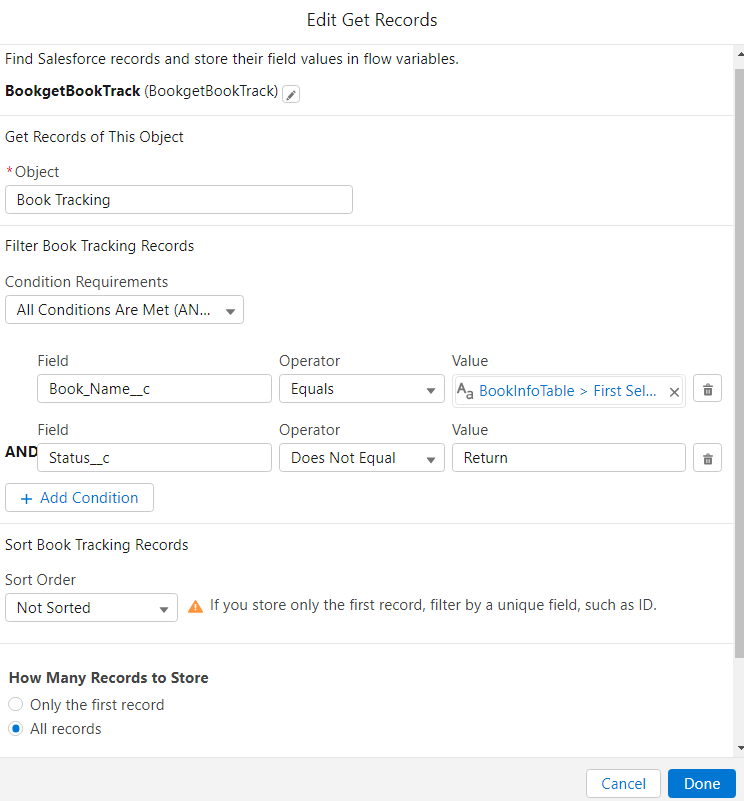




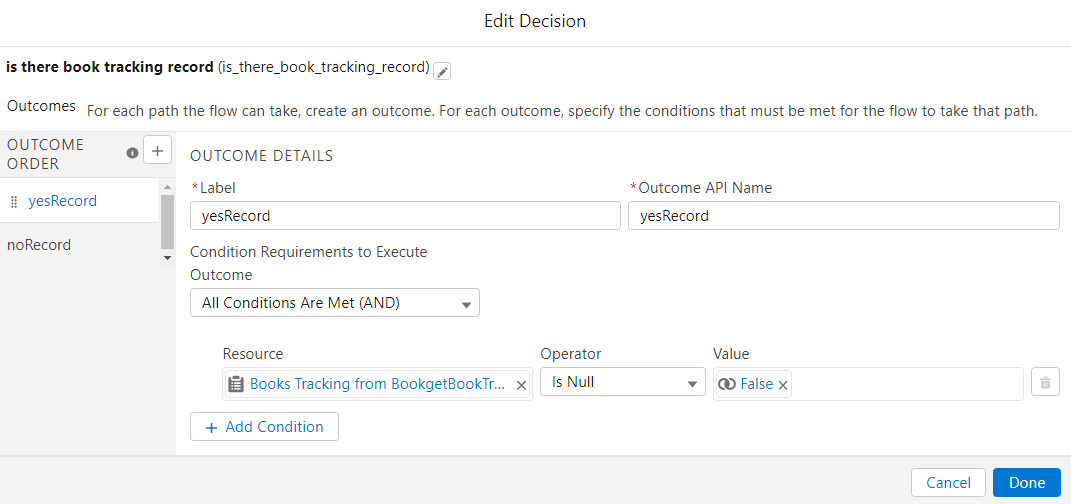
Eğer kitap sistemde kayıtlı ise Flow soldan devam eder ve “Screen” elementine konulan “Data Table” komponenti bize sistemde kayıtlı kitap bilgilerini gösterir. Burada işlem yapılmak istenen kitap kaydına ait satır seçilip devam edilmelidir.



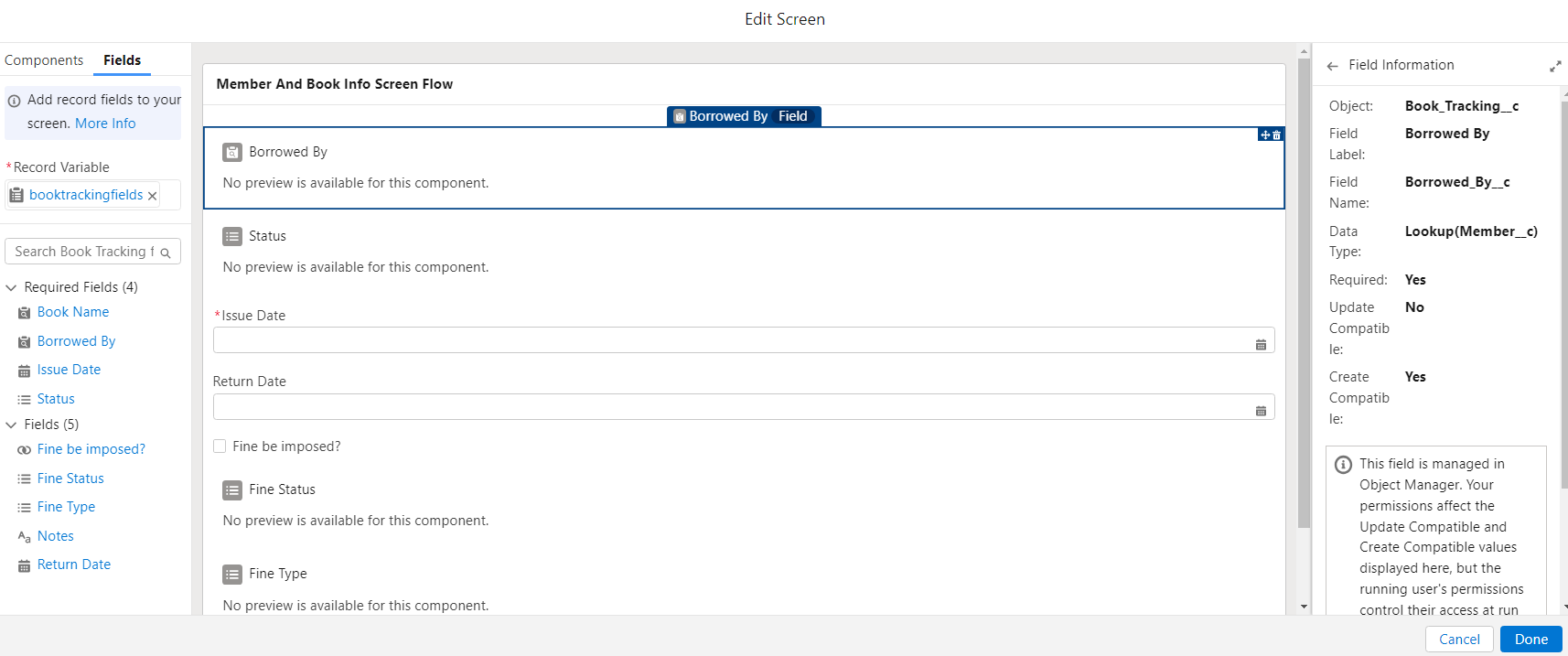
Seçilen kitaba ait Book Tracking kaydı varsa “Get Record” elementi ile bu bilgileri Flow’a taşırız. Bu elementin içindeki condition’da “Data Table” komponentinde seçilen kitaptan “Status” alanı “Return” olmayanlar belirtilmiştir. Burada sadece güncelleme yapılabilecek kayıtların Flow’a aktarılması amaçlanmıştır.



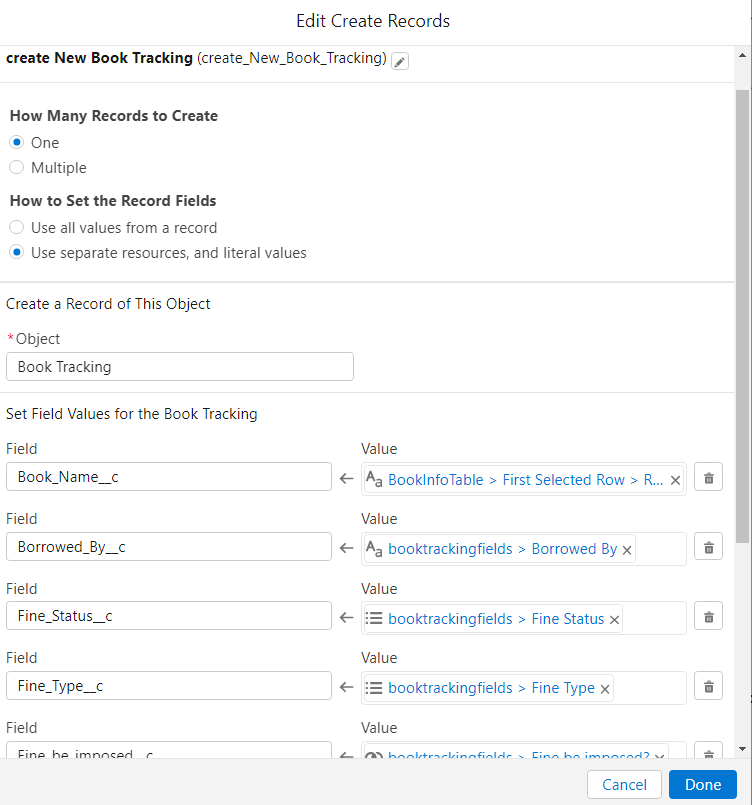
“Desicion” elementi ile burada Flow iki yola ayrılır. “Get Record” elementi ilgili Book Tracking kaydını bulamadıysa yani kitapla ilgili Book Tracking işlemi yapılmadıysa Flow sağ yoldan devam eder.

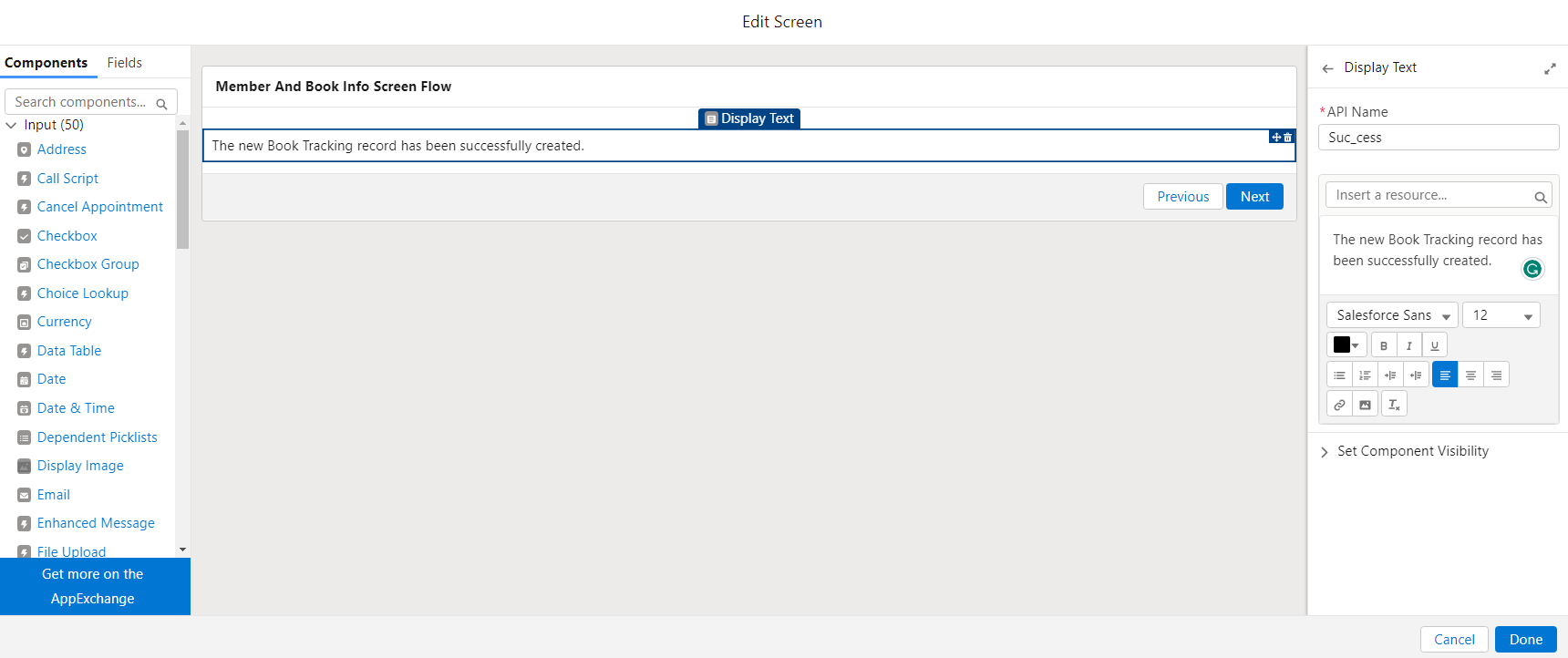


Eğer Book Tracking kaydı yok ise bunu oluşturabileceğimiz bir “Screen” elementi ile Flow devam eder.

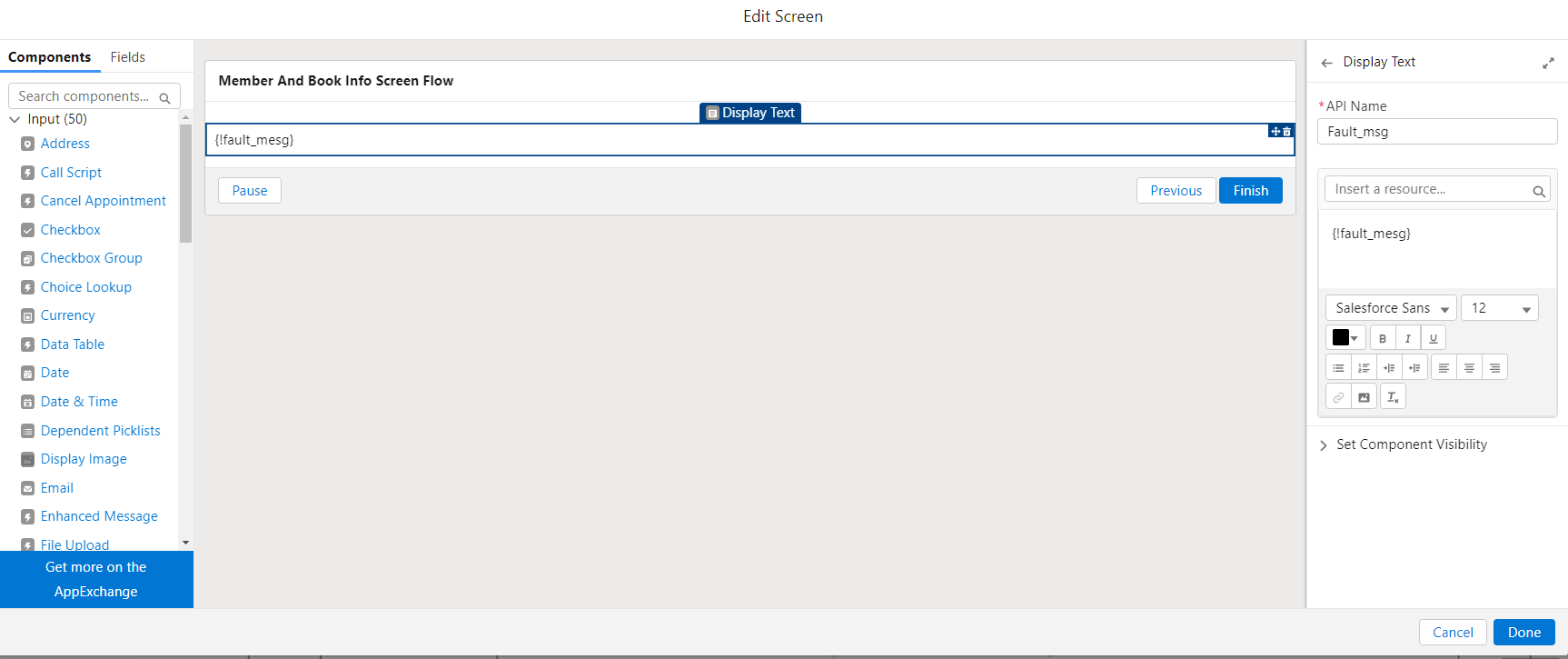


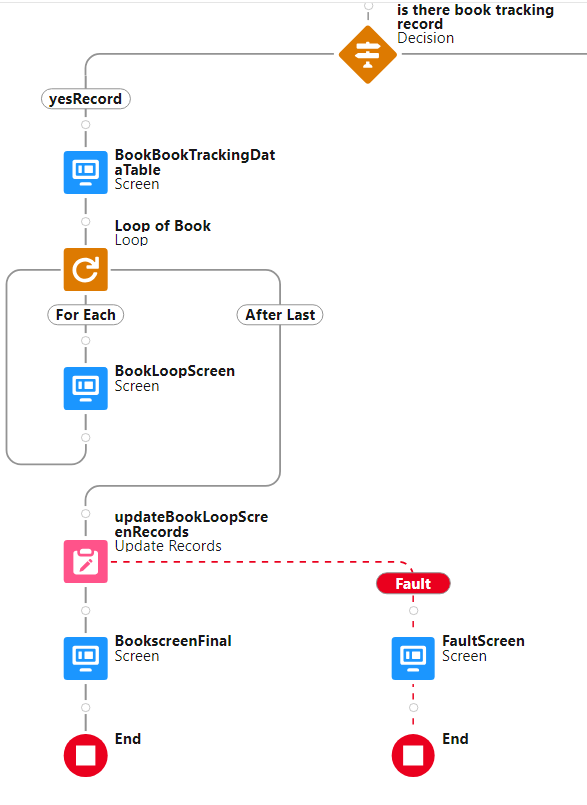
Sonrasında screen elementinden girilen bilgiler ve bir önceki ekranda seçilen member bilgisi “Create Record” elementinde birleştirilerek Book Tracking kaydı oluşturulur.



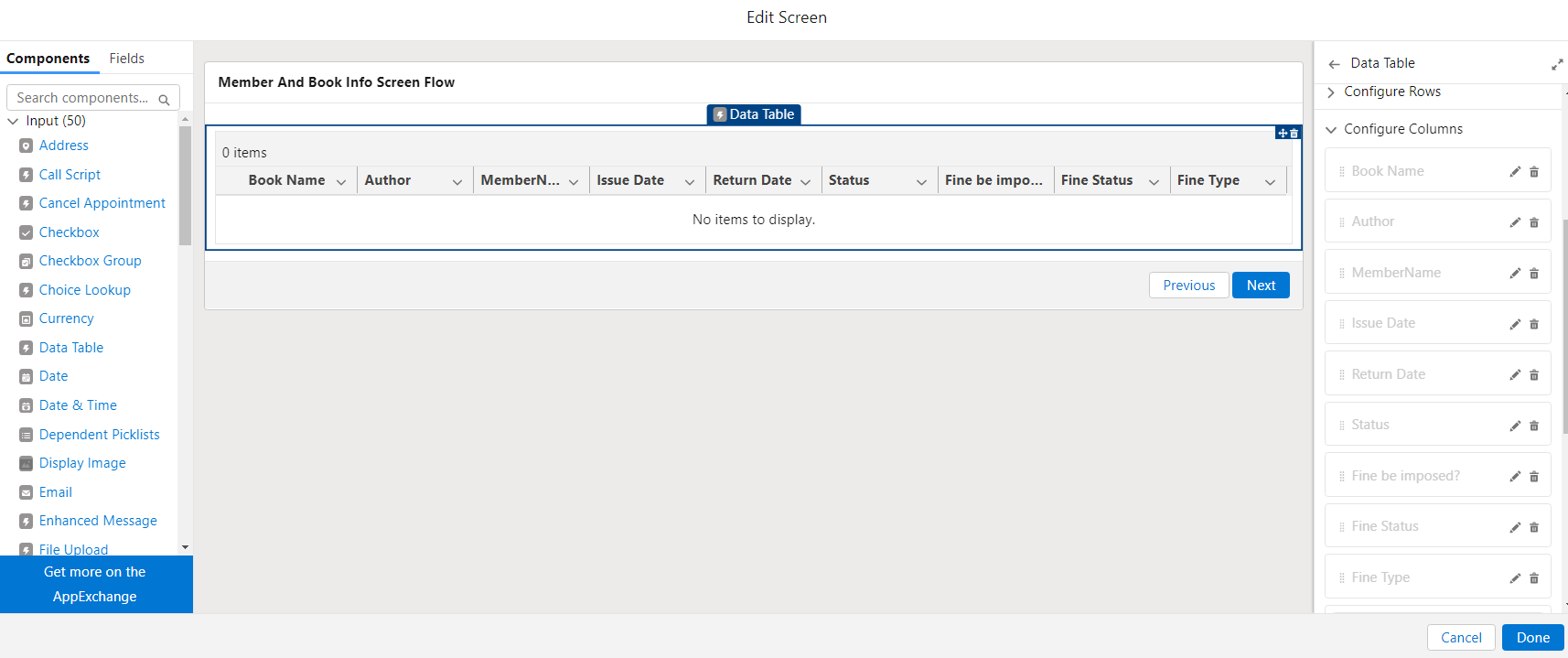


Kayıt başarısız ise “Fault Path” eklenerek burada kullanılan “Screen” elementinde hata mesajı bir formül yardımıyla ekrana yazdırılır. Bu formülde hata mesajının sadece ilgili kısmının ekrana yazdırılması amaçlanmıştır. Hata mesajını gören kullanıcı Flow’dan ayrılmadan bir önceki ekrana geri dönerek hatayı düzeltip Flow’u başarı ile sonlandırabilir.

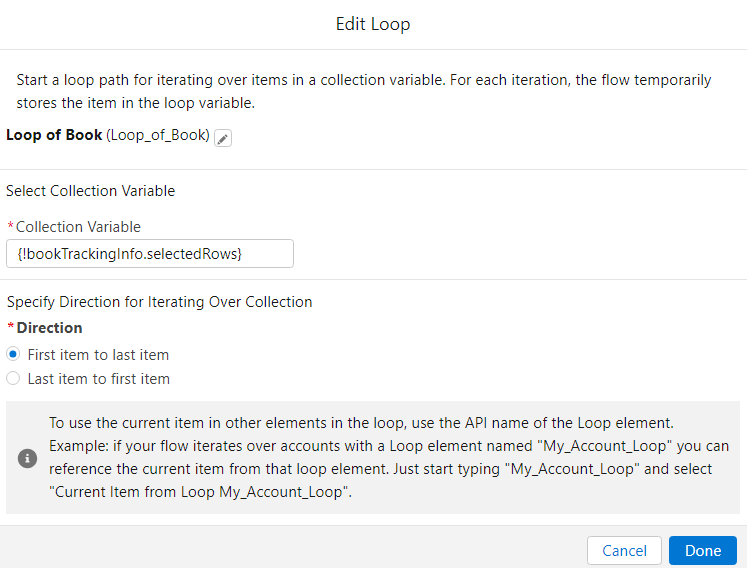




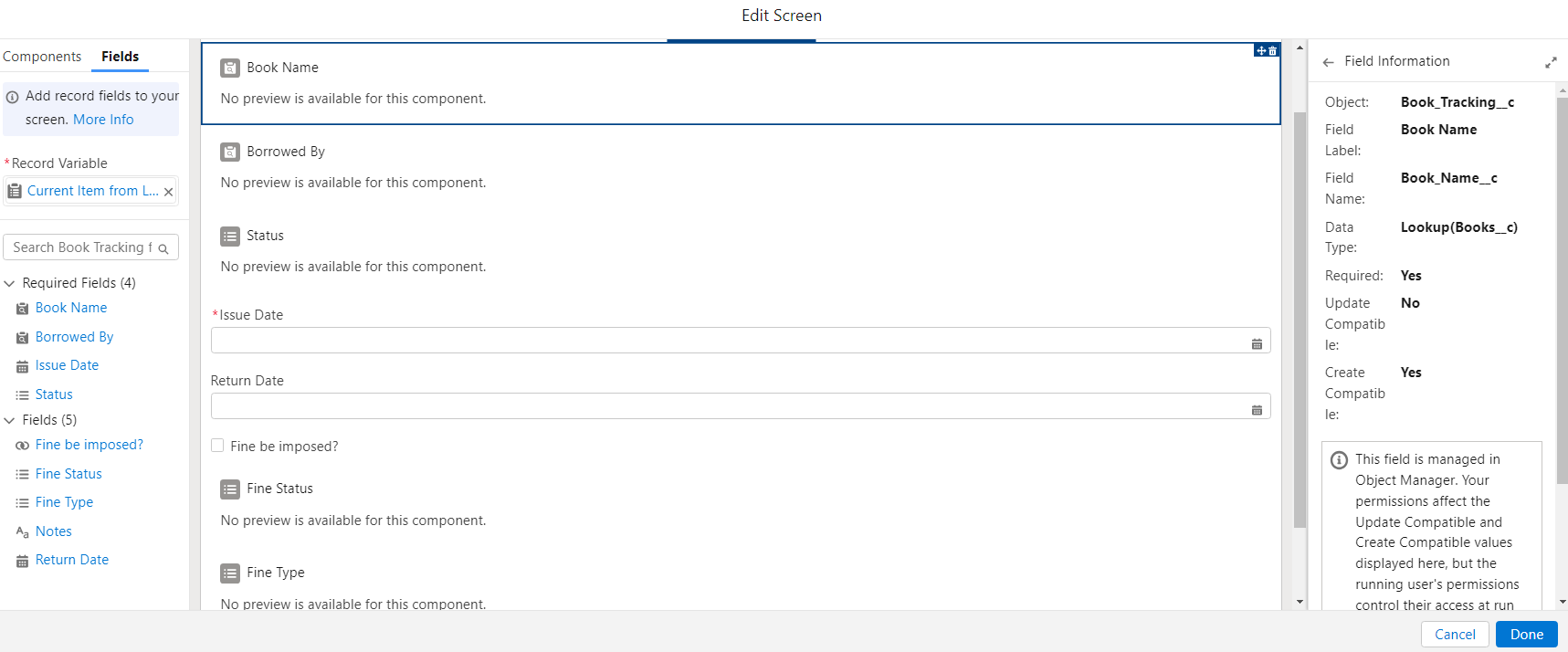
Eğer aratılan kitaba ait Book Tracking kaydı var ise Flow desicion elementinin sol tarafından devam edecektir. Burada “Screen” elementinin içine konulan “Data Table” komponenti ile bulunan kayıt veya kayıtlara ait ilgili fieldların ekrana yazdırılması amaçlanmıştır. Burada kullanıcı değiştirmek istediği kayıt varsa bu kayda ait satırı seçerek next butonuna basmalıdır.



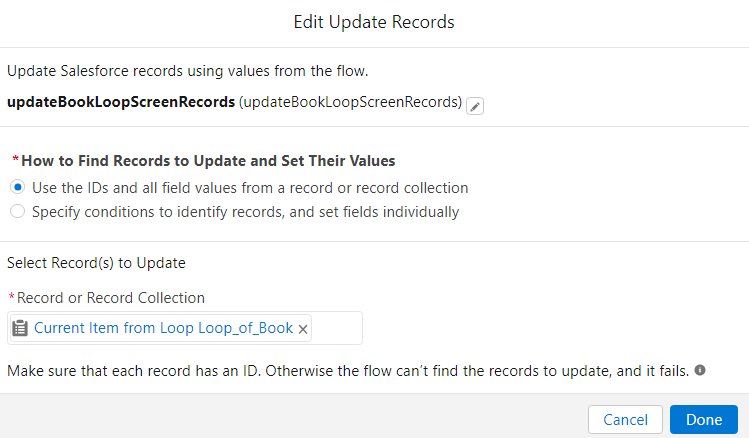
Burada karşımıza “Loop” elementi çıkar. Loop elementi ile tabloda birden fazla satır seçilmesi durumunda bunların hepsinin güncellenmesi amaçlanmıştır.



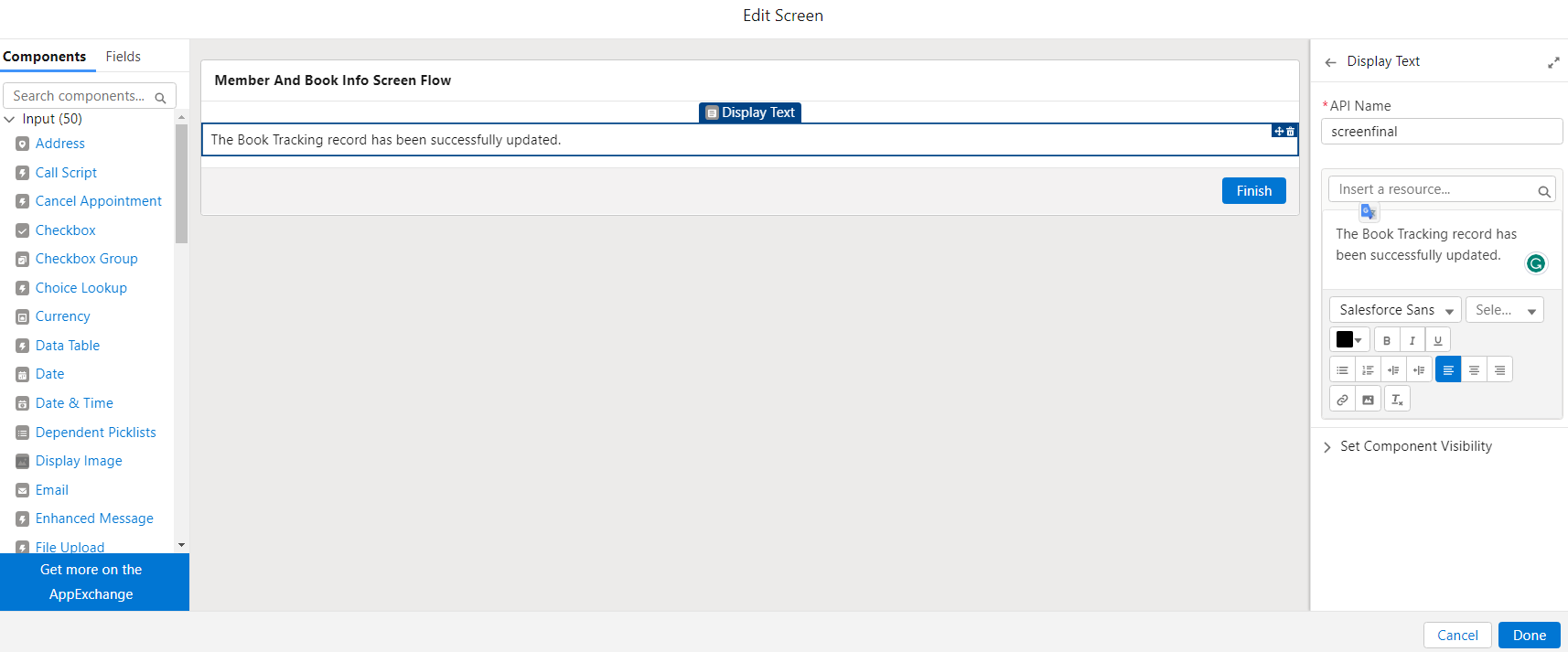
Loop’tan sonra “Screen” elementi ile bir önceki ekranda seçilen Book Tracking kaydına ait bilgiler ekrana getirilir. Kullanıcı burada değiştirmek istediği bilgileri değiştirebilir.

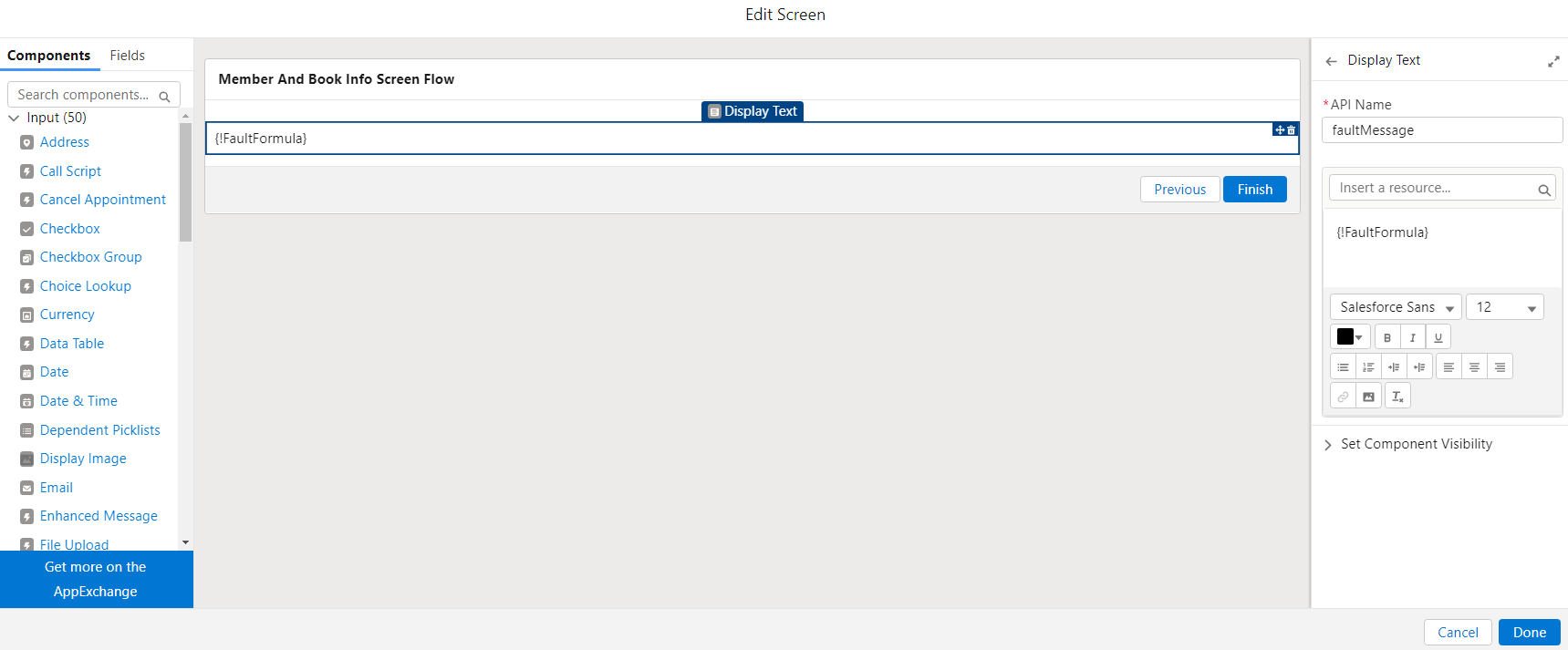


Daha sonra değiştirilen bu bilgiler “Update” elementi yardımıyla sisteme kaydedilir.



Kayıt başarılı ise bu “Screen” elementi ile kullanıcıya bildirilir.



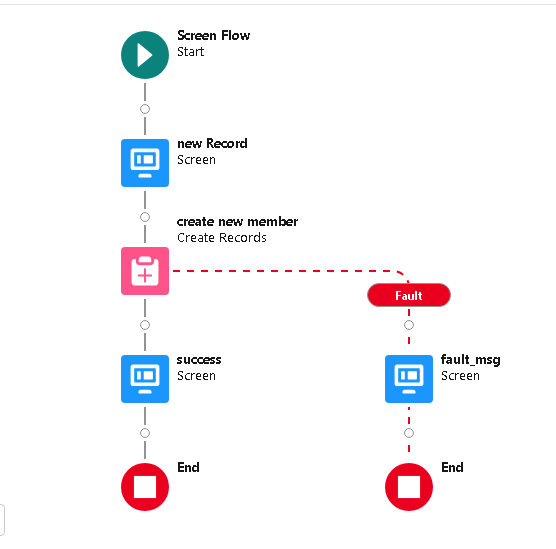


Kayıt başarısız ise “Fault Path” eklenerek burada kullanılan “Screen” elementinde hata mesajı bir formül yardımıyla ekrana yazdırılır. Bu formülde hata mesajının sadece ilgili kısmının ekrana yazdırılması amaçlanmıştır. Hata mesajını gören kullanıcı Flow’dan ayrılmadan bir önceki ekrana geri dönerek hatayı düzeltip Flow’u başarı ile sonlandırabilir.

1. **NEW MEMBER RECORD SCREEN FLOW**

**Description:** This flow is used to register members. Records created on the screen are reflected on the Salesforce page.

**Amaç:** Member\_\_c objesine yeni kayıt eklemek. Ekranda gerekli alanlar; Name, Surname, Email, Address, Phone bilgilerini girdikten sonra otomatik olarak yeni bir kayıt oluşacak. Bu kayıt Member\_\_c objesinde gözükecek.



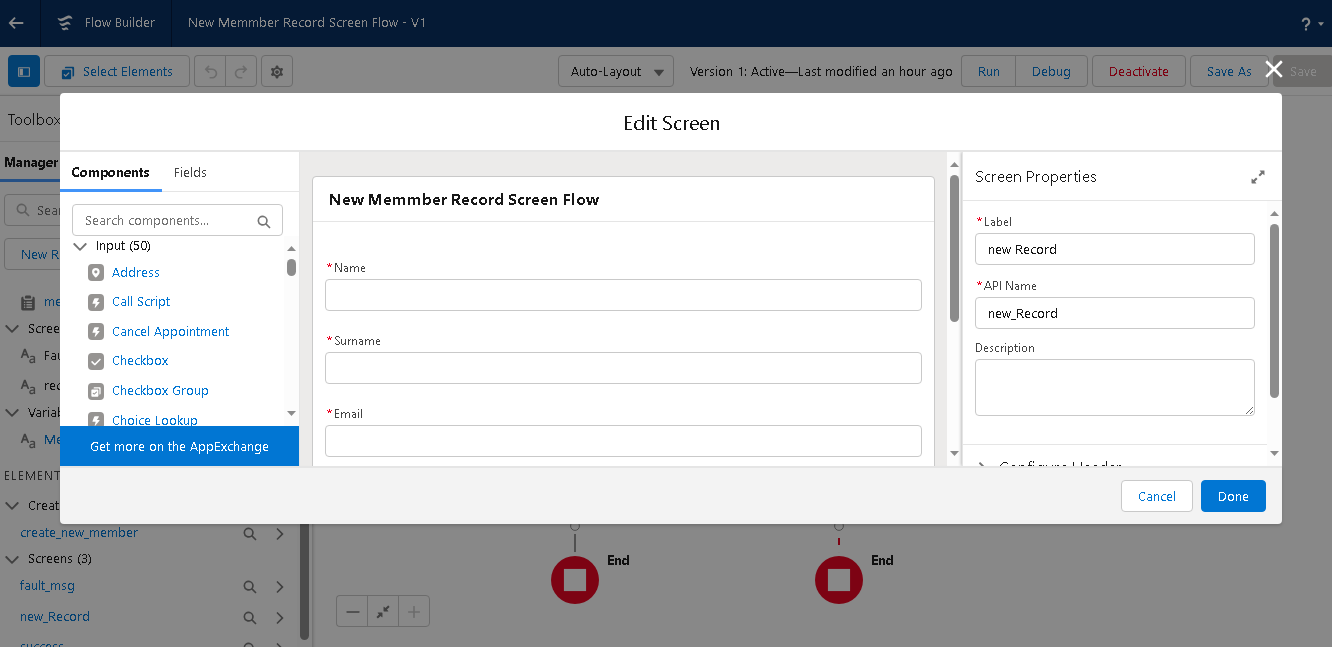
1

2-SCREEN ELEMENT

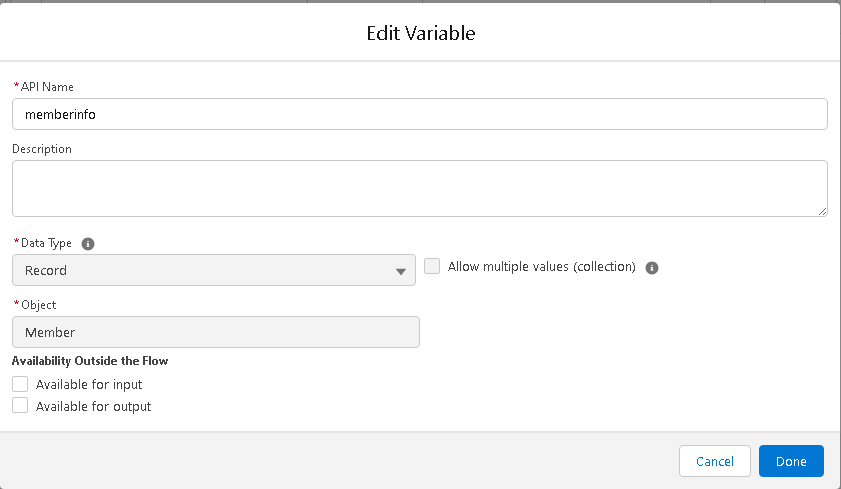
Fields kısmından Name, Surname, Email, Address ve Phone alanları seçilerek Screen ekaranına eklendi.

1-

Bu flowu oluştururken **‘Screen Flow’** template kullanıldı.



2



3-VARIABLE

memberinfo variable oluşturuldu.

**Data Type:** Record

**Object:** Member

3

4-CREATE RECORD ELEMENT

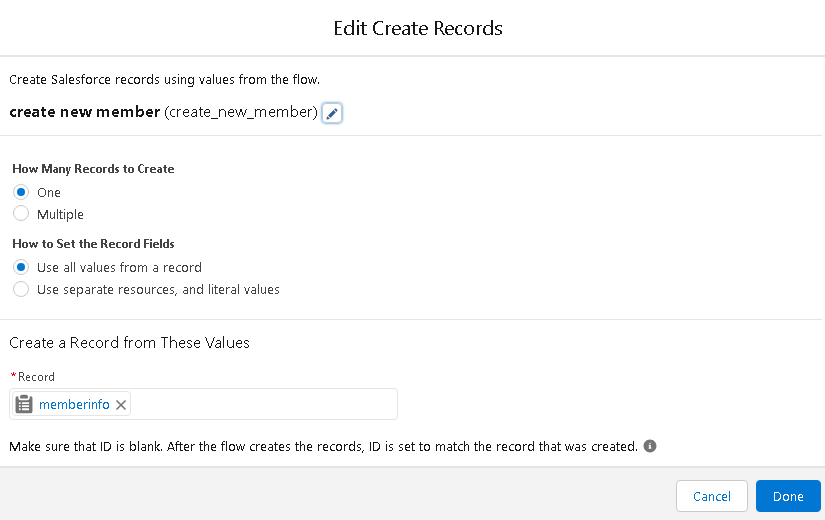
create new member oluşturuldu.

**How Many Records to Create:** One

**How to Set the Record Fields:** Use all values from record

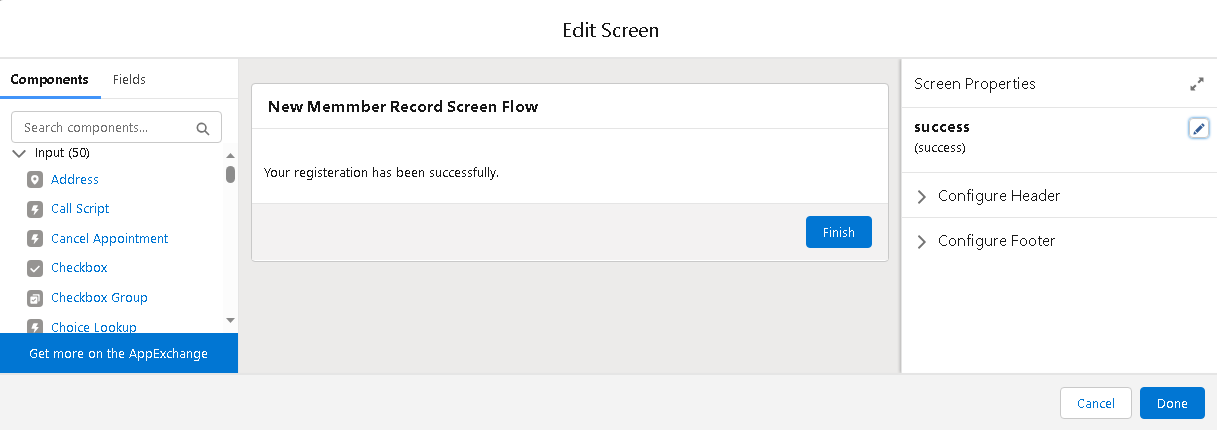
**Record:**memberinfo

4



5-SCREEN ELEMENT

Kaydın başarılı bir şekilde yapıldığını göstermek için Screen ekaranına **‘Your registration has been done successfully’** mesajı eklendi.



5

**APEX CODES (Developer Console)**

**APEX TRİGGER UpdateIdAsVolume**

/\*Bu tetikleyici, özel Books\_\_c nesnesi üzerinde gerçekleşen before insert ve before update olaylarını ele almak üzere tasarlanmıştır.

Bir yeni kayıt veya güncellenen kaydın Volume\_\_c alanı 1'den büyükse, bu tetikleyici devreye girer.

Her ekstra hacim için, orijinal ile aynı detaylara sahip yeni bir kayıt oluşturur, ancak hacim 1 olarak ayarlanır.

Tetikleyici aynı zamanda orijinal kaydın hacmini 1 olarak günceller ve çoğaltılmış kayıtları ekler.\*/

trigger UpdateIdAsVolume on Books\_\_c (before insert,before update) {

List<Books\_\_c> booksToUpdate = new List<Books\_\_c>();

for (Books\_\_c book : Trigger.new) {

if (book.Volume\_\_c > 1) {

// For new records with more than one volume, create new records with the same name

for (Integer i = 1; i < book.Volume\_\_c; i++) {

Books\_\_c updatedBook = new Books\_\_c(

Name\_\_c = book.Name\_\_c,

Author\_\_c = book.Author\_\_c, // Set the author from the original record

Price\_\_c = book.Price\_\_c,

Volume\_\_c = 1, // Set the volume to 1 for the duplicated records

Pages\_\_c=book.Pages\_\_c,

Genre\_\_c=book.Genre\_\_c,

Published\_By\_\_c=book.Published\_By\_\_c,

Language\_\_c=book.Language\_\_c

);

booksToUpdate.add(updatedBook);

System.debug(booksToUpdate);

}

}

}

// Update the volume of the original records

for (Books\_\_c book : Trigger.new) {

book.Volume\_\_c = 1;

}

// Insert the duplicated records

insert booksToUpdate;

}

**APEX TEST CLASS TestUpdateIdAsVolume**

@isTest

public class TestUpdateIdAsVolume {

@isTest

static void testTrigger() {

List<Author\_\_c> authors = new List<Author\_\_c>();

for (Integer i = 0; i < 5; i++) {

Author\_\_c author = new Author\_\_c(

Name\_\_c = 'Author ' + i,

Surname\_\_c = 'Surname ' + i

);

authors.add(author);

}

insert authors;

// Create a test record

Books\_\_c testBook = new Books\_\_c(

Name\_\_c = 'Test Book',

Author\_\_c = authors[0].id,

Price\_\_c = 29.99,

Volume\_\_c = 3,

Pages\_\_c = 200,

Genre\_\_c = 'Roman',

Language\_\_c = 'English'

);

// Insert the test record

insert testBook;

// Check if additional records were created

List<Books\_\_c> duplicatedBooks = [SELECT Name\_\_c, Author\_\_c,

Price\_\_c, Volume\_\_c, Pages\_\_c, Genre\_\_c, Published\_By\_\_c, Language\_\_c

FROM Books\_\_c WHERE Name\_\_c = 'Test Book'

AND Volume\_\_c = 1];

System.assertEquals(3, duplicatedBooks.size(), 'Two additional records should have been created');

// Update the test record's volume

testBook.Volume\_\_c = 2;

update testBook;

// Check if additional records were created after update

duplicatedBooks = [SELECT Name\_\_c, Author\_\_c, Price\_\_c,

Volume\_\_c, Pages\_\_c, Genre\_\_c, Published\_By\_\_c, Language\_\_c

FROM Books\_\_c WHERE Name\_\_c = 'Test Book' AND

Volume\_\_c = 1];

System.assertEquals(4, duplicatedBooks.size(), 'One additional record should have been created after update');

}

}

**APEX TRİGGER UpdateMemberLastBookReadDate**

/\*Bu tetikleyici, Book\_Tracking\_\_c özel nesnesi üzerinde gerçekleşen before insert ve before update olaylarını ele almak üzere tasarlanmıştır.

Tetikleyici, üyelerin en son kitap satın alma tarihini güncellemekten sorumludur.

Yeni bir Book\_Tracking\_\_c kaydı eklenirken veya güncellenirken, ilgili üyenin en son satın alma tarihi kontrol edilir.

Her üye için, ilgili tüm Book\_Tracking\_\_c kayıtları arasında en son Issue\_Date\_\_c bulunur.

Bu tarih, üyenin Last\_Book\_Read\_Date\_\_c alanına yazılır.\*/

trigger UpdateMemberLastBookReadDate on Book\_Tracking\_\_c (before insert, before update) {

// Collect the Member\_\_c record Ids for all inserted Book\_Tracking\_\_c records

Set<Id> memberIds = new Set<Id>();

for (Book\_Tracking\_\_c bookTracking : Trigger.new) {

if (bookTracking.Borrowed\_By\_\_c != null) {

memberIds.add(bookTracking.Borrowed\_By\_\_c);

}

}

// Query the latest Issue\_Date\_\_c for each Member\_\_c record

Map<Id, Date> memberLastBookReadDates = new Map<Id, Date>();

for (AggregateResult result : [

SELECT Borrowed\_By\_\_c, MAX(Issue\_Date\_\_c) maxIssueDate

FROM Book\_Tracking\_\_c

WHERE Borrowed\_By\_\_c IN :memberIds

GROUP BY Borrowed\_By\_\_c

]) {

Id memberId = (Id)result.get('Borrowed\_By\_\_c');

Date maxIssueDate = (Date)result.get('maxIssueDate');

memberLastBookPurchaseDates.put(memberId, maxIssueDate);

}

// Update the Member\_\_c records with the latest Issue\_Date\_\_c

List<Member\_\_c> membersToUpdate = new List<Member\_\_c>();

for (Id memberId : memberIds) {

if (memberLastBookPurchaseDates.containsKey(memberId)) {

Member\_\_c member = new Member\_\_c(Id = memberId);

member.Last\_Book\_Read\_Date\_\_c =

memberLastBookPurchaseDates.get(memberId);

membersToUpdate.add(member);

}

}

// Perform the updates

update membersToUpdate;

}

**APEX TEST CLASS TestUpdateMemberLastBookReadDate**

@isTest

public class TestUpdateMemberLastBookReadDate {

@isTest

static void testUpdateLastBookPurchaseDate() {

// Create a test Member\_\_c record

Member\_\_c testMember = new Member\_\_c(Name\_\_c = 'mehmet',

Surname\_\_c = 'gel', Email\_\_c = 'mhm@gl.tr');

insert testMember;

// Create a test Author\_\_c record

Author\_\_c testAuthor = new Author\_\_c();

insert testAuthor;

// Create a test Book\_\_c record

List<Books\_\_c> books = new List<Books\_\_c>();

books.add(new Books\_\_c(Name\_\_c = 'Book1', Author\_\_c =

testAuthor.Id, Volume\_\_c = 20, Price\_\_c = 50));

books.add(new Books\_\_c(Name\_\_c = 'Book2', Author\_\_c =

testAuthor.Id, Volume\_\_c = 20, Price\_\_c = 50));

books.add(new Books\_\_c(Name\_\_c = 'Book3', Author\_\_c =

testAuthor.Id, Volume\_\_c = 20, Price\_\_c = 50));

insert books;

// Create test Book\_Tracking\_\_c records associated with the test

Member\_\_c record

Book\_Tracking\_\_c book1 = new Book\_Tracking\_\_c(Borrowed\_By\_\_c =

testMember.Id, Book\_Name\_\_c= books[0].id,

Issue\_Date\_\_c =

Date.newInstance(2023,12,1), Status\_\_c='Issued');

insert book1;

// Introduce a delay (e.g., 2 seconds) before inserting the second

Book\_Tracking\_\_c record

Test.startTest();

Long delayMillis = 2000; // 2 seconds

Long startTime = System.now().getTime();

while (System.now().getTime() < startTime + delayMillis) {

// Wait for the specified delay

}

Book\_Tracking\_\_c book2 = new Book\_Tracking\_\_c(Borrowed\_By\_\_c =

testMember.Id, Book\_Name\_\_c= books[1].id,

Issue\_Date\_\_c =

Date.newInstance(2023,12,15),Status\_\_c='Issued');

insert book2;

// Introduce a delay (e.g., 2 seconds) before inserting the second

Book\_Tracking\_\_c record

Long delayMilliss = 2000; // 2 seconds

Long startTimee = System.now().getTime();

while (System.now().getTime() < startTime + delayMillis) {

// Wait for the specified delay

}

Test.stopTest();

Book\_Tracking\_\_c book3 = new Book\_Tracking\_\_c(Borrowed\_By\_\_c =

testMember.Id,

Book\_Name\_\_c= books[2].id,

Issue\_Date\_\_c = Date.newInstance(2023,12,30),

Status\_\_c='Issued');

insert book3;

// Trigger should have updated the Last\_Book\_Read\_Date\_\_c on the

Member\_\_c record

testMember = [SELECT Id, Last\_Book\_Read\_Date\_\_c FROM

Member\_\_c WHERE Id = :testMember.Id];

Date expectedLastBookReadDate = Date.newInstance(2023,12,15);

System.assertEquals(expectedLastBookReadDate,

testMember.Last\_Book\_Read\_Date\_\_c, 'Last\_Book\_Read\_Date\_\_c should be updated');

}

}

**APEX CLASS LibraryOperations**

/\*

Bu Class, kütüphane işlemleri için çeşitli metotlar içerir.

Bu metotlar ziyaretçi üyelik kontrolü yapmak ve yanlış yazılmış yazar adını düzeltmek gibi işlemleri gerçekleştirir.

\*/

public class LibraryOperations {

// Verilen ad ve soyad ile bir ziyaretçinin üyeliğini kontrol eder

public static void **checkVisitorMembership**(String firstName, String lastName) {

List<Member\_\_c> members = [SELECT Id, Name FROM Member\_\_c

WHERE Name\_\_c = :firstName AND Surname\_\_c = :lastName];

if (!members.isEmpty()) {

List<Map<String, String>> booksRead = new List<Map<String,String>>();

List<Book\_Tracking\_\_c> bookTrackings = [SELECT Id, Book\_Name\_\_r.Name\_\_c, Book\_Name\_\_r.Author\_\_r.Name\_\_c, Book\_Name\_\_r.Author\_\_r.Surname\_\_c, Borrowed\_By\_\_r.Name\_\_c,Borrowed\_By\_\_r.Surname\_\_c FROM Book\_Tracking\_\_c

WHERE Borrowed\_By\_\_c = :members];

for (Book\_Tracking\_\_c bookTracking : bookTrackings) {

Map<String, String> bookInfo = new Map<String, String>();

bookInfo.put('Kitap Adı', bookTracking.Book\_Name\_\_r.Name\_\_c);

bookInfo.put('Yazar Adı ve Soyadı', bookTracking.Book\_Name\_\_r.Author\_\_r.Name\_\_c +' '+

bookTracking.Book\_Name\_\_r.Author\_\_r.Surname\_\_c);

booksRead.add(bookInfo);

}

System.debug(firstName + ' ' + lastName + ' kütüphane üyesidir.');

System.debug('Üye tarafından okunan kitaplar:'+booksRead);

} else {

System.debug(firstName + ' ' + lastName + ' kütüphane üyesi değildir.');

}

}

// Yanlış yazılmış yazar adını düzeltir

public static void **correctAuthorName**(String incorrectName, String correctName) {

List<Author\_\_c> authorsToUpdate = [SELECT Id, Name\_\_c FROM Author\_\_c

WHERE Name\_\_c = :incorrectName];

for (Author\_\_c author : authorsToUpdate) {

author.Name\_\_c = correctName;

}

update authorsToUpdate;

}

}

**APEX TEST CLASS TestLibraryOperations**

@isTest

public class TestLibraryOperations{

@isTest

static void testCheckVisitorMembership() {

// Create test data: Member\_\_c and Author\_\_c records

Member\_\_c testMember = new Member\_\_c(Name\_\_c = 'John',Surname\_\_c = 'Doe', Email\_\_c = 'john.doe@example.com');

insert testMember;

system.debug(testMember);

Author\_\_c testAuthor = createTestAuthor();

insert testAuthor;

system.debug(testAuthor);

// Create a test Book\_\_c record

Books\_\_c testBook = new Books\_\_c(

Name\_\_c = 'Test Book',

Author\_\_c = testAuthor.Id,

Price\_\_c = 50.0,

Volume\_\_c = 15

);

insert testBook;

system.debug(testBook);

// Create a test Book\_Tracking\_\_c record associated with the test Book\_\_c record

Book\_Tracking\_\_c testBookTracking = new Book\_Tracking\_\_c(

Book\_Name\_\_c = testBook.Id,

Borrowed\_By\_\_c = testMember.Id,

Status\_\_c='Issued',

Issue\_Date\_\_c= date.today()

);

insert testBookTracking;

system.debug(testBookTracking);

// Test the checkVisitorMembership method

Test.startTest();

System.debug('Test Case 1: Existing Member');

LibraryOperations.checkVisitorMembership('John', 'Doe');

System.debug('Test Case 2: Non-existing Member');

LibraryOperations.checkVisitorMembership('Jane', 'Smith');

Test.stopTest();

}

@isTest

static void testCorrectAuthorName() {

// Create test data: Author\_\_c record

Author\_\_c testAuthor = createTestAuthor();

insert testAuthor;

// Test the correctAuthorName method

Test.startTest();

System.debug('Test Case 1: Correct the author name');

LibraryOperations.correctAuthorName(testAuthor.Name\_\_c, 'CorrectedAuthor');

Test.stopTest();

// Verify that the author's name is updated correctly

Author\_\_c updatedAuthor = [SELECT Id, Name\_\_c FROM Author\_\_c WHERE Id = :testAuthor.Id];

System.assertEquals('Corrected Author', updatedAuthor.Name\_\_c);

}

// Helper method to create a test Author\_\_c record

static Author\_\_c createTestAuthor() {

return new Author\_\_c(Name\_\_c = 'Test Author');

}

}

**APEX CLASS CapitalizeMemberNamesBatch AND CapitalizeMemberNamesScheduler**

/\*

\* **CapitalizeMemberNamesBatch Class**

Bu Class, üye isimlerini büyük harfle düzenlemek için bir toplu işlem gerçekleştirir.

Veritabanı kayıtlarını toplu olarak alır, isim ve soyisim alanlarını büyük harfe çevirir ve günceller.

\*/

public class CapitalizeMemberNamesBatch implements

Database.Batchable<sObject> {

// Adım 1: İşlem başladığında yürütülen sorgu belirlenir

public Database.QueryLocator start(Database.BatchableContext context) {

String query = 'SELECT Id, Name\_\_c, Surname\_\_c FROM Member\_\_c';

return Database.getQueryLocator(query);

}

// Adım 2: Belirtilen kapsam içindeki üye isimlerini büyük harfe çevirir ve günceller

public void execute(Database.BatchableContext context,

List<Member\_\_c> scope) {

for (Member\_\_c member : scope) {

member.Name\_\_c = member.Name\_\_c.toUpperCase();

member.Surname\_\_c = member.Surname\_\_c.toUpperCase();

}

update scope; // Güncelleme işlemi

}

// Adım 3: İşlem tamamlandığında çalışır, isteğe bağlı olarak son işlemler yapılabilir

public void finish(Database.BatchableContext context) {

// İsteğe bağlı: Gerekirse herhangi bir son işlem mantığı ekleyebilirsiniz.

}

}

/\*

\* **CapitalizeMemberNamesScheduler Class**

Bu Class, CapitalizeMemberNamesBatch sınıfını bir zamanlayıcı ile çalıştırmak için tasarlanmıştır.

Zamanlayıcı tetiklemesi gerçekleştiğinde, toplu işlemi başlatır ve üye isimlerini büyük harfe dönüştürür.

\*/

public class CapitalizeMemberNamesScheduler implements Schedulable {

// Adım 1: Zamanlayıcı tetiklemesi gerçekleştiğinde yürütülen metot

public void execute(SchedulableContext context) {

// CapitalizeMemberNamesBatch sınıfından bir örnek oluştur

CapitalizeMemberNamesBatch batchJob = new CapitalizeMemberNamesBatch();

// Toplu işlemi yürüt

Database.executeBatch(batchJob);

}

}

**APEX TEST CLASS TestCapitalizeMemberNamesBatch**

@isTest

public class TestCapitalizeMemberNamesBatch {

@isTest

static void test\_BatchJob() {

// Create test data

List<Member\_\_c> testMembers = new List<Member\_\_c>();

for (Integer i = 0; i < 200; i++) {

testMembers.add(new Member\_\_c(Name\_\_c = 'Name' + i, Surname\_\_c = 'Surname' + i, Email\_\_c='test@test.tr' +i));

}

insert testMembers;

Test.startTest();

// Enqueue the batch job

CapitalizeMemberNamesBatch batchJob = new CapitalizeMemberNamesBatch();

Database.executeBatch(batchJob);

Test.stopTest();

// Retrieve the updated records and assert the changes

List<Member\_\_c> updatedMembers = [SELECT Id, Name\_\_c, Surname\_\_c FROM Member\_\_c WHERE Id IN :testMembers];

for (Member\_\_c member : updatedMembers) {

// Assert that names have been capitalized

System.assertEquals(member.Name\_\_c, member.Name\_\_c.toUpperCase());

System.assertEquals(member.Surname\_\_c, member.Surname\_\_c.toUpperCase());

}}}

**APEX TEST CLASS TestCapitalizeMemberNamesScheduler**

@isTest

public class TestCapitalizeMemberNamesScheduler {

@isTest

static void test\_Scheduler() {

// Create test data

List<Member\_\_c> testMembers = new List<Member\_\_c>();

for (Integer i = 0; i < 200; i++) {

testMembers.add(new Member\_\_c(Name\_\_c = 'Name' + i, Surname\_\_c = 'Surname' + i, Email\_\_c='test@test.tr'+i));

}

insert testMembers;

system.debug(testMembers);

String cronExp = '0 0 0 \* \* ?'; // Set the cron expression to a date in thefuture

Test.startTest();

CapitalizeMemberNamesScheduler capName= new CapitalizeMemberNamesScheduler();

system.schedule('CapitalizeMemberNamesScheduler', cronExp, capName);

Test.stopTest();

system.debug(capName);

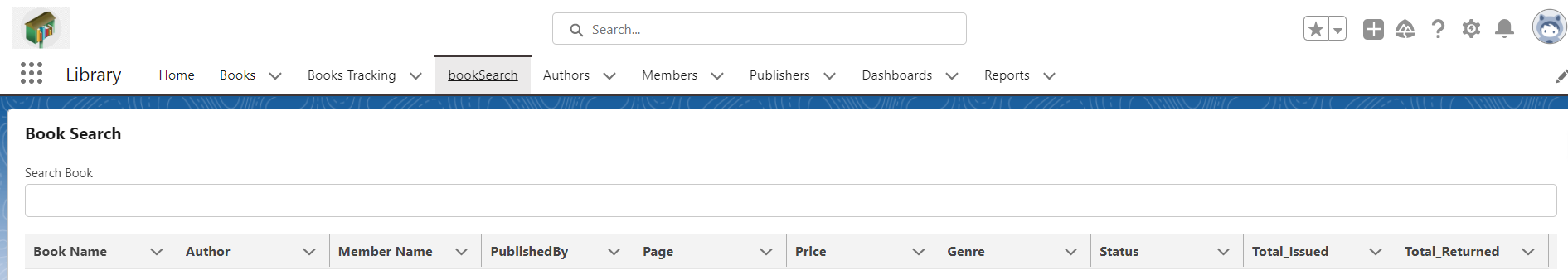
}

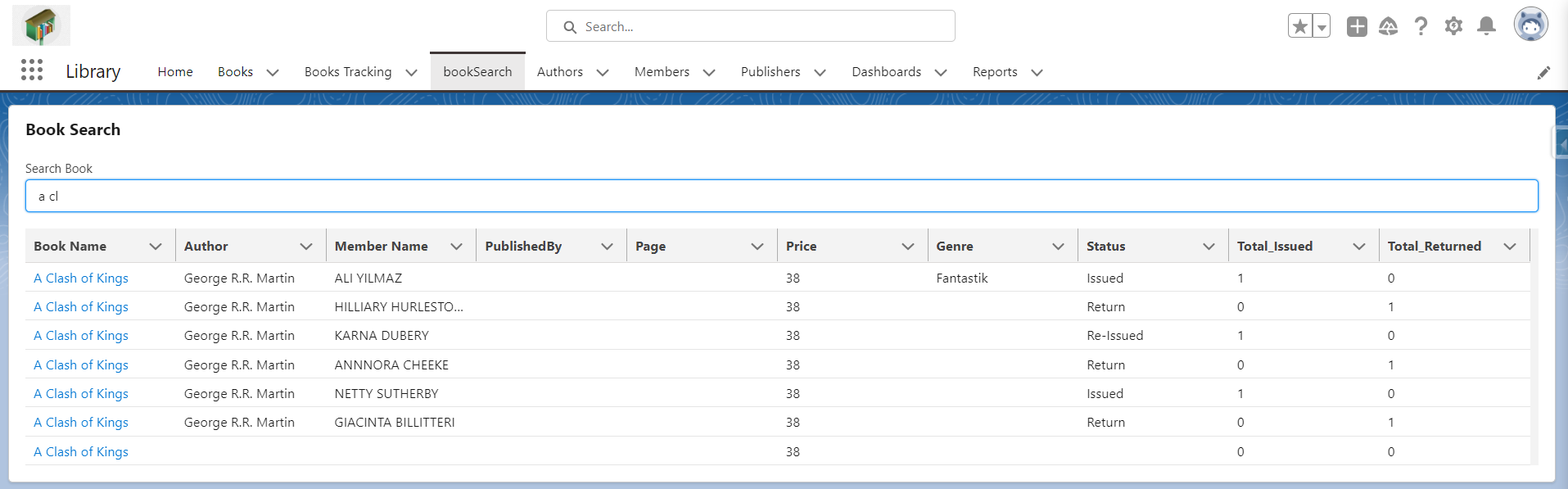
}

**BOOK SEARCH LİGHTNİNG WEB COMPENENT (LWC)**

**Salesforce Lightning Web Components (LWC),** Salesforce platformunda kullanılmak üzere özel olarak geliştirilen modern ve hafif web bileşenlerini tanımlayan bir geliştirme çerçevesidir. LWC, Salesforce uygulamalarını daha hızlı, daha kullanıcı dostu ve daha özelleştirilebilir hale getirmek için tasarlanmıştır.

Burada kulanıcı için rahat ulaşabileceği ,tab olarak bir LWC compenent ile bir ‘Search’ olanağı sunuldu. Ayrıca BookName için link özelliği ile ilgili kitap detaylarına rahatça ulaşacaktır.

****

****

**BookSearch LWC ; Salesforce Developer Console Ve Visual Studio Code (VsCode) kullanılarak hazırlanmıştır. Vscode içerisinde**

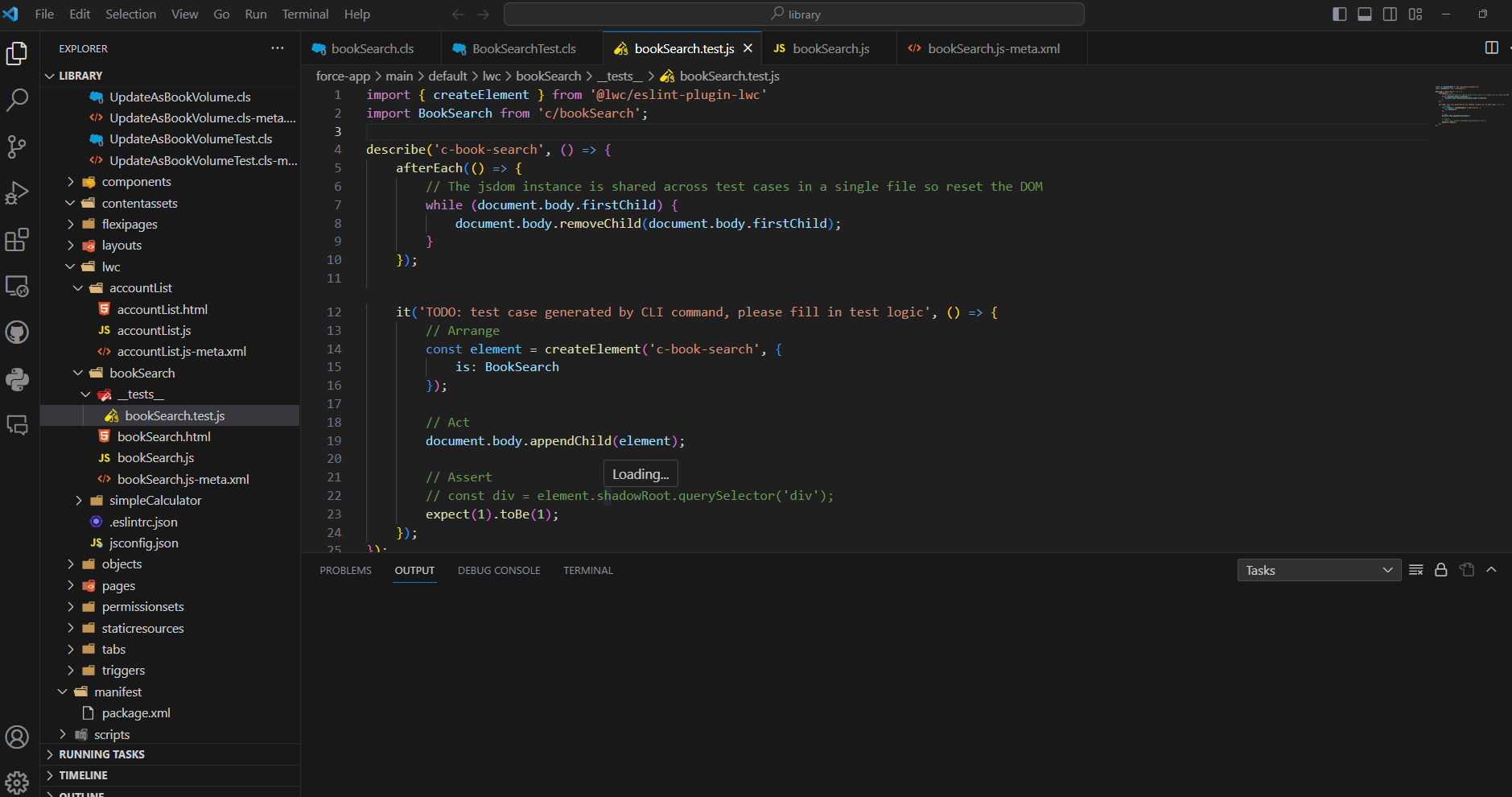
**SFDX:Create Lightning Web Component,**

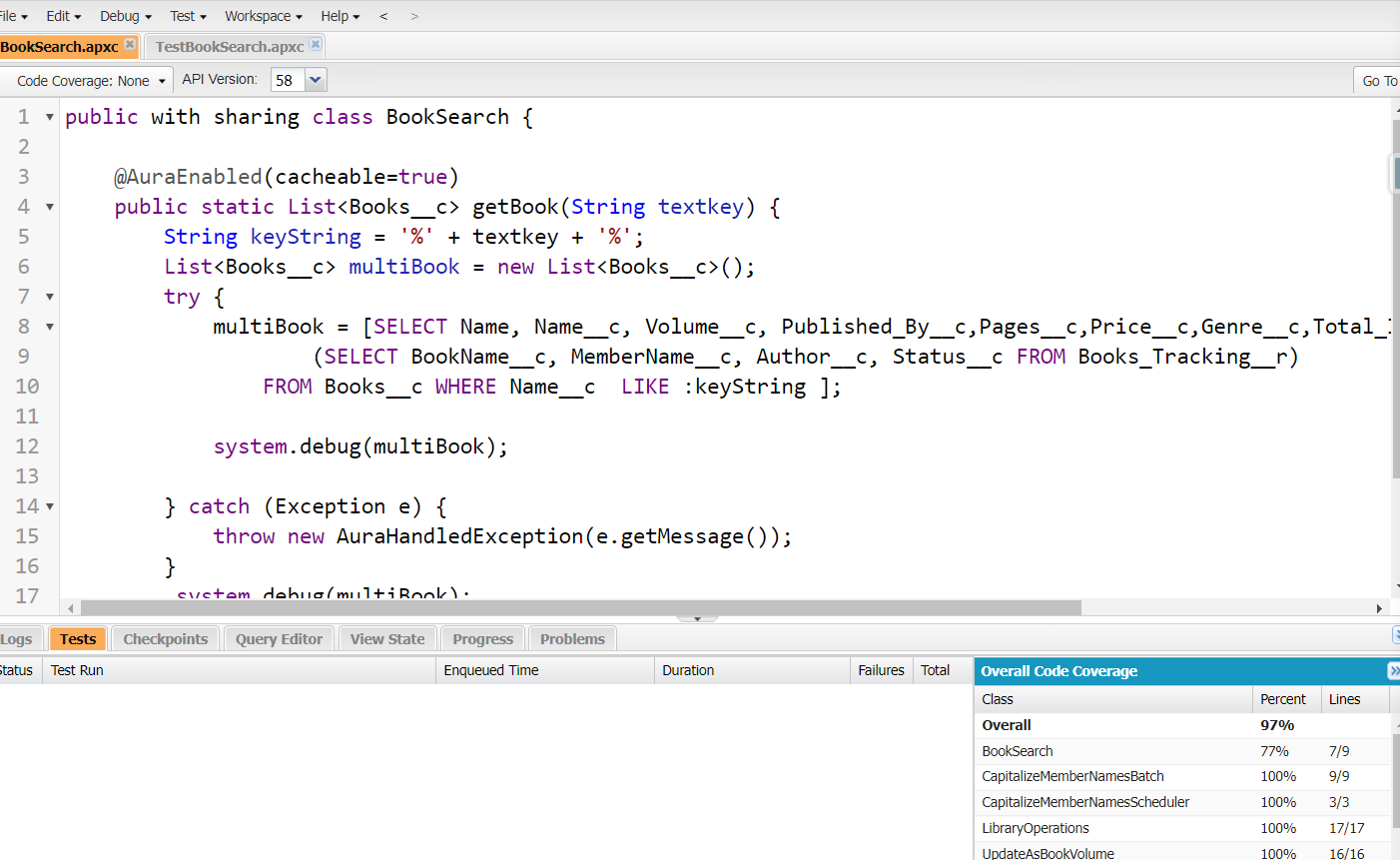
**SFDX: Authorize an Org**

**SFDX: Deploy Source the Org ,**

**SFDX:Retrieve Source in Manifest from Org**

**Yapıları kullanılarak Vscode to Org , Org to Vscode karşılıklı olarak ilişki kurulmuştur.**

****

****

**BookSearch LWC Component yapılandırılırken aşağıdaki kod yapıları oluşturulmuştur.**

**APEX CLASS BookSearch (Developer Console)**

**getBook** adlı bir Aura Enabled yöntemi tanımlar. Bu yöntem, kullanıcı tarafından sağlanan arama metni ile eşleşen kitapları veri tabanından getirir ve bir liste olarak döndürür.

**textkey** parametresi, arama için kullanılacak metin anahtarını alır.

**LIKE :keyString** ifadesi, veri tabanında metin anahtarı ile eşleşen kitapları sorgular.

Veri tabanından çekilen kitap verileri, hata ayıklama amacıyla system.debug ile konsola yazdırılır.

Hata durumunda, AuraHandledException kullanılarak bir hata mesajı yakalanır ve döndürülür.

Bu Apex Class, Salesforce LWC bileşenlerinin kullanabileceği kitap arama işlevselliğini sağlar.

public with sharing class BookSearch {

/\*\*

\* @AuraEnabled(cacheable=true)

\* Bu yöntem, kitap araması yapmak için çağrılır ve veritabanından kitap verilerini getirir.

\* @param textkey Arama için kullanılacak metin anahtarı

\* @return Kitap verilerini içeren bir Liste

\*/

public static List<Books\_\_c> getBook(String textkey) {

// Arama metni ile eşleşen kitapları getirmek için kullanılacak metin anahtarı oluşturuluyor.

String keyString = '%' + textkey + '%';

List<Books\_\_c> multiBook = new List<Books\_\_c>();

try {

// Veritabanından kitapları çekme sorgusu yapılıyor.

multiBook = [SELECT Name, Name\_\_c, Volume\_\_c, Published\_By\_\_c, Pages\_\_c, Price\_\_c, Genre\_\_c, Total\_Issued\_\_c, Total\_Returned\_\_c,

(SELECT BookName\_\_c, MemberName\_\_c, Author\_\_c, Status\_\_c FROM Books\_Tracking\_\_r)

FROM Books\_\_c WHERE Name\_\_c LIKE :keyString ];

// Elde edilen kitap verileri hata ayıklama amacıyla konsola yazdırılıyor.

system.debug(multiBook);

} catch (Exception e) {

// Hata durumunda hata mesajı yakalanıp bir AuraHandledException ile döndürülüyor.

throw new AuraHandledException(e.getMessage());

}

// Kitap verileri döndürülüyor.

return multiBook;

}

}

**APEX TEST CLASS TestBookSearch (Developer Console)**

@isTest

public class TestBookSearch{

@testSetup

static void setupTestData() {

// Create test data for Authors\_\_c (assuming this is the object where Authors are stored)

List<Author\_\_c> authors = new List<Author\_\_c>();

for (Integer i = 0; i < 5; i++) {

Author\_\_c author = new Author\_\_c(

Name\_\_c = 'Author ' + i,

Surname\_\_c = 'Test ' + i

);

authors.add(author);

}

insert authors;

// Create test data for Books\_\_c

List<Books\_\_c> books = new List<Books\_\_c>();

for (Integer i = 0; i < 5; i++) {

Books\_\_c book = new Books\_\_c(

Name\_\_c = 'Book ' + i + ' Name',

Volume\_\_c = 1,

Author\_\_c = authors[i].Id, // Use the Id of the corresponding Author

Price\_\_c = 120.00,

Pages\_\_c=245,

Genre\_\_c='Roman'

);

books.add(book);

}

insert books;

// Create test data for Borrowers\_\_c (assuming this is the object where Borrowers are stored)

List<Member\_\_c> borrowers = new List<Member\_\_c>();

for (Integer i = 0; i < 5; i++) {

Member\_\_c borrower = new Member\_\_c(

Name\_\_c = 'Borrower ' + i,

Surname\_\_c = 'Test ' + i,

Email\_\_c= 'hy@hy.hy'+i

);

borrowers.add(borrower);

}

insert borrowers;

// Create related Book\_Tracking\_\_c records

List<Book\_Tracking\_\_c> trackingRecords = new List<Book\_Tracking\_\_c>();

for (Books\_\_c book : books) {

for (Integer j = 0; j < 2; j++) {

Book\_Tracking\_\_c tracking = new Book\_Tracking\_\_c(

Book\_Name\_\_c = book.Id,

Borrowed\_By\_\_c = borrowers[j].Id, // Use the Id of the corresponding Borrower

Status\_\_c = 'Issued',

Issue\_Date\_\_c=date.today()

);

trackingRecords.add(tracking);

}

}

insert trackingRecords;

}

@isTest

static void testGetBook() {

// Test querying books based on search key

Test.startTest();

List<Books\_\_c> result = BookSearch.getBook('Book 1');

Test.stopTest();

// Verify results

System.assertEquals(1, result.size());

System.assertEquals('Book 1 Name', result[0].Name\_\_c);

System.assertEquals(1.0, result[0].Volume\_\_c); // Volume\_\_c is of type Decimal

System.assertEquals(2, result[0].Books\_Tracking\_\_r.size());

}

@isTest

static void testExceptionHandling() {

Test.startTest();

try {

List<Books\_\_c> result = BookSearch.getBook(null);

} catch (AuraHandledException e) {

System.assert(e.getMessage() != null);

}

Test.stopTest();

}

}

**bookSearch.js-meta.xml (VsCode)**

Bu XML dosyası, bir Salesforce Lightning bileşeninin özelliklerini ve kullanımını tanımlar:

* <apiVersion>: Bu etiket, bileşenin hangi Salesforce Metadata API sürümüyle uyumlu olduğunu belirtir. Bu sürüm, bileşenin yapılandırma ve özelliklerini destekler.
* <isExposed>: Bu etiket, bileşenin dışa açık (kullanılabilir) olup olmadığını belirtir. "true" olarak ayarlandığında, bileşen Salesforce içinde kullanılabilir ve dağıtılabilir hale gelir.
* <targets>: Bu etiket, bileşenin hangi hedeflerde (kullanım senaryolarında) kullanılabileceğini belirtir. Örneğin, lightning App Page hedefi, bileşenin uygulama sayfalarında kullanılabileceğini gösterir. Bu hedefler, bileşenin nerede ve nasıl kullanılabileceğini kontrol etmek için kullanılır.

Bu XML dosyası, bir Salesforce Lightning bileşeninin tanımını içerir ve bu bileşenin ne tür uygulamalarda ve sayfalarda kullanılabileceğini belirler. Bileşen paketlemesi ve dağıtımı sırasında bu tür bir tanımlama, bileşenin doğru şekilde entegre edilmesini ve kullanılmasını sağlar.

<?xml version="1.0" encoding="UTF-8"?>

<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">

<!-- Salesforce Metadata API sürümü -->

<apiVersion>57.0</apiVersion>

<!-- Bileşen paketinin dışa açık (kullanılabilir) olup olmadığını belirtir -->

<isExposed>true</isExposed>

<!-- Bileşenin hedeflerini (kullanım senaryolarını) belirtir -->

<targets>

<!-- Uygulama sayfası (App Page) hedefi -->

<target>lightning\_\_AppPage</target>

<!-- Ana sayfa (Home Page) hedefi -->

<target>lightning\_\_HomePage</target>

<!-- Kayıt sayfası (Record Page) hedefi -->

<target>lightning\_\_RecordPage</target>

<!-- Sekme (Tab) hedefi -->

<target>lightning\_\_Tab</target>

</targets>

</LightningComponentBundle>

**bookSearch.html (VsCode)**

Bu kod bloğu, Salesforce Lightning Web Components (LWC) kullanarak bir kitap arama bileşeninin kullanıcı arayüzünü tanımlar. Bileşen, kullanıcıların bir kitap adıyla kitapları arayabileceği ve sonuçları bir tablo şeklinde görebileceği bir kullanıcı arayüzü sağlar

Bu kullanıcı arayüzü şunları içerir:

**lightning-card:** Bu bileşen, bir kart görünümü sağlar ve "Book Search" başlığını taşır. Bunu, kullanıcı arayüzünün ana bileşeni olarak kullanır.

**lightning-input:** "Search Book" etiketli bir arama girdisi içerir. Kullanıcılar buraya kitap adını girebilirler. onchange olayı, kullanıcının girdiyi değiştirdiğinde çağrılacak handleSearch işlevini tetikler. value özelliği, girdinin değerini searchText değişkenine bağlar.

**lightning-datatable:** Bu bileşen, kitap sonuçlarını görüntülemek için bir tablo sağlar. key-field, verilerin anahtar alanını belirtir. data, kitap sonuçlarını bookList değişkeninden alır. columns, tablonun sütun yapılandırmasını columns değişkeninden alır. hide-checkbox-column özelliği, tabloya varsayılan olarak eklenen seçim kutularını gizler. is-loading, veriler yüklenirken bir yükleniyor göstergesi göstermek için isNoRecordFound durumunu izler.

**template if:true={isNoRecordFound}**: Bu şablon, isNoRecordFound değişkeni true olduğunda, yani kayıt bulunamadığında görünen bir mesajı içerir. Bu, kullanıcılara hiçbir kaydın bulunamadığını bildirir.

Bu kod bloğu, kullanıcıların kitap aramalarını yapmalarını ve sonuçları kullanıcı dostu bir şekilde görüntülemelerini sağlayan bir arayüzü tanımlar.

<template>

<lightning-card title="Book Search">

<!-- Kitap arama kartı başlığı -->

<div class="slds-m-around\_medium">

<!-- Arama girdisi ve olay dinleyici -->

<lightning-input

label="Search Book"

onchange={handleSearch}

value={searchText}

></lightning-input>

</div>

<div class="slds-m-around\_medium">

<!-- Sonuçları görüntülemek için veri tablosu -->

<lightning-datatable

key-field="Id"

data={bookList}

columns={columns}

hide-checkbox-column="true"

is-loading={isNoRecordFound}

></lightning-datatable>

<!-- Kayıt bulunamadı durumunu kontrol etmek için şablon -->

<template if:true={isNoRecordFound}>

<p>No records found.</p>

</template>

</div>

</lightning-card>

</template>

**bookSearch.js (VsCode)**

Bu kod, Salesforce Lightning Web Components kullanarak bir kitap arama bileşeni oluşturur ve kullanıcıların aradıkları kitapları bulmalarını sağlar. Bileşen, bir arama metni girdisi ve sonuçları bir tablo şeklinde görüntülemek için kullanılır. @api ve @track özellikleri, bileşenin dışarıdan alınan verilere ve bileşen içindeki duruma erişimini sağlar. Apex yöntemi olan getBook ise arama sorgusunu yürüten ve sonuçları getiren bir sunucu tarafı işlemi yapar.

import { LightningElement, track, api } from 'lwc';

import getBook from '@salesforce/apex/BookSearch.getBook';

const columns = [

    { label: 'Book Name', fieldName: 'BookURL', type: 'url', typeAttributes: { label: { fieldName: 'Name\_\_c' } } },

    { label: 'Author', fieldName: 'Author\_\_c', type: 'text' },

    { label: 'Member Name', fieldName: 'MemberName\_\_c', type: 'text' },

    { label: 'PublishedBy', fieldName: 'Published\_By\_\_c', type: 'text' },

    { label: 'Page', fieldName: 'Pages\_\_c', type: 'text' },

    { label: 'Price', fieldName: 'Price\_\_c', type: 'text' },

    { label: 'Genre', fieldName: 'Genre\_\_c', type: 'text' },

    { label: 'Status', fieldName: 'Status\_\_c', type: 'text' },

    { label: 'Total\_Issued', fieldName: 'Total\_Issued\_\_c', type: 'text' },

    { label: 'Total\_Returned', fieldName: 'Total\_Returned\_\_c', type: 'text' }

];

export default class BookSearch extends LightningElement {

@track searchText = ''; // Kullanıcının arama metnini tutar

@track bookList = []; // Kitap sonuçlarını saklar

@track isNoRecordFound = false; // Kayıt bulunamadı durumunu izler

columns = columns; // Sütun tanımlarını bileşen içinde kullanmak için atama yapar

@api title; // Bileşen başlığını dışarıdan alır

// Kullanıcının arama girdisini dinleyen işleyici

handleSearch(event) {

this.searchText = event.target.value;

this.loadBooks(); // Kitapları yükleme işlemini başlatır

}

// Kitapları yüklemek için Apex yöntemini çağırır

loadBooks() {

getBook({ textkey: this.searchText }) // Apex yöntemine arama metnini ileterek çağırır

.then(result => {

if (result && result.length > 0) {

// Sonuçlar bulunduysa

this.bookList = result.map(book => {

const { Name\_\_c, Published\_By\_\_c, Pages\_\_c, Price\_\_c, Genre\_\_c, Total\_Issued\_\_c, Total\_Returned\_\_c, Books\_Tracking\_\_r } = book;

if (!Books\_Tracking\_\_r || Books\_Tracking\_\_r.length === 0) {

return {

Name\_\_c,

Published\_By\_\_c,

Pages\_\_c,

Price\_\_c,

Genre\_\_c,

Total\_Issued\_\_c,

Total\_Returned\_\_c,

MemberName\_\_c: null,

Author\_\_c: null,

Status\_\_c: null,

BookURL: '/lightning/r/Books\_c/' + book.Id + '/view'

};

} else {

return {

Name\_\_c,

Published\_By\_\_c,

Pages\_\_c,

Price\_\_c,

Genre\_\_c,

Total\_Issued\_\_c,

Total\_Returned\_\_c,

...Books\_Tracking\_\_r[0],

BookURL: '/lightning/r/Books\_c/' + book.Id + '/view'

};

}

});

this.isNoRecordFound = false; // Kayıt bulunamadı durumunu kapatır

} else {

// Sonuç yoksa

this.bookList = []; // Sonuçları temizler

this.isNoRecordFound = true; // Kayıt bulunamadı durumunu açar

}

})

.catch(error => {

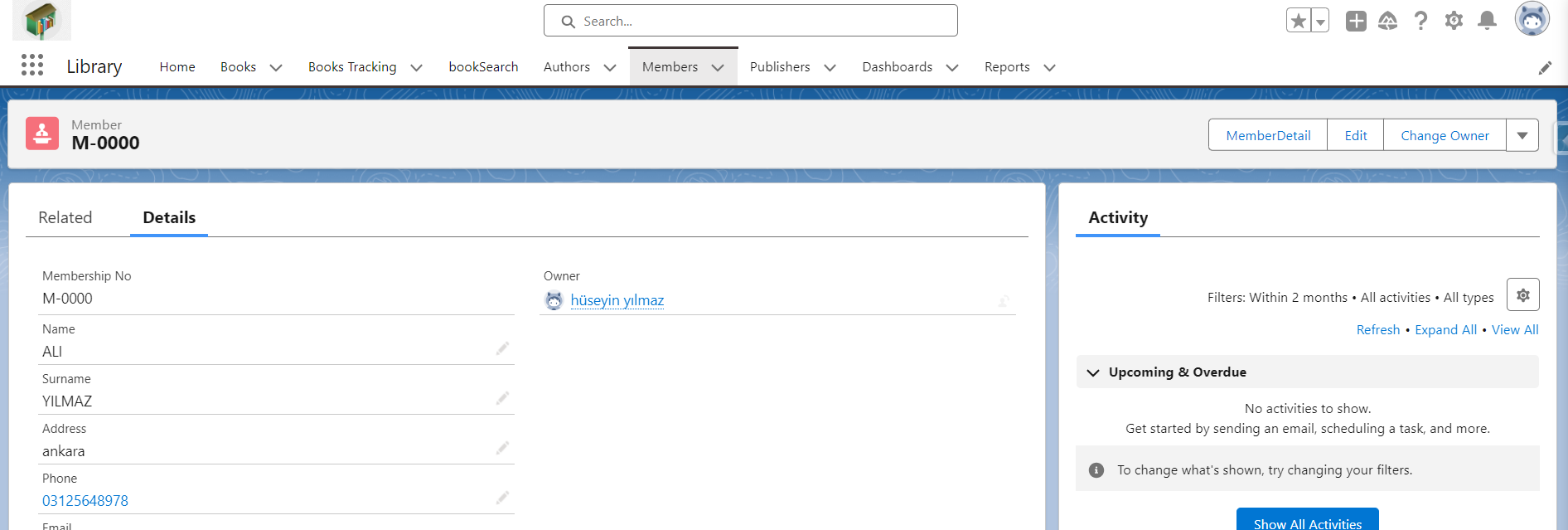
this.isNoRecordFound = true; // Hata durumunu açar

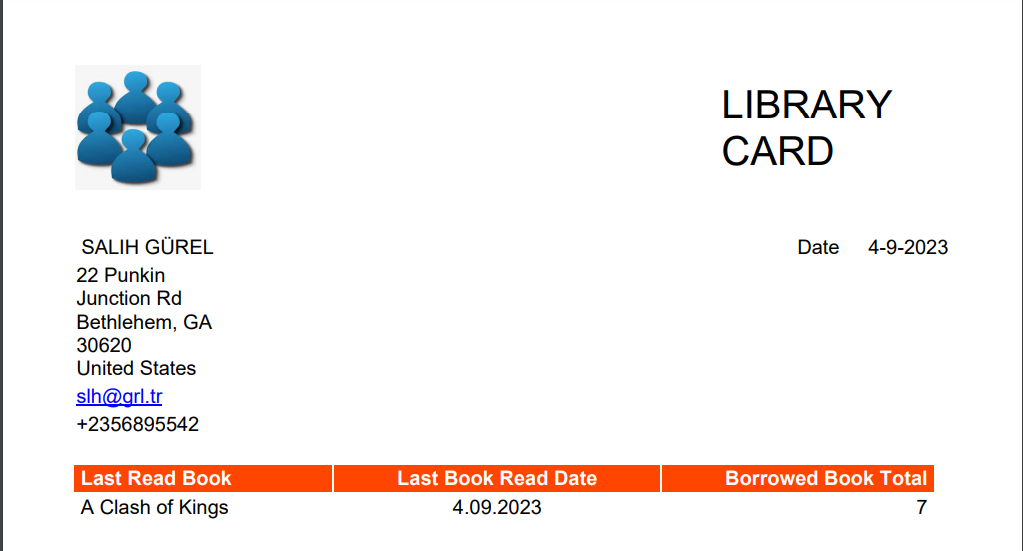
console.error(error); // Hata mesajını konsola yazdırır

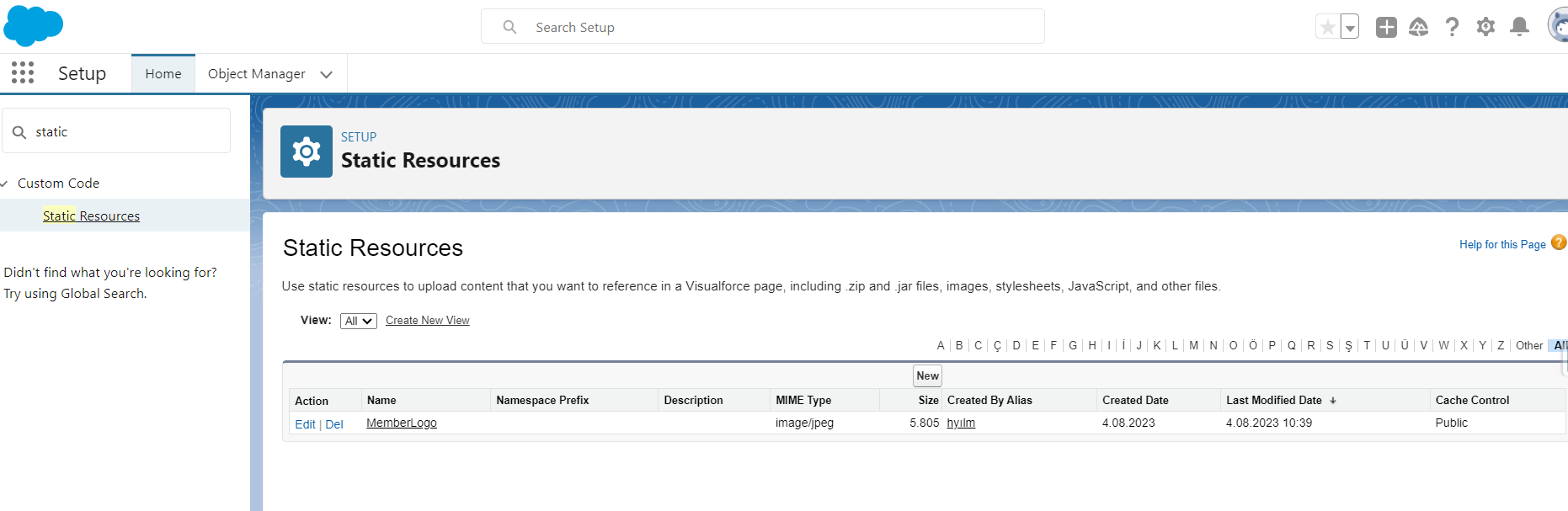
});

}}

**MEMBERDETAİLCARDS VİSUALFORCEPAGE**







Visualforce sayfası oluşturulurken Developer console kulanılmıştır. Ayrıca logo Static Resource ile hazırlandı.

Bu Visualforce sayfası, bir "Member\_\_c" özel denetleyicisi (custom controller) ile çalışan ve PDF olarak render edilen bir sayfa oluşturur.

**<apex:page> Etiketi:**

**standardController:** Sayfanın hangi özel denetleyiciyi kullanacağını belirler. Bu sayfa, "Member\_\_c" özel denetleyicisini kullanır.

**renderAs:** Sayfanın nasıl görüntüleneceğini belirler. Bu sayfa PDF olarak render edilecektir.

**İlk <table> Etiketi:**

Bu tablo, başlık ve resim içerir.

**<apex:image> etiketi**, bir resmi görüntüler. id, value, width, ve height gibi özelliklerle konfigüre edilir.

**İkinci <td>** içerisinde boş bir hücre bulunur ve CSS stili ayarları vardır.

**Üçüncü <td>** içerisinde "LIBRARY CARD" metni bulunur.

**İkinci <table> Etiketi:**

Bu tablo, üyeye ait bilgileri içerir.

**İlk satırda**, üyenin adı ve soyadı görüntülenir. <apex:outputField> etiketi ile veri alanlarına erişilir.

Tarih bilgisi için "Date" etiketi görüntülenir.

Tarih bilgisi "TODAY()" fonksiyonu kullanılarak oluşturulur.

**İkinci satırda**, üyenin adresi görüntülenir.

**Sonraki satırlarda**, üyenin e-posta ve telefon bilgileri görüntülenir.

**Üçüncü <table> Etiketi:**

Bu tablo, üyenin kitap okuma bilgilerini içerir.

Başlık satırı, arka plan rengi ve hücre stil ayarları ile belirginleştirilir.

İlk satırda, son okunan kitabın adı (Last\_read\_book\_\_c), son kitap okuma tarihi (Last\_Book\_Read\_Date\_\_c) ve ödünç alınan toplam kitap sayısı (Borrowed\_Book\_Total\_\_c) görüntülenir.

Bu Visualforce sayfası, bir "Member\_\_c" kaydının PDF formatında görüntülenmesi için tasarlanmıştır. Sayfa, özel denetleyici tarafından sağlanan verilere dayalı olarak üyenin bilgilerini ve kitap okuma geçmişini görüntüler. Ayrıca, sayfanın stilini ve düzenini CSS ile özelleştiren bazı stil ayarları içerir. Bu sayfa, Salesforce platformunda kullanılabilir ve "Member\_\_c" nesnesine bağlı olarak çalışır.

**MemberDetailCards VisualforcePage kodları:**

<apex:page standardController="Member\_\_c" renderAs="pdf">

<table style="font-family:sans-serif; padding-bottom:30px;">

<tr>

<td><apex:image id="MemberLogo" value="{!$Resource.MemberLogo}" width="100" height="100"/></td>

<td style="font:24pt; padding-left:10px; padding-right:400px;"></td>

<td style="font:24pt;">LIBRARY CARD</td>

</tr>

</table>

<table style="border:0.5px solid #fff; font-family:sans-serif;">

<tr>

<td style="padding-left:5px;"><apex:outputField value="{!Member\_\_c.Name\_\_c}"/> <span><apex:outputField value="{!Member\_\_c.Surname\_\_c}"/></span></td>

<td style="test-align:right; padding-right:20px;">Date</td>

<td style="text-align:right;"><apex:outputText value="{!DAY(TODAY())}-{!MONTH(TODAY())}-{!YEAR(TODAY())}"/></td>

</tr>

<tr>

<td style="padding-right:450px;"><apex:outputField value="{!Member\_\_c.Address\_\_c}"/> </td>

</tr>

<tr>

<td><apex:outputField value="{!Member\_\_c.Email\_\_c}"/></td>

</tr>

<tr>

<td><apex:outputField value="{!Member\_\_c.Phone\_\_c}"/></td>

</tr>

</table>

<br></br>

<table style="font-family:sans-serif;">

<tr style="color:#fff; background-color:#ff4500;">

<th style="padding-left:5px; padding-right:80px;">Last Read Book</th>

<th style="text-align:center; padding-left:50px; padding-right:50px;">Last Book Read Date</th>

<th style="text-align:right; padding-left:50px; padding-right:5px;">Borrowed Book Total</th>

</tr>

<tr>

<td style="padding-left:5px;"><apex:outputField value="{!Member\_\_c.Last\_read\_book\_\_c}"/></td>

<td style="text-align:center;"><apex:outputField value="{!Member\_\_c.Last\_Book\_Read\_Date\_\_c}"/></td>

<td style="text-align:right; padding-right:5px;"><apex:outputField value="{!Member\_\_c.Borrowed\_Book\_Total\_\_c}"/></td>

</tr>

</table>

</apex:page>