### BLM 4531 RAPOR

Bu proje kapsamında görev listenizin yapılma aşamasını 3 kısma ayırarak (to do,in progress ve done ) şeklinde iş akışlarının takip edilmesini kolaylaştırmayı amaçladım. Aynı zamanda günlük haberleri api yoluyla kullanıcıya aktararak kullanıcının haberleri takip etmesini sağladım. Bu projeyi iş tanımına uygun olarak hem görevlerin zamansal anlamda takip edilmesini, görev tanımlanmasını ve iş tanımına uygun olarak çıkan önemli makale ve haberleri kullanıcıya ileten bir uygulama yaptım.

## Veri Tabanı:

Uygulamam da list, progress, done şeklinde aynı database içerisinde 3 farklı table oluşturdum. Bu tabloların her biri içerisinde ayrı coloumns olarak id,title ve description oluşturdum. Id'yi primary key olarak işaretleyerek ve sequence oluşturarak gelen bir görev için idyi arttırdım. Böylece gelen her bir özellik için id farklı olacaktır.

#### BackEnd:

Gelen bilgileri veri tabanına eklemek, silmek, güncellemek için her bir bahsedilen table için ayrı fonksiyonlar tanımladım. Database connection sağladıktan sonra ilgili id üzerinden verileri çekerek üzerinde ekleme ya da silme işlemlerini gerçekleştirdim. Aşağıda örnek olarak edit aşamasının kodunu görebilirsiniz.

```
[HttpPost]
public ActionResult Edit(List edit_liste)
    todoEntities db = new todoEntities();
    List Saved_db = db.List.FirstOrDefault(x => x.id == edit_liste.id);
    Saved_db.title=edit_liste.title;
    Saved_db.description = edit_liste.description;
    db.SaveChanges();
    return RedirectToAction("show_list");
[HttpPost]
O başvuru
public ActionResult Progress_Edit(List edit_liste)
    todoEntities db = new todoEntities();
    var Saved_db = db.Progress.FirstOrDefault(x => x.id == edit_liste.id);
    Saved_db.title = edit_liste.title;
    Saved_db.description = edit_liste.description;
    db.SaveChanges();
    return RedirectToAction("show_in_progress");
```

Ayrıca tablelar arasında veri geçişini sağlamak amacıyla ayrı fonksiyonlar yazarak iş akış aşamalarında geçiş sağlamayı amaçladım. Örneğin aşağıdaki kodda progress

table aldığım veriyi Done table içerisine eklemek için Done table tipinde bir nesne oluşturarak o nesneyi Done tabelına ekledim.

```
public ActionResult Send_do_to_done(int id)
    todoEntities db = new todoEntities();
   List Send_data = db.List.FirstOrDefault(x => x.id == id);
   var model = new Done { description = Send_data.description, id = Send_data.id, title = Send_data.title };
   db.Done.Add(model);
   db.List.Remove(Send_data);
   db.SaveChanges();
   return RedirectToAction("show_list");
public ActionResult Send_Done(int id)
    todoEntities db = new todoEntities();
    var model = db.Progress.FirstOrDefault(x => x.id == id);
    var done_model=new Done {id=model.id,title=model.title,description=model.description};
    db.Done.Add(done_model);
    db.Progress.Remove(model);
   db.SaveChanges();
    return RedirectToAction("show_in_progress");
```

Bu aşamalar arasındaki her bir geçişi iş akış tanımına göre tanımladım.

### Api:

Haberlerin ve makalelerin takip edilmesini sağlamak için newsapi.orgdan connection sağlayarak ilgili habeleri json objesi olarak çektim. Daha sonra bu verileri Deserialize

Etmek amacıyla rootjson classını oluşturdum. Dönen veri içerisinde source, id olarak bir json array dönmektedir. Dönen json arrayın içindekiler içinde ayrı class tanımlamam gerekti. Yani bir article classı ve source classlarında oluşturdum. Bu durumda rootjsono classı list tipinde articlar olacak ve bunun yanında id si ve kaynağı da olacaktır. Aşağıda tanımlaması verilmiştir.

```
public class Article
    0 basvuru
    public Source source { get; set; }
    public string author { get; set; }
    public string title { get; set; }
    0 başvuru
    public string description { get; set; }
    0 başvuru
    public string url { get; set; }
    0 başvuru
    public string urlToImage { get; set; }
    public DateTime publishedAt { get; set; }
    O başvuru
public string content { get; set; }
}
1 başvuru
public class RootJson
    0 başvuru
    public string status { get; set; }
    public int totalResults { get; set; }
    public List<Article> articles { get; set; }
}
1 başvuru
public class Source
    0 başvuru
    public string id { get; set; }
    public string name { get; set; }
```

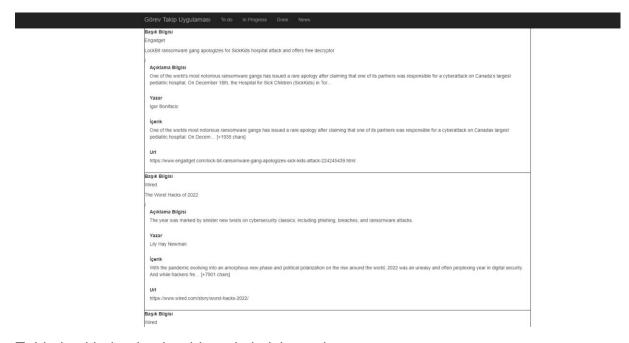
```
///
var url = "https://newsapi.org/v2/everything?q=cybersecurity&sortBy=popularity&apiKey=97da344f557e483f8e64c69bc0b256e2";

WebClient client = new WebClient();
client.Encoding = Encoding.UTF8;

string json = client.DownloadString(url);

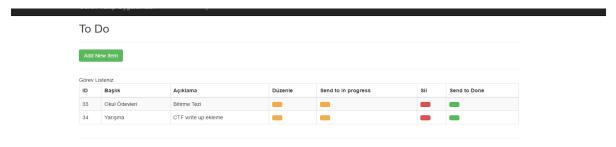
var deserialized = JsonConvert.DeserializeDbject<RootJson>(json);
return View(deserialized);
```

Deserialeze ettiğim veriyi de ön kısımda her bir makale için ayrı table olacak şekilde ayırarak gösteriyorum. En sonunda ise görünüm şu şekilde oluyor.

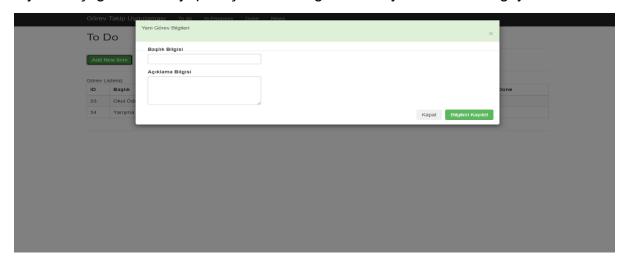


Tablo kesitleri gelen her bir makale için ayrılıyor.

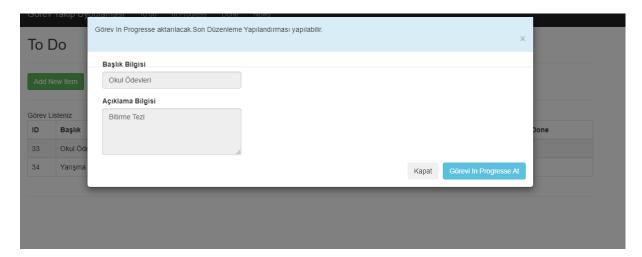
# FrontEnd:



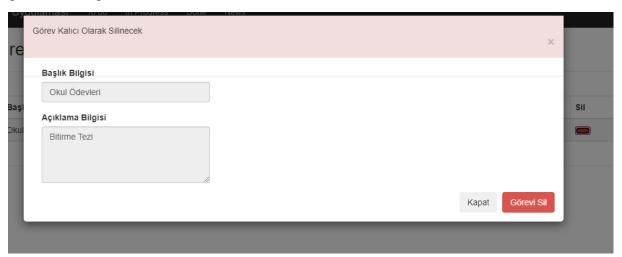
MVC kullanarak yukarıdaki gibi button ve model tanımlamalarımı yaptım. Add new item diyerek aşağıdaki card yapısı içerisinde bilgilerinin kaydedilmesini sağlıyorum.



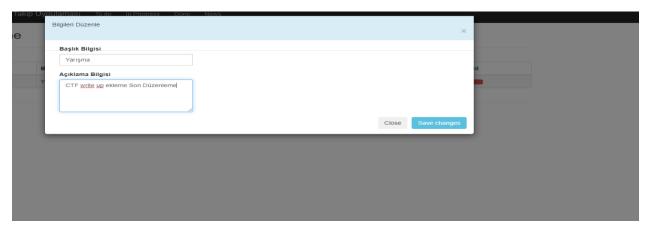
Progress ve Done tabela atılacağı zaman ise aşağıdaki gibi bir card modeli tanımladım.



Görev kalıcı olarak silineceğinde ise aşağıdaki gibi danger modunda bir card yapısı ile gösterimi sağladım.



Görev Düzenleme ekranında ise aşağıdaki gibi bir editleme yaparak database kaydettim.



Aşağıda da örnek olarak to do ekranının front end kısmını görebilirsiniz. List tabelında ilgili verileri viewbag yapısına daha önceden atayarak verileri bu şekilde alabiliyoruz.

```
circ class-real-public
class-rea
```