T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

KOD ŞABLONLARINA DAYALI OTOMATİK WEB UYGULAMASI OLUŞTURMA MEKANİZMASI

YÜKSEK LİSANS TEZİ

Burak UYANIK

Enstitü Anabilim Dalı	**:**	**BİLGİSAYAR VE BİLİŞİM MÜHENDİSLİĞİ**
Tez Danışmanı	**:**	**Dr.Öğr. Üyesi Veysel Harun ŞAHİN**

Aralık 2018

T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

KOD ŞABLONLARINA DAYALI OTOMATİK WEB UYGULAMASI OLUŞTURMA MEKANİZMASI

YÜKSEK LİSANS TEZİ

Burak UYANIK

Enstitü Anabilim Dalı : BİLGİSAYAR VE BİLİŞİM MÜHENDİSLİĞİ

Bu tez 24/12/2018 tarihinde aşağıdaki jüri tarafından oybirliği / ~~oyçokluğu~~ ile kabul edilmiştir.

Dr. Öğr. Üyesi
Veysel Harun ŞAHİN
Jüri Başkanı

Prof.Dr.
Celal ÇEKEN
Üye

Doç.Dr.
Ahmet SAYAR
Üye

BEYAN

Tez içindeki tüm verilerin akademik kurallar çerçevesinde tarafımdan elde edildiğini, görsel ve yazılı tüm bilgi ve sonuçların akademik ve etik kurallara uygun şekilde sunulduğunu, kullanılan verilerde herhangi bir tahrifat yapılmadığını, başkalarının eserlerinden yararlanılması durumunda bilimsel normlara uygun olarak atıfta bulunulduğunu, tezde yer alan verilerin bu üniversite veya başka bir üniversitede herhangi bir tez çalışmasında kullanılmadığını beyan ederim.

Burak UYANIK

3.12.2018

TEŞEKKÜR

Bu tezde sunulan çalışma, TÜBİTAK TÜSSİDE'de geliştirilmiş olan TÜSSİDE Bütünleşik Yönetim Sistemi (TBYS) adlı projenin bir bölümü olarak geliştirilmiştir. İlgili sistem halen TÜSSİDE'de geliştirilmekte ve aktif olarak kullanılmaktadır.

Yüksek lisans eğitimim boyunca değerli bilgi ve deneyimlerinden yararlandığım, her konuda bilgi ve desteğini almaktan çekinmediğim, araştırmanın planlanmasından yazılmasına kadar tüm aşamalarında yardımlarını esirgemeyen, teşvik eden, aynı titizlikte beni yönlendiren değerli danışman hocam Dr. Veysel Harun ŞAHİN'e teşekkürlerimi sunarım.

Son olarak, bu günlere ulaşmamda emeklerini hiçbir zaman ödeyemeyeceğim aileme şükranlarımı sunarım. Sevgili eşim Gizem'e sonsuz desteği ve anlayışından dolayı minnettarlığımı ifade etmek isterim.

İÇİNDEKİLER

SİMGELER VE KISALTMALAR LİSTESİ

API	: Uygulama Programlama Arayüzü - (Application Programming Interface)
CRUD	: Oluştur - Oku - Güncelle - Sil (Create-Read-Update-Delete)
DOM	: Belge Nesnesi Modeli (Document Object Model)
HTML	: Hiper Metin İşaret Dili (Hyper Text Markup Language)
HTTP	: Üstmetin transfer protokolü (Hyper Text Transfer Protocol)
IIS	: Internet Information Services
IOT	: Nesnelerin interneti (Internet of Things)
JSON	: JavaScript Nesne Gösterimi (JavaScript Object Notation)
MVC	: Model Görünüm Denetleyici (Model View Controller)
ODATA	: Açık Veri Protokolü (Open Data Protocol)
REST	: Temsili Durum Transferi (Representational State Transfer)
SCG	: Kaynak Kod Üretimi (Source Code Generator)
SQL	: Yapılandırılmış Sorgu Dili (Structured Query Language)
T-SQL	: Görülmüş Yapılandırılmış Sorgu Dili (Transact Structured Query Language)
UI	: Kullanıcı Arayüzü (User Interface)
UML	: Birleşik Modelleme Dili (Unified Modelling Language)
URL	: Tekdüze Kaynak Bulucu (Uniform Resource Locator)
XML	: Genişletilebilir İşaretleme Dili (Extensible Markup Language)
XSLT	: Genişletilebilir Biçimlendirme Dili Dönüşümleri (Extensible Stylesheet Language Transformations)

ŞEKİLLER LİSTESİ

TABLOLAR LİSTESİ

ÖZET

Anahtar kelimeler: Yazılım Mühendisliği, Kod Üretim Mekanizması, Otomatik Kod Üretimi, AngularJS

Gelişmekte olan teknoloji web uygulamalarını daha çok ön plana çıkarmaktadır. Masaüstü uygulama programlarının yerini web uygulama programları almaya başlamıştır. Web uygulamaları büyük ölçekli kurumsal firmalarıda yaygın olarak kullanılmaktadır. Çok katmanlı mimari, entegrasyonu ve erişimi kolaylaştırmaktadır. Bu kapsamda, çok zaman alan kodlama işlemlerinin daha kısa sürede tamamlanabilir olması çalışmanın kalitesini ve verimliliğini arttırmaya yardımcı olmaktadır. Bu tez çerçevesinde "kod şablonlarına dayalı kod üretimi" isminde bir sistem tasarlanmış ve geliştirilmiştir. İlgili sistem, TÜBİTAK TÜSSİDE'nin TÜSSİDE Bütünleşik Yönetim Sistemi (TBYS) projesinde halen kullanılmaktadır. Bu çalışmada sunulan sistemde, geliştirici tarafından oluşturulmuş olan veritabanında bulunan sütun tipine göre tablo ve saklı yordam nesnelerini referans alarak, web uygulamasında doğrudan kullanılabilecek HTML kodları üretilmektedir. Sonuç olarak çok katmanlı mimari belirli bir kalıplar içerisinde gerçekleştirildiğinde, entegrasyon kolaylığı ve zaman tasarrufu açısından verimli bir çalışma ortaya çıkmaktadır.

WEB APPLICATION GENERATION MECHANISM BASED ON CODE TEMPLATES

SUMMARY

Keywords: Code Production Mechanism, Automatic Code Generation, AngularJS

Web applications have become more prominent in the advancing technology era. Today, Web application programs are taking place of desktop application programs. Web applications are widely used in large-scale companies. Multi-layer architecture facilitates integration and access. In this context, completing time-consuming coding processes in shorter time helps to increase the quality and efficiency of the work. In the framework of this thesis, a system called “code generation based on code templates” was designed and developed. This system is still being used in TUBITAK TUSSIDE’s TUSSIDE Integrated Management System (TBYS) project. In the system presented in this study, HTML codes are generated which can be used directly in the web application, referring to the table and stored procedure objects according to the column type found in the database created by the developer. As a result, when multi-layered architecture is carried out in certain molds, an efficient study comes out, in terms of ease of integration and time saving.

BÖLÜM 1. GİRİŞ

Yazılım geliştirme teknikleri hakkında son dönemde, geliştiriciler tarafından hızlı ve pratik kullanma yöntemleri için yeni fikirler ortaya atılmaktadır. Bu fikirlerin amacı hem zaman tasarrufu sağlamak hem de kaliteli ve verimli iş yapılmasını sağlamaktır. Otomatik kod üretimi bir kod yazma görevini basitleştirir. Kod üretim mekanizmalarının altyapı mimarileri ve otomatik kod üretimi konularında [1-4] numaralı kaynaklarda detaylı bilgiler mevcuttur. Otomatik kod üretim sistemlerinde, ihtiyaç duyulan kod parçaları bileştirilerek yeni bir kod üretilir. Üretilen kodlar sonrasında başka bir uygulama için kullanılabilir hale gelir. Buna ihtiyaç duyulmasının temel nedeni, yazılım geliştirme sürecinde, yinelenen işlemlerin kodlarının tekrar tekrar yazılmasının önüne geçmek ve dolayısıyla geliştiriciye yardımcı olmaktır. Gerek bu tez çalışmasındaki gerekse diğer otomatik kod üretiminin bize sağladığı faydalar şunlardır;

- Arayüz geliştiricisi ve arka plan geliştiricisi olarak 2 farklı geliştiricinin olması proje maliyetini arttırır. Otomatik kod üretimi mekanizmalarından faydalanıldığında tek geliştirici ile hem arka plan hem de arayüz kod üretimi gerçekleştirilebilir.

- Çok zaman alan arayüz tasarımı için otomatik kod üretimini kullanarak zaman tasarrufu yapılabilir.

- Otomatik kod üretimi kullanılmadığında her yeni bir değişiklik için sistemin baştan kodlanması gerekir. Böyle bir durumda da hata oranı artar. Öte yandan otomatik kod üretimi modeli kullanıldığı zaman belirlenmiş kalıplar içerisinde kod üretimini gerçekleştireceğinden dolayı hata oranı en aza inecektir. Ayrıca

otomatik kod üretimi ile oluşturulmuş olan kodlar uygulamanın kalitesini arttırır.

- Belirlenmiş kalıplar içerisinde belirli veri tipleri ile oluşturulduğu için aynı isimle kod üretimi yapılır. Bu sebeple her işlem sonrası aynı kalıp ile aynı veri tipleriyle kod üretimi gerçekleşir.

- Üretilen kod kullanıcının girdi değerleriyle oluşturulduğu için kodları doğrudan uygulamada kullanılabilir.

Kod üretimi daha önce birçok alanda kullanılmıştır. Halen de yeni alanlarda farklı şekillerde kod üretimi devam etmektedir. Kod üretiminin amacı kod yazma işlemin kalitesini arttırmaktır. Kalite arttığında geliştiricinin üretkenliği artar ve zamandan tasarruf eder. Otomatik kod üretimi hakkında yazılan bir çok yayın mevcuttur. [5-15] numaralı makalelerde çeşitli altyapılar kullanarak otomatik kod üretimi oluşturmaktan bahsedilmektedir.

Zhang, veritabanı kıyaslamaları için otomatik kod üreten bir mekanizma geliştirmiştir [8]. İlgili çalışmada bir ön modül ve dört arka modül mevcuttur: Çevre, Şablon Motoru, Şablonlar ve Soyut Kıyaslama. Kullanıcı veritabanı konfigürasyon belirtme aşamasında, kısıtlamalar esas alınarak isim-değer çiftleri olan ortam modülünde saklanır. Kod üretimi aşamasına geçtiğinde kısıtlar ortamdan okunur ve şablon motoruna verilir. Apache Velocity Engine'in (Java tabanlı bir şablon motoru) [16] üzerine kurulmuş olan Şablon Motoru, önceden tanımlanmış şablonu dosyalarındaki değişkenleri kısıtlamalara bağlar ve karşılık gelen Java kaynak dosyalarını oluşturur.

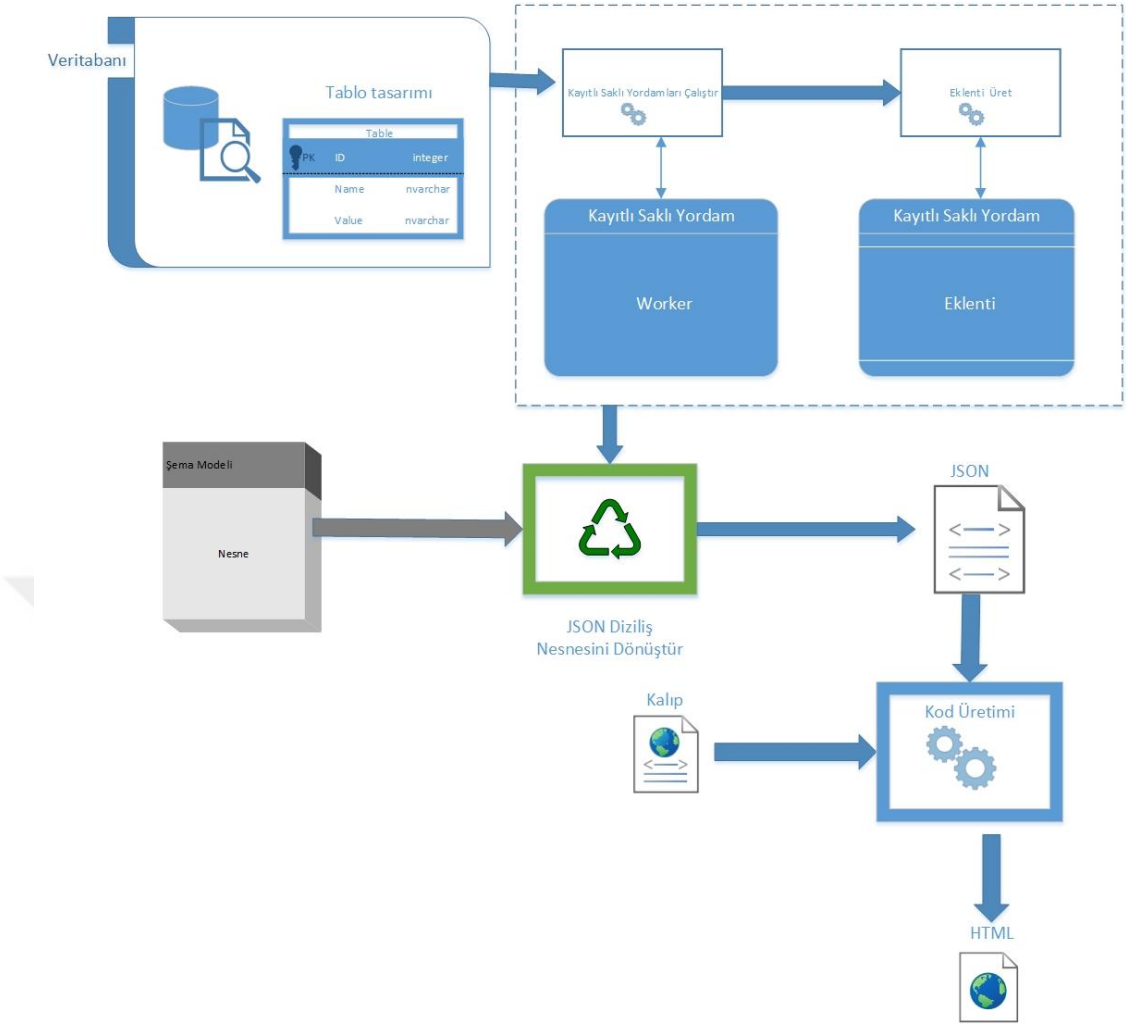
Akbulut vd., son kullanıcı geliştirmesi için otomatik kod üretimi gerçeklemesi yapmıştır [11]. İlgili çalışmada kullanıcı tarafından oluşturulan Birleştirilmiş Modelleme Dili (UML) diyagramları Genişletilebilir İşaretleme Dili (XML) belgesine çevirilmektedir. Daha sonra oluşturulan XML dosyasından kod üretimi gerçekleşir.

Balçiçek vd. çalışma gereksinimlerine odaklı kod üretimi yapısı modeli geliştirmiştir.[13]. İlgili çalışmada iş gereksinimlere odaklı kaynak kod üretimi

modelinden bahsedilmektedir. Arayüzde belirlenen alan ve veri tipleri XML dosyasını oluşturur. Oluşturulan XML dosyası Genişletilebilir Biçimlendirme Dili Dönüşümleri (XSLT) kalıplarına işlenir. İstenilen kodlar, saklı yordamlar ve tetikleyiciler oluşturulur. Uygulama esnasında gerçekleşen veri erişimleri, veri taşımacılığı vb. için veri paketleri otomatik olarak oluşturulur.

Altıparmak vd. "büyük ölçekli uygulamalar için kaynak kod üretimi üzerine çalışma yapmıştır[12]. Franky ve Pavlich-Mariscal düzenli ifadeler (regular expression)[10]. yardımıyla kaynak kod üretme sistemi geliştirmiştir. Bileşenleri ve aksiyonları ekran arayüzünden referans alınarak gereken görünüm, saklı yordam, tablo vb. kodlarının üretilebileceği bir çalışmayı anlatmaktadır.

Bu tezde sunulan "kod şablonlarına dayalı kod üretim mekanizması" çalışmasında geliştirici tarafından oluşturulmuş veritabanında mevcut olan sütun tipine göre tablo ve saklı yordam nesnelerini referans alarak, web uygulamasında doğrudan kullanabileceğimiz HTML kodları üreten model tasarlanmış ve geliştirilmiştir. İlgili mekanizma Şekil 1.1.'de görülmektedir.



Şekil 1.1. Veritabanından HTML kod dönüşüm süreci

Bölüm 2'de, kod şablonlarına dayalı kod mekanizmasını oluşturan model geliştirilirken kullanılan teknolojilerden bahsedileecektir. 3. Bölüm olan Yöntem bölümünde otomatik kod üretiminin gerçekleşme aşamaları anlatılacaktır. Sonuç bölümünde ise elde edilen sonuç, uygulamanın genel akışı, avantajları vb. konulardan bahsedilecektir.

BÖLÜM 2. KULLANILAN TEKNOLOJİLER

2.1. Saklı Yordam

Saklı yordamlar, veritabanı yönetim sistemi içerisinde tanımlanan ve belirli bir görevi icra eden kodlardır [17]. Geliştirici tarafından saklı yordam yazılarak oluşturulduktan sonra ilk derleme dışında yeniden derleme yapmaz. İstenildiği zaman çağrılarak kullanabilir. Saklı yordam kullanmanın faydaları aşağıdaki şekilde özetlenebilir;

1. Başarımı arttırır. Yazıldığı zaman derleme yapıldığı için, her koşturmada yeniden yorumlanmaz, derlenmez.

2. Geliştirici verimliliğini artırır. Uygulama programlarında yinelenecek olan yapılandırılmış sorug dili (SQL) gruplarının tekrar yazılmasına gerek kalmaz. Yeniden kullanılabilirlik (reusability) özelliği katar.

3. Geliştirici tarafından oluşturulan kodun okunurluğunu arttırır.

4. N-katmanlı mimarilerde daha verimli iş üretir.

5. Veritabanı yönetim sistemine gönderilen sorguların boyutu küçüleceğinden dolayı ağ trafiğini azaltır.

6. SQL'e göre daha güvenlidir. Veri girişi ve çıkışı sadece uygulama platformunda yapılır.

7. Temel programlama dili yapılarını destekler. "If—Else" vb.

8. Yetkilendirme ve kota işlemleri yapılabilir.

Saklı yordamın en çok kullandığı başlıca alanları; Veri çekme, Veri Silme,Veri ekleme,Veri güncelleme. SQL değişkenlerine otomatik olarak parametre tanımlanır. SQL değişkenlerin olduğu yerlerdeki parametrelerin yerine saklı yordam parametrelerini veya uygun görülen prosedürleri istenildiği şeklinde değiştirip kullanılabilir. Bu sayede kodda oluşabilecek herhangi bir karmaşıklığı önler. Her döngüde tekrarlanan inceleme,derleme vb. işlevlerden kurtulup daha hızlı bir şekilde çalışır.

2.2. CRUD

CRUD kısaltması, Oluşturma (Create), Okuma (Read), Güncelleme (Update) ve Silme (Delete) kelimelerinin İngilizce'deki baş harflerinin birleşmesinden oluşmaktadır. Oluşturma, okuma, güncelleme ve silmeden oluşan veri döngüsüdür [18]. Okurlar, CRUD hakkında Kilov'un 1990 yılındaki makalesinden detaylı bilgi alabilirler [19]. CRUD döngüsü bir dört işlevinin aşağıdaki şekilde özetleyebiliriz;

1. Oluşturma (Create): Oluşturma prosedürü, yeni kayıt gibi ifadelerde kullanılır. Örneğin, bir elektronik ticaret web uygulamasında kişi kişisel bilgileriyle ilgili bir sistemde üyeliğini oluşturur. Girdi parametreleriyle kaydet düğmesine basıldığında kişinin veritabanına kaydı yapılır ve üyeliğini oluşturur.

2. Okuma (Read): Okuma prosedürü, önceden oluşturulmuş olan verileri okuma, listeleme vb. işlemlerde kullanılır. Örneğin, bir elektronik ticaret web uygulamasının veritabanında kişinin daha önce kaydedilmiş olan üyelik detay bilgileri, sonraki zaman dilimlerinde sorgulanarak gösterilebilir.

3. Güncelleme (Update) : Güncelleme prosedürü, daha önce kaydedilmiş mevcut bir bilgiyi değiştirme eylemidir. Örneğin, bir elektronik ticaret web uygulamasının veritabanında kayıtlı bir üyenin isim bilgisi farklı bir isimle değiştirip kaydedilebilir.

4. Silme (Delete): Silme prosedürü, önceden kaydedilmiş olan veri(ler)in silinmesi işlemidir. Örneğin, bir elektronik ticaret web uygulamasında, bir üye,

üyeliğinin iptalinin yapılmasını isteyip “Hesabımı Sil” düğmesine tıklayarak silme işlemini gerçekleştirebilir.

Yüksek seviyeli C#, Java, PHP vb. bir dilde CRUD işlemleri yazarak gerçekleştirmek için Veri Depolama Katmanı ismi verilen bir katman kullanılır. Veri depolama katmanı kullanılmasının başlıca nedeleri; başarımı artırmak ve saldırıları önlemektir. RESTful mimarisinin anlaşılması için CRUD oluşturulmalıdır. Bölüm 2.4’de ve [20] numaralı kaynakta RESTful hakkında detaylı bilgiler mevcuttur. CRUD işlemlerinin üstmetin transfer protokolündeki (HTTP) karşılıkları aşağıda özetlenmiş ve Tablo 2.1.’de verilmiştir.

1. Oluşturma(Create) - PUT: İstenilen veri sınıfına yeni veri eklenir.

2. Okuma(Read) - GET: Bir veriyi okumak için kullanılır. Bu veriyi üst üste aynı kez GET olarak çağırırsanız bile ilk çağırmada gösterilen veriyi gösterir.

3. Güncelleme(Update) - POST: Var olan bir veri bilgisini değiştirmek için kullanılır.

4. Silme(Delete) - DELETE: Var olan bir veri bilgisini sistemden silmek için kullanılır.

Tablo 2.1. CRUD'un HTTP'deki karşılığı

İŞLEM	HTTP
OLUŞTURMA (CREATE)	PUT
OKUMA (READ)	GET
GÜNCELLEME (UPDATE)	POST
SİLME (DELETE)	DELETE

CRUD işlemlerinin SQL ifadelerinde de karşılıkları vardır. Tablo 2.3.’de bunlar gösterilmektedir. CRUD bu prosedürlerin otomatikleştirilmesi için gereklidir.

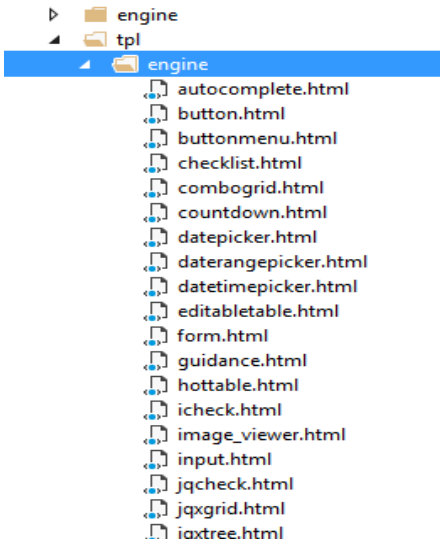
CRUD'u yöneten kişi yine geliştiricidir. Saklı yordam yazılırken CRUD'un yerinde kullanılması önemlidir.

Tablo 2.2. CRUD'un SQL'deki karşılığı

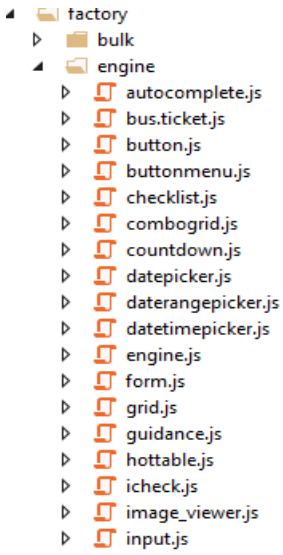
İŞLEM	SQL
OLUŞTURMA (CREATE)	INSERT
OKUMA (READ)	SELECT
GÜNCELLEME (UPDATE)	UPDATE
SİLME (DELETE)	DELETE

2.3. Eklenti

Bir uygulamaya ek işlevsellik katmak için geliştirilen yazılıma eklenti (plugin) ismi verilir. Eklentilerin, eklendikleri uygulama dışında tek başlarına bir değeri yoktur ve herhangi bir işlem gerçekleştirmezler. Diğer bir deyişle eklentiler bağımsız yazılımlar değildir. Eklentiler, geliştiricileri uğraştan kurtarıp programların daha hızlı ve verimli yazılmasına yardımcı olurlar.



Şekil 2.1. HTML Eklentileri



Şekil 2.2. Javascript Eklentileri

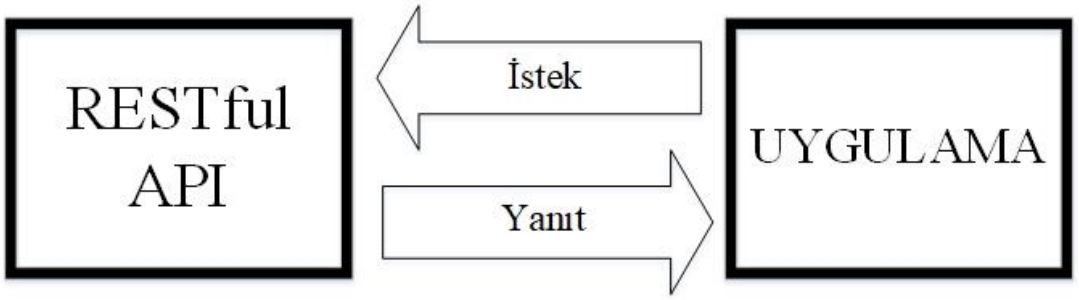
Şekil 2.1.'de görünüm (view) eklenti dosyalarının bir listesi gösterilmektedir. Bu eklentiler HTML biçiminde yazılmıştır. Bu dosyalar, arayüz bileşenlerinin nasıl gösterileceğine ilişkin kodları içerir. Örneğin; düğme (button) vb. Şekil 2.2.'de aksiyon işlevlerine dair eklenti dosyalarının bir listesi gösterilmektedir. Bu dosyalar JavaScript biçimindedir. Arayüzünde gördüğümüz bileşenlerin icra edeceği görevleri ifade eden aksiyonları içerir. Örneğin; düğmenin çalışması için gereken kodlar bu eklentiler içerisinde yer alır.

2.4. RESTful

Temsili durum transferi (REST), istemci-sunucu iletişimine dair mimari bir yapıdır [20]. REST iletişimi kurmak için HTTP prosedürlerini kullanır. RESTful, REST'e benzeyen mimari yapısıyla web servislerinde veya uygulamalarda tanım yapılırken kullanılan terimdir. RESTful da HTTP prosedürlerini Tablo 2.3.'de görüldüğü gibi farklı şekilde kullanmaktadır. Tekdüze kaynak bulucu (URL) ile aşağıda verilen HTTP'nin dört prosedüründen biri şeçilerek talep edilen sunucu üzerindeki etkiye göre HTTP isteği belirlenir.

Tablo 2.3. HTTP Prosedürü RESTful Karşılığı

HTTP	RESTful
GET	OKUMA - READ
POST	EKLEME - INSERT
PUT	GÜNCELLEME - UPDATE
DELETE	SİLME - DELETE



Şekil 2.3. RESTful ile Uygulama arasındaki iletişim

RESTful servisleri ile uygulama yazılımları, JavaScript nesne gösterimi (JSON) aracıyla ile haberleşir (Şekil 2.3.). Böylece servis katmanında diğer katmanlardan bağımsız olarak çalışır. Hem HTML hem de başka bir dil kullanabilir. RESTful servislerine aşağıdaki örnekleri verebiliriz.

1. Kod kaynağına erişim sağlayan bir URL (/services/control)

2. Kod kaynağı üzerinde işlem yapmamızı sağlayan metotların tanımlanması
   - Sisteme yeni kod girme -> HTTP POST
   - Sistemden kod silme -> HTTP DELETE
   - Sistemde var olan bir kodlarını güncelleme -> HTTP PUT
   - Sistemde var olan bir kodlarını görüntüleme -> HTTP GET

RESTful servislerin avantajları olduğu gibi dezavantajı vardır. Önce avantajlarından anlatalım;

- Basit yapıda olması

- Bir uygulamaya kolayca uygulanabilme

- Uygulamanın daha hızlı çalışması

- Uygulamaya uygulandığında esnek çalışabilme imkanı

Dezavantajları;

- Güvenlik önlemlerini almak için yazılım geliştirilmesi gerekir.

2.5. JSON

JSON, XML biçiminin alternatifi olarak üretilmiş olan JavaScript tabanlı bir veri saklama ve taşıma biçimidir [21]. Platformlardan bağımsız olarak çalışır. Günümüzde JSON sadece JavaScript uygulamalarında değil birçok farklı yazılımda kullanılmaya başlanmıştır. Örneğin PHP, .NET, web servis vb. birçok uygulamada veri taşımak için JSON kullanılmaktadır.

JSON'un kullanılmasındaki esas amaç daha küçük boyutlarla veri alışverişini gerçekleştirmesidir. XML'in aksine JSON dosyaları daha küçük boyuttadır. Bu sebeple dosyalar XML'den daha az yer kaplamaktadır. JSON türündeki veriler Anahtar (Key) ve Değer (Value) biçimindedir. "Anahtar:Değer (Key:Value)" ilişkisi yardımıyla referans belirleyip veri saklanabilir. JSON'da iki temel yapı mevcuttur;

1. JSON Nesne (Object) Yapısı: Her JSON nesnesi süslü parantez içerisinde bulunur. İçerisinde sonsuz sayıda "Anahtar:Değer (Key:Value)" mevcuttur. Her nesne virgül ile ayırılır. Aşağıda iki adet JSON nesnesi görülmektedir.

   {
   "Name": "Project1",
   "ProjectNumber" : "1938"

},
{
“Name”: “Project2”,
“ProjectNumber”: “1939”
}

2. JSON Dizi (Array) Yapısı: Her JSON dizisi köşeli parantez içerisinde bulunur. İçerisinde “Değer (Value)” mevcuttur. Değerler birbirinden virgül ile ayrılır. Aşağıda bir adet JSON adet dizisi görülmektedir.
   [“Project1”, “Project2”]

JSON biçiminin desteklediği veri tipleri; karakter katarı (string), sayı (number), mantıksal (boolean) (doğru (true), yanlış (False)), boş (null), nesne (object) ve dizidir (array). Bu veri tiplerine ilişkin değerler iç içe kullanılabilirler. Örneğin nesne içerisinde dizi veya dizi içerisinde nesne gibi.

JSON dosyaları birçok uygulamada kullanılabilir ve uzantısı daima “.json” olmak zorunda değildir. Öyle ki bazı geliştiriciler uzantıyı kullanılmaya gerek duymamaktadır. JSON dosyaları, uzantı bulunmasa bile kullanılabilir. JSON kullanılmasının avantajları aşağıda belirtilmiştir.

1. Yazılması ve okunması kolaydır. Sade ve anlaşılabilirdir.

2. Dosya boyutu küçüktür.

3. Hızlı bir şekilde oluşturulabilir.

4. Programlama dilinden bağımsızdır.

JSON, web servisleri, paket yöneticileri, NuGet paket yazılımları, mobil uygulamalar vb. alanlarda çok yaygın bir şekilde kullanılmaktadır.

2.6. HandlebarsJS

Yer tutucu olarak bilinen HandlebarsJS tüm yazılım dillerinde kullanılabilen şablon kütüphanesidir [22]. HandlebarsJS mimarisi üç ana yapından oluşturmaktadır.

1. JSON; Veri kaynağıdır.

2. HTML Şablonları; “<script>” … “</script>” etkiketleri arasındadır. Buna “Gösterge Alanı” denir.

3. Gösterge; JavaScript nesnedeki veriyi yer tutucuda veri gösteren kısmıdır.

2.7. ASP.NET Web API

API, “Uygulama Programlama Arayüzü (Application Programming Interface)” ifadesinin kısaltmış halidir.Geliştirme arayüzü anlamına gelir. Var olan servisler veya verileri dış dünyaya sunan arayüzdür. Günümüzde insanlar yoğun olarak web tabanlı uygulamalar kullanılmaktadır. Günümüzde nesnelerin interneti (IOT) teknolojileri yardımı ile insanlar cihazlarına internet üzerinden bağlanabilmektedir. Bu teknolojinin kullanımı ilerleyen zamanlarda daha da artacaktır. Bunun anlamı tahmin edilenden daha çok API geliştirilmesi gerekeceğidir.

ASP.NET Web API [23] , farklı platformlar üzerinden HTTP protokolleriyle veri alışverişini gerçekletirmek için kullanılan sistemdir. RESTful servisleri Web API platformu üzerinden farklı bir araca ihtiyaç duymadan gerçekleşir. HTTP’nin mevcut protokülündeki bütün özelliklerini kullanr. ASP.NET Web API’nin temel özellikleri aşağıda verilmiştir.

- CRUD yapısı kullanılabilir.

- HTTP durum kodu (HTTP Status Code) ve HTTP istek başlığı (HTTP accept header) parametreleri cevap (response) içerisinde mevcuttur. Web

uygulamalarında kullanıcı tarafından hangi veri tipleri kabul edildiği HTTP istek başlığı ile belirlenir.

- Geliştirici tarafından cevaplar ortam biçimlendiricileri (MediaTypeFormatter) ile sıfırlanabilir. Ortam biçimlendiricileri, asenkron okuma ve yazma yöntemlerini kullanır.

- Açık Veri Protokolü (Odata) mevcuttur. Açık Veri Protokolü, web uygulamaları için veri erişimi protokolüdür. Veri kümelerini sorgulamak ve değiştirmek için kullanılır.

- Sorgu yazması basittir.

- Internet Information Services (IIS) üzerinden veya bir uygulama üzerinden konfigure edilmiş bir alan ayırabilir.

- Model Görünüm Denetleyici (Model View Controller - MVC) özelliklerini destekler.

- XML, JSON vb. destekleri mevcuttur.

2.8. AngularJS

Google tarafından geliştirilmiş açık kaynaklı bir platformdur [24]. MVC mimarisine sahip, büyük boyutta ve yüksek performanslı web uygulamalarını oluşturmamıza yardımcı olur. HTML'den bilgi alışverişi yapılmasına yardımcı olan belge nesnesi modeli (DOM), AngularJS üzerinden kontrol edilebilmektedir. Hem kolayca yönetebilir hem de düzenleme yapılabilir. Böylece daha sade bir HTML DOM oluşturulmuş olur. AngularJS platformunun önemli temel özellikleri aşağıda verilmiştir.

- Veri Bağlama: Model ile Görünüm bölümleri arasındaki veri eşleme.

- Kapsam: Denetleyici ile görünüm arasındaki referans nesneleri.

- Denetleyici: Belirli bir model içerisine bağlı olan JavaScript işlemleri.

- Bağımlılık Enjeksiyonu: AngularJS'nin uygulamaya yardımcı olan alt sistemidir. Bu durumda geliştiriciye uygulamanın daha anlaşılır, geliştirebilir ve kolay kontrol edilebilir hale getirir.

- Hizmetler: XMLHttpRequests oluşturmak için farklı yerleşik hizmetler.

- Şablonlar: Denetleyici ve modelden gelen veri bilgileri işleyerek elde edilen görünümler.

- Model görünümü: Model ve denetleyici kalıpları.

- Derin bağlantı: Kullanıcıları belirli bir uygulama sayfasına götüren yapı.

- Filtreler: Seçtiği alt grupların dizileri içerisine yeni bir dizi oluşturup döndürülmesine yardımı olan yapı.

- Yönergeler: DOM öğelerindeki işaretleyiciler. Özel HTML etiketleri oluştururken kullanılır.

- TypeScript dilinin kullanılıyor olması.

AngularJS avantajları ise aşağıdaki şekilde özetlenebilir.

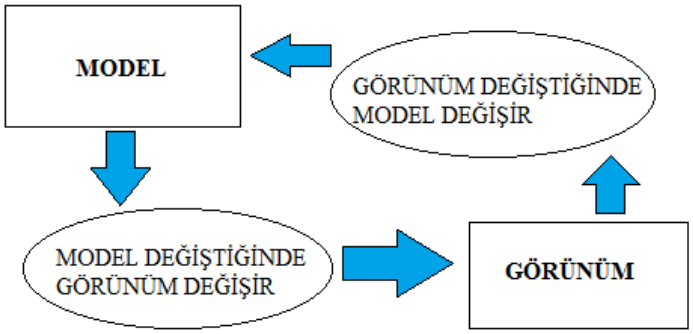
- HTML'ye veri bağlama özelliği sağlar.

- Kodlar çok sade ve anlaşılır olduğu için Birim Testi (Unit Test) yapılması uygundur.

- Daha az kod ile daha çok iş elde edilebilir.

- JavaScript ile yazılmış kodlar AngularJS'de denetleyici işlemleri sağlar.

AngularJS iki yönlü veri bağlama özelliği olduğu için Şekil 2.4.'de görüldüğü gibi senkronize olma özelliğine sahip olur.



Şekil 2.4. AngularJS Model - Görünüm Senkronizasyonu

BÖLÜM 3. YÖNTEM

Bu bölümde, bu tezde yapılan çalışmanın yöntemleri anlatılmaktadır. Kod şablonlarına dayalı kod üretme mekanizmanın temel gereksinimlerinden bahsedildikten sonra altyapı mimarisi anlatılmakta ve devamında diğer bileşenlerinden değinilmektedir.

Kod şablonlarına dayalı kod üretimi mekanziması geliştirici verimliliğini ve kod kalitesini arttırır. Örneğin geliştiricinin belirlemiş olduğu HTML kalıplarındaki alana veri tipleri yerleştirerek istenilen kod üretiminin gerçekleştirilmesi sağlanır. Bu esnasında tekrar gerektiren kodlar da şablonlarına dayalı olarak otomatik olarak üretilir. Şablonlara dayalı kod üretimi mekanizmasının temel gereksinim listesi aşağıda verilmiştir.

- Kaynak ve veri girişleri iyi tanımlanmış söz dizimine sahip olmalıdır.

- Üretilen kodu doğrulamak için işlevler ve argümanlar olmalıdır.

- Üretilen kodlar iyi yapılandırılmış, iyi belgelenmiş ve orijinal belirtime göre kolayca izlenebilir olmalıdır.

- Sorunsuz bir kod üretimi için gerekli işlevler ve argümanlar olmalıdır.

- İlgili veritabanındaki işlemler için CRUD saklı yordamları olmalıdır.

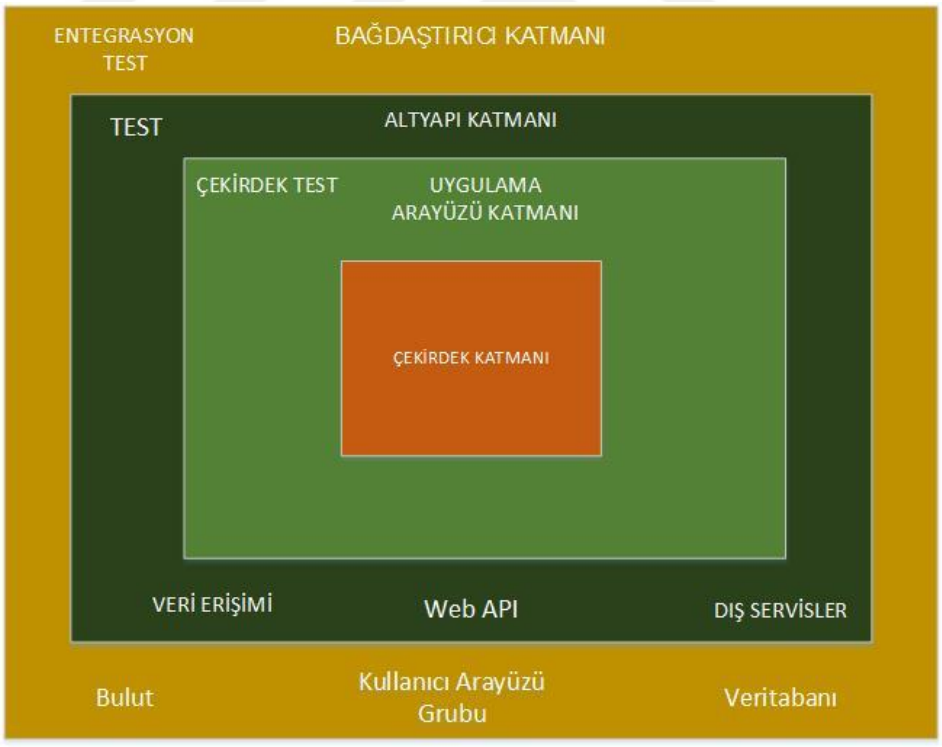
- Belirlenmiş kalıplarının kod hatası olmamalıdır.

- Eklentiler, JavaScript ve HTML kütüphanesi olmalıdır.

Otomatik kod üretim işlemi geliştiricinin üretmek istediği koda (arayüze) ilişkin bir tablo oluşturması ile başlar. Devamında mekanizma tarafaından otomaik olarak bu tabloya göre saklı yordamlar oluşturulur. Sonrasında yine otomatik olarak JSON şemaları oluşturulur. Bu arada geliştirici arayüz için bir kalıp seçer. Daha sonra mekanizma seçilen kalıba ve oluşturulmuş olan JSON şemalarına göre otomatik olarak HTML üretir. Bu bölümün devam eden alt bölümlerinde bu çalışma sistemi detaylı bir şekilde anlatılacaktır. Bu aşamalar Şekil 1.1.'de gösterilmektedir.

3.1. TÜSSİDE Bütünleşik Yönetim Sistemi'nin Altyapısı

Bu tezde yapılan otomatik kod üretimi mekanizması TBYS içinde kullanılmıştır. Mekanizmanın daha iyi anlaşılabilmesi için bu bölümde kısaca TBYS mimarisinden bahsedilecektir. Otomatik kod üretimi mekaniması için TÜSSİDE Bütünleşik Yönetim Sistemi'nin altyapı mimarisi modeli Şekil 3.1.'de gösterilmektedir.



Şekil 3.1. TÜSSİDE Bütünleşik Yönetim Sistemi'nin Altyapı Mimarisi

Bu çalışmada farklı katmanlar birbirlerinden ayrıldı. Ayırma işlemi her bir katmana kendi .NET projesini vererek ve referanslarını kontrol ederek yerine getirildi. Her

.NET projesi ayrı katmanını üretmektedir. Katman referansları, refere edilen katmanın genel arayüz ve sınıflarını kullanma araçlarıdır.

- Bağdaştırcı katmanı: Entegrasyon testi, kullanıcı arayüz grubu, bulut ve veritabanı bulunan katmandır. Arayüz HTTP(S) üzerinden altyapı katmanları API'sini çağırır, gösterim için gerekli verileri alır ve kullanıcı arayüzü üzerinden işlem yaptığında da WebAPI'ye verilen komutu izler. Kullanıcının gördüğü ekran ve veri doğrulama yapılan bölüm kullanıcı arayüz grubudur. Entegrasyon testi, birbirinden bağımsız olarak test edilmiş olan iki yazılım ürününün birbirine bağlanması, birbiri arasında veri aktarımının sağlanması için birleştirilen modüllerin test edilmesidir. Entegrasyon testinin amacı bu birleşim sırasında ortaya çıkabilecek hataları gidererek sorunsuz birleşimi sağlamaktır.

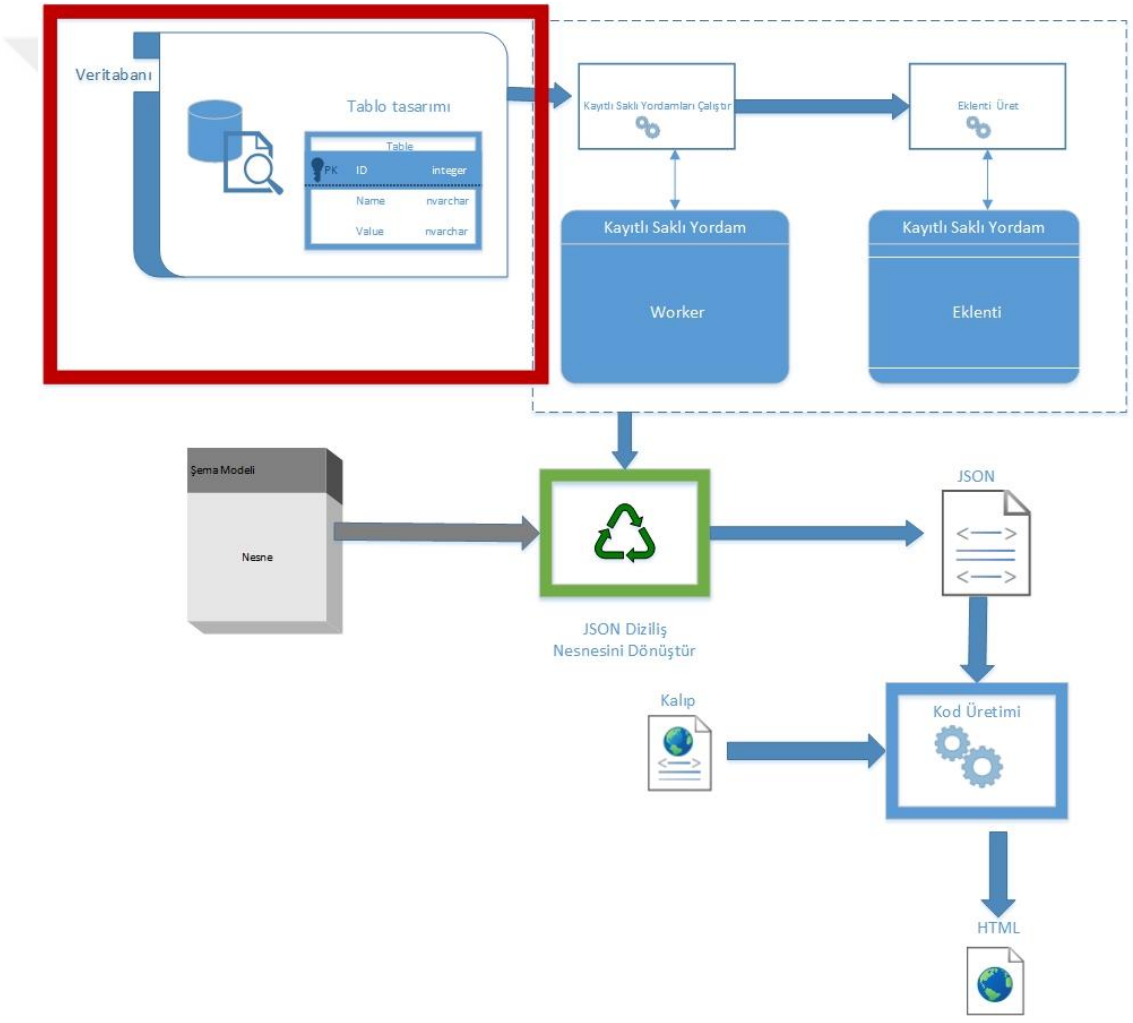
- Altyapı katmanı: Dış servisler, veri erişimi, test ve Web API bulunan katmandır. Test, birim testini kolaylaştırır ve aynı kullanıcı ismi ile tek bir komutla gerçekleştirir. Aynı zamanda mevcut kullanıcı adı ve mevcut zaman bilgisini sunmak için yardımcı sınıfı bu katmana tanıtılır. Çekirdek ve uygulama hizmetleri altyapı katmanına bağlanır. Bu katmanda tüm arayüzlere konfigürasyon tarafından uygulama sınıfları verilir ve uygulama arayüzü tarafından talep edildiğinde barındırıcı bu bilgi yardımıyla doğru bilgiyi sunabilir.

- Uygulama arayüzü katmanı: Çekirdek test alanını barındıran katmandır. Çekirdek test alanı bu katmanda yer alır ve program çalıştırılırken doğal olarak atlanır. Uygulama arayüzü katmanının ana sorumluluğu, birbirine bağlantılı nesnelerin bir araya gelerek oluşturduğu büyük yazılım modülünü kolayca kullanılabilir birimler halinde paketlemek ve ek olarak iş biriminin kapsamını da yönetmektir. Bunun anlamı da gerekli veritabanı bağlantılarını yapmak, veritabanı işlemlerini gerçekleştirmek, bağlantılı nesnelerin bir araya gelerek oluşturduğu büyük yazılım parçasını almak, bu büyük parçanın etki alanı

operasyonlarını çağırmak, değişmiş durumu kalıcı ortama kaydetmek ve değişimleri işlemek, son olarak da kullanılan kaynakları serbest bırakmaktır.

- Çekirdek katmanı: Bu katman tüm iş hususlarını ilişkili varlık sınıflarının yardımıyla bağlamayı sağlayan bağlantı nesnelerinin bir araya gelerek oluşturduğu büyük yazılım parçasını içermektedir. Çekirdek katmanı, tıpkı Şekil 3.1.'te de görüldüğü üzere alanı bulunmayan tek katmandır.

3.2. Veritabanı Tablo Tasarımı



Şekil 3.2. Veritabanından HTML kod dönüşüm süreci – İlk aşama

Otomatik kod üretimin ilk aşaması Şekil 3.2.'de kırmızı ile çerçevelenmiş alanda gösterilmektedir. Kod üretimini gerçekleştirirken yeni bir arayüz oluşturmak için

(HTML kodu) ilk aşamada yeni bir tablo eklemek gerekir. Devamında bu eklenen tabloya bağlı modüller üretilir. Bu aşamada oluşturulan tablolarının verileri sorunsuz bir şeklinde uygulamayla veri alışverişi yapabilmek amacıyla kullanılır. Örnek bir tablo Şekil 3.3.'de gösterilmektedir. Tabloda depolamak istediğiniz her özellik için şunları tanımlamanız gerekir:

- Sütun Adı (Column Name): Otomatik olarak oluşturulacak olan arayüzde (HTML) bulunması istenilen özelliklerin isimleri sütun adı kısmına yazılır.

- Veri Türü (Data Type): İlgili özelliklerin değerelerinin veri tipi burada belirlenir.

- Boş Değerlere İzin Ver (Allow Nulls): İlgili özelliğe boş değer verilip verilemeyeceği bildirilir.

Column Name	Data Type	Allow Nulls
FaultRequestID	int	☐
FaultJobRequestTypeID	int	☑
FaultRequestUser	int	☑
FaultRequestUserDepart...	int	☑
FaultRequestTypeID	int	☑
FaultRequestDescription	nvarchar(MAX)	☑
FaultRequestDate	datetime	☑
FaultAppointedUser	int	☑
FaultAppointedDate	datetime	☑
FaultAppointedDeadline...	datetime	☑
FaultNecessarySale	bit	☑
FixBeforeExplanation	nvarchar(MAX)	☑
FixAfterExplanation	nvarchar(MAX)	☑
WorkBeginDate	datetime	☑
WorkEndDate	datetime	☑
FaultRequestStatusID	int	☑
LockStatus	bit	☑
Status	tinyint	☑
CreateDate	datetime	☑
KullaniciID	uniqueidentifier	☑
CreateUserID	uniqueidentifier	☑

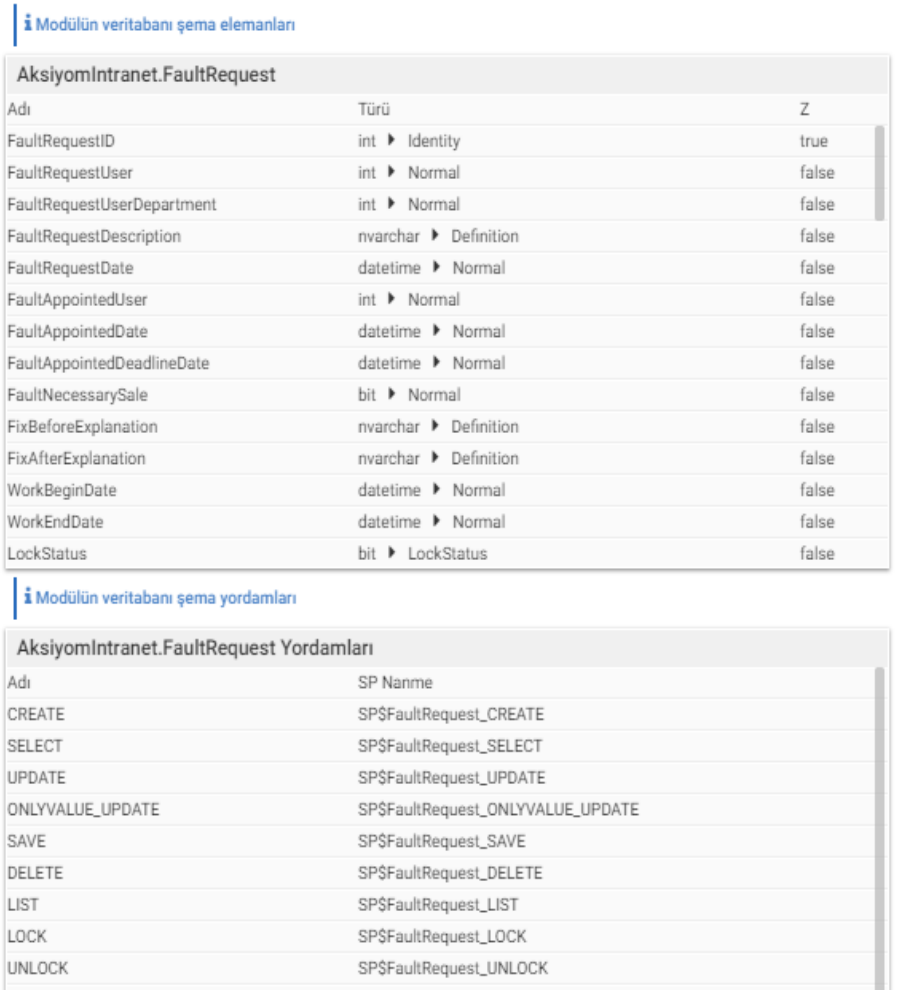
Şekil 3.3. Veritabanı Tablo Özellik Listesi

Örneğin; Bir izin formu oluşturmak istenildiği zaman izin formunu oluşturmak için tablodan izin tablosunu oluşturmamız gerekir. Bu izin tablosunun "sütun adı" sütunun içerisinde olması gereken parametreler; izin başlangıç tarihi, izin bitiş tarihi, açıklama… vb. parametrelerdir. Bu parametrelerin "veri türü" sütununda veri türlerini tanımlanır. Tarih parametleri için olarak tanımlanan veri tipi "date" veya açıklama parametreleri için tanımlanan veri tipi "nvarchar(max)" seçilir. "Boş değerlere izin ver" sütununda geliştiricinin uygun gördüğü şekilde işaretleme yapılır. Yani bir izin formunu doldururken tarih kısmına boş değer verilmeyeceğinden dolayı için boş değerlere izin ver kutusu işaretlenilmez.

Oluşturulan tablolarda birincil anahtar alnına isim vermek için özel bir isimlendirme kuralı kullanılmalı. Birincil anahtar alanının ismi tablo isminin ve "ID" kelimesinin birleşiminden oluşmalıdır. Kod mekanizması tarafından üretilen arayüzün veritabanı ile sorunsuz bir şekilde veri alışverişi yapabilmesi için bu böyle olmalıdır. Örneğin tablo ismi FaultRequest ise birincil anahtar sütununun ismi FaultRequestID olmalıdır.

3.3. Saklı Yordam Üretimi

Geliştiricinin veritabanından herhangi bir işlem yapabilmesi için saklı yordama ihtiyaç duyulur. Çünkü yapacak olduğu her işlem tekrar gerekeceğinden hazır üretilmiş saklı yordam kullanılarak oluşacak olan iş yükü hafifletilir. Aynı zamanda gelişitirme hızı artar. Bu tezde sunulan çalışmada Şekil 3.4.'de görüldüğü gibi saklı yordamlar kullanılmıştır.



Modülün veritabanı şema elemanları

AksiyomIntranet.FaultRequest

Adı	Türü	Z
FaultRequestID	int ▸ Identity	true
FaultRequestUser	int ▸ Normal	false
FaultRequestUserDepartment	int ▸ Normal	false
FaultRequestDescription	nvarchar ▸ Definition	false
FaultRequestDate	datetime ▸ Normal	false
FaultAppointedUser	int ▸ Normal	false
FaultAppointedDate	datetime ▸ Normal	false
FaultAppointedDeadlineDate	datetime ▸ Normal	false
FaultNecessarySale	bit ▸ Normal	false
FixBeforeExplanation	nvarchar ▸ Definition	false
FixAfterExplanation	nvarchar ▸ Definition	false
WorkBeginDate	datetime ▸ Normal	false
WorkEndDate	datetime ▸ Normal	false
LockStatus	bit ▸ LockStatus	false

Modülün veritabanı şema yordamları

AksiyomIntranet.FaultRequest Yordamları

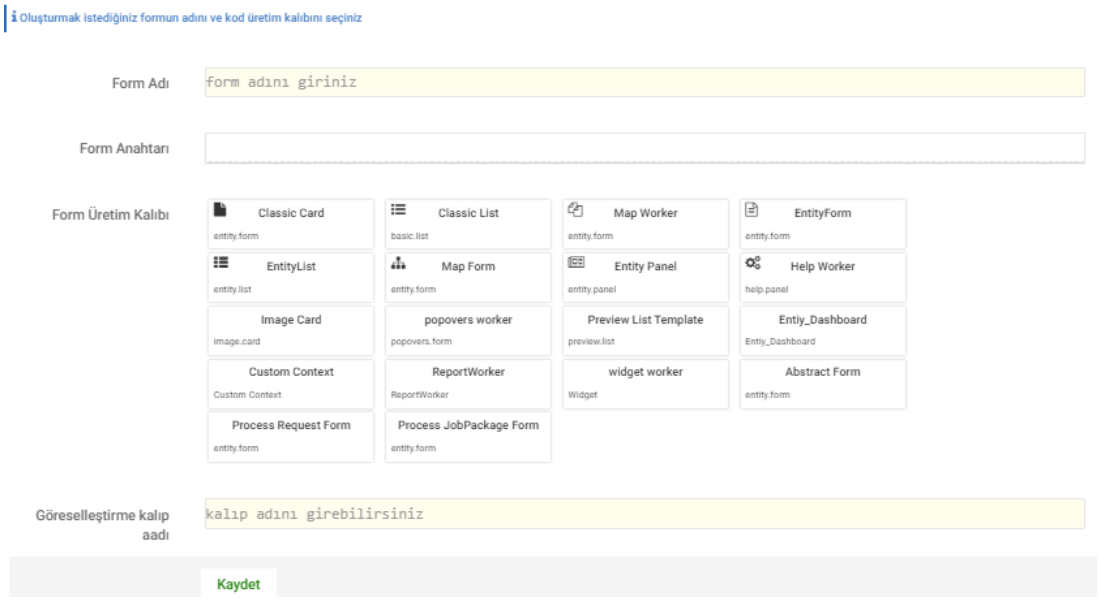
Adı	SP Nanme
CREATE	SP$FaultRequest_CREATE
SELECT	SP$FaultRequest_SELECT
UPDATE	SP$FaultRequest_UPDATE
ONLYVALUE_UPDATE	SP$FaultRequest_ONLYVALUE_UPDATE
SAVE	SP$FaultRequest_SAVE
DELETE	SP$FaultRequest_DELETE
LIST	SP$FaultRequest_LIST
LOCK	SP$FaultRequest_LOCK
UNLOCK	SP$FaultRequest_UNLOCK

Şekil 3.4. Üretilmiş Saklı Yordamlar

Kod üretiminin gerçekleşmesi için oluşturulan tablodaki sütun adı bilgisinden faydalanılır. Bu bilgi, ekran üzerindeki alanları, aksiyonların gerçekleştirilmesi için var olan düğmeleri ve eklentileri ifade ederler. İlgili tabloya özel saklı yordmlar bu bilgiler yardımı ile otomatik olarak oluşturulur (Şekil 3.4.). Bu saklı yordamlar birçok görevi yerine getirmelerinin yanı sıra özel olarak CRUD işlevlerini yerine getirmek için de oluşturulur. Otomatik kod üretimi sisteminde CRUD doğru bir şekilde tanımlanmalıdır.

Saklı yordamlar görevini yerine getirirken öncesinde tanımladığımız veri değişkenlerinin gereksinimine göre veritabanından kayıtları çağırır. Çağırılan kayıtlar, Şekil 3.5.'te verilen kalıp biçimlerine göre arayüz tasarımında belirlenmiş kalıplardır.

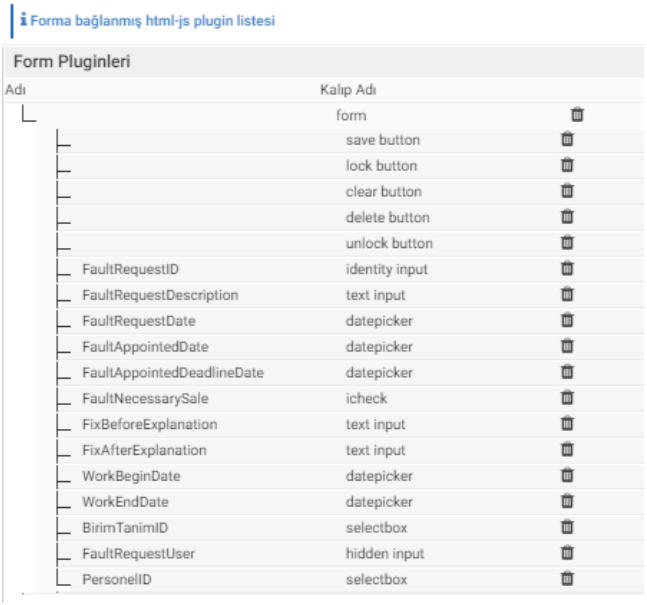
Kalıp kavramı ileride HTML bölümünde açıklanmıştır. Kod üretimi sürecindeki gerekli tüm adreslemeler belirlenmiş veri tipleriyle yapılır. Aynı zamanda HTML kalıp dosyalarının oluşturulmuş olması gerekir. Bu kapsamda, kullanıcı arayüzü ekranında hangi menünün altında olması gerektiği, aksiyonların ve saklı yordamların neler olduğu vb. bilgiler önceden belirtilmiş olmalıdır.



Şekil 3.5. Kalıp Seçme Ekranı

3.4. Eklenti Üretimi

Eklenti, yazılım programı sürecini hızlandıran bir yazılım paketidir. Eklentiler üretilmediğinde uygulama kodları beklenildiğinden daha uzun ve karmaşık olabilir. Zaman tasarufu ve basitliği hedeflediğimiz için eklentileri oluşturmamız gerekir. Eklenti kullanıldığı zaman kodun kapladığı alan küçülür. Kod, anlaşılır, kaliteli ve verimli bir hale gelir. Bu tezde yapılan çalışmada eklentiler veritabanındaki tabloların sütun bilgilerine göre uygun eklentiler üretilir.



Şekil 3.6. Üretilmiş Eklenti Listesi

Kod üretimi sisteminde oluşturulan eklentiler bir klasörde bulunur. Bu klasörde eklentilere ait kaynak dosyaları bulunur. Bu kaynak dosyalarda ise HTML ve JavaScript kodları bulunur.

Eklentiler, kullanıcı arayüzünde bulunan HTML bileşenlerinin (button, textbox vb.) tasarımını ve işlevlerini tanımlar. Eklentiler hem işlev parametreleri hem de nesne sınıflarının özellikleri için kullanılabilirler. Belirlenmiş olan işlevlerin kodsal karşılığı olması gerekir.

Geliştirici önceden her bir kalıbın olaylarını oluşturur, belirler. Daha sonra otomatik kod üretiminin eklenti üretimi süreci esnasında hem eklentiler üretilir hem de bu olaylara ilişkin aksiyon işlevleri oluşturulur. Şekil 3.6.'da örnek eklentiler görülmektedir. Eklenti üretim sisteminin çalışma şekli aşağıda anlatılmıştır.

İlk adımda geliştiricinin seçtiği HTML kalıbına göre eklentiler üretilir. Örneğin HTML kalıbı form olarak tanımlandıysa "Kaydet", "Sil" vb. şeklinde; HTML kalıbı liste olarak tanımlandıysa "sayfalama", "listeleme" vb. şeklinde eklentiler üretilir. Üretilen eklentiler, tablo ismi ve bileşen ismi birleştirilerek isimlendirilir. Örneğin tablo ismi "TabloIzin" ve düğme ismi "kaydet" ise "TabloIzin_kaydet" ismi ile bir eklenti üretilir.

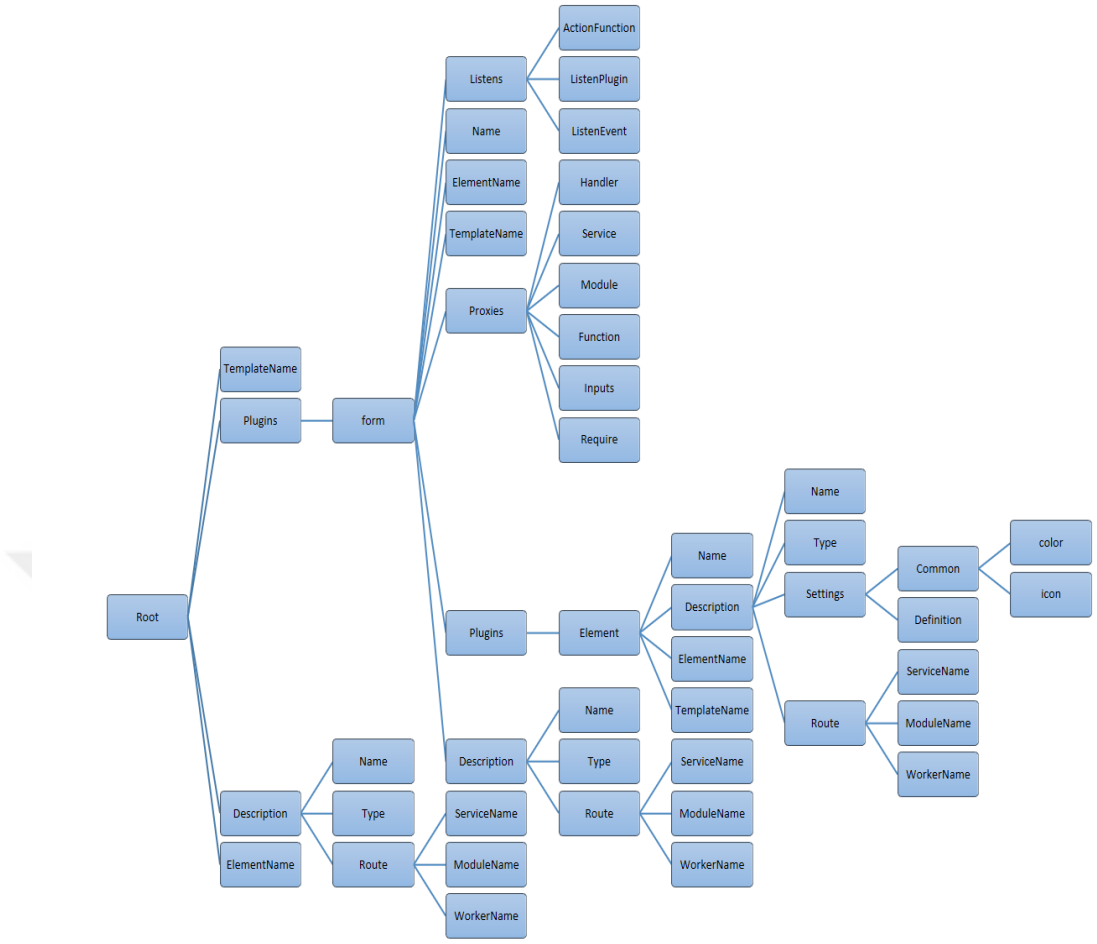
İkinci adımda veri tiplerine göre uygun HTML bileşenlerinin eklentileri üretilir. Yani veri tipi "string" formatındaysa "input" ya da veri tipi "date" ise "datetimepicker" olarak gösterilmektedir. Üretilen eklentiler, sütun isimleri ile bileşen isimleri birleştirilerek isimlendirilir. Örneğin tablodaki sütun ismi "adres" ve veri tipi "string" ise "adres_input" adında eklenti üretilir.

3.5. JSON Şeması ve Üretimi

JSON, veri depolamak ve verileri farklı platformlara taşımak için günümüzde yaygın olarak kullanılmaktadır. Bu tezde anlatılan çalışmada da JSON tercih edilmiştir. JSON tercih edilmesinin nedeni aşağıda belirtilmiştir.

- Sadelik.
- Hem veri modelini hem de kullanıcı ara yüzünü açıklar.
- Geliştirici, ilgili kod üretimini verimli ve basit bir şekilde geliştirilebilir.
- Tanımlama yapılırken kendini tekrar eden kodların tekraren yazılmasına ihtiyaç duyulmaz.
- Alan geliştirilebilir. Yeni kodlar üretilerek yeni biçimle üretilmiş bir HTML kodu oluşturabilir.
- JSON şemalarını liste görünümlü veya form görünümlü olabilir.

JSON, alt yapı ile uygulama arasında veri alışverişinin yapılmasını sağlar. Sistem, bileşen oluşturmak istediği her tablo için bir JSON dosyası oluşturur. Çünkü liste ve form görünümleri için sunucudan veri almak adına gerekli alt yapıyı oluşturulmak gerekir.



Şekil 3.7. JSON Şema Yapısı

Bu tezde yapılan çalışmada kullanılan JSON şemasının üç ana bileşeni Şekil 3.7.'de gösterilmektedir: Description (Açıklama), TemplateName (KalıpAdı) ve ElementName (ElemanAdı). Bunlardan farklı tanımlanması gereken bir özelliğe ihtiyaç duyulursa yeni bir ara bileşen tanımlanır. Böylece üretilebileceği bileşenler basit ve alt dizeler halinde oluşabilir. JSON yardımı ile bileşenlerin ve alt gruplarının karmaşık biçimleri daha basit, anlaşılır ve yönetilebilir şekilde gruplandırabilir. Yukarıda verilen JSON şemasını kullanarak web uygulamasının JSON bileşenleri geliştirici tarafından oluşturulur. Geliştirici, elde edilen uygulamayı üretmek için JSON ve kalıp dosyalarını kullanarak giriş modelini elde eder. Bazı veri girdisi (ör. servis adı, worker name) dışında, JSON şeması için yeni bir ek bilgi gerekmez. Sisteme yeni bir bileşen eklenmesi gerekiyorsa şemaya uygun bileşen olması gerekir. Bu kapsamda, JSON şema modeline yeni bileşen ekleme yapılabilir.

```
}, {
    "Name": "NewsRequest_delete",
    "Description": {
        "Name": "button",
        "Type": "delete",
        "Settings": {
            "Common": {
                "color": "btn-default"
            },
            "Definition": {}
        },
        "Route": {
            "ServiceName": "AksiyomIntranet",
            "ModuleName": "NewsRequest",
            "WorkerName": "yayntalepformu_10_07_2017_www"
        }
    },
    "TemplateName": "button",
    "ElementName": "NewsRequest_delete"
```

Şekil 3.8. JSON kalıba göre oluşturan kod bölümü

JSON dosyası, HTML'e sorunsuz dönüşüm için gerekli JavaScript işlev adları içerir. Örneğin; Şekil 3.8.'de verilen kodda ("Name":"NewsRequest_delete") JavaScript işlevin adıdır. JSON dosyası aynı zamanda model sınıfı için arka planda RESTful bağlantı bilgilerini içerir. Örneğin; Şekil 3.8.'de verilen ("Route") ve alt satırında bulunan ("ServiceName", "ModuleName", "WorkerName") bağlantı bilgileridir. Bu bilgiler birleştirilerek yönledirme bilgisi (route information) oluşturulur. Bu bağlantı verilerin modelinde bulunur.

```
{
 "Name": "NewsRequestTitlle_input",
 "Description": {
     "Name": "input",
     "Type": "string",
     "Settings": {
         "Common": {
             "icon": "fa-pencil-square-o ",
             "icon_type": "input"
         },
         "Definition": {}
     },
     "Route": {
         "ServiceName": "AksiyomIntranet",
         "ModuleName": "NewsRequest",
         "WorkerName": "yayntalepformu_10_07_2017_www"
     }
 },
 "TemplateName": "input",
 "ElementName": "NewsRequestTitlle"
```

Şekil 3.9. JSON veritabanına göre oluşturan kod bölümü

JSON dosyasındaki ilk anahtar ID'dir. Son olarak her model nesnesi, veri tiplerine ve ilk anahtarına bağlı olarak veri nesnesinin URI'sini oluşur. Şekil 3.8.'de JSON

kalıbındaki kodun bir bölümü gösterilmektedir. HTML kalıp biçimlerine göre çeşitli standartlaştırılmış düğmeler mevcuttur. Bu örnekte geliştiricinin belirlediği kalıbın standartlaştırılmış üç farklı düğmesi vardır: sil, ekle ve temizle. Şekil 3.8.'de görülen tanımlanmış ("Name" : "button") düğmeyi ifade eder. Bu düğme ise silme türünde bir düğmedir ("Type" : "delete"). Şekil 3.9.'da ise JSON formatında oluşturulmuş kodun son halinin bir parçasını gösterilmektedir. Veritabanında tanımlı olan ("Name": "input") girdiyi ifade eder. Bu girdi veritabanında ("Type" : "string") ile tanımlıdır. Bu girdi değeri için veritabanından gerekli saklı yordamları, veri tiplerini ve eklentilerini çağrılarak JSON formatına dönüştürlür. Belirli bir JSON kalbının içerisine veritabanından okunan bilgilere göre eklemeler yapılarak nihai JSON dosyasını oluşturulur.JSON dosyalarını, yukarıda bahsedilen bir liste görünümü sayfası ve form görünümü sayfası içeren modüller, bağımsız bileşenler üretir.

3.6. HTML Kod Üretimi

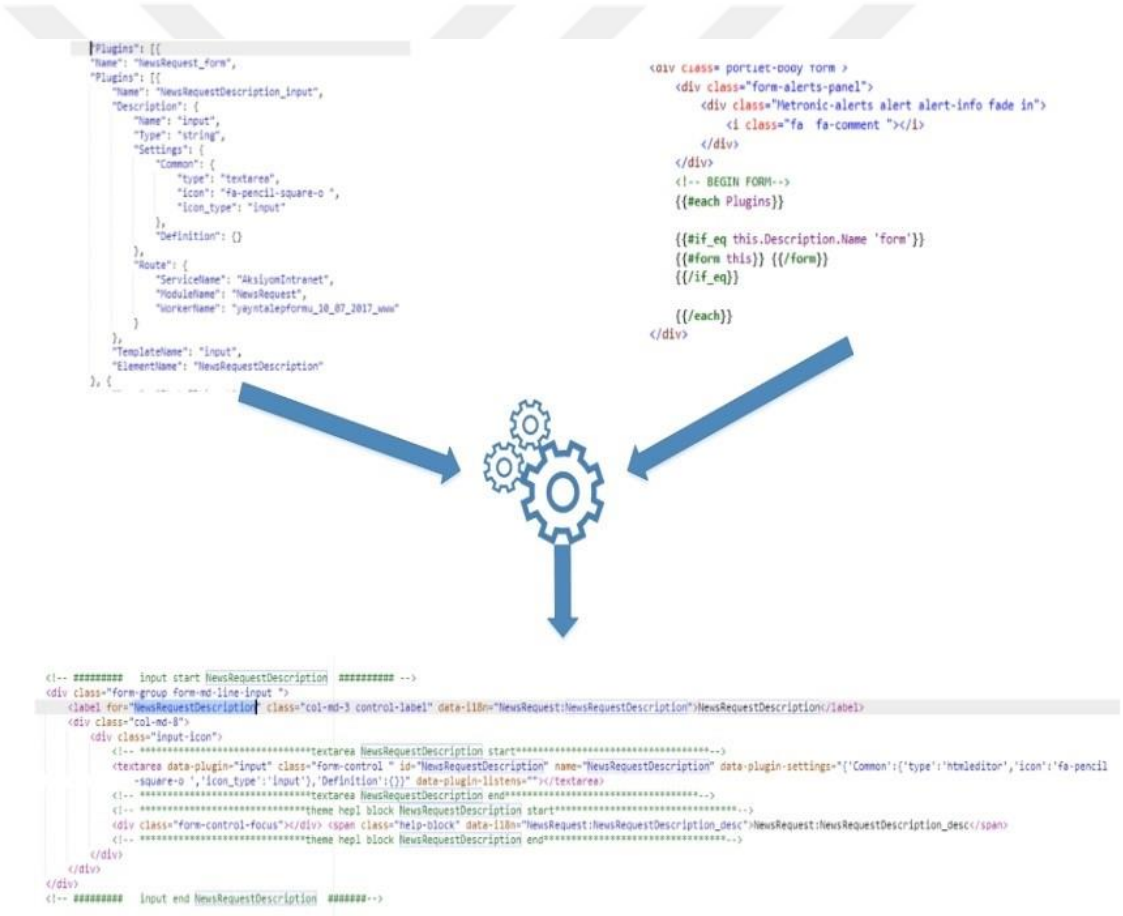
HTML belgelerin birbirlerine nasıl bağlanacaklarını ve belge içindeki metin ve resimlerin nasıl yerleşeceklerini belirleyen ve etiket (tag) denilen kod parçalarından oluşan bir sistemdir. Şekil 3.10.'da örnek bir HTML kalıp biçimi gösterilmektedir. Bu örnekte gösterilen kalıp, ekran aksiyonlarının ve gösterimde olan düğmenin HTML formatında kod üretimi gerçekleştirilmesi için kullanılan kalıptır. Bu kalıba göre düğme bileşeni için veritabanındaki kayıtlar baz alınarak aksiyon işlevleri ve eklentiler otomatik olarak üretilir. "{{#eachPlugins}}" satırı bu düğme için tanımladığımız eklentiyi çağırır. Bu düğme için "{#if_eqthis.Description.Name'form'}}" satırı eklenti eşlemesini yapar. Doğruluk kontrolüyle birlikte HandlebarsJS temel değişken referanslarına veritabanından tanımlanmış değişken ekler. Bu formatla ve bu eklentiye ait her düğme için tekrardan kodları yazmaya gerek kalmadan "{{#eachthis.Plugins}}" komutuyla önceki HTML kod mekanizmasını bozmadan altına ekleme yaparak üretir yeni kod üretilir.

```
<div class="actions action-form-buttonset">
    {{#each Plugins}}
    {{#if_eq this.Description.Name 'form'}}
    <div class="btn-group">
        {{#each this.Plugins}}
        {{#if_eq this.Description.Name 'button'}}
        {{#button this}} {{/button}}
        {{/if_eq}}
        {{/each}}
    </div>
    {{/if_eq}}
    {{/each}}
</div>
```

Şekil 3.10. HTML Kalıp Formatı

Bu aşamada, oluşturulan JSON ve eklenti dosyasındaki değişken veri tipleri çözümlenir. Kalıptaki bağımsız değişken ayıklanıp bir işleve dönüştürülür. ASP.NET Web API bu kalıptaki isteklere göre oluşturulur. Bu istek parametreleri kalıptaki değişkenlerden çıkarılır ve URL ile birlikte istek işlevine iletilir. Üretim sürecinde öncelikle kalıp oluşturmak gerekir. Nasıl bir kalıp seçmesi gerektiğini geliştirici belirlemelidir. Kalıp seçme ekranı Şekil 3.5.'de gösterilmektedir. Belirlenen kalıp arayüzü temsil eder. Bu kalıpın verilerini, değişkenlerini tanımlayabilmek için

ASP.NET Web API'leri tanımlanmış olmalıdır. Bu işlem yapılırken HandlebarsJS'den (mustache) yararlanılmaktadır. HandlebarsJS, HTML kodun oluşturulduğu sırada dosyaların oluşturulduğu kodu belirler. Ayrıca kalıplardaki değişken içeriğinin nereye yerleştirilmesi gerektiğini de gösterir. Kalıptaki bağlaçlar "{{ … }}" değişkeni temsil eden kod bölümüdür. JSON'un nesneleri, HTML kalıplarının bağlaçlarındaki değişkenlere tanıtılmalıdır. HTML kodun oluşturma aşamasında geçici olarak dosyada tuttuğumuz eklentileri ve JSON dosyaları HTML kalıplarına eşlenerek HTML kod üretimi gerçekleşir (Şekil 3.11.). Her nesne grubu " #each" döngüsü içerisinde devam ederek bir önceki kod satırın altına eklenir. Bunun sonucunda derlenmiş ve üretilmiş uygulama ortamında kullanılma hazır HTML dosyası oluşturur.



Şekil 3.11. HTML Kod Üretim Süreci

Her ne kadar otomatik kod üretimi yapılsa da geliştiriciler manuel olarak da sistemde düzenleme yapmak isteyebilirler. Küçük düzenlemelere izin verilmektedir. Bu durumda manuel olarak kodlanan kodlar için öncelikli olarak ASP.NET Web API isteği

tanımlanmalıdır. Sunucu tarafından gelen veri ile istemci tarafındaki veri eşlenmelidir. Bu durumda RESTful bağlantı bilgilerinden faydalanılmalıdır. Veri dönüşümlerinde dikkat edilmesi gereken nokta verilerin hangisinin alıcı, hangisinin verici olacağına karar vererek ilgili kalıplarda belirtilmelidir. Veri dönüşümü yapıldıktan sonra belirlenen kalıbı tekrar düzenlemek mümkün değildir.

BÖLÜM 4. SONUÇ

Bu tez çalışması çerçevesinde "kod şablonlarına dayalı otomatik kod üretimi mekanizması" tasarlanmış ve geliştirilmiştir. Geliştirilen mekanizma web uygulamaları için önceden belirlenmiş olan şablolara göre otomatik olarak kod üretimi yapmakta ve dolayısıyla ilgili arayüzler ve işlevleri otomatik olarak oluşmaktadır. Geliştirilen sistemin avantajları ve kazanımları aşağıda açıklanmıştır.

- Kod üretme sistemlerinde otomatik kod üretilir. Dolayısıyla tekrar kod yazmaya gerek kalmaz. Bu çalışmada geliştirilen sistemde de bu anlmada geliştirici verimliliği artırılmıştır. Benzer şekilde saklı yordamlar, kullanıcı ara yüz kodları ve uygulama sunucu kodları gibi tüm parçalar otomatik olarak üretilmesi de bir avantajıdır.

- Geliştirilme sürecinde sistemin katmanlarının kolay geliştirebilir olması önemli bir özelliğidir. Otomatik kod üretiminde geliştirciden kaynaklı hatalar engellendiği için hata oranının en az seviyede olur.

- Ek olarak standartların önceden tanımlanmış olması gerekir. Dolayısıyla kurumun iş süreçleri henüz projeye başlamadan standartlaştırılmış olur. Bunun yanı sıra standartlara uygun uygulama geliştirilmiş olur.

Otomatik kod üretimi sisteminin bir dezavantajı ise, belirlenmiş kalıpların dışında bir kalıp yapılmasının zor olmasıdır. Böyle bir durumda gereken eklentiler, kodlar vb. geliştirici tarafından manuel olarak hazırlanması gerekir.

KAYNAKLAR

[1] G. Young, CQRS Documents by Greg Young, https://cqrs.files.wordpress.com/2010/11/cqrs_documents.pdf Erişim Tarihi:10.10.2018.

[2] H. Kovanen, Infrastructure and asset management using modern web-technologies, JAMK University of Applied Sciences, Finlandiya, 2014.

[3] G. S. Benato, F. J. Affonso ve E. Y. Nakagawa, Infrastructure Based on Template Engines for Automatic Generation of Source Code for Self-adaptive Software Domain, 2017.

[4] J. P. Alfonso ve F. R. Calle, Automatic Source Code Generation for Web-based Process-oriented, pp. 103-113, 2017.

[5] C. Akdemir, HLA Nesne Model Şablonu Temelli FOM-ÇEVİK Kod Üretimi, İstanbul Teknik Üniversitesi: Fen bilimleri Enstitüsü, 2009.

[6] J. P. A. Hoyos ve F. R.-C. , Automatic Source Code Generation for Web-based Process-oriented Information Systems, In Proceedings of the 12th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2017), pp. 103-113, 2017.

[7] M. Livraghi , Automatic Generation of Web CRUD Applications, Politecnico Di Milano: Master of Science in Computer Engineering, 2016.

[8] Z. Zhang, An Automatic Source Code Generation Tool for OLTP Database Benchmarks,http://cs.brown.edu/research/pubs/theses/masters/2010/zhang.pdf. Erişim Tarihi:29 Temmuz 2018.

[9] iMatix, Template-based Code Generation https://imatix-legacy.github.io/twp/codegen.pdf. Erişim Tarihi:6 Eylül 2018.

[10] M. C. Franky ve J. A. Pavlich-Mariscal, Improving Implementation of Code Generators: A Regular-Expression Approach, Pontificia Universidad Javeriana Bogotá, Colombia, pp. 978-1-4673-0793-2/12/ ©2012 IEEE., 2012.

[11] A.Akbulut, F. P. Akbulut, H. Köseokur ve Ç. Çatal, Son Kullanıcı Geliştirme için Otomatik Kod Üretim Aracının Tasarımı ve Gerçeklenmesi, Dokuz Eylül Üniversitesi Fen ve Mühendislik Dergisi, cilt 19, no. 55.1, pp. 76-88, 2017.

[12] H. Ç. Altıparmak, B. Tokgöz, Ö. E. Balçiçek, A. Arslan ve A. Özkaya, Source Code Generation for Larger Scale Application, IEEE International Conference on Technological Advances in Electrical,Electronics and Computer Engineering (TAEECE), pp. 404-410, 2013.

[13] Ö. E. Balçiçek, B. Tokgöz ve H. Ç. Altıparmak, İş Gereksinimi Odaklı Kaynak Kod Üretme Sistemi, 20 Eylül 2016

[14] F. S. Farooji, Evaluation of Code Generation Tools, Stockholm: KTH Information and Communication Systems, 2014.

[15] C. Lemaire, Parsing and Code Generation Techniques to Deal with Uncertainty: Experıences from Highly-Evolving and Complex Systems, 5th Workshop on Domain-Specific Modeling, 2005.

[16] Apache Velcity Engine, http://velocity.apache.org/engine/. Erişim Tarihi:1 Eylül 2018.

[17] SQL Stored Procedures for SQL Server, w3schools.com, https://www.w3schools.com/sql/sql_stored_procedures.asp Erişim Tarihi:5 Eylül 2018.

[18] What is CRUD?, Codecademy, https://www.codecademy.com/articles/what-is-crud Erişim Tarihi:27 Ağustos 2018.

[19] H. Kilov , From semantic to object-oriented data modeling, Bell Communications Research, cilt 1E, no. 243, pp. 385-393, 1990.

[20] Learn REST: A RESTful Tutorial, REST API TUTORIAL, https://www.restapitutorial.com/ Erişim Tarihi:2 Eylül 2018.

[21] Introducing JSON, https://www.json.org/ Erişim Tarihi:24 Ağustos 2018.

[22] Handlebars, https://handlebarsjs.com/ Erişim Tarihi:25 Ağustos 2018.

[23] ASP.NET genel bakış, Microsoft, https://docs.microsoft.com/tr-tr/aspnet/overview Erişim Tarihi:2 Eylül 2018.

[24] AngularJS, https://angularjs.org/ Erişim Tarihi:1 Eylül 2018.

ÖZGEÇMİŞ

Burak UYANIK, 03.01.1988'de İzmir'de doğdu. İlk, orta ve lise eğitimini Eskişehir'de tamamladı.Anadolu Üniversitesi Bilgisayar Teknolojisi ve Programlama Bölümü'nü 2007 de mezun oldu. 2008 yılında Dikey geçiş ile Bilgisayar Mühendisliği Bölümü'nü 2011 yılında bitirdi. 2011 yılında KoçSistem Ar-Ge Mühendisi olarak çalışmaya başladı.2014 yılında KoçSistem'den ayrılıp TÜBİTAK TÜSSİDE Yazılım Mühendisi olarak çalışmaya başladı.2015 yılında Sakarya Üniversitesi Bilgisayar ve Bilişim Mühendisliği Bölümü'nde yüksek lisans eğitimine başladı.

Halen TÜBİTAK TÜSSİDE araştırmacı olarak görev yapmaktadır.