T.C.

YILDIZ TEKNİK ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

MODEL GÜDÜMLÜ GELİŞTİRME YAKLAŞIMI İLE OTOMATİK KOD ÜRETİMİ ARAÇLARININ KARŞILAŞTIRILMASI

Büşra İÇÖZ

YÜKSEK LİSANS TEZİ

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

Danışman

Prof. Dr. Oya KALIPSIZ

Mart, 2021

T.C.

YILDIZ TEKNİK ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

MODEL GÜDÜMLÜ GELİŞTİRME YAKLAŞIMI İLE OTOMATİK KOD ÜRETİMİ ARAÇLARININ KARŞILAŞTIRILMASI

Büşra İÇÖZ tarafından hazırlanan tez çalışması 03.03.2021 tarihinde aşağıdaki jüri tarafından Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı, Bilgisayar Mühendisliği Programı **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Prof. Dr. Oya KALIPSIZ
Yıldız Teknik Üniversitesi
Danışman

Jüri Üyeleri

Prof. Dr. Oya KALIPSIZ, Danışman
Yıldız Teknik Üniversitesi _____

Doç. Dr. Mehmet Sıddık AKTAŞ, Üye
Yıldız Teknik Üniversitesi _____

Prof. Dr. Selim AKYOKUŞ, Üye
İstanbul Medipol Üniversitesi _____

Danışmanım Prof. Dr. Oya KALIPSIZ sorumluluğunda tarafımca hazırlanan Model Güdümlü Geliştirme Yaklaşımı İle Otomatik Kod Üretimi Araçlarının Karşılaştırılması başlıklı çalışmada veri toplama ve veri kullanımında gerekli yasal izinleri aldığımı, diğer kaynaklardan aldığım bilgileri ana metin ve referanslarda eksiksiz gösterdiğimi, araştırma verilerine ve sonuçlarına ilişkin çarpıtma ve/veya sahtecilik yapmadığımı, çalışmam süresince bilimsel araştırma ve etik ilkelerine uygun davrandığımı beyan ederim. Beyanımın aksinin ispatı halinde her türlü yasal sonucu kabul ederim.

Büşra İÇÖZ

İmza

Değerli aileme...

TEŞEKKÜR

Yüksek lisans eğitimim sürecinde ve tez çalışmam sırasında bilgi ve tecrübeleriyle bana yol gösteren, desteğini esirgemeyen danışman hocam sayın Prof. Dr. Oya KALIPSIZ'a ve hayatımın her evresinde beni hep destekleyen sevgili aileme sonsuz teşekkürlerimi sunarım.

Büşra İÇÖZ

İÇİNDEKİLER

KISALTMA LİSTESİ

API	Application Programming Interface
CASE	Computer-Aided Software Engineering
CI/CD	Continuous Integration, Continuous Deployment
CLI	Command Line Interface
DSL	Domain Specific Language
EDL	Epsilon Dönüşüm Dili
EGMÇ	Eclipse Grafik Modelleme Çerçevesi
EMF	Eclipse Modeling Framework
GPL	General-Purpose Programming Language
IDE	Integrated Development Environment
MDSD	Model Driven Software Development
MGG	Model Güdümlü Geliştirme
MGM	Model Güdümlü Mimari
MGMH	Model Güdümlü Mühendislik
MGY	Model Güdümlü Yaklaşım
MGYG	Model Güdümlü Yazılım Geliştirme
MTL	Model to Text Language
SysML	Systems Modeling Language
UML	Unified Modeling Language
XML	Extensible Markup Language

ŞEKİL LİSTESİ

TABLO LİSTESİ

ÖZET

Model Güdümlü Geliştirme Yaklaşımı ile Otomatik Kod Üretimi Araçlarının Karşılaştırılması

Büşra İÇÖZ

Bilgisayar Mühendisliği Anabilim Dalı

Yüksek Lisans Tezi

Danışman: Prof. Dr. Oya KALIPSIZ

Model Güdümlü Yazılım Geliştirme (MGYG) bir yazılım projesinin hızlı, verimli, yüksek kalitede, minimum maliyet ve zamanda geliştirilmesi için kullanılan bir yazılım geliştirme yaklaşımıdır. Model güdümlü yazılım geliştirme yaklaşımında, yazılım geliştirme işlemine model oluşturularak başlanır. Oluşturulan model ile yazılımın kaynak kodlarının, birim testlerinin ve fonksiyonel test senaryolarının tamamının ya da büyük bir kısmının yazılım kaynak kodu üretme aracı kullanılarak otomatik olarak oluşturulması amaçlanır. Bu sayede, doğrudan yazılım projesinin geliştirilmesi sırasında zaman ve maliyetten kazanç elde edilmesi hedeflenir.

Bu tez çalışmasında model güdümlü yazılım geliştirme üzerinde durulmuş ve model güdümlü yazılım geliştirme yaklaşımının en önemli aşamalarından biri olan yazılım kaynak kodlarının otomatik olarak üretilmesi aşamasında kullanılan otomatik yazılım kaynak kodu üretme araçları karşılaştırılmıştır. Otomatik yazılım kaynak kodu üreten araçlar için birtakım kalite ölçütleri tanımlanmış ve bu kalite ölçütleri için gereksinimler belirlenmiştir. Belirlenen gereksinimler doğrultusunda açık kaynak kodlu ve ücretsiz olan otomatik yazılım kaynak kodu üretme araçları karşılaştırılmıştır. Araçların karşılaştırılması sonucunda, kalite ölçütleri için

belirlenen gereksinimleri en çok karşılayan araç olan Telosys aracı ile mikroservis mimari yapısında RESTful bir web servisin geliştirilmesi sırasında yazılım geliştirme yaşam döngüsünde model güdümlü yaklaşımdan yararlanarak örnek bir uygulama çalışması yapılmıştır. Ayrıca, yapılan uygulama çalışması üzerinden model güdümlü geliştirme yaklaşımının mikroservis mimarisi için uygulanabilirliği değerlendirilmiştir.

Anahtar Kelimeler: Model güdümlü mühendislik, model güdümlü geliştirme, otomatik kod üretimi, otomatik kod üretme aracı, mikroservis.

ABSTRACT

Comparison of Automatic Code Generation Tools with Model Driven Approach

Büşra İÇÖZ

Department of Computer Engineering

Master of Science Thesis

Advisor: Prof. Dr. Oya KALIPSIZ

Model Driven Software Development (MDSD) is a software development approach used for a fast, efficient, high quality, minimum cost and time process of a software project. In the model-driven software development approach, the software development process is started by creating a model. With the created model and software source code generation tool, it is aimed to automatically generate software source code for all or most of the source codes, unit tests and functional test scenarios. In this way, it is aimed to directly save time and cost during the development of the software project.

In this thesis, model-driven software development is emphasized and the automatic software source code generation tools used in the automatic generation of software source codes, which is one of the most important stages of the model-driven software development approach, have been compared. A number of quality criteria have been defined for the automatic software source code generating tools and the requirements have been determined for these quality criteria. In line with the specified requirements, open source and free automatic software source code generation tools were compared. As a result of the comparison of the tools, a sample case study was carried out using the model-driven approach in the software

development life cycle during the development of a RESTful web service in the microservice architecture structure with the Telosys tool, which is the tool that meets the requirements for quality criteria the most. In addition, the applicability of the model-driven development approach for microservice architecture was evaluated through the case study.

Keywords: Model-Driven engineering, model-driven development, automatic code generation, automatic code generation tool, microservice.

1
GİRİŞ

1.1 Literatür Özeti

MGG'nin temel amacı minimum maliyet ile kısa zamanda hızlı ve verimli uygulamalar geliştirebilmektir. Model güdümlü geliştirme yaklaşımında yazılım geliştirme işlemine tasarım aşamasında model oluşturmak ile başlanılır ve oluşturulan modeller referans alınarak yazılımın kaynak kodlarının otomatik olarak üretilmesi amaçlanır. Model Güdümlü Mimari (MGM) olarak da adlandırılan bu model güdümlü yazılım geliştirme yöntemi, OMG (Object Management Group) tarafından desteklenmektedir. OMG grubunun oluşturduğu standartlar bir dizi modelin, gösterimlerinin ve dönüşüm kurallarının nasıl tanımlanacağına dair standartlardan oluşmaktadır. Bu sayede standartlar doğrultusunda farklı platformlarda uygulamalar kolaylıkla geliştirilebilir.

Model güdümlü geliştirme tekniği ile yazılım geliştirilmesi alanında yapılmış çeşitli çalışmalar mevcuttur. Bu alanda yapılan çalışmalar birbirinden genel olarak kullanılan kaynak kod üretme yöntemi, kullanılan kaynak kod üretme aracı ve model güdümlü geliştirme tekniğinin uygulanma alanlarına göre ayrılmaktadır. Vitaliy Schreibmann ve arkadaşlarının 2015 yılında yaptığı "Model-driven Development of RESTful APIs" [1] çalışmasında REST servislerinin geliştirilmesi sırasında model güdümlü yaklaşımdan nasıl yararlandıklarından ve bu yaklaşımın yazılım geliştirme sırasında sağladığı faydalardan bahsedilmiştir. Çalışmada, ilk olarak Roy Fielding'in 2000 yılında doktora tezinde tanımladığı REST kavramı için Fielding'in kısıtlamalarını ve önerilerini temel alarak REST servisler için soyut bir model oluşturulmuştur. UML tabanlı olarak oluşturulan modelde, RESTful API'lerin modellenmesinde diğer modellerin aksine, uygulamanın durumlarına ve aralarındaki geçişlere odaklanılmıştır. Çalışmada oluşturulan modellerden yazılım kaynak kodunun otomatik olarak üretilmesi için açık kaynak kodlu olan Xtend otomatik yazılım kaynak kodu üretim aracı kullanılmıştır. Rosales-Morales ve arkadaşlarının 2015 yılında yaptığı "An analysis of tools for automatic software

development and automatic code generation" [2] çalışmasında ise otomatik yazılım kaynak kodu üreten araçlar incelenmiş ve otomatik kaynak kod üreten araçlar üç kategoriye ayrılmıştır. Bu kategoriler; CASE (Computer-Aided Software Engineering) araçları, çerçeve araçları ve yazılım geliştirme ortamlarında entegre olarak kurulumu gerçekleştirilebilen araçlardır. Çalışmada, bu araçlar için birtakım kalite ölçütlerini referans alarak kendi belirledikleri kalite ölçütlerine göre niteliksel ve nicelik açısından araçların değerlendirmeleri yapılmıştır. Gonçalves ve arkadaşlarının 2019 yılında yaptığı "RESTful Web Services Development with a Model-Driven Engineering Approach" [3] çalışmasında da yazılım kaynak kodunun otomatik olarak üretimi için DSL kullanılmış ve RESTful web servisler için otomatik kod üretme araçları detaylı bir şekilde karşılaştırılmıştır. Karşılaştırma sonucunda Xtext ve Xtend araçları seçilip bu araçlar ile örnek bir uygulama için otomatik kod üretimi yapılmıştır. Ayrıca, model güdümlü yazılım geliştirme yaklaşımı ile uygulamalar geliştirmek için açık kaynak kodlu projeleri destekleyen ve ücretiz bir platform olan Eclipse, model güdümlü geliştirme için çeşitli entegrasyon olanaklarını sunan çalışmalar yapmaktadır. Eclipse, bu alanda yaptığı çalışmaları EMF (Eclipse Modeling Framework) [4] projesi altında toplamıştır. Bu projede çeşitli modelleme dillerini (UML, DSL) ve metodolojileri desteklemek için tasarlanmış genişletilebilir çerçeveler ve örnek araçlar üretilmektedir. EMF projesinde genel olarak model oluşturulması ve yazılım kaynak kodunun otomatik üretilmesi için kullanılabilecek araçlar üzerinde çalışılır.

1.2 Tezin Amacı

Model güdümlü olarak yazılım geliştirmenin sağladığı birçok fayda vardır. Bunların en başında zaman ve maliyetten edilen kazanç gösterilebilir. Bunun önemli nedenlerinden biri de yazılım kaynak kodlarının bir kısmının ya da tamamının otomatik olarak üretilmesidir. Model güdümlü olarak yazılım geliştirilmesi sırasında oluşturulan modeller üzerinden popüler programlama dilleri için (Örneğin; C, C++, Java, Python vb.) otomatik kod üreten çeşitli araçlar bulunmaktadır. Bu nedenle programlama dillerine özel tasarım yapılması gerekli değildir, birçok programlama dili için bir model üzerinden otomatik olarak yazılım kaynak kodları üretilebilir. Bu aşamada önemli olan uygun yazılım kaynak kodu

üretim aracıdır. Yazılım kaynak kodu üretim aracı tercih edilirken, uygulamanın gereksinimleri ve kullanılan model referans alınarak bu doğrultuda uygun programlama dillerine istenilen şekilde çıktı verebilen yazılım kaynak kodu üretim araçları tercih edilmelidir. Bu tez çalışmasının amacı açık kaynak kodlu ve ücretsiz olan otomatik yazılım kaynak kodu üretim araçlarını karşılaştırmak ve bunlar arasında birtakım yazılım kalite ölçütlerine göre en uygun olanını tespit etmektir. Aynı zamanda mikroservis mimari modeli kullanılarak tasarlanmış bir uygulamanın mikroservislerini, otomatik yazılım kaynak kodu üretme araçlarının karşılaştırılması sonucu uygun olarak tespit edilen araç ile geliştirmektir. Ayrıca, geliştirilen uygulama üzerinden otomatik kaynak kod üretimi aracından hangi aşamalarda nasıl yararlanılabileceğini ve yöntemin uygulanabilirliğini tespit etmektir.

1.3 Hipotez

Bir uygulamanın yazılımı geliştirilirken farklı yöntemlerden yararlanılır. Model Güdümlü Mühendislik (MGMH)' de bunlar biridir. Bir uygulama geliştirilirken; kısa zamanda geliştirmesi, maliyetin minimum seviyede tutulması, yüksek kalitede olması ve sonrasında kolay bir şekilde bakım yapılabilir olması gibi çeşitli gereksinimler vardır. Model güdümlü mühendislik ile bu gereksinimlerin büyük bir kısmı karşılanmaktadır. Bu çalışmada 2. Bölüm'de, model güdümlü mühendislik ve model güdümlü mühendisliğin bir uygulama geliştirilirken yazılım yaşam döngüsünde nasıl kullanılabileceğinden bahsedilmiştir. Sonrasında 3. Bölüm'de, model güdümlü yazılım geliştirme aşamalarından olan otomatik yazılım kaynak kodu üretimi ve otomatik yazılım kaynak kodu üretimi sırasında kullanılan modelleme dillerinden bahsedilmiştir. Ayrıca, açık kaynak kodlu olan yazılım kaynak kodu üretim araçları incelenmiş ve birtakım kalite ölçütlerine göre uygulamalar puanlanarak karşılaştırılmıştır. Çalışmanın son bölümünde, belirlenen kalite ölçütlerine göre en yüksek puanı alan otomatik yazılım kaynak kodu üretim aracı ile model güdümlü olarak bir uygulama geliştirilmiştir. Geliştirilen uygulama üzerinden otomatik kaynak kod üretimi aracından hangi aşamalardan nasıl yararlanılabileceğinden ve uygulanabilirliğinden bahsedilmiştir.

2
MODEL GÜDÜMLÜ GELİŞTİRME

Bu bölümde model güdümlü geliştirmenin temellerinden, model güdümlü geliştirmenin sağladığı faydalardan ve model güdümlü yazılım geliştirme yazılım yaşam döngüsünden bahsedilmiştir.

2.1 Model Güdümlü Geliştirmenin Temelleri

Model Güdümlü Mühendislik (MGMH), uygulamanın alanına özgü modellerin oluşturulmasına ve bu modellerin kullanılmasına odaklanan bir yazılım geliştirme yöntemidir. Modeller, uygulama alanını yöneten bilgilerin soyut temsili olarak nitelendirilebilirler [5]. Modeller, tasarımcılar, geliştiriciler, yöneticiler ve uygulama alanına özgü kullanıcılar tarafından yazılım geliştirme yaşam döngüsünün aşamalarında kullanılabilir. Model Güdümlü Yazılım Geliştirme (MGYG) yaklaşımı, yazılım yaşam döngüsünün her bir aşamasını hızlı, minimum maliyet ve zaman ile tamamlamak için kullanılan bir yazılım geliştirme yaklaşımıdır. Model güdümlü yazılım geliştirme yaklaşımında, uygulamanın ilk olarak modeli oluşturulur ve bu doğrultuda uygulamanın yazılım kaynak kodlarının tamamının ya da belirli bir kısmının otomatik olarak üretilmesi hedeflenir. Model güdümlü mimari olarak da bilinen model güdümlü geliştirme, OMG'nin öncülüğünü yaptığı yazılım tasarımı, geliştirmesi ve uygulanmasına yönelik bir yaklaşımdır. Model güdümlü mimari, model olarak ifade edilen yazılım gereksinim tanımlarının yapılandırılması için yönergeler sağlar [6].

OMG'ye göre model güdümlü geliştirmenin dört temel ilkesi vardır. Bunlar aşağıdaki gibidir;

• İyi tanımlanmış bir gösterim ile ifade edilen modeller, kurumsal ölçekli çözümler için sistemleri anlamak için önemlidir.

• Sistemlerin geliştirilmesi, modeller arasında bir dizi dönüşümü empoze ederek bir dizi model etrafında organize edilebilirler. Ayrıca, mimari bir katman ve dönüşüm çerçevesi içinde de organize edilebilirler.

• Bir dizi modeldeki modelleri açıklamak için resmi bir temel, modeller arasında anlamlı entegrasyonu ve dönüşümü kolaylaştırır. Ayrıca, birtakım model güdümlü geliştirme araçları aracılığıyla otomasyonun temelini oluşturabilir.

• Bir model güdümlü yaklaşımın kabulü ve geniş çapta benimsenmesi, tüketicilere açıklık sağlamak ve satıcılar arasındaki rekabete teşvik etmek için endüstri standartlarını gerektirebilir.

2.2 Model Güdümlü Geliştirmenin Sağladığı Faydalar

Model güdümlü olarak uygulama geliştirmenin sağladığı birçok fayda vardır. Model güdümlü olarak yazılım geliştirmenin sağladı faydalardan bazıları aşağıdaki gibidir:

- MGG kullanarak uygulama geliştirmek daha hızlıdır. Model güdümlü geliştirmede, bir uygulamanın yazılım modeli, geleneksel programlama dillerinden daha yüksek bir soyutlama düzeyinde belirtilir. Bu model, otomatik kod üreterek veya modeli yorumlayarak otomatik olarak çalışan bir yazılım uygulamasına dönüştürülür. Kullanılan model daha yüksek bir soyutlama düzeyinde iken, kodda ifade edilen aynı modele kıyasla çok daha küçüktür. Başka bir deyişle; modeldeki her öğe birden çok kod satırını temsil eder. Böylece, aynı anda daha fazla işlevsellik oluşturulabilir. Bir model üzerinden çeşitli programlama dillerini destekleyen otomatik kod üretim araçları ile uygun kodlar üretilebilir.

-MGG ile uygulama geliştirmek daha az maliyetlidir. İlk neden, MGG ile model üzerinden otomatik olarak yazılım kaynak kodu üretilebildiği için yazılım geliştirme süresi kısalır ve daha kısa bir sürede pazara sunulabilir. İkinci neden, daha düşük maliyet (daha az insan, kısa geliştirme süresi vb.) ile uygulama geliştirmenin mümkün olmasıdır. MGG yaklaşımı kullanılarak geliştirilmiş uygulamaların bakımının yapılması daha az maliyetlidir. Yüksek seviyeli modelleri okuyarak bir uygulamanın davranışını anlamak daha kolaydır. Ayrıca, üst düzey bir dil kullanarak işlevsellik eklemek veya mevcut işlevselliği değiştirmek daha hızlıdır.

-MGG uygulamanın kalitesinin artmasına yol açar. Bir yazılım uygulaması, bir kod üretim aracı tarafından üst düzey bir model ile oluşturulduğundan; uygulamanın (teknik) kalitesi yazılım kaynak kod üretim aracına bağlıdır. Bu nedenle, kod kalitesinde artış olur çünkü en iyi deneyimler kod üretme aracı üzerine eklenebilir

ve tek bir elden çıkmış daha az kod tekrarı olan yazılım kaynak kodları elde edilebilir.

-MGG yaklaşımı ile uygulama geliştirildiğinde, yazılım geliştirme sürecinde ortaya çıkan yazılım geliştirici değişikliklerinden daha az etkilenilir. Yazılım geliştirmek için daha az geliştiriciye ihtiyaç vardır. Ayrıca, geliştirme ekibine yeni bir üye katıldığı durumda, uygulamanın üst düzey modelini anlamak, kaynak kodunu okuyarak uygulamanın davranışını anlamaya çalışmaktan çok daha kolay ve hızlıdır.

-MGG gelişmiş geliştiricilerin zor şeylere odaklanmasını sağlar. MGG'de, gelişmiş geliştiriciler daha az tekrarlayan işler yaparlar, böylece çalışmalarının yaratıcı yönlerine odaklanabilirler.

-MGG ile modeller üzerinden otomatik test durumları üretimi de yapılabilir. Bu durum, bütün test durumlarının üzerinden geçilmesine ve yazılım test maliyetinin büyük oranda azaltılmasına fayda sağlamaktadır. Böylece yazılım hataları test sırasında kolay bir şekilde tespit edilebilir ve hızlı bir şekilde düzeltilebilir.

2.3 Model Güdümlü Geliştirme için Yazılım Yaşam Döngüsü

Model güdümlü olarak yazılım geliştirilirken sıra ile gereksinim analizi, tasarım (model oluşturma), otomatik kod üretimi, otomatik test senaryosu üretimi, fonksiyonel test ve uygulamanın çalışacağı ortama kurulumu aşamaları izlenir.

Şekil 2.1 Model güdümlü yazılım geliştirme yaşam döngüsü

Model güdümlü geliştirme yazılım yaşam döngüsünde Şekil 2.1'de de görüldüğü gibi gereksinim analizi yapıldıktan sonra uygulamanın gereksinimleri doğrultusunda tasarımı yapılır. UML, DSL ya da veri tabanı modeli oluşturulur. Oluşturulan model üzerinden otomatik kod üretme araçları yardımı ile otomatik olarak yazılımın kaynak kodları ve test senaryoları üretilir. Sonrasında yazılım kaynak kodları otomatik olarak üretilen uygulama için otomatik olarak üretilmiş fonksiyonel test senaryoları ile fonksiyonel test işlemi yapılır. Son olarak, uygulamanın bulut ortama ya da fiziksel ortama kurulum ve dağıtım işlemi gerçekleştirilir.

3
OTOMATİK KOD ÜRETİMİ

Yazılım kaynak kodunun geliştirilmesi, yazılım mühendisliğinde önemli bir alandır, bu nedenle yazılım kaynak kodunun geliştirilmesini hızlı, verimli ve en az maliyet ile gerçekleştirmek için çok çeşitli teknik, yöntem ve yaklaşımlar ortaya çıkmıştır. Yazılım kaynak kodunun Şekil 3.1'deki gibi çeşitli modeller oluşturarak bir araç yardımı ile otomatik olarak üretilmesi de bu yöntemlerden biridir.

Şekil 3.1 Model güdümlü olarak otomatik yazılım kaynak kodu üretme

Bu bölümde, otomatik kod üretme araçlarında kullanılan modelleme dillerinden, otomatik kaynak kodu üretmenin geliştirme sırasında ve sonrasında sağladığı faydalardan ve açık kaynak kodlu otomatik kod üretme araçlarından bahsedilecektir. Ayrıca, ISO / IEC 9126 standardı [7] referans alınarak açık kaynak kodlu otomatik kod üretme araçlarının belirlenen bir dizi kalite ölçütünü karşılayıp karşılamadıklarını belirlemek için analiz ve değerlendirilmesi yapılacaktır. Ayrıca, otomatik kod üretme araçları belirlenen kalite ölçütlerine göre puanlanıp en yüksek puanı alan araç için çalışmanın geri kalanında örnek bir uygulama çalışması yapılacaktır.

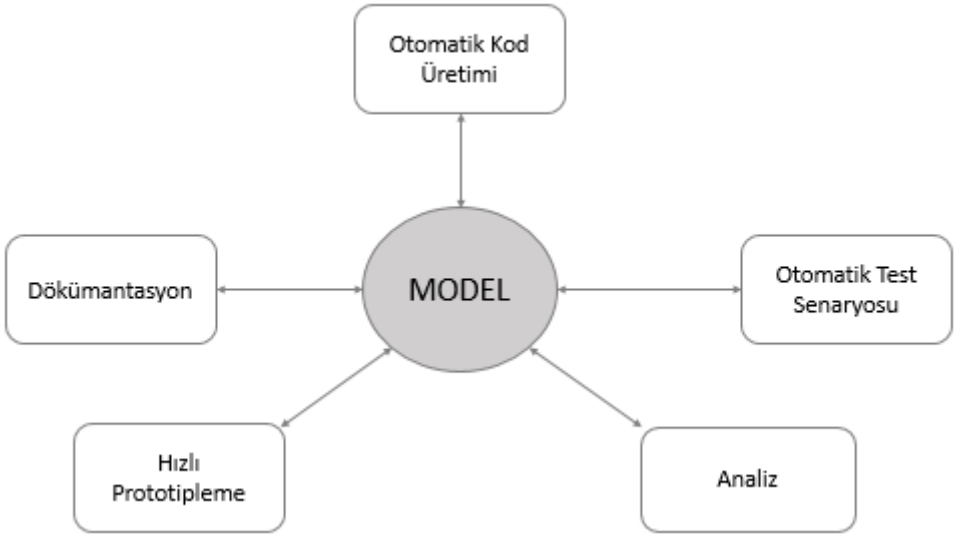
3.1 Yazılım Kaynak Kodunun Otomatik Olarak Üretilmesinin Sağladığı Faydalar

Model güdümlü geliştirmenin en önemli adımlarından biri olan yazılım kaynak kodunun otomatik olarak üretilmesinin, yazılım kaynak kodunu üreten araçların en iyi deneyimleri referans alarak geliştirildiği göz önünde bulundurulduğunda yazılımın geliştirilmesi sırasında sağladığı birçok fayda vardır. Bunlardan birkaçı maddeler halinde aşağıdaki şekilde belirtilmiştir.

- Yazılım kodu, geliştirme bir insan tarafından yapılmadığı için daha az sentaks hatası içerir.
- Daha büyük ve daha karmaşık modelleri uygularken yazılım kodunun otomatik olarak üretilmesi ile büyük zaman tasarrufu sağlanır.
- Basit uygulamalar için hızlı pazara çıkılması sağlanır.
- Kod oluşturma, herhangi bir CI/CD (Continuous Integration, Continuous Deployment) ardışık düzenine veya geliştirme iş akışına bir adım olarak eklenebilir. Böylece kod yazımı için harcanan efor minimum seviyeye indirilmiş olur.
- Uçtan uca oluşturma, birden çok kaynak dosya ve dosya türü ile uğraşırken hataları ortadan kaldırır.
- Tek bir yapıda büyük kod tabanlarında uygulamak için temel modellere yeni işlevler eklenerek otomatik kod üretimi tekrarlanabilir, böylece uygulamanın bakımı kolaylaşır.
- Modellerin anlaşılırlığı yazılım kaynak kodundan daha kolaydır, yazılım ekibindeki herhangi bir kişi model üzerinde değişiklik yaparak otomatik olarak kod üretimi gerçekleştirebilir.

3.2 Modelleme Dilleri

Modeller Şekil 3.2'de de gösterildiği gibi çeşitli amaçlarla kullanılır. Modelleme aynı zamanda yazılımın kodlamasına başlamadan önce yazılım uygulamalarının tasarlanması aşamasında da sıklıkla kullanılır. Modelleme, yazılım projelerinin önemli bir parçasıdır. Yazılım geliştirmeye başlanırken model kullanıldığında, bir yazılım geliştirme projesinin başarısından sorumlu olan kişiler, uygulama geliştirmesine başlamadan önce işlevselliğinin eksiksiz ve doğru olduğundan, son kullanıcı ihtiyaçlarının karşılandığından ve program tasarımının ölçeklenebilirlik, sağlamlık, güvenlik, genişletilebilirlik ve diğer özellikler için gereksinimleri desteklediğinden emin olabilirler. Bu bölümde, model güdümlü otomatik kod üretme araçlarında sıklıkla girdi olarak kullanılan UML ve DSL modellerinden bahsedilecektir.



Şekil 3.2 Modellerin kullanım alanları

3.2.1 DSL (Domain Specific Language)

Alana özgü dil olarak da bilinen DSL, belirli bir problem sınıfı için optimize edilmiş daha yüksek düzeyde soyutlamaya sahip bir programlama dilidir [8]. Bir DSL, alan veya etki alanındaki kavramları ve kuralları kullanır. Belirli bir problemi çözmeye odaklanır. DSL, Java ve C gibi genel bir amaca hizmet eden dillerin aksine bir yazılım geliştirici tarafından değil alanında uzman herhangi biri tarafından geliştirilebilir ve DSL kullanılarak uygun araçlar ile yazılımın amacına ve beklentilerine uygun programlama dilinde otomatik kod üretimi gerçekleştirilebilir.

Bir DSL her amaca hizmet etmez. DSL'ler Java gibi GPL (Genereal Purpose Language)'in aksine sınırlı bir uygulanabilirlik ve kullanım alanı için yaratılmıştır, ancak bu alandaki sorunları ve çözümleri temsil edecek ve ele alacak kadar güçlüdür. DSL'e örnek olarak HTML verilebilir. HTML web uygulama alanı için yaygın olarak kullanılan bir dildir.

DSL oluşturmak için sıklıkla kullanılan açık kaynak kodlu araçlara örnek olarak Xtext [9] ve JetBrains MPS [10] verilebilir.

Xtext: Xtext, DSL'lerin geliştirilmesini sağlayan Eclipse ile entegre çalışan bir yazılım çerçevesidir. Şekil 3.3 'de Xtext ile oluşturulmuş bir DSL örneği gösterilmiştir.

```
typedef String
typedef Integer
typedef Date mapsto java.util.Date

entity Person {
  String name
  String surName
  Date birthDay
  Address home
  Address work
}

entity Boss extends Person {
  Person* employees
}

entity Address {
  String street
  String number
  String city
  String ZIP
}
```

Şekil 3.3 Xtext ile oluşturulmuş DSL örneği

JetBrains MPS: JetBrains MPS, DSL'ler oluşturmak için entegre bir geliştirme ortamıdır (IDE). Kendisini, bir belgeyi temeldeki soyut ağaç yapısı olarak saklayan bir projeksiyonel düzenleyici olarak adlandırır (Bu kavram, Microsoft Word gibi programlar tarafından da kullanılır). JetBrains MPS ayrıca Java, C, JavaScript veya XML'e kod oluşturmayı da destekler. Şekil 3.4'de JetBrains MPS ile oluşturulmuş ve uygulama dağıtımı için kullanılan DSL gösterilmektedir.

```
text gen component for concept DockerImage {
file name : (node)->string {
  "Dockerfile";
}
  (node)->void {
    append {FROM } ${node.From} \n ;
    append {MAINTAINER } ${node.Maintainer} \n ;
    append {EXPOSE } ${node.Expose} \n ;
    append {COPY } ${node.microservice.microservice.name} \n ;
    append {apt -get update} \n ;
    append {CMD java -jar} ${service.name}\n ;
  }
}
```

Şekil 3.4 JetBrains MPS ile uygulama dağıtımı için oluşturulmuş DSL örneği

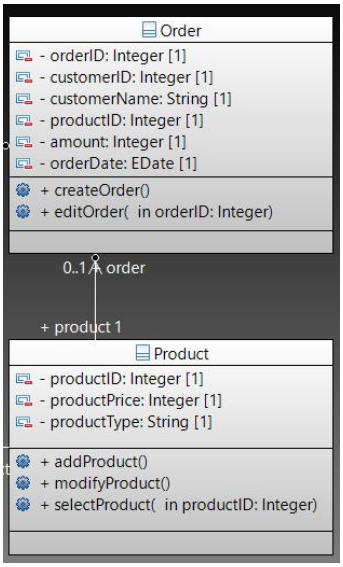
3.2.2 UML (Unified Modelling Language)

UML (Unified Modelling Language) olarak da bilinen iş sistemlerinin nasıl modellenebileceğini belirleyen ve açıklayan yöntemlerin bir araya toplanmış halini açıklayan modelleme dilidir.

UML 2.0 [11], üç kategoriye ayrılmış on üç tip diyagramı barındırır: Altı diyagram tipi statik uygulama yapısını temsil eder; üçü genel davranış türlerini temsil eder; ve dördü, etkileşimlerin farklı yönlerini temsil eder [11]:

- Yapı diyagramları; sınıf şeması, nesne şeması, bileşen şeması, kompozit yapı şeması, paket şeması ve dağıtım şemasını içerir.
- Davranış diyagramları; kullanım durumu şeması (gereksinimlerin toplanması sırasında bazı metodolojiler tarafından kullanılır), etkinlik şeması ve durum makinesi şemasını içerir.
- Etkileşim diyagramları; sıra şeması, iletişim şeması, zamanlama şeması ve etkileşime genel bakış şemasını içerir.

Şekil 3.5'de Eclipse platformunda oluşturulmuş örnek bir UML sınıf şeması gösterilmiştir.



Şekil 3.5 Eclipse Papyrus aracı ile oluşturulmuş UML sınıf şeması

3.3 Otomatik Yazılım Kaynak Kod Üretme Araçları için Yazılım Kalite Ölçütleri

Bu bölümde, çalışmada yazılım kaynak kod üretme araçları için değerlendirme sürecinde kullanılan yazılım kalite ölçütlerinden bahsedilecektir. Yazılım kaynak kod üretme araçları değerlendirilirken uygulama kalite değerlendirmesi için yaygın

olarak tanınan ve uluslararası bir standart olan ISO / IEC 9126 [7] yazılım kalite standartlarından yararlanılmıştır.

ISO / IEC 9126 [7] standardının temel amacı, bir yazılım geliştirme projesinin müşteriye teslimi̇nde ve yazılımın uygunluğunun ölçülmesi sırasında genel olarak insan önyargılarını ele alarak bu doğrultuda başarılı sayılabilecek bir yazılım projesinin uygunluğunu ölçmek için bir standart oluşturulmuştur. ISO / IEC 9126 [7] soyut öncelikleri (uyumluluk) ölçülebilir değerlere dönüştürerek, projenin hedefleri ve hedefleri hakkında ortak bir anlayış geliştirmeye çalışılmıştır.

Standart dört bölüme ayrılmıştır:

ISO 9126-1 Kalite Modeli [7]

ISO 9126-2 Dış Metrikler [7]

ISO 9126-3 Dahili Metrikler [7]

ISO 9126-4 Kullanımdaki Kalite Metrikleri [7]

Bu çalışmada yazılım kaynak kod üretme araçlarını karşılaştırırken Şekil 3.6'da da gösterilen kalite modelini kullanılmıştır.

Şekil 3.6 ISO 9126'ya göre kalite kriterleri [12]

ISO / IEC 9126-1 [12] olarak da tanımlanan kalite standardı, yazılım kalitesini yapılandırılmış bir dizi özellik ve alt özellikte sınıflandırır. ISO/IEC 9126-1 2011 yılında ISO/IEC 25010 standardı olarak güncellenmiştir ve ISO/IEC 9126-1'den farklı olarak kalite faktörlerine uyumluluk ve güvenilirlik ölçütleri eklenmiştir [13]. Bu tez çalışmasında, uyumluluk ve güvenilirlik ölçütleri kalite standartlarının belirlenmesi sırasında değerlendirmeye katılmamıştır. Çalışmada genel olarak ISO/IEC 9126-1 standartları referans alınarak otomatik yazılım kodu üretme araçları için inceleme ve karşılaştırma gerçekleştirilmiştir.

ISO / IEC 9126 standardının ilk bölümünde sunulan kalite modeli, ISO / IEC 9126-1 [12] yazılım kalitesini yapılandırılmış özellikler, alt özellikler ve açıklaması Tablo 3.1 deki gibidir.

Tablo 3.1 ISO / IEC 9126-1 standardına göre yazılım kalite ölçütleri ve alt özellikleri

Özellik	Alt Özellikler	Özellik Açıklaması
İşlevsellik	1. Uygunluk 2. Doğruluk 3. Birlikte çalışabilirlik 4. Güvenlik 5. İşlevsellik Uyumu	İşlevler kümesinin ve bunların belirlenmiş özelliklerinin varlığına dayanan bir öznitelik kümesidir. İşlevler, belirtilen veya ima edilen ihtiyaçları karşılayan işlevlerdir [12] [7].
Güvenilirlik	1. Olgunluk 2. Hata toleransı 3. Kurtarılabilirlik 4. Güvenilirlik uyumluluğu	Yazılımın, belirtilen koşullar altında belirli bir süre boyunca performans düzeyini sürdürme kapasitesini ölçen bir dizi özellik [12] [7].

Tablo 3.1 ISO / IEC 9126-1 standardına göre yazılım kalite ölçütleri ve alt özellikleri (devamı)

Kullanılabilirlik	1. Anlaşılabilirlik 2. Öğrenilebilirlik 3. İşletilebilirlik 4. Çekicilik 5. Kullanılabilirlik uyumluluğu	Belirtilen bir kullanıcı grubu tarafından kullanım için gereken bireysel çabanın değerlendirmesini ölçen bir dizi özellik [12] [7].
Verimlilik	1. Zaman davranışı 2. Kaynak kullanımı 3. Verimlilik uyumu	Belirtilen koşullar altında, yazılımın performans seviyesi ile kullanılan kaynak miktarı arasındaki ilişkiyi ölçen bir dizi özellik [12] [7].
Sürdürülebilirlik	1. Analiz edilebilirlik 2. Değiştirilebilirlik 3. İstikrar 4. Test edilebilirlik 5. Sürdürülebilirlik uyumluluğu	Belirtilen değişiklikleri yapmak için gereken çabayı ölçen bir dizi özellik [12] [7].
Taşınabilirlik	1. Uyarlanabilirlik 2. Kurulabilirlik 3. Birlikte çalışma 4. Değiştirilebilirlik 5. Taşınabilirlik uyumluluğu	Yazılımın bir ortamdan diğerine aktarılma yeteneğini ölçen bir dizi özellik [12] [7].

Tablo 3.1 de gösterilen alt özellikler de kendi içinde öznitelikler içermektedir fakat standart da bunlar yazılım ürününe göre farklılık gösterebileceği için belirtilmemiştir [7] [14]. Bir sonraki bölümde ISO / IEC 9126-1 standartları referans alınarak otomatik yazılım kodu üretme araçları değerlendirilecektir.

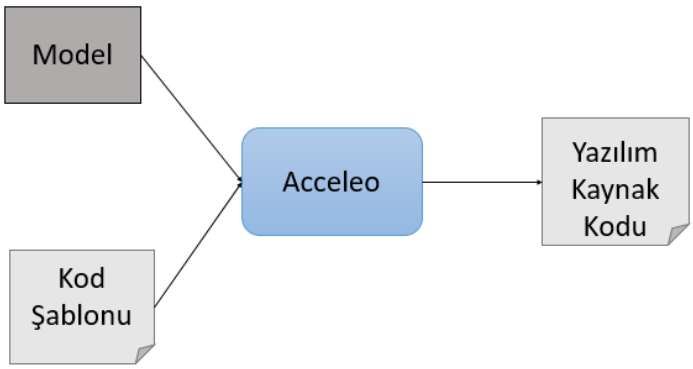
3.4 Açık Kaynak Kodlu Otomatik Kod Üretme Araçları

Model odaklı bir yaklaşımla yazılım geliştirirken oluşturulan modellere dayalı olarak popüler programlama dilleri (örneğin, C, C ++, Java, Python vb.) için otomatik olarak kod oluşturan çeşitli araçlar vardır. Bu sayede yazılım geliştirme sürecinde, yazılım kaynakları modellemeye odaklanır, yazılım tasarımının kalitesi artar, geliştirme maliyeti düşer ve programlama dilleri için özel bir tasarıma gerek kalmaz,

bir dizi otomatik kod üretilebilir. Otomatik kod üreten yazılım araçlarının çoğu, veri tabanı modeli, UML, sınıf diyagramı, DSL (Domain Specific Language) gibi çeşitli modeller kullanır ve bu modeller için bir kod çerçevesi oluşturur. Böylece geliştirme süreci ve maliyeti azalır. Ayrıca otomatik olarak üretilen kodlar en iyi uygulamalara göre oluşturulduğundan, kod kalitesi artar ve tekrar eden kod azalır. Bu bölümün geri kalanında popüler açık kaynak kodlu otomatik yazılım kaynak kodu üreten araçlardan kısaca bahsedilecektir.

Acceleo

Acceleo, Eclipse Vakfı'na ait, yazılım kaynak kodunu otomatik olarak oluşturmak için model odaklı bir yaklaşım kullanılmasına olanak sağlayan açık kaynaklı bir otomatik kod üreticidir. Modelden metne dönüşüm gerçekleştirmek için OMG'nin MOF Model to Text Language (MTL) [2] standardını uygulayan ve çeşitli programlama dillerindeki kodu (Java, C++, Scala, Ruby ...) oluşturmak için Şekil 4'de de görüldüğü gibi herhangi bir EMF (Eclipse Modeling Framework) tabanlı modeli (UML, SysML, DSL ...) MTL şablon ile birlikte kullanarak otomatik kod üretmeye olanak sağlayan, Obeo firmasının Eclipse Vakfı ile birlikte geliştirdiği açık kaynaklı bir otomatik kod üretme aracıdır[3].



Şekil 3.7 Acceleo ile otomatik kod üretimi

Acceleo Eclipse üzerinde bir eklenti olarak kullanılabileceği gibi eclipse dışında da bir maven projesi olarak kullanılabilir. Şekil 3.8'de Eclipse platformunda oluşturulmuş bir Acceleo projesi için MTL şablonu örneği vardır [15].

```
[comment encoding = UTF-8 /]
[module generate('http://www.eclipse.org/uml2/5.0.0/UML')/]

[template public generate(aClass : Class)]
[file (aClass.name.concat('.java'), false)]
  public class [aClass.name.toUpperFirst()/] {
  [for (p: Property | aClass.attribute) separator('\n')]
    private [p.type.name/] [p.name/];
  [/for]

  [for (p: Property | aClass.attribute) separator('\n')]
    public [p.type.name/] get[p.name.toUpperFirst()/]() {
      return this.[p.name/];
    }
  [/for]

  [for (o: Operation | aClass.ownedOperation) separator('\n')]
    public [o.type.name/] [o.name/]() {
      // TODO should be implemented
    }
  [/for]
  }
[/file]
[/template]
```

Şekil 3.8 Acceleo kod şablonu olarak kullanılan .mtl dosyası örneği

Şekil 3.9'da Şekil 3.8'deki kod şablonu kullanılarak örnek bir UML modeli için üretilmiş bir java sınıfının kod parçacığı örneği gösterilmiştir.

```
public class City {
  private City city_at;

  private Street street_2;

  public City getCity_at() {
    return this.city_at;
  }

  public Street getStreet_2() {
    return this.street_2;
  }

  public Boolean isBig() {
    // TODO should be implemented
  }

  public Integer TestSequence() {
    // TODO should be implemented
  }
}
```

Şekil 3.9 Acceleo kod şablonu kullanılarak bir UML modeli için üretilen kod parçacığı örneği

Acceleo, artımlı olarak kod çerçevesi üretimi gerçekleştirir. Kullanıcı kendi kodunu bu çerçeveye ekleyebilir. Kullanıcı kodu belirli etiketler arasında olduğu için kod şablonunda oluşan değişiklikten etkilenmez.

Telosys

Telosys basit ve pratik bir açık kaynak kodlu otomatik kod üretme aracıdır. Komut satırı arabirimi aracı (CLI) ve Eclipse eklentisi olarak kullanılabilmektedir. Telosys model olarak DSL kullanmaktadır. Birçok dil ve çerçevede kolay bir şekilde otomatik kod üretilmesini sağlar.

Telosys aracı "Geliştiriciler için geliştiriciler tarafından tasarlanmıştır ve hızlı bir şekilde çalışmaya odaklanmıştır "sloganını kullanır [16].

Şekil 3.10 'da Eclipse platformunda Telosys eklentisi ile birlikte otomatik kod üretmek için oluşturulmuş DSL modeli gösterilmiştir.

```
1 // Entity Depo
2
3 Depo {
4   urunFiyati : long ;
5   urunTipi:    string;
6   urunNumarası: int;
7   urunEkle(Depo urun)   : void
8   urunSil(int urunNumarasi)   : void
9   urunGuncelle(Depo urun)   : void
10  urunFlitrele(String urunTipi):string
11 }
12
```

Şekil 3.10 Telosys aracı ile oluşturulmuş DSL örneği

Şekil 3.11'de, Telosys kullanarak Eclipse'de otomatik olarak oluşturulan kod parçasının bir örneği gösterilmektedir.

```
@RequestMapping(value = "/sil/") // GET or POST
public String urunSil(RedirectAttributes redirectAttributes) {
    log("Action 'sil'" );
    try {
        boolean deleted = urunService.deleteById(  );
        log("Urun deleted. Key : " + toString() + " result = " + deleted );
        //--- Set the result message
        messageHelper.addMessage(redirectAttributes,
        new Message(MessageType.SUCCESS,"delete.ok"));
    } catch(Exception e) {
        messageHelper.addException(redirectAttributes, "stock.error.sil", e);
    }
    return redirectToList();
}
```

Şekil 3.11 Teloys aracı ile oluşturulmuş kod parçası örneği

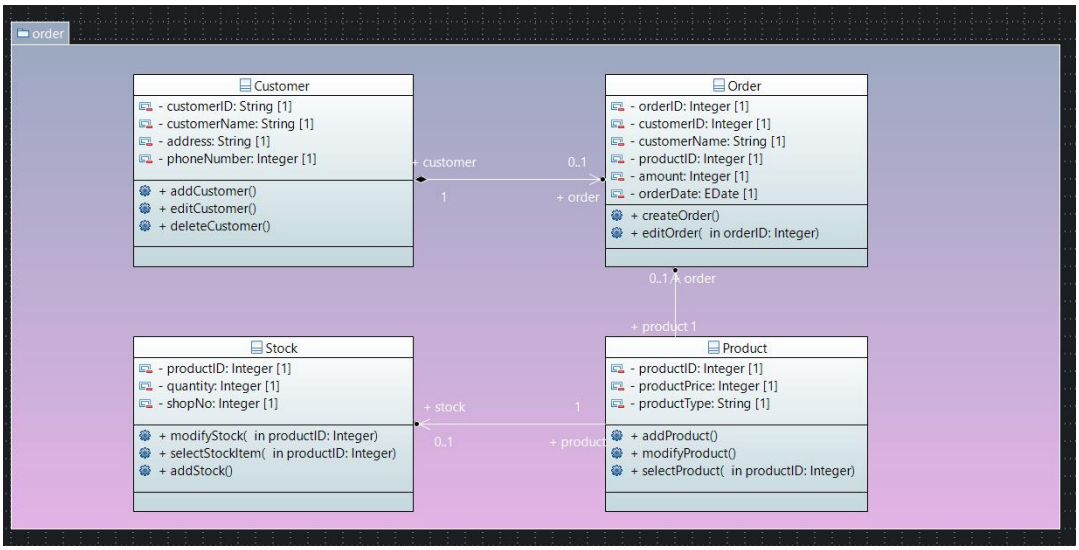
Umple

Umple, model güdümlü yazılım geliştirmeye olanak sağlayan bir otomatik kod üretme aracıdır. UML kullanılarak Java, C ++, PHP ve Ruby gibi nesne yönelimli programlama dillerinde otomatik kod üretimi sağlar. Umple ayrıca UML sınıfını ve durum diyagramlarını metinsel olarak oluşturmak için de kullanılabilir [17].

Umple, açık kaynaklı ve ücretsiz bir araçtır. Umple Eclipse eklentisi olarak kullanılabilir ya da komut satırı arabirim aracı (CLI) olarak çevrim içi de kullanılabilir.

Papyrus

Papyrus model güdümlü ücretsiz ve açık kaynak kodlu otomatik kod üretme aracıdır. Eclipse eklentisi olarak kullanılabilmektedir. UML model'i kullanarak Java kod çerçevesi üretmemize olanak sağlar [18]. Şekil 3.12'de Eclipse üzerine eklenti olarak kurulan Papyrus aracı ile basit bir ürün yönetimi yazılımı için otomatik Java kodu üretmek için UML sınıf modeli oluşturulmuş ve sınıflar arasındaki ilişkiler tanımlanmıştır.



Şekil 3.12 Eclipse platformunda Papryus aracı ile oluşturulan UML modeli

Şekil 3.13'de Şekil 3.12'deki UML modeli kullanılarak otomatik olarak oluşturulmuş kod parçacığının örneği paylaşılmıştır.

```
1  package order;
2
3  public class Customer {
4
5      /**
6       *
7       */
8      private String customerID;
9      /**
10      *
11      */
12     private String customerName;
13     /**
14      *
15      */
16     private String address;
17     /**
18      *
19      */
20     public Order order;
21     /**
22      *
23      */
24     private Integer phoneNumber;
25     /**
26      * Getter of customerID
27      */
28     public String getCustomerID() {
29         return customerID;
30     }
31     /**
32      * Setter of customerID
33      */
34     public void setCustomerID(String customerID) {
35         this.customerID = customerID;
36     }
```

Şekil 3.13 Eclipse'de Papryus aracı ile otomatik oluşturulan kod parçası örneği

Xtext

Xtext, programlama dilleri ve alana özgü diller (DSL'ler) geliştirmek için açık kaynaklı bir çerçevedir. Standart ayrıştırıcı üreticilerinden farklı olarak, Xtext

yalnızca bir ayrıştırıcı değil, aynı zamanda özet söz dizimi ağacı ve tam özellikli, özelleştirilebilir bir Eclipse tabanlı bir IDE için de bir sınıf modeli oluşturur.

Xtext, Eclipse projesinde, Eclipse modelleme çerçeve projesi kapsamında geliştirilmekte ve Eclipse kamu lisansı altında lisanslanmaktadır [10].

EMF-REST

EMF-REST, EMF modelleri için RESTful API'ler oluşturulur. Açık kaynaklı ve ücretsiz bir araçtır. Eclipse eklentisi olarak kullanılabilir [19].

JET

JET Eclipse platformu üzerinde eklenti olarak kullanılabilen açık kaynak kodlu ücretsiz bir model güdümlü bir otomatik kod çerçevesi üretme aracıdır. Jet otomatik kod üretimi sırasında aşağıdaki özellikleri sağlar [20]:

- Özel etiketleri desteklemek için JET dili genişletilebilir.
- Özel etiket kütüphaneleri bildirmek için Java arayüzleri ve Eclipse Uzantıları tanımlanabilir.
- JET ayrı bir çerçeve editörü sağlar (EMFT JET Editor).

3.5 Otomatik Kod Üretim Araçlarının Karşılaştırılması

Bu bölümde otomatik kod üretme araçları ISO / IEC 9126-1 [12] standartları da referans alınarak beş yönden karşılaştırılacaktır. Değerlendirme yapılırken bir yazılım kaynak kodunun gereksinimleri göz önüne alınarak, yazılım kod üretme araçlarının bu gereksinimleri karşılama durumuna göre puanlama yapılacaktır. Puanlama sırasında aşağıdaki gibi bir metrik tutulmuştur;

Yazılım kaynak kodu üretici aracının belirli bir gereksinimi karşılaması durumunda 3 puan, kısmen karşılaması durumunda 2 puan ve karşılamaması durumunda 1 puan olacak şekilde yazılım kaynak kodu üretme araçları puanlanmıştır. En yüksek puanı alan araç için çalışmanın geri kalanında örnek bir vaka çalışması yapılacaktır. Araçları karşılaştırılması sırasında kullanılan durumlar ve bu durumların gereksinimleri aşağıda belirtilmiştir;

Durum 1 (D1): Araçların kullanılabilirliği

Gereksinim 1. Kullanım kolaylığı

Gereksinim 2. Öğrenme kolaylığı

Gereksinim 3. Teknik bilgi ve beceri gerekliliği

Gereksinim 4. Kurulum kolaylığı

Durum 2 (D2): İşlevsellik

Gereksinim 1. Modülerlik ilkelerinin kullanımı

Gereksinim 2. Fonksiyonların amacına uygun tanımlanması

Durum 3 (D3): Sürdürülebilirlik

Gereksinim 1. Modellerin tekrar kullanılabilmesi

Gereksinim 2. Modeller üzerinden otomatik test durumu üretimi

Durum 4 (D4): Taşınabilirlik

Gereksinim 1. Farklı programlama dilleri için otomatik kod üretimi

Gereksinim 2. Farklı platformlarda çalışabilme.

Durum 5 (D5): Dokümantasyon

Gereksinim 1. Yazılım kod üretme aracı yeterli dokümantasyona sahip olması

Gereksinim 2. Otomatik olarak üretilen kodun yeterli dokümantasyona sahip olması

Tablo 2'deki her bir yazılım kaynak kodu üretim aracı için Bölüm 5'de de yapılan uygulama çalışması olan Ürün Yönetimi Sistemi uygulamasının Depo Yönetimi mikro servisi için otomatik kod üretimi yapılmıştır. D1-G2, D1-G3, D2-G1, D2-G3, D3-G1, D3-G2, D5-G2 gereksinimleri doğrultusunda yazılım kaynak kodu üretim araçları ile yapılan uygulama çalışması 5 yazılım geliştirici 2 test mühendisinden oluşan yedi kişilik bir yazılım ekibi tarafından değerlendirilmiştir.

Tablo 3.2 Açık kaynak kodlu otomatik yazılım kaynak kodu üretim araçlarının karşılaştırılması

Durum	Gereksinim	Acceleo	Telosys	JET	Umple	Papyrus	Xtext	EMF-REST
D1	G1	3	3	2	2	3	2	2
	G2	2	3	2	1	3	2	2
	G3	2	2	2	2	2	2	2
	G4	3	3	3	3	3	3	3
D2	G1	3	3	3	3	3	3	3
	G2	3	3	3	3	3	3	3
D3	G1	1	3	2	2	3	2	2
	G2	1	3	1	1	1	1	2
D4	G1	3	3	1	3	1	1	3
	G2	3	3	3	3	3	3	3
D5	G1	3	3	2	2	2	3	3
	G2	3	3	2	2	3	2	3
Toplam Puan		30	35	25	27	30	27	31

Puanlama; Gereksinimi Karşılıyor: 3, Gereksinimi Kısmen Karşılıyor: 2, Gereksinimi Karşılamıyor: 1.

Tablo 2'de de görüldüğü gibi yazılım araçları birtakım kalite ölçütü olan D1, D2, D3, D4, D5 durumlarına göre karşılaştırılmıştır.

D1 durumu genel olarak araçların kullanılabilirliği için gerek olan gereksinimleri içermektedir. Bu gereksinimler;

D1-G1: Kullanım kolaylığı gereksiniminde uygulamaların açık kaynak dokümanları ve örnek çalışmalar incelenmiştir ve Depo Yönetimi mikro servisi için otomatik kod üretimi gerçekleştirilmiştir. D1-G1 gereksinimini Acceleo, Telosys ve Papyrus

araçları karşılamaktadır. JET, Umple, Xtext ve EMF-REST araçları için yeterli sayıda örnek çalışma olmadığı için gereksinimi kısmen karşılamaktadır.

D1-G2: Öğrenme kolaylığı gereksinimi için D1-G1'de de olduğu gibi açık kaynaklı dokümanlar ve yapılan örnek çalışmalar incelenmiştir. Her bir araç için Depo Yönetimi mikro servisi geliştirilirken harcanan süreye göre bir puanlama yapılmıştır. Telosys aracı için harcanan süre en az olduğu için en yüksek puan ona verilerek diğer araçlar için Telosys aracı için harcanan süre referans alınmıştır ve bu doğrultuda bir değerlendirme yapılmıştır.

D1-G3: Teknik bilgi ve beceri gerekliliği her servis için kısmen gereklidir. Otomatik olarak üretilen yazılım kaynak kodunun doğruluğunu ölçmek için en az bir programlama dili ile ilgili temel bilgilere sahip olunması gereklidir.

D1-G4: Kurulum kolaylığı gereksiniminde; araçların kurulum kolaylığı değerlendirilirken araçların tümü Intel core i5 işlemci, 8 GB ram, 256 MB SSD depolama alanı ve Windows işletim sistemine sahip olan bir bilgisayar üzerine kurulumu yapılmıştır. Uygulamaların kurulum aşamasında uygulamaların açık kaynaklı kurulum dokümanlarından yararlanılmıştır. Bu aşamada uygulamaların tümü D1-G4 için gereksinimini karşılamaktadır. Kurulumlarda bir zorluk ile karşılaşılmamıştır.

D2 durumu araçlar ile otomatik olarak üretilen yazılım kaynak kodlarının işlevsellik gereksinimlerini içermektedir. D2 durumu için bütün araçlar D2-G1 (modülerlik ilkelerinin kullanımı) ve D2-G2 (fonksiyonların amacına uygun tanımlanması) gereksinimlerini karşılamaktadır.

D3 durumu sürdürülebilirlik gereksinimlerini içermektedir. Bu gereksinimler;

D3-G1: Modellerin tekrar kullanılabilmesi gereksinimi benzer uygulamanın aynı model güncellenerek geliştirilmesi ve uygulamanın daha sonrasında kolay bir şekilde bakım yapılabilir olmasına göre değerlendirilmiştir. Bu değerlendirme sonucunda Telosys ve Papyrus aracı gereksinimi karşılamaktadır. Acceleo aracı gereksinimi karşılamamaktadır. Diğer araçlar ise kısmen karşılamaktadır.

D3-G2: Modeller üzerinden otomatik test durumu üretimi gereksinimi bir model üzerinden aynı zamanda yazılımın birim testleri ve fonksiyonel test senaryolarının

üretiminin gerçekleşmesi olarak değerlendirilmiştir. Telosys aracı bu iki durumu da karşılamaktadır. EMF-REST aracı kısmen karşılamaktadır. Diğer araçlar karşılamamaktadır.

D4 durumu taşınabilirlik gereksinimlerini içermektedir. D4-G1 gereksinimi Acceleo, Telosys, Umple ve EMF-REST araçları karşılamaktadır, diğer araçlar ile sadece Java dili için kod üretimi gerçekleştirilebildiği için karşılamamaktadır. D4-G2 (Farklı platformlarda çalışabilme) gereksinimi bütün araçlar karşılamaktadır. D4-G2 gereksinimi için uygulamalar Windows ve Linux işletim sistemine ait ortamlarda test edilmiştir.

D5 durumu yazılım kaynak kodu üretim aracının ve oluşturulan kaynak kodlarının dokümantasyonu için gereksinimler içermektedir. Bu gereksinimler;

D5-G1: Yazılım kaynak kod üretme aracı yeterli dokümantasyona sahip olması gereksinimi için araçların büyük çoğunluğu gereksinimi karşılamaktadır. JET, Umple ve Papyrus araçları için diğer araçlara göre yapılan çalışmalar daha az rastlanılmıştır. Bu araçlar D5-G1 gereksinimi kısmen karşılamaktadır.

D5-G2: Otomatik olarak üretilen kodun yeterli dokümantasyona sahip olması gereksinimi için her bir servis için yapılan uygulama çalışması sonucu oluşturulan yazılım kaynak kodları incelenmiştir. İnceleme sonucunda Acceleo, Telosys, Papyrus ve EMF-REST ile oluşturulan kodlarda her bir sınıf ve metot için yeterli dokümanın oluştuğu gözlemlenmiştir ve bu araçlar gereksinimi karşılamaktadır. Diğer araçlar sadece sınıflar için doküman oluşturmuşlardır ve bu gereksinimi kısmen karşılamaktadırlar.

Sonuç olarak, yapılan karşılaştırmalar ve puanlama sonucu en yüksek puanı alan ve kalite ölçütlerini büyük oranda karşılayan servis olarak Telosys aracı seçilmiştir. Telosys aracının en yüksek puanı almasının ve belirlenen kalite kriterlerine göre diğer uygulamalara göre ön plana çıkmasının en önemli nedenlerinin başında tek bir model üzerinden birçok çerçeve ve programlama dili için yazılım kaynak kodu, birim test ve fonksiyonel testlerini otomatik olarak üretebilmesidir. Aynı zamanda Eclipse ile entegre olarak kullanılabildiği için kurulumu basit ve hızlıdır. Bölüm 5'de

model güdümlü yazılım yaşam döngüsünde Telosys aracı kullanılarak bir uygulama çalışması yapılmıştır.

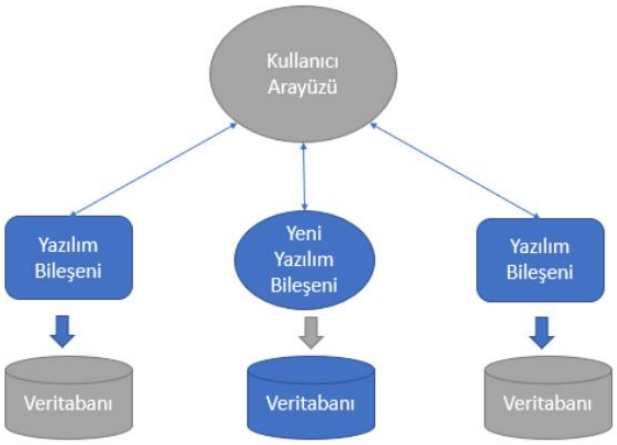
4
MGY İLE MİKROSERVİS GELİŞTİRİLMESİ

4.1 Mikroservis Mimarisi

Mikroservis mimari modeli, monolitik mimariye sahip uygulamaları daha küçük, bağımsız tek bir amaca hizmet eden parçalara bölme felsefesine dayanan modüler mimari stilinin bir örneğidir. Mikroservis mimari modeli, uygulama ve veri tabanı arasındaki ilişkiyi önemli ölçüde etkiler. Tek bir veri tabanı şemasını diğer servisler ile paylaşmak yerine, her servisin kendi veri tabanı şeması vardır ve her servis ihtiyaçlarına en uygun veri tabanı türünü kullanabilir. Bu yaklaşım çoğu zaman bazı verilerin çoğaltılarak tekrarlanmasına neden olabilir. Ancak, mikroservislerden yararlanılmak isteniyor ise, servis başına bir veri tabanı şemasına sahip olmak önemlidir, çünkü bu durum servislerin birbirlerine bağımlılığını önemli ölçüde azaltır. Netflix'in monolitik mimaride olan servis yapısını mikroservis mimarisi ile değiştirmesindeki en önemli nedenlerden biri de tek bir veri tabanına sahip olması ve bu durumdan kaynaklanan bir hatadan dolayı birkaç gün hizmet veremediğinde aldığı ciddi zarar olarak biliniyor [21]. Bu sebepten Netflix Şekil 4.1'de gösterildiği gibi monolitik yapıda olan servislerini mikroservislere bölmüştür.

Şekil 4.1 Netflix mikroservis mimari yapısı [21]

Netflix gibi birçok şirket benzer sebeplerden dolayı mikroservis yapısını tercih etmişlerdir. Örneğin; yeni teknolojilere kolay adapte olabilmek, Şekil 4.2'de gösterildiği gibi yeni bir servisi en az maliyetle çalışan uygulamaya dahil edebilmek vb.



Şekil 4.2 Mikroservis mimarisi örneği

Bu bölümün geri kalanında mikroservis mimarisinin sağladığı faydalardan bahsedilmiştir.

Mikroservis Mimarisinin Uygulama Geliştirirken Sağladığı Faydalar

Mikroservisler, yazılım geliştirilirken sağladığı faydalardan dolayı Amazon, Netflix ve eBay gibi büyük teknoloji şirketlerinin de uygulamalarını mikroservis mimari yapısına geçirmesine neden olmuştur. Monolitik mimari ile karşılaştırıldığında mikroservislerin sağladığı faydalar:

- Büyük uygulamalar birbirinden bağımsız servislerden oluştukları için tek bir yazılım bileşeninin arızalanmasından çoğunlukla etkilenmezler.

- Servisler işlevine uygun olan teknolojiler kullanılarak geliştirilebilir. Bu sayede servisler arasındaki teknoloji bağımlılığı ortadan kalkmaktadır.

-Mikroservislerin monolitik servislere göre daha anlaşılırdır. Bu sayede uygulamaya yeni bir özellik eklemek ya da iyileştirme yapmak daha kolaydır.

4.2 Mikro Servis Mimarisi ile Uygulama Geliştirilirken Model Güdümlü Yaklaşımın Sağladığı Faydalar

Mikroservis mimarisi kullanılarak model güdümlü yazılım geliştirmenin sağladığı faydalar aşağıdaki gibidir:

- Model güdümlü geliştirme kullanarak uygulama geliştirmek daha hızlıdır. Model güdümlü geliştirmede, bir yazılım uygulamasının modeli, geleneksel programlama dillerinden daha yüksek bir soyutlama düzeyinde belirtilir. Bu model ile otomatik olarak üretilen yazılım kaynak kodları sayesinde çalışan bir yazılım uygulaması elde edilir. Kullanılan model daha yüksek bir soyutlama düzeyinde iken, kodda ifade edilen aynı modele kıyasla çok daha küçüktür. Başka bir deyişle; modeldeki her öğe birden çok kod satırını temsil eder. Mikroservis mimarisine ile geliştirilen bir uygulama için oluşturulacak modeller daha basit olacağı için mikroservisler için sağlam ve kaliteli yazılım kodları elde edilebilir. Bir model üzerinden çeşitli programlama dillerini destekleyen otomatik kod üretim araçları ile mikroservislerin amacına ve işlevine uygun kodlar üretilebilir. Bu durum mikroservis mimarisi ile uygulama geliştirmenin temel özelliklerinden olan amacına uygun programlama dili ile servislerin geliştirilmesi gereksinimi bire bir karşılamaktadır.

- Model güdümlü geliştirme yaklaşımı ile geliştirilen uygulamalar daha az maliyetlidir. İlk neden, model güdümlü geliştirme yaklaşımı yazılım geliştirme süresini kısalttığı için uygulamalar daha kısa bir pazara sunum süresine sahip olurlar. İkinci neden, daha düşük maliyetle (daha az insan gücü, kısa geliştirme süresi vb.) ile uygulama geliştirmenin mümkün olmasıdır. Model güdümlü geliştirme yaklaşımı kullanılarak geliştirilen mikroservislerde değişiklik yapılması ve servislerin bakımı da daha az maliyetli olur. Mikroservis mimarisi kullanılarak geliştirilen uygulamalar birbirinden bağımsız servislere sahip olduğu için bakım maliyeti model güdümlü geliştirme ile daha da düşürülebilir. Uygulamanın tamamını güncellemek yerine ilgili servisin modeli güncellenerek aynı doğrultuda yazılım kaynak kodları otomatik olarak üretilebilir.

- Model güdümlü geliştirme yaklaşımı yazılım kaynak kodunun kalitenin artmasına yol açar. Model güdümlü yaklaşım ile geliştirilen bir uygulamanın temelini üst düzey soyut bir model oluşturduğu için; uygulamanın (teknik) kalitesi model üzerinden yazılım kaynak kodunu otomatik olarak üreten araca bağlıdır. Yazılım kaynak kodlarını otomatik olarak üreten araçlar en iyi deneyimlere göre yazılım kaynak kodlarını üretirler, ayrıca yazılım kodunun dokümantasyonunu da sağlarlar. Bu nedenle, model güdümlü yaklaşım ile geliştirilmiş mikroservislerin kod kalitesinde artış olur çünkü en iyi deneyimler ile üretilmiş ve tek bir elden çıkmış daha az kod tekrarı olan yazılım kaynak kodları elde edilir.

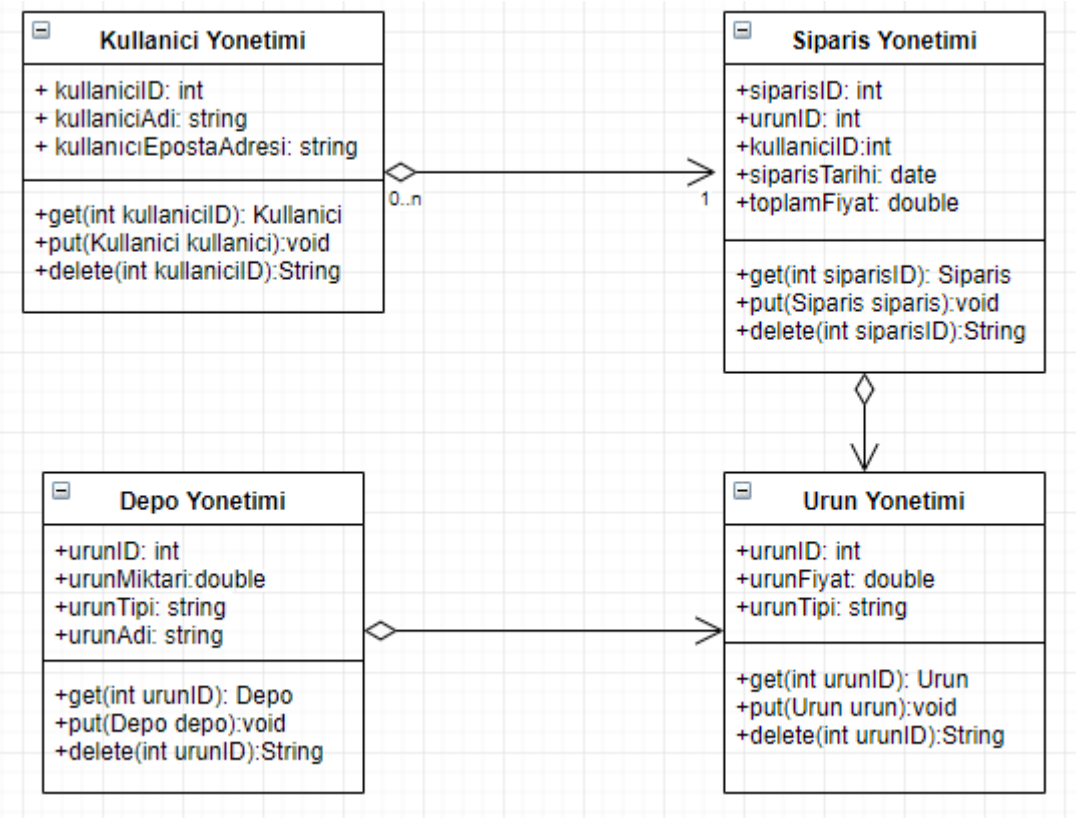
- Model güdümlü geliştirme yaklaşımı ile modeller üzerinden otomatik test durumları üretimi de gerçekleştirilebilir. Bu durum özellikle tek bir işleve sahip mikroservisler için bütün durumların üzerinden geçilmesini ve yazılım test maliyetinin büyük oranda azaltılmasına fayda sağlamaktadır. Mikro servisler tek bir amaca hizmet ettiği için karmaşık modellere sahip monolitik servislere göre oluşturulan test durumları kodun büyük bir kısmını kapsar. Böylece yazılım hataları test sırasında kolay bir şekilde tespit edilebilir ve hızlı bir şekilde düzeltilebilir.

5
UYGULAMA ÇALIŞMASI

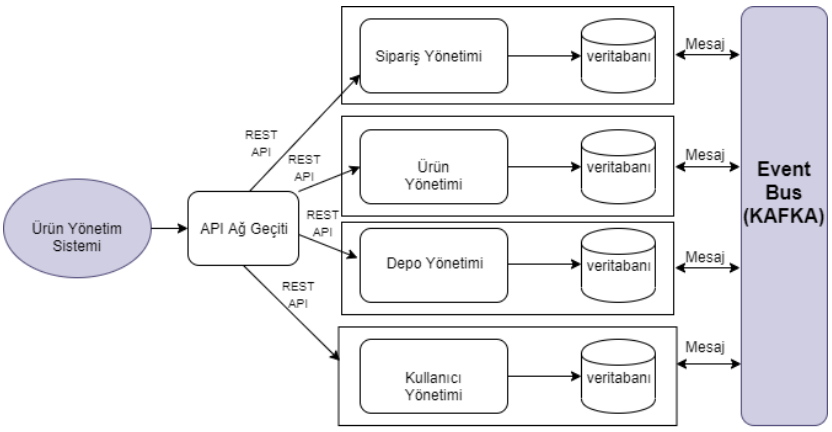
Mikroservisler model güdümlü bir yaklaşım ile geliştirilirken; model oluşturma, model üzerinden otomatik kod üretme, model üzerinden otomatik testlerin oluşturulması ve canlı ortama dağıtılması adımları izlenir. Bu bölümde model güdümlü yaklaşım kullanılarak Şekil 5.1'de de UML sınıf diyagramı gösterilen monolitik mimari modeline sahip bir ürün yönetim sistemi uygulaması Şekil 5.2' deki gibi mikroservislere bölünerek tekrardan geliştirilmiştir.

Çalışmada mikroservis mimarisinin tercih edilmesinin nedeni mikroservis mimarisinin uygulama geliştirirken sağladığı faydalardan yararlanmaktır. Fakat bu çalışmada tercih edilmesinin başlıca sebebi mikroservislerin genellikle tek bir işleve sahip basit uygulamalar olduğu için daha az karmaşıklığa sahip olmalarıdır. Model güdümlü olarak uygulama geliştirilirken karmaşık modeler yerine daha basit modeler üzerinden üretilen kodların güvenilirlik ve kalitesini ölçmek daha basittir ve otomatik olarak üretilen yazılım kaynak kodları daha anlaşılırdır. Ayrıca model güdümlü olarak mikroservisler geliştirilirken benzer modeler ile benzer servisler birbirlerinden bağımsız olarak amaçlarına uygun olan farklı programlama dilleri ile oluşturulabilirler. Bu sebeple Bölüm 3.5 'de yazılım kodunu otomatik üreten araçlar karşılaştırılırken D4-G1 gereksinimi bir ölçüt olarak değerlendirilmiştir. Bu ölçüt özellikle mikroservis mimarisi ile uygulama geliştirirken önem kazanmaktadır. Bu şekilde aynı ya da benzer modeler ile farklı programlama dilleri ile yazılım kaynak kodları üretilebilir. Ayrıca mikroservis mimarisinin sağladığı diğer bir fayda ise servisler paralel bir şekilde geliştirilebilirler. Böylece bütün bir sistem modelinin tek bir seferde oluşturulmasına gerek kalmamaktadır. Modeller daha basit ve anlaşılır olarak bölünebilirler. Basit modeler üzerinden otomatik kod üretmek ya da modeler referans alınarak geliştirme yapmak uygulama geliştiricilerine de kolaylık sağlamaktadır. Çalışmanın bu kısmında Şekil 5.2'de de mikroservis mimari modeli gösterilen RESTful bir ürün yönetimi sistemi uygulamasının depo yönetimi mikroservisi geliştirilmiştir. Uygulama geliştirilirken izlenilen adımlarda bölüm 3.5'de otomatik kod üretim araçlarının karşılaştırılması sonucunda en yüksek puanı

alan otomatik kod üretme aracı olan Telosys aracı kullanılmıştır. Ayrıca, mikroservis geliştirilmesi sırasında her bir adımda model güdümlü yaklaşımın sağladığı faydalar ve model güdümlü yaklaşımın uygulanabilirliğinden bahsedilmiştir.



Şekil 5.1 Ürün yönetimi sistemi UML sınıf diyagramı



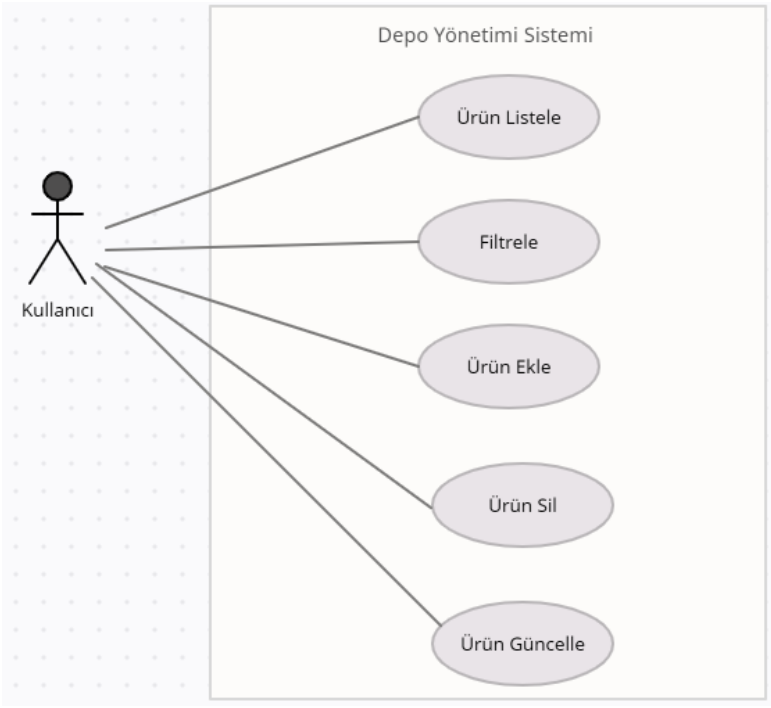
Şekil 5.2 Ürün yönetimi sistemi mikroservis mimari modeli

5.1 Gereksinim Analizi

Ürün yönetimi sistemi; sipariş yönetimi, ürün yönetimi, depo yönetimi ve kullanıcı yönetimini içeren dört adet mikroservisten oluşan bir uygulamadır. Şekil 5.3 'de kullanım senaryosu diyagramı paylaşılan Depo Yönetimi Sistemi mikroservisinin gereksinimleri aşağıdaki gibidir;

1. Kullanıcılar depodaki ürünlerin tamamını listeleyebilirler.
2. Kullanıcılar depodaki ürünleri kategorilerine göre listeleyebilirler.
3. Kullanıcılar depoya ürün ekleyebilirler.
4. Kullanıcılar depodaki ürünleri silebilirler.
5. Kullanıcılar depodaki ürünleri güncelleyebilirler.

Yukarıda gereksinimleri tanımlanan depo yönetimi mikroservisi çalışmanın geri kalanında model güdümlü yaklaşım ile Telosys aracı kullanılarak geliştirilecektir.



Şekil 5.3 Depo yönetimi sistemi mikroservisi kullanım senaryosu diyagramı

5.2 Tasarım

Depo yönetimi mikroservisinin gereksinimleri çıkartıldıktan sonra gereksinimler doğrultusunda UML diyagramı, veri tabanı şeması ve DSL (Alana özel dil) gibi model

yapıları kullanılarak modeller oluşturulabilir. Depo yönetimi mikroservisinin geliştirilmesi sırasında DSL kullanılmıştır.

DSL

Alana özgü dil olarak da adlandırılan DSL, belirli bir amaç için kullanılan özel bir dildir. Belirli bir sorunu çözmek için kullanılır. Bu çalışmada Şekil 5.4'de de görüldüğü gibi Eclipse'e eklenti olarak kurulan Telosys aracı kullanılarak DSL oluşturulmuştur.

```
// Entity Depo

 Depo {
  urunID: int {@AutoIncremented,@Id,@InitialValue(100)};
  urunMiktari : double {@NotNull }  ;
  urunTipi : string {@NotNull }  ;
  urunAdi : string {@NotNull, @SizeMax(50)}  ;
}
```

Şekil 5.4 Telosys aracı ile Eclipse platformunda oluşturulmuş DSL örneği

Mikroservisler karmaşık sistemler olmadığı için depo yönetimi mikroservisinin gereksinimleri doğrultusunda oluşturulan model örneği Şekil 5.4'de de görüldüğü gibi yazılım geliştirme ekibindeki tüm üyeler tarafından anlaşılır ve güncellenebilir basitliktedir. Oluşturulan model doğrultusunda, depo yönetimi mikroservisi için RESTful bir web servis oluşturulmuştur. Bu servisin işlevleri şekil 5.3'de kullanım senaryosu diyagramında gösterildiği gibidir. Mikroservise daha sonradan yeni bir özellik eklemek basit bir modele sahip olduğu için daha az maliyetlidir. Yazılım ekibindeki herhangi bir kişi tarafından model güncellenerek yeni özellik uygulamaya eklenebilir.

Şekil 5.4'deki DSL'deki alanlardan bahsetmek gerekirse "urunID" alanı integer (tam sayı) tipinde ilk değer olarak 100 'den başlayan ve otomatik olarak birer birer artan, benzersiz bir alandır. "urunMiktari" alanı double (kesirli sayı) tipindedir ve boş bırakılamaz. "urunTipi" alanı string(dizi) tipindedir ve boş bırakılamaz. "urunAdi" alanı string(dizi) tipindedir ve boş bırakılamaz.

Çalışmanın geri kalanında Şekil 5.4'deki DSL modeli kullanılarak otomatik olarak yazılım kaynak kodları, birim testleri ve fonksiyonel test senaryoları üretilecektir.

5.3 Otomatik Kod Üretme

Bu kısımda Eclipse ile birlikte Telosys aracı kullanılmıştır. Telosys UML, sınıf diyagramı, DSL gibi modelleri kullanarak otomatik bir Java kod çerçevesi oluşturur. Ücretsiz ve açık kaynaklı bir kod üretme aracıdır. Şekil 5.5'de, Telosys kullanarak Eclipse'de otomatik olarak oluşturulan kod parçasının bir örneği gösterilmektedir.

```
/**
 * 'DELETE' action processing. <br>
 * This action is based on the 'Post/Redirect/Get (PRG)' pattern, so it ends by 'http redirect'<br>
 * @param redirectAttributes
 * @param urunID  primary key element
 * @return
 */
@RequestMapping(value = "/delete/{urunID}") // GET or POST
public String delete(RedirectAttributes redirectAttributes, @PathVariable("urunID") int urunID) {
    log("Action 'delete'" );
    try {
        boolean deleted = depoService.deleteById( urunID );
        log("Depo deleted. Key : " + toString(urunID) + " result = " + deleted );
        //--- Set the result message
        messageHelper.addMessage(redirectAttributes, new Message(MessageType.SUCCESS,"delete.ok"));
    } catch(Exception e) {
        messageHelper.addException(redirectAttributes, "depo.error.delete", e);
    }
    return redirectToList();
}
```

Şekil 5.5 Eclipse'de Telosys ile oluşturulmuş kod parçası

Şekil 5.5'de de göründüğü gibi depo yönetimi mikroservisinin ürün silme gereksinimi için oluşturulmuş basit bir kod parçası örneği vardır. Otomatik üretilmiş kodların tamamı Ek A'daki gibidir. Otomatik üretilen kodlar Şekil 5.5' de de görüldüğü gibi kod içinde açıklayıcı dokümantasyonlar içermektedir. Bu durum yazılım geliştiricilerine, yazılım kodunda ekleme ya da güncelleme yaptıkları sırada büyük ölçüde kolaylık sağlamaktadır.

5.4 Birim Testi

Yazılımın en küçük fonksiyonunu test ettiğimiz birim testleri, yazılımın kalitesine önemli ölçüde fayda sağlar. Birim testleri kusurların düşük bir maliyetle erken zamanda bulunması için kullanılır. Model güdümlü geliştirme yönteminde, otomatik kod üretme araçları genellikle model üzerinden otomatik birim test üretimi de gerçekleştirirler. Bu sayede, birim testler tüm kodu kapsayacak şekilde otomatik olarak oluşturulur ve kod kapsamını artırmak için fazladan çabaya ihtiyaç duyulmaz. Şekil 5.6'da Eclipse üzerinde Telosys aracı kullanılarak otomatik üretilmiş birim test örneği gösterilmiştir;

```
@Test
public void testPutGetRemove() {

    System.out.println("Testing class DepoCache ..." );

    Depo depo = new Depo();
    populate(depo);
    System.out.println("Entity populated : " + depo );

    DepoCache.putDepo(depo) ;   // Store in cache

    Depo depo2 = DepoCache.getDepo( depo.getUrunID() );
    assertTrue( depo == depo2 ) ; // Same instance

    DepoCache.removeDepo(  depo.getUrunID() ) ; // Remove from cache

    Depo depo3 = DepoCache.getDepo( depo.getUrunID() );
    assertTrue( null == depo3 ) ; // Not in cache
    assertNull(depo3); // Not in cache

}
```

Şekil 5.6 Eclipse'de Telosys ile otomatik olarak üretilmiş birim testi örneği

Telosys aracı ile Şekil 5.4'deki modelde bulunan her bir gereksinimi kapsayacak şekilde Ek B'de paylaşılan iki adet otomatik test senaryosu üretilmiştir. Otomatik oluşturulan birim testlerinin tümü Eclipse platformunda koşulmuştur ve tümü başarılı olmuştur.

5.5 Fonksiyonel Test

Fonksiyonel test, yazılımın işlevselliğini test etmek için kullanılan bir yöntemdir. Manuel ya da otomatik olarak gerçekleştirilebilir. Bu çalışmada Telosys aracı ile Şekil 5.3'deki model üzerinden otomatik olarak test senaryosu oluşturulmuştur. Ek C'deki otomatik oluşturulan test senaryosu REST uygulamaları test etmek için kullanılan Postman [22] uygulaması ile uyumludur. Postman, RESTful uygulamaların arka planda çalışan servislerinin durumunun ve çalışma hızının test edilebildiği ücretsiz bir uygulamadır. Bu aşama test senaryolarının model üzerinden otomatik üretilmesi uygulamanın geliştirilmesi sırasında test maliyetini ciddi bir şekilde düşürür. Ayrıca bütün senaryolar için test durumu üretildiğinden yazılımın kalitesini artırır ve bakım maliyetini azaltır. Uygulamaya yeni bir özellik eklendiğinde test senaryoları da otomatik olarak güncellenir ve sistemin kalitesi

korunur. Depo yönetimi servisi için Postman uygulamasında koşulan test senaryolarının tümü başarılı olmuştur.

5.6 Dağıtım ve Hizmet Yönetimi

Model güdümlü yaklaşım ile bulut ortamına ve fiziksel ortama dağıtım için literatürde yapılan çalışmalar mevcuttur, fakat bu çalışmada oluşturduğumuz model üzerinden doğrudan otomatik dağıtım dosyasını oluşturabilecek bir araç kullanılmamıştır. Dağıtım manuel bir şekilde oluşturulmuş Ek D'de paylaşılan docker dosyası ile depo yönetimi uygulaması için bir docker image oluşturulmuştur ve dağıtımda kullanabilmek için dockerhub'a yüklenmiştir. Sonrasında Ek D'de paylaşılan bir dağıtım dosyası ile Openshift bulut ortamına dağıtımı manuel olarak gerçekleştirilmiştir. Bu aşamada her bir mikro servis için ortam gereksinimleri farklılık gösterebileceğinden model güdümlü geliştirme yaklaşımı ile oluşturulmuş modellerin her bir servis için güncellenmesi gerekebilir ve bu yaklaşım geliştiriciler açısından kullanışlı olamayabilir. Bu aşamada model güdümlü geliştirmenin mikro servislerin dağıtımı sırasında yetersiz kaldığı gözlemlendi.

5.7 Uygulama Değerlendirmesi

Yapılan çalışmada gözlemlenen, model güdümlü yaklaşım, özellikle yazılım kodu ve otomatik test senaryolarının üretiminde yazılımın verimliliğini ve hızını arttırmaktadır. Bu durum mikroservislerin geliştirilmesinde önemli bir fayda sağlamaktır. Ayrıca, kod şablonunun otomatik oluşturulması, yinelenen kodların azaltılması, geliştirme süresinin kısalması gibi birçok fayda sağladığı gözlemlenmiştir. Mikroservisler platformdan bağımsız uygulamalardır. Model güdümlü bir yaklaşım ile geliştirme yapılarak platform bağımlılığı ortadan kaldırılmıştır. Model güdümlü yaklaşım ile benzer mikroservisler kısa bir süre içinde modeller güncellenerek oluşturulabilir. Mikroservisler tek bir işleve sahip olan basit servisler olduğu için model güdümlü yazılım geliştirme yaklaşımı ile uygulama geliştirilmesinin uygun olduğu gözlemlenmiştir. Büyük ve karmaşık uygulamalarda aynı doğrultuda model karmaşıklığı da artacağından otomatik oluşturulan yazılım kaynak kodunun doğruluğunun ölçülmesi ve kaynak kodunun

güncellenmesi zorlaşmaktadır, bu nedenle model güdümlü yaklaşımın daha basit uygulamalar için daha verimli sonuçlar verdiği gözlemlenmiştir.

6
SONUÇ VE ÖNERİLER

Model Güdümlü Yazılım Geliştirme (MGYG) yaklaşımı, uygulamanın yazılım yaşam döngüsünü hızlı, verimli, minimum maliyet ve sürede tamamlamak için kullanılan bir yazılım geliştirme yaklaşımıdır. Model güdümlü yazılım geliştirme yaklaşımında, uygulamanın ilk olarak modeli oluşturulur ve bu doğrultuda uygulamanın kaynak kodlarının üretilmesi hedeflenir. Bu çalışmada, model güdümlü yaklaşım ile yazılım geliştirilirken en önemli aşamalardan olan yazılım kaynak kodlarının modeller üzerinden otomatik olarak üretilmesi sırasında kullanılan açık kaynak kodlu otomatik yazılım kaynak kod üretme araçları tanıtılmış ve birtakım kalite ölçütlerine göre karşılaştırılmıştır. Yazılım kaynak kodu üretme araçları karşılaştırılırken seçilen kalite ölçütleri genel olarak araçların kullanım kolaylığı, dokümantasyonu, platformdan bağımsız olması ve oluşturulan kodun kalitesi ile ilişkilidir. Belirlenen kalite ölçütleri için birtakım gereksinimler çıkarılmıştır ve bu gereksinimler üzerinden 1 ile 3 aralığında bir puanlama sistemi ile araçlara gereksinimleri karşılama durumda 3, kısmen karşılama durumunda 2, karşılamama durumda 1 puan olacak şekilde bir puanlama yapılmıştır. Ayrıca, en yüksek puan alan araç olan Telosys yazılım kaynak kodu üretme aracı için bir uygulama çalışması yapılmıştır. Yapılan uygulama çalışmasında gözlemlenen, model güdümlü yaklaşım, özellikle yazılım kaynak kodu ve otomatik test senaryolarının üretiminde yazılımın verimliliğini ve hızını artırmaktadır. Bu durum uygulama maliyetini ciddi oranda azaltmaktadır. Ayrıca, kod şablonunun otomatik oluşturulması, yinelenen kodların azaltılması, geliştirme süresinin kısalması gibi birçok alanda fayda sağladığı gözlemlenmiştir. Uygun bir yazılım kaynak kodu üretme aracı ile benzer uygulamaların kısa sürede geliştirilebildiği gözlemlenmiştir. Bu durumun yazılım geliştiricilerine de büyük kolaylık sağladığı gözlemlenmiştir. Yazılım geliştiriciler yazılım geliştirme süresinin kısalması ile tasarıma daha çok odaklanıp daha bakım yapılabilir, kaliteli yazılımlar elde edilmesi için çalışabilirler. Ayrıca, otomatik kod üretme araçlarının iyileştirilmesi ve yeni özellikler eklenmesi için de çalışmalar yapabilirler.

Uygulama çalışması bölümünde kullanılan mikroservis mimarisinin model güdümlü yaklaşım ile geliştirilmesinin uygulama geliştirilmesi aşamasında sağladığı faydalardan da bahsedilmiştir. Mikroservisler küçük ve basit servisler olduklarından monolitik uygulamalara göre model güdümlü geliştirmenin daha uygulanabilir olduğu gözlemlenmiştir. Monolitik servisler mikroservislere göre daha karmaşık tasarımlara sahiptir. Monolitik servisler için üretilen otomatik kod satırı daha fazla olacağı için otomatik olarak oluşturulan yazılım kaynak kodunun doğruluğunun ve kalitesinin de ölçülmesinin zorlaştığı gözlemlenmiştir. Bu nedenle mikroservisler için otomatik üretilen kodlar daha bakım yapılabilir ve anlaşılırdır.

Yapılan çalışmalar ve gözlemler sonucunda; model güdümlü yazılım geliştirmenin bir geliştirici dahil olmadan sadece model üzerinden çalıştırılabilir bir uygulama geliştirilmesi için literatürde yapılan çalışmalar ve otomatik kod üretim araçlarının yetersiz kaldığı gözlemlenmiştir. Otomatik olarak oluşturulan kodlar yazılım geliştiriciye zaman kazandırsa da otomatik oluşturulan kodlar iş mantığının tümünü kapsamadığı için bir geliştiricinin yazılım geliştirme sürecine dahil olması gerekmektedir. Ayrıca otomatik olarak oluşturulan kodların birtakım tasarım kalıplarını (factory pattern, builder pattern vb.) da desteklemediği gözlemlenmiştir. Tasarım kalıpları özelinde otomatik kod üretim araçlarının iyileştirilmesi ve yazılım geliştiricisi dahil olmadan iş mantığını kapsayan çalıştırılabilir uygulamalar üreten otomatik kod üretme araçlarının geliştirilmesi için çalışmalar yapılabilir. Son olarak, uygulama modeli üzerinden otomatik olarak kurulumun sağlanabilmesi için de bir kurulum dosyası oluşturabilen otomatik kod üretme aracı geliştirilebilir.

KAYNAKÇA

[1] Schreibmann, Vitaliy & Braun, Peter. (2015). Model-driven development of RESTful APIs. 5-14. 10.5220/0005411200050014.

[2] Viviana Yarel Rosales-Morales, Giner Alor-Hernández, Jorge Luis García-Alcaráz, Ramón Zatarain-Cabada, María Lucía Barrón-Estrada. (2015). An analysis of tools for automatic software development and automatic code generation. Revista Facultad de Ingeniería, Universidad de Antioquia, No. 77, pp. 75-87, 2015.

[3] Gonçalves, Rafael & Azevedo, Isabel. (2019). RESTful Web Services Development With a Model-Driven Engineering Approach. 10.4018/978-1-5225-7455-2.ch009.

[4] Eclipse Modeling Framework (EMF). [Online], Available: https://www.eclipse.org/modeling/emf/ [Accessed:10-01-2021].

[5] Viet-Cuong Nguyen.(2015). Model Driven Testing of Web Applications Using Domain Specific Language.

[6] MDA. [Online], Available: https://www.omg.org/mda [Accessed:10-01-2021].

[7] International Organization for Standardization (ISO), Software engineering - Product quality – Part 1: Quality Model, ISO/IEC 9126-1:2001, 2001.

[8] "Domain-Specific Languages" [Online], Available: https://www.jetbrains.com/mps/concepts/domain-specific-languages[Accessed:07-12 2020].

[9] xText. [Online], Available: http://xtext.com/ [Accessed 12-01-2021].

[10] MPS. [Online], Available: https://www.jetbrains.com/mps [Accessed:12-01-2021].

[11] Introduction To Omg's Unified Modeling Language. [Online], Available: https://www.uml.org/what-is-uml.htm[Accessed:07-12 2020].

[12] ISO 9126 Software Quality Characteristics. [Online], Available: http://www.sqa.net/iso9126.html [Accessed:10-01-2021].

[13] ISO/IEC 25010:2011. [Online], Available: https://www.iso.org/standard/35733.html [Accessed:07-12-2020].

[14] An analysis of tools for automatic software development and automatic code generation, Revista Facultad de Ingeniería, Universidad de Antioquia, No. 77, pp. 75-87, 2015

[15] Acceleo. [Online], Available: https://www.eclipse.org/acceleo/ [Accessed:07-12-2020].

[16] Telosys. [Online], Available: https://www.telosys.org/eclipsePlugin.html [Accessed: 12-01-2021].

[17] Umple. [Online], Available: https://cruise.eecs.uottawa.ca/umple/ [Accessed:07-12-2020].

[18] Papyrus. [Online], Available: https://www.eclipse.org/papyrus/ [Accessed:07-12-2020].

[19] EMF-REST. [Online], Available: https://som-research.uoc.edu/tools/emf-rest/ [Accessed:07-12-2020].

[20] JET .[Online], Available: https://www.eclipse.org/modeling/m2t/?project=jet [Accessed:12-01 2021].

[21] 10 companies that implemented the microservice architecture and paved the way for others. [Online], Available https://divante.com/blog/10-companies-that-implemented-the-microservice-architecture-andpaved-the-way-for-others/ [Accessed: 12-01-2021].

[22] Postman. [Online], Available: https://www.postman.com/ [Accessed: 12-01 2021].

A
DEPO YÖNETİM SİSTEMİ İÇİN OTOMATİK ÜRETİLMİŞ JAVA KODU

DepoController.java sınıfı;

```
/*
 * Created on 2020-12-20 ( Date ISO 2020-12-20 - Time 16:14:31 )
 * Generated by Telosys ( http://www.telosys.org/ ) version 3.1.2
 */


package main.java.org.demo.web.mvc.spring.controller;

import java.util.List;

import javax.annotation.Resource;
import javax.servlet.http.HttpServletRequest;
import javax.validation.Valid;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;
```

```
//--- Common classes
import org.demo.web.mvc.spring.commons.AbstractController;
import org.demo.web.mvc.spring.commons.FormMode;
import org.demo.web.mvc.spring.commons.Message;
import org.demo.web.mvc.spring.commons.MessageType;

//--- Entities
import org.demo.data.record.DepoRecord;

//--- Persistence services
import org.demo.persistence.DepoPersistence;



/**
 * Spring MVC controller for 'Depo' management.
 */
@Controller
@RequestMapping("/depo")
public class DepoController extends AbstractController {

        //--- Variables names ( to be used in JSP with Expression Language )
        private static final String MAIN_ENTITY_NAME = "depo";
        private static final String MAIN_LIST_NAME   = "list";

        //--- JSP pages names ( View name in the MVC model )
```

```
    private static final String JSP_FORM   = "depo/form";
    private static final String JSP_LIST   = "depo/list";

    //--- SAVE ACTION ( in the HTML form )
    private static final String SAVE_ACTION_CREATE   = "/depo/create";
    private static final String SAVE_ACTION_UPDATE   = "/depo/update";

    //--- Main entity service
    @Resource
private DepoPersistence depoService; // Injected by Spring
    //--- Other service(s)

    //----------------------------------------------------------------------------------
    /**
     * Default constructor
     */
    public DepoController() {
        super(DepoController.class, MAIN_ENTITY_NAME );
        log("DepoController created.");
    }

    //----------------------------------------------------------------------------------
    // Spring MVC model management
    //----------------------------------------------------------------------------------
```

```
    /**
     * Populates the Spring MVC model with the given entity and eventually other useful data
     * @param model
     * @param depo
     */
    private void populateModel(Model model, DepoRecord depo, FormMode formMode) {
        //--- Main entity
        model.addAttribute(MAIN_ENTITY_NAME, depo);
        if ( formMode == FormMode.CREATE ) {
            model.addAttribute(MODE, MODE_CREATE); // The form is in "create" mode
            model.addAttribute(SAVE_ACTION, SAVE_ACTION_CREATE);

            //--- Other data useful in this screen in "create" mode (all fields)
        }
        else if ( formMode == FormMode.UPDATE ) {
            model.addAttribute(MODE, MODE_UPDATE); // The form is in "update" mode
            model.addAttribute(SAVE_ACTION, SAVE_ACTION_UPDATE);

            //--- Other data useful in this screen in "update" mode (only non-pk fields)
        }
    }
```

```
//-------------------------------------------------------------------------------
// Request mapping
//-------------------------------------------------------------------------------
/**
 * Shows a list with all the occurrences of Depo found in the database
 * @param model Spring MVC model
 * @return
 */
@RequestMapping()
public String list(Model model) {
        log("Action 'list'");
        List<DepoRecord> list = depoService.findAll();
        model.addAttribute(MAIN_LIST_NAME, list);
        return JSP_LIST;
}

/**
 * Shows a form page in order to create a new Depo
 * @param model Spring MVC model
 * @return
 */
@RequestMapping("/form")
public String formForCreate(Model model) {
        log("Action 'formForCreate'");
```

```
        //--- New record instance (it will be used to initialize the web form
fields )
        DepoRecord depo = new DepoRecord();
        //--- Initialize the instance here if necessary...
        // depo.setXxxx("XX");
        //--- Populates the model with the new instance
        populateModel( model, depo, FormMode.CREATE);
        //--- Redirect to the 'FORM VIEW' ( JSP )
        return JSP_FORM;
    }

    /**
     * Shows a form page in order to update an existing Depo
     * @param model Spring MVC model
     * @param urunID  primary key element
     * @return
     */
    @RequestMapping(value = "/form/{urunID}")
    public String formForUpdate(Model model, @PathVariable("urunID") int
urunID ) {
        log("Action 'formForUpdate'");
        //--- Search the entity by its primary key
        DepoRecord depo = depoService.findById(urunID);
        //--- Populates the model with the instance
        populateModel( model, depo, FormMode.UPDATE);
```

```
            //--- Redirect to the 'FORM VIEW' ( JSP )
            return JSP_FORM;
        }

        /**
         * 'CREATE' action processing. <br>
         * This action is based on the 'Post/Redirect/Get (PRG)' pattern, so it ends by 'http redirect'<br>
         * @param depo  entity to be created
         * @param bindingResult Spring MVC binding result (to retrieve validation errors)
         * @param model Spring MVC model
         * @param redirectAttributes Spring MVC redirect attributes
         * @param httpServletRequest
         * @return
         */
        @RequestMapping(value = "/create" ) // GET or POST
        public String create(@Valid DepoRecord depo, BindingResult bindingResult, Model model, RedirectAttributes redirectAttributes, HttpServletRequest httpServletRequest) {
            log("Action 'create'");
            try {
                if (!bindingResult.hasErrors()) {
                    DepoRecord recordCreated = depoService.create(depo);
                    log("Depo created : " + recordCreated );
```

```
                        model.addAttribute(MAIN_ENTITY_NAME,
recordCreated);


                        //---
                        messageHelper.addMessage(redirectAttributes, new
Message(MessageType.SUCCESS,"save.ok"));
                        return redirectToForm(httpServletRequest,
depo.getUrunID() );
                    } else {
                        log("Action 'create' : binding error(s) " );
                        logBindingErrors(bindingResult);
                        populateModel( model, depo, FormMode.CREATE);
                        return JSP_FORM;
                    }
                } catch(Exception e) {
                    log("Action 'create' : Exception - " + e.getMessage() );
                    messageHelper.addException(model, "depo.error.create", e);
                    populateModel( model, depo, FormMode.CREATE);
                    return JSP_FORM;
                }
            }


            /**
             * 'UPDATE' action processing. <br>
             * This action is based on the 'Post/Redirect/Get (PRG)' pattern, so it ends
by 'http redirect'<br>
```

```
     * @param depo  entity to be updated
     * @param bindingResult Spring MVC binding result (to retrieve validation errors)
     * @param model Spring MVC model
     * @param redirectAttributes Spring MVC redirect attributes
     * @param httpServletRequest
     * @return
     */
    @RequestMapping(value = "/update" ) // GET or POST
    public String update(@Valid DepoRecord depo, BindingResult bindingResult, Model model, RedirectAttributes redirectAttributes, HttpServletRequest httpServletRequest) {
        log("Action 'update'");
        try {
            if (!bindingResult.hasErrors()) {
                //--- Perform database operations
                boolean updated = depoService.update(depo);
                log("Depo updated : result = " + updated );
                model.addAttribute(MAIN_ENTITY_NAME, depo);
                //--- Set the result message
                messageHelper.addMessage(redirectAttributes, new Message(MessageType.SUCCESS,"save.ok"));
                log("Action 'update' : update done - redirect");
                return redirectToForm(httpServletRequest, depo.getUrunID());
            } else {
```

```
                        log("Action 'update' : binding errors");
                        logBindingErrors(bindingResult);
                        populateModel( model, depo, FormMode.UPDATE);
                        return JSP_FORM;
                    }
                } catch(Exception e) {
                    messageHelper.addException(model, "depo.error.update", e);
                    log("Action 'update' : Exception - " + e.getMessage() );
                    populateModel( model, depo, FormMode.UPDATE);
                    return JSP_FORM;
                }
            }

            /**
             * 'DELETE' action processing. <br>
             * This action is based on the 'Post/Redirect/Get (PRG)' pattern, so it ends
by 'http redirect'<br>
             * @param redirectAttributes
             * @param urunID  primary key element
             * @return
             */
            @RequestMapping(value = "/delete/{urunID}") // GET or POST
            public String delete(RedirectAttributes redirectAttributes,
@PathVariable("urunID") int urunID) {
                log("Action 'delete'" );
```

```
                try {
                        boolean deleted = depoService.deleteById( urunID );
                        log("Depo deleted. Key : " + toString(urunID) + " result = " +
deleted );
                        //--- Set the result message
                        messageHelper.addMessage(redirectAttributes, new
Message(MessageType.SUCCESS,"delete.ok"));
                } catch(Exception e) {
                        messageHelper.addException(redirectAttributes,
"depo.error.delete", e);
                }
                return redirectToList();
        }

}
```

pom.xml dosyası:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">

        <!-- File generated by Telosys Tools Generator ( version 3.1.2 ) - Date 2020-
12-20 ( Time 15:28:46 ) -->

        <modelVersion>4.0.0</modelVersion>
```

```
	<groupId>group.to.be.defined</groupId>
	<artifactId>DepoYonetimi</artifactId>
	<version>0.1</version>
	<packaging>jar</packaging>

	<properties>
		<project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
		<java.version>1.8</java.version>
	</properties>

	<dependencies>
		<!-- Spring MVC -->
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-webmvc</artifactId>
			<version>4.2.5.RELEASE</version>
		</dependency>

		<!-- SLF4J implementation required by Spring MVC -->
		<!-- "SLF4J Simple" implementation : sends log messages to the
console -->
		<dependency>
			<groupId>org.slf4j</groupId>
			<artifactId>slf4j-simple</artifactId>
```

```
            <version>1.7.5</version>
        </dependency>

        <!-- Tiles -->
        <dependency>
            <groupId>org.apache.tiles</groupId>
            <artifactId>tiles-core</artifactId>
            <version>3.0.5</version>
        </dependency>
    <dependency>
   <groupId>org.apache.tiles</groupId>
   <artifactId>tiles-jsp</artifactId>
   <version>3.0.5</version>
</dependency>

        <!-- Servlet/JSP -->
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>javax.servlet-api</artifactId>
            <version>3.0.1</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>javax.servlet.jsp</groupId>
            <artifactId>javax.servlet.jsp-api</artifactId>
```

```
            <version>2.2.1</version>
            <scope>provided</scope>
        </dependency>
        <dependency>
            <groupId>jstl</groupId>
            <artifactId>jstl</artifactId>
            <version>1.2</version>
        </dependency>

        <!-- Dependencies for Unit Testing -->
        <dependency>
          <groupId>org.mockito</groupId>
            <artifactId>mockito-all</artifactId>
          <version>1.9.5</version>
          <scope>test</scope>
        </dependency>

        <dependency>
          <groupId>org.springframework</groupId>
          <artifactId>spring-test</artifactId>
          <version>3.2.6.RELEASE</version>
          <scope>test</scope>
        </dependency>
    </dependencies>
```

```
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>2.5.1</version>
                <configuration>
                    <source>${java.version}</source>
                    <target>${java.version}</target>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

B
DEPO YÖNETİM SİSTEMİ İÇİN OTOMATİK ÜRETİLMİŞ BİRİM TEST SENARYOLARI

DepoTest.java sınıfı;

```
/*
 * JUnit test case for bean Depo
 * Created on 2020-12-20 ( Date ISO 2020-12-20 - Time 18:03:53 )
 * Generated by Telosys Tools Generator ( version 3.1.2 )
 */

package org.demo.basic.bean;

import org.junit.Assert;
import org.junit.Test;

/**
 * JUnit test case for bean Depo
 *
 * @author Telosys Tools Generator
 *
 */
public class DepoTest
{
```

```
@Test
public void testSettersAndGetters() {

        System.out.println("Checking class Depo getters and setters ..." );

        Depo depo = new Depo();

/*
        //--- Test setter/getter for field "urunID"  ( type : int )
        // System.out.println(" checking field urunID ..." );
        depo.setUrunID( intValue ) ;
        Assert.assertTrue( intValue == depo.getUrunID() ) ;

        //--- Test setter/getter for field "urunMiktari"  ( type : double )
        // System.out.println(" checking field urunMiktari..." );
        depo.setUrunMiktari ( doubleValue ) ;
        Assert.assertTrue( doubleValue == depo.getUrunMiktari() ) ;

        //--- Test setter/getter for field "urunTipi"  ( type : String )
        // System.out.println(" checking field urunTipi ..." );
        depo.setUrunTipi( stringValue ) ;
        Assert.assertTrue( stringValue.equals( depo.getUrunTipi() ) ) ;

        //--- Test setter/getter for field "urunAdi"  ( type : String )
```

```
// System.out.println(" checking field urunAdi ..." );
depo.setUrunAdi( stringValue ) ;
Assert.assertTrue( stringValue.equals( depo.getUrunAdi() ) ) ;

*/

//--- Test setter/getter for attribute "urunID"  ( type : int )
depo.setUrunID( 100 ) ;
Assert.assertEquals( 100, depo.getUrunID() ) ;



//--- Test setter/getter for attribute "urunMiktari"  ( type : double )
depo.setUrunMiktari ( 1000.66D ) ;
Assert.assertEquals( 1000.66D, depo.getUrunMiktari() ) ;



//--- Test setter/getter for attribute "urunTipi"  ( type : String )
depo.setUrunTipi( "A" ) ;
Assert.assertEquals( "A", depo.getUrunTipi() ) ;



//--- Test setter/getter for attribute "urunAdi"  ( type : String )
depo.setUrunAdi(
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" ) ;
```

```
                Assert.assertEquals(
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA",
depo.getUrunAdi() ) ;



        }
}
```

DepoCacheTest.java sınıfı;

```
/*
 * JUnit test case for Depo caching service
 * Created on 2020-12-20 ( Date ISO 2020-12-20 - Time 18:03:53 )
 * Generated by Telosys Tools Generator ( version 3.1.2 )
 */

package org.demo.basic.cache;

import static org.junit.Assert.assertNull;
import static org.junit.Assert.assertTrue;

import org.demo.basic.bean.Depo ;

import org.junit.Test;
```

```
/**
 * JUnit test case for Depo caching service
 *
 * @author Telosys Tools Generator
 *
 */
public class DepoCacheTest
{
	//protected static final java.util.Date now = new java.util.Date();

	private final static void populate(Depo depo) {
		depo.setUrunID( 100 ) ;
		depo.setUrunMiktari ( 1000.66D ) ;
		depo.setUrunTipi( "A" ) ;
		depo.setUrunAdi(
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA" ) ;
	}

	@Test
	public void testPutGetRemove() {

		System.out.println("Testing class DepoCache ..." );

		Depo depo = new Depo();
		populate(depo);
```

```
                System.out.println("Entity populated : " + depo );

                DepoCache.putDepo(depo) ;          // Store in cache

                Depo depo2 = DepoCache.getDepo( depo.getUrunID() );
                assertTrue( depo == depo2 ) ; // Same instance

                DepoCache.removeDepo(  depo.getUrunID() ) ; // Remove from
cache

                Depo depo3 = DepoCache.getDepo( depo.getUrunID() );
                assertTrue( null == depo3 ) ; // Not in cache
                assertNull(depo3); // Not in cache

        }
}
```

C
DEPO YÖNETİM SİSTEMİ İÇİN OTOMATİK ÜRETİLMİŞ FONKSİYONEL TEST SENARYOLARI

Otomatik Postman için oluşturulmuş test senaryosu **DepoPostmanTest.json** dosyası;


```
{
	"variables": [],
	"info": {
		"name": "REST API tests for entity Depo",
		"description": "PostMan tests generated by Telosys",
		"schema":
"https://schema.getpostman.com/json/collection/v2.0.0/collection.json"
	},
	"item": [
		{
			"name": "FIND ALL",
			"event": [
				{
					"listen": "test",
					"script": {
						"type": "text/javascript",
						"exec": [
```

```
                                                "// Test if response code is always
200 (a list is expected, void list if nothing) ",
                                                "tests[\"Status code is always
200\"] = ( responseCode.code === 200 ) ;",
                                                "",


                                                "if ( responseCode.code === 200 )
{",
                                                "// 200 OK (found) => Body
expected in the response",
                                                        "tests[\"Has Content-
Type\"] = responseHeaders.hasOwnProperty(\"Content-Type\");",
                                                        "tests[\"Content-Type is
application/json\"] = responseHeaders[\"Content-
Type\"].has(\"application/json\");",
                                                "tests[\"Has body\"] =
responseBody ;",
                                                        "tests[\"Body is valid JSON
\"] = JSON.parse(responseBody) ;",
                                                "}",
                                                ""
                                        ]
                                }
                        }
                ],
                "request": {
                        "url": "http://localhost:3000/api/v1/depo",
```

```
                "method": "GET",
                "header": [],
                "body": {
                    "mode": "raw",
                    "raw": ""
                },
                "description": "FIND ALL with GET method"
            },
            "response": []
        },

        {
            "name": "FIND ONE",
            "event": [
                {
                    "listen": "test",
                    "script": {
                        "type": "text/javascript",
                        "exec": [

                            "// Test if response code is 200 or
404",
                            "tests[\"Status code is 200 or
404\"] = ( responseCode.code === 200 || responseCode.code === 404 ) ;",
                            "",
```

```
                                                    "if ( responseCode.code === 200 )
{",
                                                    "// 200 OK (found) => Body
expected in the response",
                                                        "tests[\"200 => Has
Content-Type\"] = responseHeaders.hasOwnProperty(\"Content-Type\");",
                                                        "tests[\"200 => Content-
Type is application/json\"] = responseHeaders[\"Content-
Type\"].has(\"application/json\");",
                                                    "tests[\"200 => Has body\"] =
responseBody ;",
                                                        "tests[\"200 => Body is
valid JSON \"] = JSON.parse(responseBody) ;",
                                                    "}",
                                                    "",


                                                    "if ( responseCode.code === 404 )
{",
                                                    "// 404 NOT FOUND => No body
expected in the response",
                                                        "tests[\"404 => Content-
Length is ZERO\"] = responseHeaders[\"Content-Length\"].has(\"0\");" ,
                                                    "tests[\"404 => No body\"] = ( !
responseBody ) ;",
                                                    "}",
                                                    "",
```

```
                                        ""
                                    ]
                                }
                            }
                        ],
                        "request": {
                            "url": "http://localhost:3000/api/v1/depo/100",
                            "method": "GET",
                            "header": [],
                            "body": {},
                            "description": "FIND ONE with GET method"
                        },
                        "response": []
                    },

                    {
                        "name": "CREATE",
                        "event": [
                            {
                                "listen": "test",
                                "script": {
                                    "type": "text/javascript",
                                    "exec": [
```

```
                                                    "// Test if response code is 201 or
409",
                                                    "tests[\"Status code is 201 or
409\"] = ( responseCode.code === 201 || responseCode.code === 409 ) ;",
                                                    "",


                                                    "if ( responseCode.code === 201 )
{",
                                                    "// 201 CREATED => Body
expected in the response",
                                                        "tests[\"201 => Has
Content-Type\"] = responseHeaders.hasOwnProperty(\"Content-Type\");",
                                                        "tests[\"201 => Content-
Type is application/json\"] = responseHeaders[\"Content-
Type\"].has(\"application/json\");",
                                                    "tests[\"201 => Has body\"] =
responseBody ;",
                                                        "tests[\"201 => Body is
valid JSON \"] = JSON.parse(responseBody) ;",
                                                    "}",
                                                    "",
                                                    "if ( responseCode.code === 409 )
{",
                                                        "// 409 CONFLICT => No
body expected in the response",
                                                        "tests[\"409 => Content-
Length is ZERO\"] = responseHeaders[\"Content-Length\"].has(\"0\");" ,
```

```
                                                    "tests[\"409 => No body in response\"] = ( ! responseBody ) ;",
                                                    "}",
                                                    ""
                                                ]
                                            }
                                        }
                                    ],
                                    "request": {
                                        "url": "http://localhost:3000/api/v1/depo",
                                        "method": "POST",
                                        "header": [
                                            {
                                                "key": "Content-Type",
                                                "value": "application/json",
                                                "description": ""
                                            }
                                        ],
                                        "body": {
                                            "mode": "raw",
                                            "raw": "{\n\t\"urunID\":100, \n\t\"urunMiktari\":1000.66, \n\t\"urunTipi\":\"A\", \n\t\"urunAdi\":\"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\"\n}"
                                        },
                                        "description": "CREATE with POST method"
```

```
            },
            "response": []
        },

        {
            "name": "UPDATE ",
            "event": [
                {
                    "listen": "test",
                    "script": {
                        "type": "text/javascript",
                        "exec": [
                            "// Test if response code is 200 or
404",
                            "tests[\"Status code is 200 or
404\"] = ( responseCode.code === 200 || responseCode.code === 404 ) ;",
                            "",

                            "// No body in the response (in
any case) ",
                            "tests[\"No body in response\"] =
( ! responseBody ) ;",
                            "",

                            ""
                        ]
```

```
                }
            }
        ],
        "request": {
            "url": "http://localhost:3000/api/v1/depo/100",
            "method": "PUT",
            "header": [
                {
                    "key": "Content-Type",
                    "value": "application/json",
                    "description": ""
                }
            ],
            "body": {
                "mode": "raw",
                "raw": "{\n\t\"urunID\":100,
\n\t\"urunMiktari\":1000.66, \n\t\"urunTipi\":\"A\",
\n\t\"urunAdi\":\"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\"\n}"
            },
            "description": "UPDATE with PUT method"
        },
        "response": []
    },

    {
```

```
"name": "SAVE",
"event": [
{
"listen": "test",
"script": {
"type": "text/javascript",
"exec": [
"// Test if response code is 200 or 201",
"tests[\"Status code is 200 or 201\"] = ( responseCode.code === 200 || responseCode.code === 201 ) ;",
"",
"// 200 or 201 : Body always expected in the response",
"tests[\"Has Content-Type\"] = responseHeaders.hasOwnProperty(\"Content-Type\");",
"tests[\"Content-Type is application/json\"] = responseHeaders[\"Content-Type\"].has(\"application/json\");",
"tests[\"Has body\"] = responseBody ;",
"tests[\"Body is valid JSON \"] = JSON.parse(responseBody) ;",
"",
""
```

```
                                ]
                            }
                        }
                    ],
                    "request": {
                        "url": "http://localhost:3000/api/v1/depo",
                        "method": "PUT",
                        "header": [
                            {
                                "key": "Content-Type",
                                "value": "application/json",
                                "description": ""
                            }
                        ],
                        "body": {
                            "mode": "raw",
                            "raw": "{\n\t\"urunID\":100,
\n\t\"urunMiktari\":1000.66, \n\t\"urunTipi\":\"A\",
\n\t\"urunAdi\":\"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\"\n}"
                        },
                        "description": "SAVE with PUT method"
                    },
                    "response": []
                },
```

```
{
    "name": "DELETE",
    "event": [
        {
            "listen": "test",
            "script": {
                "type": "text/javascript",
                "exec": [
                    "// Test if response code is 204 or 404",
                    "tests[\"Status code is 204 or 404\"] = ( responseCode.code === 204 || responseCode.code === 404 ) ;",
                    "",

                    "// No body in the response (in any case) ",
                    "tests[\"No body in response\"] = ( ! responseBody ) ;",
                    "",

                    ""
                ]
            }
        }
    ],
    "request": {
```

```
                "url": "http://localhost:3000/api/v1/depo/100",
                "method": "DELETE",
                "header": [],
                "body": {
                    "mode": "raw",
                    "raw": ""
                },
                "description": "DELETE with DELETE method"
            },
            "response": []
        }


    ]
}
```

D
DEPO YÖNETİM SİSTEM İÇİN KURULUM DOSYASI

Uygulama imaj'ını oluşturmak için kullanılan docker-compose.yaml ve dağıtım için kullanılan service.yaml, deployment.yaml dosyaları:

-docker-compose.yaml;

```
FROM openjdk:8-jdk-alpine

WORKDIR /

ADD DepoYonetimi*.jar app.jar


ADD conf conf

ADD lib lib

RUN mkdir -p logs



RUN chmod -R 777 lib

RUN chmod -R 777 logs


RUN apk add --no-cache --update busybox-extras


RUN set -x \
   && apk add --no-cache --update tzdata \
   && cp /usr/share/zoneinfo/Europe/Istanbul /etc/localtime \
   && echo "Europe/Istanbul" >  /etc/timezone \
```

```
    && apk del tzdata \
    && rm -rf /var/cache/apk/*

CMD java -jar app.jar
```

-service.yaml;

```
apiVersion: v1
kind: Service
metadata:
  name: depo
spec:
  type: NodePort
  ports:
  - port: 3000
    nodePort: 3000
    name: http
  selector:
    name: depo
```

- **deployment.yaml;**

```
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: depo-master
```

```
  labels:
    app: depo
spec:
  selector:
    matchLabels:
      app: depo
      role: master
      tier: backend
  replicas: 1
  template:
    metadata:
      labels:
        app: depo
        role: master
        tier: backend
    spec:
      containers:
      - name: master
        image: busraicoz/app:latest
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        ports:
        - containerPort: 3000
```

TEZDEN ÜRETİLMİŞ YAYINLAR

Makaleler

1. İçöz, B , Kalıpsız, O . (2020). MODEL GÜDÜMLÜ YAZILIM GELİŞTİRME YAKLAŞIMI KULLANILARAK MİKRO SERVİS GELİŞTİRİLMESİ . Mühendislik Bilimleri ve Tasarım Dergisi , Özel Sayı: Uluslararası Mühendislikte Yapay Zeka ve Uygulamalı Matematik Konferansı (UMYMK 2020) , 142-148 . DOI: 10.21923/jesd.826355