

Giriş

Dizayn

Mimari

Uygulama

# Metric Visualization

**Eray Gökçe 20011079**  
Geliştirici

**DR. ÖĞR. ÜYESİ YUNUS EMRE SELÇUK**

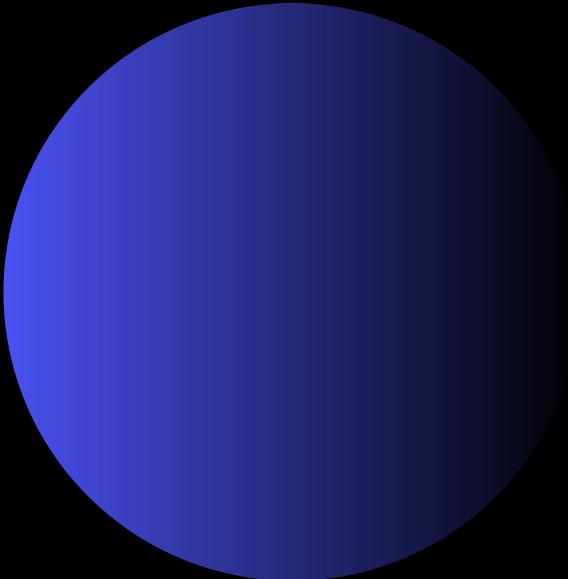
# Yazılım Metrikleri

Yazılım metrikleri, bir yazılımın ne kadar kaliteli olduğunu sayılarla anlatan ölçülerdir. Kodun karmaşıklığını, okunabilirliğini ya da ne kadar iyi tasarlandığını ölçmek için kullanılır.

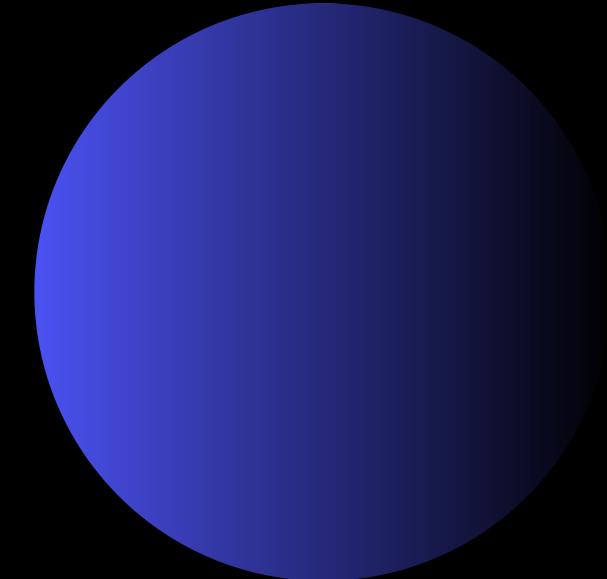
Bu projenin temel amacı, nesne yönelimli yazılım sistemlerinde yapısal kaliteyi değerlendirmek için kullanılan yazılım metriklerini analiz etmek ve bu metrikleri üç boyutlu bir ortamda görselleştirmektir.

Analiz sonuçları **kod kalitesini** artacaktır.

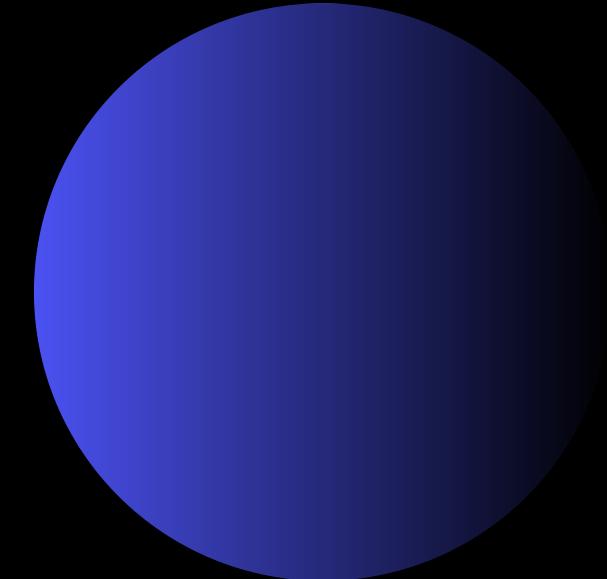
# Projenin Amacı



Sürdürülebilirlik



Ölçeklenebilirlik

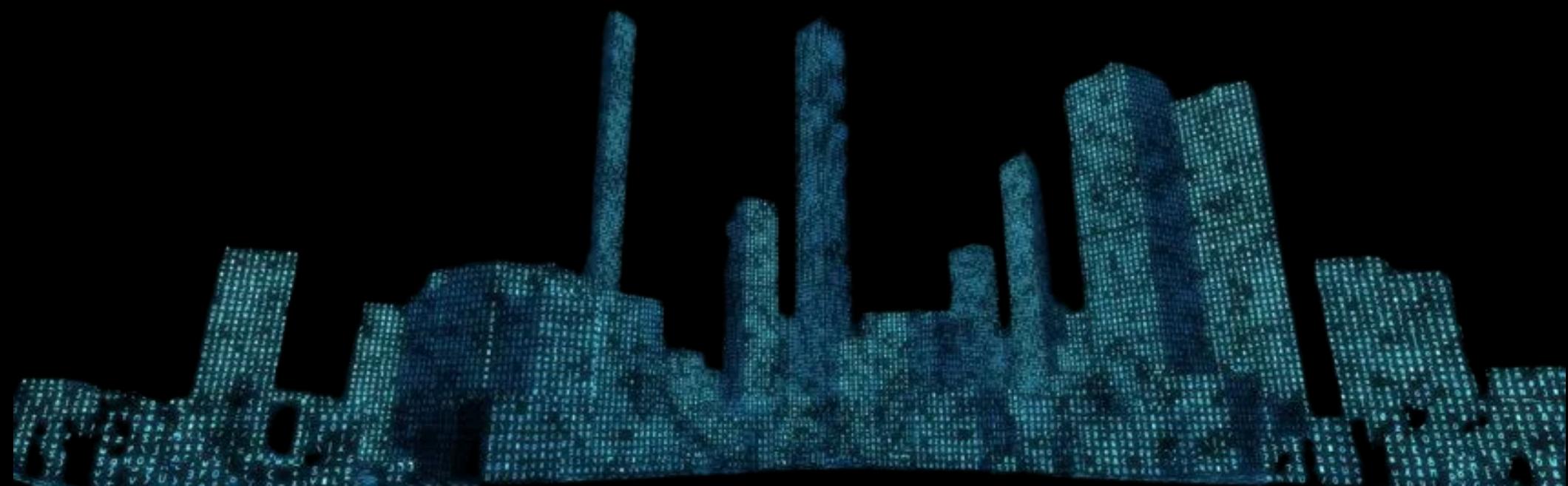


Anlaşılabilirlik

# Benzer Çalışmalar

Proje, halihazırda geliştirilmiş olan bir Java uygulamasından esinlenilmiştir. Mevcut sistem, CK metrikleri ile yazılım kalite ölçümleri gerçekleştirmeye ve bu sonuçları kullanıcıya sayısal biçimde sunmaktadır.

Projemiz, bu eksikliği gidermek amacıyla geliştirilmiş olup, görselleştirme modülü ile öne çıkmaktadır. Literatürde bu konuda öne çıkan çalışmalarlardan biri olan **CodeCity** yaklaşımı incelenmiş; yazılım bileşenlerinin sanal bir şehir görünümünde, yapısal metriklere göre temsil edilmesi fikri projemize entegre edilmiştir.



# Akış Planlanması

1

2

3

4

5

Gereksinim Analizi

Yeni Metrikler  
Hesaplanması

Eski Uygulama ile  
Entegre Edilmesi

3D Görselleştirme

Metrikler ile 3D  
Görselleştirme  
Entegre Edilmesi

# Use Case

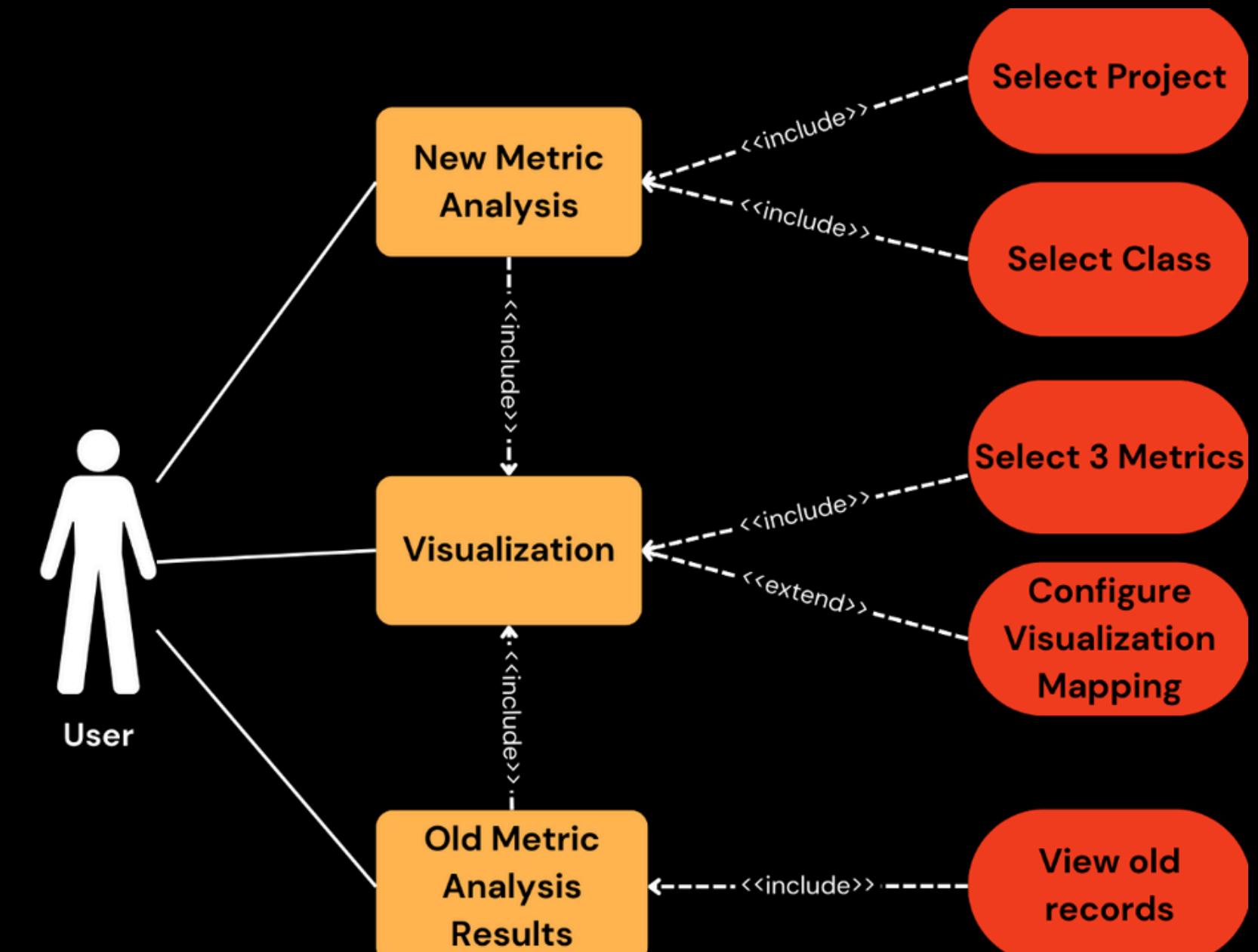
Kullanıcının, kullanacağı üç temel işlev vardır.  
Görselleştirme işlevi, Yeni Metrik Hesaplama veya  
Eski Sonuçları Görüntüle kapsamında  
gerçekleşmektedir.

Giriş

Dizayn

Mimari

Uygulama



# Kullanım Senaryoları

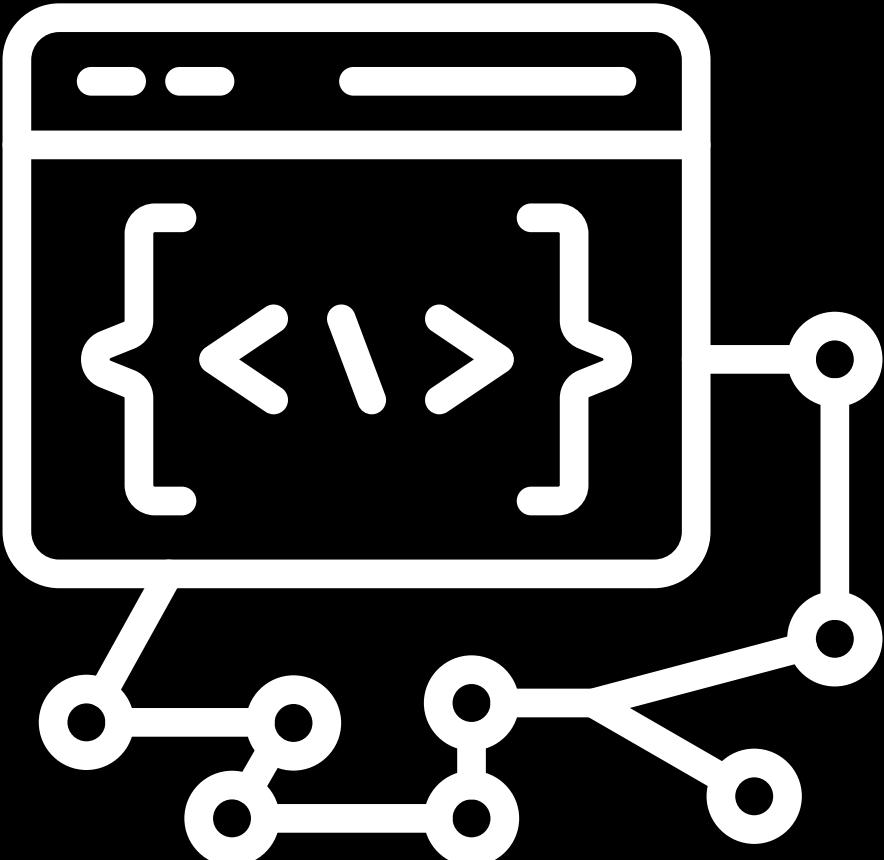
Sistem, kullanıcıya hem yeni analiz başlatma hem de daha önce yapılmış analizleri tekrar görüntüleme imkânı sunar. Kullanıcı yeni bir analiz başlatmak istediğiinde, analiz etmek istediği proje dosyasını seçer ve hangi metriklerin görselleştirileceğini belirler.

Bu metrikler, Unity ortamında üç boyutlu yapılarla temsil edilir.

Alternatif olarak sadece tek bir sınıf dosyası da analiz edilebilir. Öte yandan, sistemin öne çıkan özelliklerinden biri de geçmiş analiz kayıtlarına erişim sağladır. Kullanıcı, daha önce hesaplanmış bir projeyi yeniden seçip görselleştirme yapabilir.

# Geçmiş Kayıtları Görüntüleme

Uygulama, yapılan tüm analizleri tarihsel olarak saklar ve yeniden görselleştirilebilir şekilde sunar. Típkí Git versiyon kontrol sisteminde olduğu gibi, proje üzerindeki yapısal değişiklikler zamansal olarak takip edilebilir. Bu sayede geliştiriciler, yazılım mimarisindeki evrimi sezgisel olarak inceleyebilir ve önceki sürümlerle karşılaştırmalı analiz yapabilir.



# Görselleştirme

Projenin en ayırt edici yönlerinden biri olan görselleştirme modülü, yazılım metriklerini üç boyutlu bir şehir metaforu üzerinden sunar. Unity ile geliştirilen bu modül sayesinde her sınıf bir bina olarak temsil edilir; metrikler ise bu binaların yüksekliği, genişliği ve rengi gibi görsel özelliklere karşılık gelir. Kullanıcı, analiz sonucunda ortaya çıkan verileri sezgisel bir şekilde değerlendirebilir, karmaşık yapılar arasında ilişkileri kolayca görebilir ve olası tasarım problemlerini erkenden fark edebilir. Bu sayede soyut metrik değerler, somut ve anlaşılır bir yapıya dönüştürülür.

Giriş

Dizayn

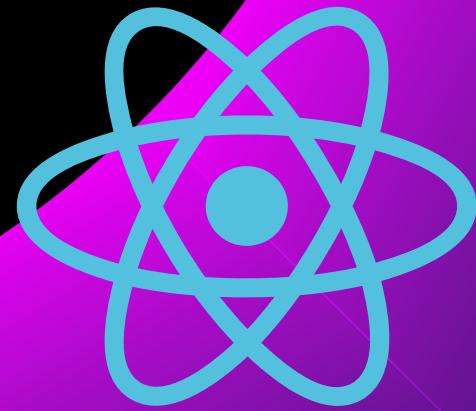
Mimari

Uygulama



`Mathf.InverseLerp(range.minVal, range.maxVal, val);`

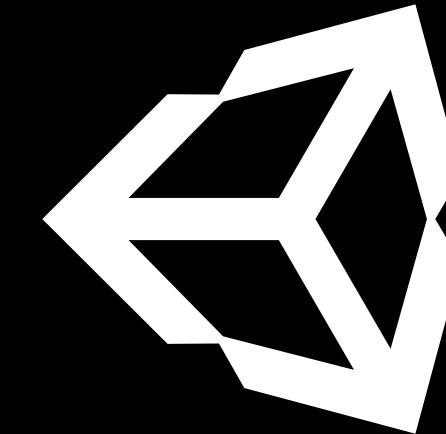
# Katmanlı Servis Odaklı Mimari



REST API



HTTP  
PUSH



Giriş

Dizayn

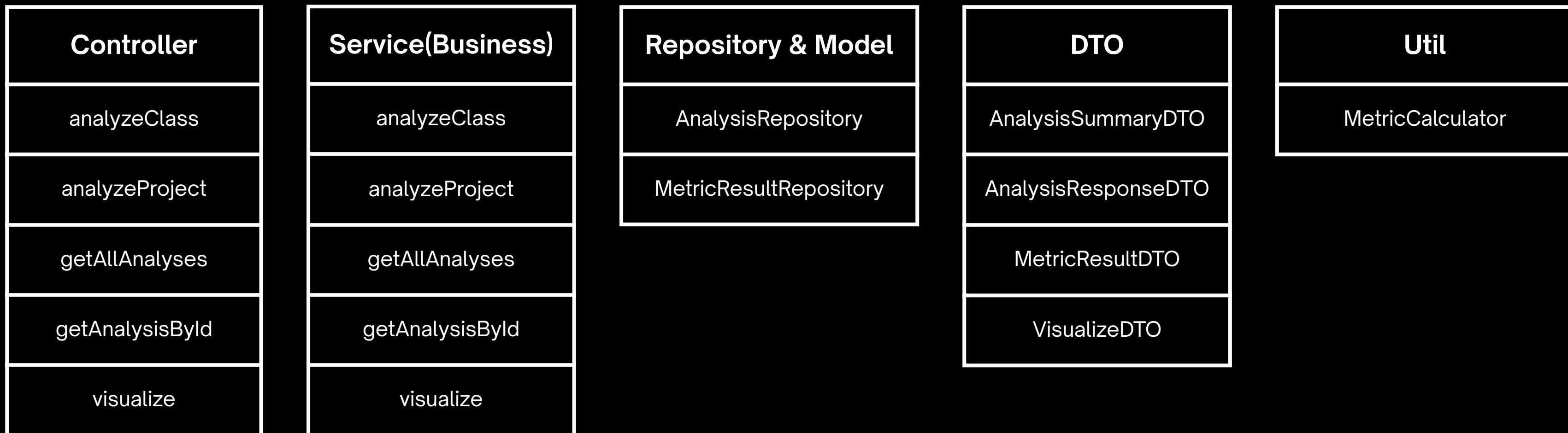
Mimari

Uygulama

Projemiz, görevleri net bir şekilde ayrılmış üç ana bileşenden oluşan modern bir mimariye sahiptir:

Kullanıcı arayüzü (React), hesaplama ve veri yönetimi (Spring Boot) ve 3D görselleştirme (Unity).

Bu "ayrık mimari" (decoupled architecture) felsefesi, her bileşenin en iyi olduğu işe odaklanmasını sağlar: React'in modern web arayüzü yetenekleri, Spring'in güçlü ve ölçeklenebilir backend altyapısı ve Unity'nin gerçek zamanlı 3D render motoru.



### Project

- └── controller // API endpoint'leri
- └── service // İş mantığı
- └── repository // Veritabanı erişimi
- └── model // JPA Entity'leri
- └── dto // Veri transfer nesneleri
- └── util // Yardımcı sınıflar



## Giriş

```
{  
    "id": "fe903677-2048-427c-8303-  
    bf6323a9fe69",  
    "type": "CLASS",  
    "targetPath": "C:\\\\Users\\\\erayg\\\\eclipse-  
    workspace",  
    "projectName": null,  
    "createdAt": "2025-06-21T01:06:38.683474"  
,  
{  
    "id": "83a5610c-8a2f-49f1-aa57-  
    baedda66d312",  
    "type": "JAR",  
    "targetPath": "C:\\\\Users\\\\erayg\\\\eclipse-  
    workspace",  
    "projectName": "CarRentalApp.jar",  
    "createdAt": "2025-06-22T00:32:01.603456"  
,
```

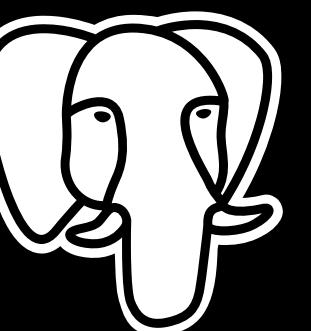
## Dizayn

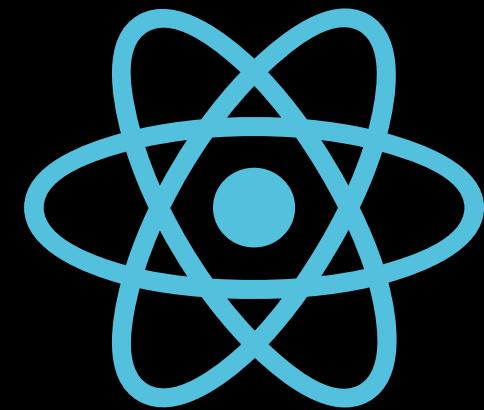
analysis	metric_result
id	id
class/project_name	analysis_id (FK)
type	class_name
target_path	metricList
created_at	

## Mimari

## Uygulama

```
"id": "83a5610c-8a2f-49f1-aa57-baedda66d312",  
    "type": "JAR",  
    "targetPath": "C:\\\\Users\\\\erayg\\\\eclipse-  
    workspace\\\\LastTestProject\\\\CarRentalApp.jar",  
    "projectName": "CarRentalApp.jar",  
    "createdAt": "2025-06-22T00:32:01.603456",  
    "results": [  
        {  
            "id": "4182c667-39c2-476e-9e98-  
            5f090ee78779",  
            "className": "com.carrental.Booking",  
            "tcc": 0.0,  
            "wmc": 2,  
            "lcom": 0.0,  
            "dit": 0,  
            ".cbo": 3,  
            "maxCyclo": 1,  
            "avgCyclo": 1.0  
        },  
        {  
            "id": "44e656fc-c9e2-41a8-8c9d-  
            5f8de818558f",  
            "className": "com.carrental.Car",  
            "tcc": 0.6,  
            "wmc": 12,  
            "lcom": 0.0,  
            "dit": 1,  
            ".cbo": 6,  
            "maxCyclo": 4,  
            "avgCyclo": 2.0  
        },  
        {
```





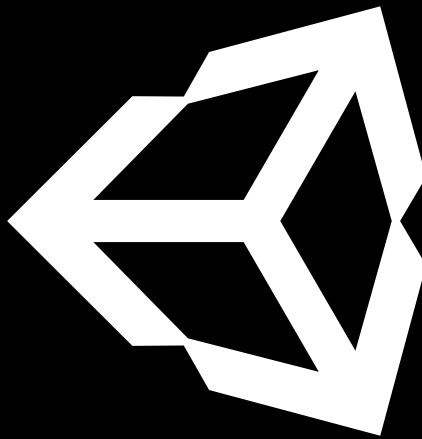
## REACT



## ELECTRON

Kullanıcının tüm etkileşimlerini yöneten modern ve dinamik bir web arayüzü sunar. Proje/sınıf seçimi, analiz başlatma, geçmiş sonuçları listeleme ve görselleştirilecek metrikleri seçme gibi işlemler burada yapılır.

Kullanıcının yerel dosya sisteminden (C:\... gibi) klasör veya dosya seçebilmesi, React web uygulamasının bir Electron kabuğu içinde çalıştırılmasıyla mümkün kılınmıştır. Bu, web teknolojileri ile masaüstü yeteneklerini birleştirir.



## UNITY

Backend'den gelen metrik verilerini alıp, bu verileri anlamlı bir 3D "kod şehri" metaforuna dönüştürür. Her bir sınıf, metrik değerlerine göre boyutları ve rengi değişen bir bina olarak temsil edilir.

Spring'ten gelen istekleri sürekli dinler. Metrik görselleştirmede özellik seçimi yapılır. Dinamik olarak class sayısına ve genişliklerine göre map genişler. Metrikler 3D olarak görselleştirilir. Bina detaylarında, metriklere sayısal olarak ulaşılabilir.

# TCC

TCC (Sıkı Sınıf Bağlılığı) metriği, bir sınıfın ne kadar bütünsel ve odaklı olduğunu ölçer. Temel prensibi, iyi tasarlanmış bir sınıfta metodların, o sınıfa ait ortak özelliklerini (alanları/değişkenleri) yoğun bir şekilde birlikte kullanması gereklidir.

Hesaplama kodumuza göre, ilk olarak Car sınıfındaki **abstract**, **native** ve **constructor** metodları haricindeki tüm metodlar analize dahil edilir. Bu metodlar şunlardır: start(), needsRefuel(), refuel(), getStatus(), logMaintenance().  
Toplamda **5 adet metot** analize alınır.

Toplamda (5,2) = **10 adet metot çifti** vardır.

start,needsrefuel,refuel,getstatus metodları ortak özelliklere erişir. (4,2) = **6 adet method çifti** vardır.

$$\text{TCC} \rightarrow 6 / 10 = 0.6$$

## Car

Kalıtım : Vehicle
Özellikleri : engine(engine.class)
numberOfDoors(int)
fuelLevel(double)
maintenanceLog(List<String>)
start() : fuelLevel, engine erişir.
1 adet if koşulu içerir
Motoru çalıştırır.
needsRefuel() : fuelLevel, engine erişir.
1 adet if, 1 adet && içerir.
Yakıt seviyesini kontrol eder.
refuel() : fuelLevel erişir.
Karmaşıklık içermez.
Yakıt seviyesini arttırır
getStatus() : fuelLevel, engine erişir.
2 adet if ve 1 adet switch-case
Status bildirir.
logMaintenance() : maintenanceLog erişir.
Karar noktası yoktur.
Bakım kaydı ekler.

# WMC

Hesaplama, sınıfındaki her bir metodun Cyclomatic Complexity (CYCLO - Döngüsel Karmaşıklık) değerini bulur ve ardından bu değerleri toplar. Yüksek bir WMC değeri, sınıfın çok fazla veya çok karmaşık metodlara sahip olduğunu, bu nedenle daha fazla hata potansiyeli taşıdığını ve bakımının maliyetli olabileceğini gösterir.

Hesaplama kodumuz, Car sınıfındaki constructor dahil olmak üzere abstract ve native olmayan tüm metodları gezer. Her bir metodun CYCLO değerini ayrı ayrı hesaplar. (**CYCLO, her metot için 1'den başlar ve her if, &&, for, switch ,try-catch gibi karar noktası için 1 artar.**)

Constructor : 1

start:  $1 + 1 = 2$

needsRefuel :  $1+1+1 = 3$

refuel : 1

getStatus :  $1 + 2(\text{if}) + 2(\text{case}) = 5$

logMaintenance : 1

**WMC = 13**

## Car

Kalıtım : Vehicle

Özellikleri : engine(engine.class)  
                  numberOfDoors(int)  
                  fuelLevel(double)  
                  maintenanceLog(List<String>)

start() : fuelLevel, engine erişir.  
1 adet if koşulu içerir  
Motoru çalıştırır.

needsRefuel() : fuelLevel, engine erişir.  
1 adet if, 1 adet && içerir.  
Yakıt seviyesini kontrol eder.

refuel() : fuelLevel erişir.  
Karmaşıklık içermez.  
Yakıt seviyesini arttırır

getStatus() : fuelLevel, engine erişir.  
2 adet if ve 1 adet switch-case  
Status bildirir.

logMaintenance() : maintenanceLog erişir.  
Karar noktası yoktur.  
Bakım kaydı ekler.

# MAXCYCLO , AVGCYCLO

MAX\_CYCLO ve AVG\_CYCLO metrikleri, bir sınıfın genel karmaşıklığını (WMC) farklı bir mercekten inceleyerek bize daha derin bir analiz sunar. WMC, sınıftaki tüm metodların karmaşıklıklarının toplamını vererek genel bir yük skoru sunarken, bu iki metrik karmaşıklığın sınıf içindeki dağılımı hakkında daha detaylı bilgi sağlar:

MAX\_CYCLO (Maksimum Karmaşıklık): Bu metrik, sınıf içerisindeki en karmaşık tek bir metodun Cyclomatic Complexity (CYCLO) değerini gösterir.

AVG\_CYCLO (Ortalama Karmaşıklık): Bu metrik, sınıftaki metodların ortalama karmaşıklık seviyesini ifade eder.

MAX\_CYCLO : 5

AVG\_CYCLO :  $13/6 = 2.16$

## Car

Kalıtım : Vehicle

Özellikleri : engine(engine.class)  
                  numberOfDoors(int)  
                  fuelLevel(double)  
                  maintenanceLog(List<String>)

start() : fuelLevel, engine erişir.  
1 adet if koşulu içerir  
Motoru çalıştırır.

needsRefuel() : fuelLevel, engine erişir.  
1 adet if, 1 adet && içerir.  
Yakıt seviyesini kontrol eder.

refuel() : fuelLevel erişir.  
Karmaşıklık içermez.  
Yakıt seviyesini arttırır

getStatus() : fuelLevel, engine erişir.  
2 adet if ve 1 adet switch-case  
Status bildirir.

logMaintenance() : maintenanceLog erişir.  
Karar noktası yoktur.  
Bakım kaydı ekler.

**DIT**

DIT (Miras Ağacı Derinliği) metriği, bir sınıfın nesne yönelimli programlamadaki kalıtım (miras) hiyerarşisindeki derinliğini, yani "atalarının" sayısını ölçer.

Değer, incelenen sınıfından, tüm Java sınıflarının en tepesindeki kök sınıf olan `java.lang.Object`'e olan uzaklığını ifade eder.

Hesaplama kodumuz, verilen sınıfından (Car) başlayarak, üst sınıf Object olana kadar yukarı doğru kaç adım çıktığını sayar.

`Car → Vehicle → Object`

**DIT : 1****Car**

Kalıtım : Vehicle

Özellikleri : `engine(engine.class)`  
`numberOfDoors(int)`  
`fuelLevel(double)`  
`maintenanceLog(List<String>)`

`start()` : fuelLevel, engine erişir.  
 1 adet if koşulu içerir  
 Motoru çalıştırır.

`needsRefuel()` : fuelLevel, engine erişir.  
 1 adet if, 1 adet && içerir.  
 Yakıt seviyesini kontrol eder.

`refuel()` : fuelLevel erişir.  
 Karmaşıklık içermez.  
 Yakıt seviyesini arttırır

`getStatus()` : fuelLevel, engine erişir.  
 2 adet if ve 1 adet switch-case  
 Status bildirir.

`logMaintenance()` : maintenanceLog erişir.  
 Karar noktası yoktur.  
 Bakım kaydı ekler.

# LCOM

LCOM (Metotlarda Bağlılık Eksikliği), bir sınıfın ne kadar "dağınık" olduğunu, yani metodlarının ne kadarının birbirinden bağımsız çalıştığını ölçer. Yüksek bir

LCOM değeri, sınıfın birbiriyle ilgisiz birden fazla sorumluluğu olduğuna ve muhtemelen "Tek Sorumluluk Prensibi"ni (Single Responsibility Principle) ihlal ettiğine işaret eder.

Hesaplama, TCC'ye benzer şekilde, sınıfındaki metot çiftlerini analiz eder.

**P: Ortak bir özelliğe erişen, yani "birbiriyle konuşan" metot çiftlerinin sayısıdır.**

**Q: Hiçbir ortak özelliğe erişmeyen, yani "birbirinden habersiz" metot çiftlerinin sayısıdır.**

Eğer dağınık çiftlerin sayısı (Q), bağlı çiftlerin sayısından (P) fazlaysa, LCOM bu ikisinin farkıdır. Aksi takdirde, sınıf yeterince bütünsel kabul edilir ve LCOM değeri 0 olur.

**P: 6**

Q : logMaintenance methodu, diğer 4 methodla hiçbir ortak özelliği yoktur. **4 adet metot çifti** oluşur.

**LCOM -> (Q > P) ? (Q - P) : 0**

**LCOM = 0**

## Car

Kalıtım : Vehicle

Özellikleri : engine(engine.class)

    numberOfDoors(int)

    fuelLevel(double)

    maintenanceLog(List<String>)

start() : fuelLevel, engine erişir.

    1 adet if koşulu içerir

    Motoru çalıştırır.

needsRefuel() : fuelLevel, engine erişir.

    1 adet if, 1 adet && içerir.

    Yakıt seviyesini kontrol eder.

refuel() : fuelLevel erişir.

    Karmaşıklık içermez.

    Yakıt seviyesini arttırır

getStatus() : fuelLevel, engine erişir.

    2 adet if ve 1 adet switch-case

    Status bildirir.

logMaintenance() : maintenanceLog erişir.

    Karar noktası yoktur.

    Bakım kaydı ekler.

# CBO

CBO (Nesneler Arası Bağlaşım), bir sınıfın ne kadar "bağımlı" olduğunu ölçer. Bir sınıfın, proje içerisindeki diğer kaç farklı sınıfa bağlı olduğunu sayar. Bir sınıfın başka bir sınıfa olan bağımlılığı şu yollarla oluşabilir:

- O sınıfından kalıtım alması (inheritance).
- O sınıfı bir özelliğinin (field) tipi olarak kullanması.
- O sınıfı bir metodunun parametresinde veya dönüş tipinde kullanması.
- Metotlarının içinde o sınıfa ait bir nesne oluşturulması veya metotlarını çağrıması.

Kalıtım : 1 (Vehicle.class)  
Özellik: 1 (Engine.class)

# CBO = 2

## Car

Kalıtım : Vehicle  
Özellikleri : engine(engine.class)  
                  numberOfDoors(int)  
                  fuelLevel(double)  
                  maintenanceLog(List<String>)  
  
start() : fuelLevel, engine erişir.  
              1 adet if koşulu içerir  
              Motoru çalıştırır.  
  
needsRefuel() : fuelLevel, engine erişir.  
              1 adet if, 1 adet && içerir.  
              Yakıt seviyesini kontrol eder.

refuel() : fuelLevel erişir.  
              Karmaşıklık içermez.  
              Yakıt seviyesini arttırır

getStatus() : fuelLevel, engine erişir.  
              2 adet if ve 1 adet switch-case  
              Status bildirir.

logMaintenance() : maintenanceLog erişir.  
              Karar noktası yoktur.  
              Bakım kaydı ekler.

Yeni analiz başlatma ekranına gider.

**START NEW ANALYSIS**

**LIST PAST RECORDS**

Eski analiz kayıtlarını görüntülemeye gider.

Jar dosyası üzerinden analiz başlatma özelliğidir.

Tekli class dosyası üzerinden analiz başlatma özelliğidir.

### Select Analysis Type

JAR

PROJECT

CLASS

BACK

Çoklu class dosyası üzerinden analiz başlatma özelliğidir.

Giriş

Dizayn

Mimari

Uygulama

Bu ekran, geçmiş analizler hakkında özet bilgi içerir.  
Kullanıcı istediği analizi detaylı olarak görüntüleyebilir.

#### 过去的分析记录

Type	Class / Project Name	Creation Date
PROJECT	LastTestProject	19.06.2025 18:05:33
PROJECT	bin	19.06.2025 19:05:19
CLASS	hobipazari\HobbyShopGUI\$2.class	19.06.2025 19:07:01
CLASS	test\MetricTest.class	21.06.2025 01:06:38
JAR	CarRentalApp.jar	22.06.2025 00:32:01
JAR	CarRentalApp.jar	22.06.2025 00:51:41
JAR	CarRentalApp.jar	22.06.2025 01:35:55

GERİ

SELEKTED ANALYSIS

Type, Class/ProjectName  
ve CreationDate gibi  
birimleri içerir.

← Project Analysis: CarRentalApp.jar  
Creation: 22.06.2025 00:32:01

Class Name	TCC	WMC	LCOM	DIT	CBO	MAXCYCLO	AVGCYCLO
com.currental.Booking	0	2	0	0	3	1	1
com.currental.Car	0,6	12	0	1	6	4	2
com.currental.ElectricCar	1	7	0	2	4	3	1,75
com.currental.Engine	1	6	0	0	3	2	1,5

Select 3 metrics to visualize:

TCC  WMC  LCOM  DIT  CBO  MAXCYCLO  AVGCYCLO

VISUALIZE RESULTS

  
Tüm detay bilgileri gösteren ekranıdır.

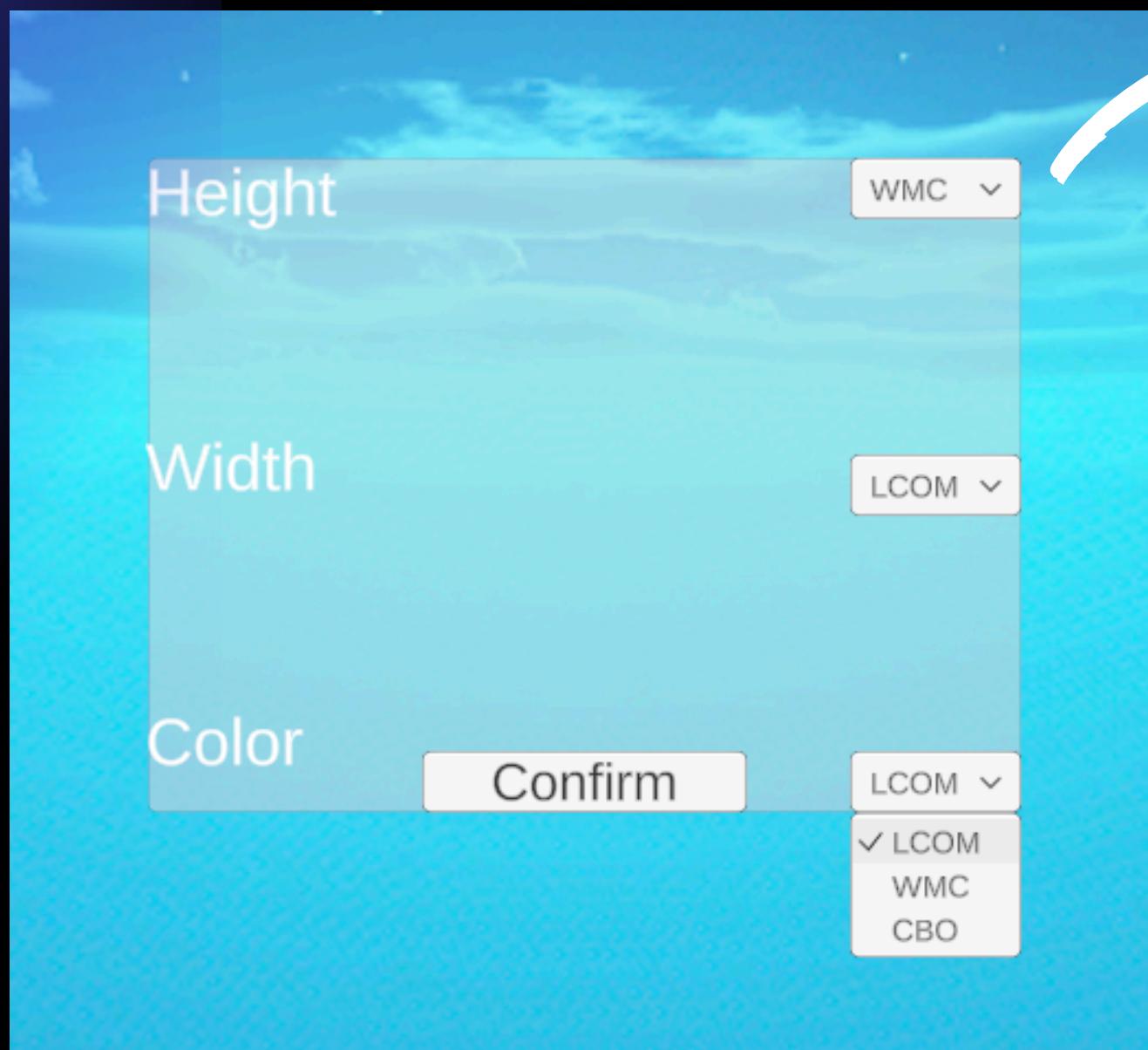
  
İstediği 3 metriği seçerek Unity'e veriyi gönderir.

Giriş

Dizayn

Mimari

**Uygulama**



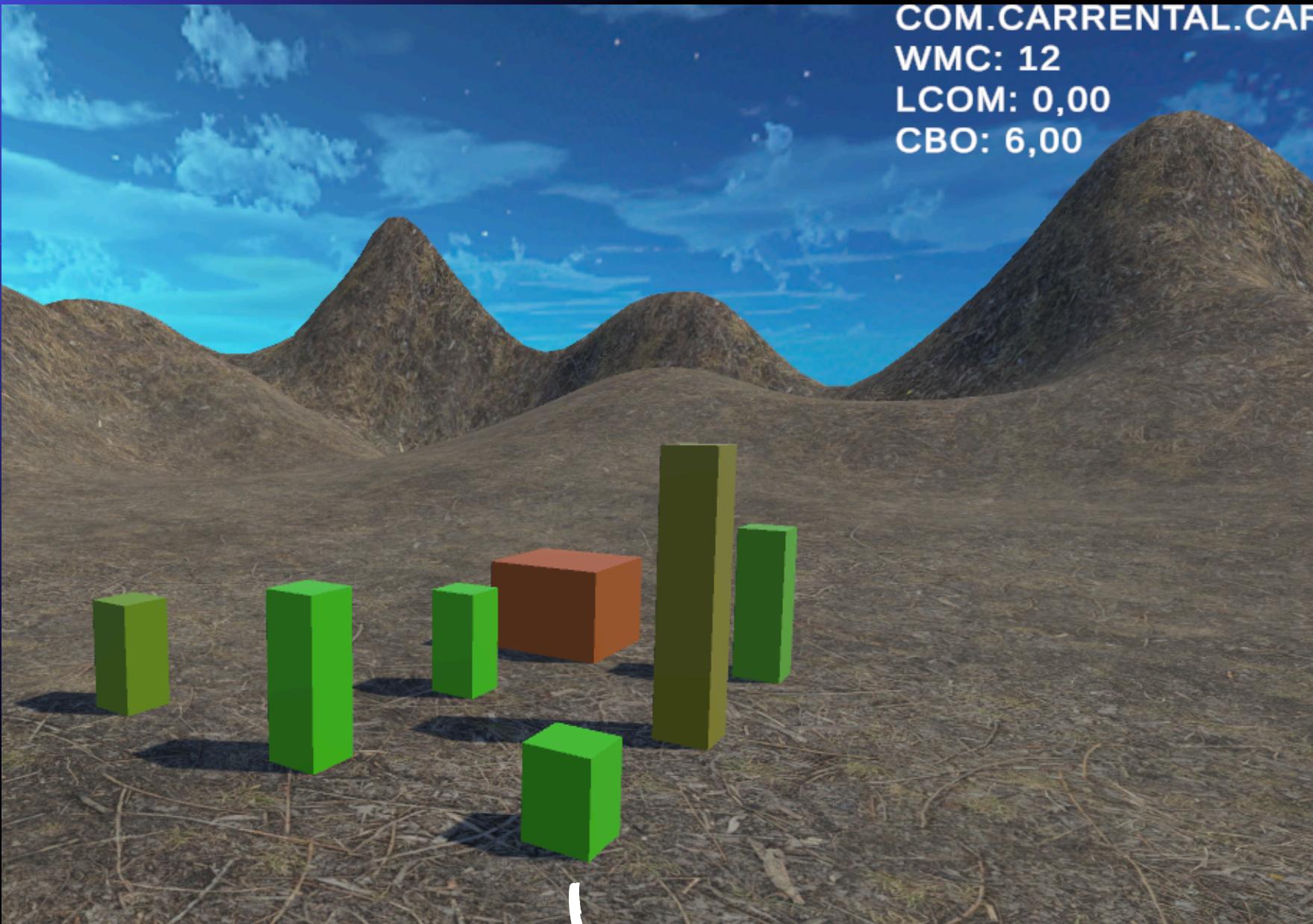
Belirlenen 3 metrik,  
kullanıcının isteğine göre  
bina özelliklerine atanır.

Giriş

Dizayn

Mimari

Uygulama



Her class, bir bina  
niteliğinde görselleştirilir.

İlgili binanın üzerine  
gelindiğinde, detay bilgiler  
görüntülenir.

Giriş

Dizayn

Mimari

Uygulama

Dinlediğiniz için  
Teşekkürler

**Eray Gökçe 20011079**  
Geliştirici

**DR. ÖĞR. ÜYESİ YUNUS EMRE SELÇUK**