TOBB EKONOMİ VE TEKNOLOJİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

MISIOT: MODÜLER, AKILLI, SUNUCU TABANLI NESNELERİN İNTERNETİ PLATFORM YAZILIMI

YÜKSEK LİSANS TEZİ

Aras Can ÖNAL

Bilgisayar Mühendisliği Anabilim Dalı

Tez Danışmanı Doç. Dr. Ahmet Murat ÖZBAYOĞLU

NİSAN 2019

Fen Bilimleri Enstitüsü Onayı

.....
Prof. Dr. Osman EROĞUL
Müdür

Bu tezin Yüksek Lisans derecesinin tüm gereksinimlerini sağladığını onaylarım.

.....
Prof. Dr. Oğuz ERGİN
Anabilimdalı Başkanı

TOBB ETÜ, Fen Bilimleri Enstitüsü'nün 151111024 numaralı Yüksek Lisans Öğrencisi **Aras Can ÖNAL** 'in ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı **"MISIOT: MODÜLER, AKILLI, SUNUCU TABANLI NESNELERİN İNTERNETİ PLATFORM YAZILIMI"** başlıklı tezi 01.04.2019 tarihinde aşağıda imzaları olan jüri tarafından kabul edilmiştir.

Tez Danışmanı:	**Doç. Dr. Ahmet Murat ÖZBAYOĞLU** TOBB Ekonomi ve Teknoloji Üniversitesi	.....
Jüri Üyeleri:	**Prof. Dr. Erdoğan DOĞDU** Çankaya Üniversitesi	.....
	Dr. Öğr. Üyesi Mehmet Tan TOBB Ekonomi ve Teknoloji Üniversitesi	.....

TEZ BİLDİRİMİ

Tez içindeki bütün bilgilerin etik davranış ve akademik kurallar çerçevesinde elde edilerek sunulduğunu, alıntı yapılan kaynaklara eksiksiz atıf yapıldığını, referansların tam olarak belirtildiğini ve ayrıca bu tezin TOBB ETÜ Fen Bilimleri Enstitüsü tez yazım kurallarına uygun olarak hazırlandığını bildiririm.

Aras Can ÖNAL

ÖZET

Yüksek Lisans Tezi

MISIOT: MODÜLER, AKILLI, SUNUCU TABANLI NESNELERİN İNTERNETİ PLATFORM YAZILIMI

Aras Can ÖNAL

TOBB Ekonomi ve Teknoloji Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Tez Danışmanı Doç. Dr. Ahmet Murat ÖZBAYOĞLU

Tarih: Nisan 2019

Bu tezde, modüler ve açık kaynak olarak geliştirilen bir nesnelerin interneti platformu anlatılmaktadır. Platform arayüz modülü, kümelenmiş sunucu modülü ve öğrenme modülü olmak üzere üç ayrı modül olarak tasarlanmıştır. Platform modüler bir yapıda tasarlandığından, platforma yeni özellikler eklemek ve mevcut özelliklerin bakımını yapmak daha kolaydır. Her modül farklı bir bilgisayar sunucusu üzerine kurulabilir ve birbirleriyle REST mimarisini kullanarak haberleşir. Arayüz modülü tüm platform özelliklerinin arayüz üzerinden kullanılabilmesine imkan tanır ve daha iyi bir kullanıcı deneyimi sunmaktadır. Sunucu modülü platformun sunduğu tüm özelliklerin yönetiminden sorumludur. Öğrenme modülü zaman serisi üzerinde LSTM algoritmasını kullanarak anomali analizi yapar. Kullanıcı CSV(Comma Separated Values) formatındaki dosyayı sisteme yükleyebilir. Buna ek olarak, herhangi bir port üzerinde soket veya REST bağlantısı oluşturarak platformun veriyi bu kaynaklardan dinlemesini sağlayabilir. Herhangi bir port üzerindeki veri akışı platform üzerinden arayüz modülü vasıtasıyla takip edilebilir. Platforma veri yüklendikten sonra, öğrenme görevleri oluşturulabilir. Platform veriyle oluşturulan görev, verisiz oluşturulan görev ve LSTM görevi olmak üzere üç farklı görev tipini desteklemektedir. Veriyle oluşturulan görev tipinde, platformda bulunan veri spesifik kriterlerle daraltılıp, istenilen Python betiğine argüman olarak verilebilir. Verisiz oluşturulan görev tipinde, herhangi bir platform verisi kullanılmadan betik doğrudan çalıştırılır. LSTM görevinde ise, çeşitli LSTM parametreleri ve spesifik bir veri kullanılarak görev oluşturulabilir. Bu seçenekte ayrıca, veri seti üzerinde anomali analizi yapılır ve analiz sonuçları arayüz modülünde grafiksel olarak görüntülenebilir.

Anahtar Kelimeler: Nesnelerin İnterneti(IoT), LSTM, Modüler IoT platformu, REST, Açık kaynak.

ABSTRACT

Master of Science

MISIOT-MODULAR INTELLIGENT SERVER BASED INTERNET OF THINGS FRAMEWORK

Aras Can ÖNAL

TOBB University of Economics and Technology
Institute of Natural and Applied Sciences
Department of Computer Engineering

Supervisor: Doç. Dr. Ahmet Murat ÖZBAYOĞLU

Date: April 2019

In this thesis, the aim is to develop modular and open source internet of things framework. The framework is designed with three separate modules as front end module, clustered server module and learning module. Since the framework is designed with modular approach, adding new features and maintaining the each module is much easier. Each module can be deployed to different host separately and communicate with using REST architecture. The front end module is responsible for offering all framework features with user interface, thus providing better user experience. The server module is responsible for managing all framework features. The learning module handles anomaly analysis of time series data with using LSTM algorithm. The user can upload new data as CSV(Comma Separated Values). In addition to bulk loading data, the user can also create socket or REST connection in desired port for listening data from various sources. Data flow on specific port can also be tracked with user interface. After uploading or listening data from various data sources, learning tasks can be created. The framework supports three different learning task types as task with framework data, task without framework data and LSTM task. Through using task with framework data option, user can create learning task with using existing data with specific criteria. Data is converted to JSON and passed as an argument to the desired Python script. Through using task without framework data option, learning task is created without using any framework data and specified script is executed directly. In LSTM option, the learning task can be created with specifying LSTM parameters and data. In this option, anomaly analysis is made on specified dataset. Result of the anomaly analysis can be seen on the graph using the front end module.

Keywords: Internet of Things(IoT), LSTM, Modular IoT framework, REST, Open source.

TEŞEKKÜR

Çalışmalarım boyunca değerli yardım ve katkılarıyla beni yönlendiren Hocalarım Doç. Dr. Murat Özbayoğlu, Dr. Ömer Sezer'e, Prof. Dr. Erdoğan Doğdu'ya, kıymetli tecrübelerinden faydalandığım TOBB Ekonomi ve Teknoloji Üniversitesi Bölümü öğretim üyelerine teşekkür ederim.

İÇİNDEKİLER

ŞEKİL LİSTESİ

Sayfa

ÇİZELGE LİSTESİ

Sayfa

KISALTMALAR

JS : JavaScript
IOT : Internet Of Things - Nesnelerin İnterneti
OB : Ortam Bilgisi
VM : Veri Madenciliği
YH : Yaygın Hesaplama
RFTS : Radyo Frekans Tanımlama Sistemi
NA : Nesnelerin Ağı
NSA : Nesnelerin Semantik Ağı
BSV : Bağlı Sensör Verisi
KSA : Kablosuz Sensör Ağları
SSA-SSN : Semantik Sensör Ağları-Semantic Sensor Networks
BMA : Bilgi Merkezli Ağ
BF : Bağlam Farkındalığı& Context Awareness
BVA : Büyük Veri Analizi
IP : Internet Protokolü
IO : Giriş Çıkış&Input Output
TSU : Tek Sayfa Uygulama
CSV : Comma Separated Value
URL : Uniform Resource Locator
LSTM : Long Short Term Memory - Uzun Kısa Dönemli Hafıza
VTK : Veri Toplama Katmanı
ÇÇY : Çıkar Çevir Yükle
VSK : Veri Saklama Katmanı
MÖA : Makine Öğrenmesi ve Analizi
GAK : Grafiksel Arayüz Katmanı

1. GİRİŞ

1.1 Problem ve Motivasyon

Son yıllarda gerek mikrodenetleyici ve benzeri gömülü sistemlerin yaygınlaşması, gerekse kablosuz ağ ve geniş bant teknolojilerindeki gelişmeler nesnelerin interneti uygulamalarının çok daha etkin bir şekilde kullanılmaya başlamasına önayak olmuştur. Bunun sonucunda çok sayıda nesnelerin interneti uygulaması ve platformu geliştirilmiştir. Bunlara örnek olarak AllJoyn, AirVantage, Arkessa, ARMmbed, Carriots, Devicehub.net, Exosite, GroveStreams IoT-Framework, IFTTT, IoTivity, IntelIoT Platforms, LinkSmart, OpenIoT, OpenMTC, OpenRemote, Platform.io, realTime.io, SensorCloud, SkySpark, Tellient, ThingSpeak, Sense Tecnic WoTkit, IBM Watson IoT Platform verilebilir.

Bu platformların birçoğu aygıt yönetimi, sensor veri analizi, gerçek zamanlı veri izleme, güvenli veri iletimi ve veri stoklama içerse de, yetersiz kaldıkları çeşitli alanlar bulunmaktadır. Bunları örnekleyecek olursak;

- **Platformların birlikte çalışabilirliği:** Çoğu platform belirli standartların veya uygun protokollerin kullanıldığı ve toplanan verilerin, üretilen bilgilerin dış dünyaya iletilemeyeceği veya sunulmadığı bağımsız araçlar olarak çalışırlar. Kapalı kutu olarak tasarlanan platformlarda, kullanıcı platform özelliklerinin dışına çıkamamaktadır. Bu sınıflandırmaya giren platformlar, kaynak kodlarını açmadıkları için, kullanıcının istediği doğrultuda geliştirme yapabilmesi de mümkün olmamaktadır.
- **Makine öğrenmesi desteği:** 2.1 tablosunda verilen platformlar çeşitli IoT standartlarını destekleseler de, önemli bir kısmı makine öğrenmesi desteğinden yoksundur. Günümüzde verilerin toplanması ve yönetimi kadar, bu verilerin işlenmesi ve analizi de önemli bir noktadır. Bazı platformlar çeşitli analiz özellikleriyle gelmelerine rağmen, kullanıcıya kendi betiklerini çalıştırma imkanı vermemektedir. Dolayısıyla kullanıcı, verileri analiz ederken platformun sunduğu makine öğrenmesi özelliklerinin dışına çıkamamaktadır.
- **Modülerlik:** Günümüzde monolitik uygulamaların yaygınlığı gittikçe azalmaktadır. Yönetilmeye ve geliştirilmeye daha müsait, modüler mimariler yaygınlaş-

maktadır. Dolayısıyla sunulan platformların tek bir kod kaynağının olmaması, kullanıcıya platformun tüm özellikleri yerine sadece ihtiyaç duyacakları modülleri kullanma imkanı sunulması hem kullanım kolaylığı, hem de geliştirme maliyetinin düşürülmesi açısından oldukça önemlidir. Tabloda verilen platformlardan açık kaynak olanların bir kısmı, kullanıcıya tüm özellikleri tek bir modül halinde sunmakta ve bu durum platformun kolay kullanılabilirliğini önemli ölçüde azaltmaktadır.

MISIoT, bu üç önemli madde üzerinde yoğunlaşmış ve bunlara çözüm amacıyla yazılmış bir platformdur. Açık kaynak olması, herhangi bir Python betiğini çalıştırabilmesi sayesinde kullanıcıları platform spesifik algoritmalarla sınırlandırmaması ve modüler bir yapıda tasarlanması ile yukarıda verilen üç maddedeki eksiklikleri gidermektedir.

Platform arayüz, sunucu ve öğrenme modülü olmak üzere üç farklı modül olarak yazılmıştır ve öncelikli olarak platformun kolay kullanılabilir olması hedeflenmiştir. Bu hedef doğrultusunda, platform içine arayüz modülü dahil edilmiş ve platform kullanılarak yapılabilen tüm işlemlerin bu modül üzerinden kullanılabilmesi sağlanmıştır. Modüler yapıda olması sayesinde, internet tarayıcısının açılmasının mümkün olmadığı durumlarda, platform REST isteği atabilen bir program veya bu isteğin oluşturulabildiği herhangi bir programlama dili üzerinden ve **sadece sunucu modülü kullanılarak** yönetilebilir. Dolayısıyla, kullanıcı sadece tek modülü kullanarak, arayüz modülü kullanmadan platformu herhangi bir işlev kaybı olmadan kullanabilir, konfigüre edebilir ve oluşturduğu görevleri yönetebilir. Modülerliğin sağladığı bir diğer önemli avantaj ise, platformu oluşturan modüllerin birbirinden bağımsız bilgisayarlar üzerinde çalıştırılabilir olmasıdır. Bu modülerlik sayesinde, platformun yönetiminden sorumlu olan sunucu modülü güçlü bir bilgisayara ve arayüz modülü internet tarayıcısı kullanılabilen herhangi farklı bir bilgisayara kurulup, platform kullanılabilir.

Platformun bir başka özelliği, Python dilinde yazılan **herhangi** bir makine öğrenmesi betiğini(script), platforma yüklenen verilerle asenkron biçimde çalıştırabilmesi ve bunların sonuçlarının platform üzerinden takip edilebilmesidir. Kullanıcının yazdığı betik, platformun JSON olarak sağladığı verileri kullanacak biçimde düzenlendikten sonra, tamamen platform üzerinden yönetilebilir, betiğin çalıştırıldığı veri seti görülebilir, betiğin oluşturduğu konsol çıktıları takip edilebilir. Modülerlik ve Node.js'in sağladığı asenkron çalışma mekanizması sayesinde istenilen sayıda, birbirinden veri ve kod olarak bağımsız olan betikler, eş zamanlı çalıştırılabilir.

Platform, tabloda verilen birçok platformun aksine modüler yapıda tasarlandığından, sadece ihtiyaç duyulan modülleri kullanılabilir veya üzerinde geliştirme yapılabilir. Böylelikle, platformun karmaşıklığı kullanıcı bazlı olarak değişkenlik göstermektedir

ve spesifik bir özellik için tüm platformun kurulmasının önüne geçilmektedir. Bunlara ek olarak, kullanıcı modüler yapı sayesinde kullanacağı modülleri farklı lokasyondaki sunuculara yükleyebilir ve platform üzerinden daha ölçeklenebilir bir yapı kurabilir.

Platform, sunduğu esnek betik çalıştırma mekanizmasına ve modülerliğe ek olarak, verilen zaman serisi üzerinde anomali analizi yapabilen, oluşturulan modelin başarımını gösterebilen bir makine öğrenmesi modülüyle sunulmaktadır. Anomali tespiti, zaman serilerinde birçok uygulama alanında önemli bir sorundur. Malhotra [1] elektrokardiyogram(ECG), uzay mekiği, multi-sensor datasetlerinde anomali tespiti için LSTM algoritmasını kullanmıştır ve %90 F-skoru başarı oranı elde etmiştir. Chauhanand ve Vig [2] ECG sinyallerinde anomali tespiti yaparak, insan kalbindeki farklı türde anormallikleri bulmayı %99 F-skoru ile göstermişlerdir. Taylor [3] ise LSTM'i otomobillerde çeşitli güvenlik açıklarını bulmak için kullanmıştır. Tüm bunlar ve diğer benzer çalışmalar, LSTM'nin zaman serileri verilerinde büyük başarı ile daha önceki çözümlerden daha iyi çalıştığını göstermektedir. Öğrenme modülünde de anomali tespiti için LSTM'den yararlanılmıştır. Bu modül, sunucu modülünde olduğu gibi asenkron bir yapıda ve çoklu iş parçacıkları(thread) kullanılarak çalışmakta olduğundan, eş zamanlı birçok görevi çalıştırabilir. Modül ayrıca arayüz modülüyle de uyumlu tasarlandığından, üretilen anomali noktaları arayüz modülünde grafiksel olarak gösterilebilmekte ve kullanıcıya modelin başarımıyla ilgili çeşitli parametreleri sunmaktadır. Opsiyonel bir modül olup, kullanılması zorunlu değildir.

Platformun güçlü olduğu bir başka nokta açık kaynak olarak planlanmasıdır. 2.1 tablosunda belirtilen platformların birçoğu açık kaynak olmadığından, kullanıcı platforma kendi ihtiyaçları doğrultusunda müdahale edememekte, bu durum platformdaki sorunların daha geç farkedilmesi ve çözülmesi, yeni özellikle ekleyememe gibi sorunlara yol açmaktadır. Dolayısıyla MISIoT, açık kaynak olmasının verdiği avantajla dünya üzerinde farklı geliştiriciler tarafından, farklı ihtiyaçlar için modifiye edilebilecek ve platformda bulunan hatalar daha hızlı giderilebilecektir. Platform üzerinde daha efektif geliştirme yapılabilmesi için Javascript(JS) dili tercih edilmiştir. Bu seçimde JS'nin dünyada en çok kod tabanına sahip olması etkin olmuştur [1].

1.2 Tezin Katkıları

Platform yazılırken, "Problem ve Motivasyon" bölümünde bahsedilen eksiklikler göz önüne alınarak, bunları gidermeyi amaçlayan bir platform yazılmıştır. Platform arayüz, sunucu ve öğrenme modülleri halinde tasarlanmış olup, her modülün kendine has görevleri bulunmaktadır. Önerilen platform ile;

[1] https://octoverse.github.com/2017/

- Platformun kullanıcı arayüzü üzerinden platforma yeni soket veya REST bağlantıları açılabilir.
- Platform soket/REST bağlantısı veya CSV formatındaki dosyalardan veri alabilir. Aldığı verileri saklayıp, daha sonra oluşturulacak öğrenme görevlerinde kullanabilir.
- Soket veya REST bağlantıları üzerinden alınan verilerin, alındıkları port üzerinden gerçek zamanlı takibi yapılabilir.
- Platform üzerinden LSTM algoritması kullanılarak, istenilen veri setinin zaman serisi analizi yapılıp, bu analizin sonuçları hem grafiksel hem de algoritmanın ürettiği anomali sayısı, test ve train skorlarının kullanıcı arayüzü üzerinden görülebilir.
- Platform üzerinde LSTM'den bağımsız olarak, kullanıcının yazdığı, herhangi bir klasör altındaki herhangi bir ML kodu çalıştırılarak, sonuçları platform üzerinden takip edilebilir.
- Çalıştırılan kodlar, platformda bulunan mevcut verilerle veya doğrudan, platform verisi olmadan çalıştırılabilir. İki durumun takibi de platform üzerinden yapılabilir.
- Platformun kullanıcı arayüzü üzerinden, platforma CSV formatında çoklu veri yüklemesi yapılabilir.
- Oluşturulan öğrenme görevleri, platformun bu iş için yazılmış ekranı üzerinden takip edilebilir, sorgulanabilir, görevlerin sonuçları platform üzerinden görülebilir.
- Açık kaynak kodu olduğundan, kullanıcılar kendi modüllerini ekleyebilirler.
- Modüler yapıda olması sebebiyle, her modül farklı bir sunucuda çalışabilir.
- Platform kümesel yapıda tasarlandığından, aynı anda farklı istekleri işleyebilir.
- Modüler yapısı sebebiyle, arayüz modülü olmadan sadece REST istekleri ile yönetilebilir. Arayüz üzerinden yapılan her işlem, arayüze ihtiyaç duyulmadan da yapılabilir.
- Asenkron yapıda(fire&forget) çalışması sebebiyle, aynı anda, platformun üzerinde çalıştığı bilgisayarın gücünün izin verdiği ölçüde öğrenme görevleri tanımlanabilir.

Platform, zaman serisi şeklinde verinin analizi için kullanılabileceği gibi, aynı zamanda kullanıcıların sadece Python betiklerini asenkron olarak platformda yüklü ve çeşitli kriterlere göre değiştirilebilen veriyle çalıştırıp, sonuçlarını takip edebilecekleri, karşılaştırabilecekleri ve saklayabilecekleri bir şekilde de kullanılabilir. Platform üzerinden çalıştırılan betiklerin hangi parametre ve verilerle çalıştırıldığı ve sonuçları arayüz modülü üzerinden görülebileceği için, yazılan betiklerin takibi oldukça kolaylaşmaktadır.

Tez altı bölüme ayrılmış olup, ikinci bölüm olan **Nesnelerin İnterneti Nedir** bölümünde nesnelerin internetinin ne olduğu, günümüzde ilgili olduğu alanlar, potansiyel uygulama alanları ve araştırma alanları ile ilgili detaylı bilgi verilmektedir.

üçüncü bölüm **MISIoT platformunda**, tez kapsamında yazılan platform detaylıca anlatılmaktadır. Bu bölümde MISIoT yazılırken kullanılan tüm teknolojiler ve MISIoT platformunu oluşturan sunucu, arayüz ve öğrenme modülleri anlatılmaktadır.

Dördüncü bölüm **Örnek Kullanım Senaryoları** olup, bu bölümde platformun desteklediği her bir görev tipi için konfigürasyon aşamasından itibaren adım adım görevler oluşturulmuş ve sonuçlar yorumlanmıştır. Bu bölüm okunarak, platformun nasıl kullanılabileceği ve konfigüre edilebileceği anlaşılabilir.

Beşinci bölüm **Planlanan Geliştirmelerde**, platformu daha performanslı ve güçlü hale getirmek için gelecekte yapılması planlanan yapısal geliştirmeler anlatılmaktadır.

Altıncı bölüm olan **Sonuç** bölümünde, platformda yapılan işler ve önerilen yapı özetlenmiştir.

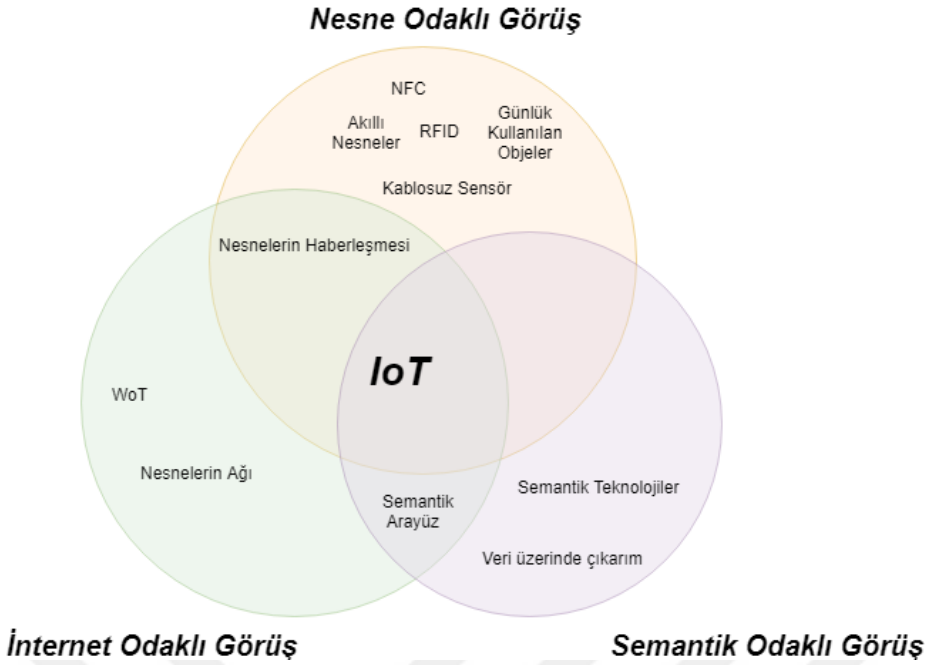
2. NESNELERİN İNTERNETİ

2.1 Nesnelerin İnterneti Nedir

Nesnelerin İnterneti (IoT), kablosuz sensör teknolojisini genişleten ve tüm dünyada cihaz bağlanırlığını sağlayan yeni bir teknoloji altyapısıdır. Literatürde IoT, insanların ve nesnelerin birbirlerine her zaman, her yerde ve herhangi bir ağı veya yolu kullanarak bağlanması olarak tanımlanmaktadır [4][5]. Gelecekte, tüm elektronik cihazların internete bağlanacağı ve birbirleriyle iletişim ağları üzerinden iletişim kuracağı beklenmektedir. IoT, farklı alanlarda ve uygulamalarda yaygın olarak kullanılmaktadır. Sağlık, çevre, trafik, havacılık, üretim, savunma, ev otomasyonu, iletişim, IoT teknolojilerini kullanan uygulama alanlarının örnekleridir. Bu alanlarda, bağlantılı elektronik IoT cihazlarının sayısı yıldan yıla artmaktadır. 2020 yılına kadar internete bağlı elektronik cihazların sayısının 50 ile 100 milyar arasında olacağı beklenmektedir [4]. Bu cihazların yaygın kullanımının bir sonucu olarak, üretilen toplam verinin 35 zettabayttan daha fazla olması beklenmektedir [6] [7].

IoT hakkında farklı tanımlamalar yapılmaktadır. Atzori'ye göre IoT, internet odaklı görüş(ara katman) nesne odaklı görüş(sensor) ve semantik odaklı görüş(bilgi) olmak üzere üç farklı görüşle tanımlanabilir [8]. Bu üç görüş, IoT paradigmasını oluşturmaktadır. IoT ancak bu üç görüşün kesişimiyle potansiyeline ulaşabilir. 2.1 ifadesi bu görüşleri göstermektedir. Bunlara ek olarak sensor ağları, RFID(Radio Frequency Identification), NFC(Near Field Communications), EPC(Electronic Product Code), WSAN(Wireles Sensor and Actuator Networks) nesne odaklı görüşlere dahil edilebilir. Yine bu şemaya göre, semantik teknolojiler ve veri üzerinde çıkarım yapma semantik odaklı görüşe dahil edilmektedir. Son olarak WoT(Web of Things) internet odaklı görüş altında yer almaktadır.

IoT üzerinde avrupa kaynaklı araştırma projelerine göre [9], "nesneler" aktif olarak iş, bilgi ve sosyal süreçlerde yer alan, birbirleriyle çevreyle etkileşim içinde bulunan, bulundukları çevreyle ilgili birbirleriyle veri alışverişinde bulunan, çevreyi sezen, çevrelerinde gerçekleşen olaylara otomatik olarak tepki verip, hali hazırda devam eden süreçlerden etkilenen ve çeşitli servisleri insan müdahalesiyle veya müdahale olmadan oluşturabilen olarak tanımlanmaktadırlar.



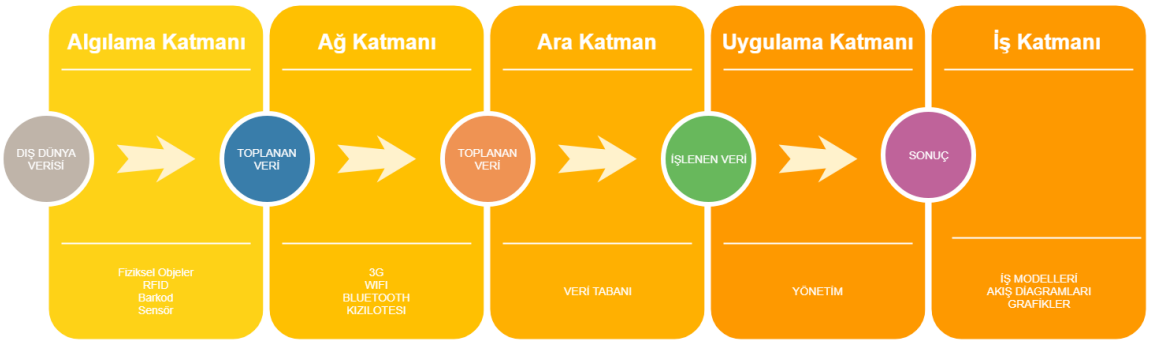
Şekil 2.1: Atzori'nin IoT Paradigması[8]

Forrester'a göre IoT, bilgi ve iletişim teknolojilerini kullanarak, kritik altyapı ve şehir yönetimi, eğitim, sağlık, güvenlik, emlak, ulaşım ve çeşitli araçları çevresine daha duyarlı, interaktif ve verimli hale getirir [10].

Bir başka tanımlamada, çevresini algılayabilen cihazların ortak bir platform üzerinden bilgi paylaşması ve daha yenilikçi uygulamalar için ortak bir çerçevede geliştirme yapılması olarak tanımlanmaktadır. Buna ulaşmanın yolu ise, büyük çaplı sensör verilerinin kullanılması, bulut programlama ve toplanan verilerin analizi olarak gösterilmiştir [11].

İnternete bağlanan IoT cihazlarının sayısı arttıkça, bunun bir sonucu olarak farklı IoT platformları, servisleri ve platformları geliştirilmekte ve kullanılmaktadır. Verileri toplamak ve analiz etmek için günlük yaşamımızda kullanılan birçok IoT platformu ve servisi bulunmaktadır. Bunlar çeşitli çalışmalarda incelenmiştir [12] [13] [14].

IoT'un günümüzde ve gelecekte oldukça büyük bir etkiye sahip olması beklenmektedir. NIC(The US National Intelligance Council)'in tahminlerine göre, 2025 yılına kadar, günlük yaşantımızda kullandığımız her eşyanın internet arayüzü olması beklenmektedir [8]. Bu durum beraberinde ciddi zorluklar da getirmektedir. Çevremizdeki nesnelerin her birinin internete bağlı olması sonucunda oluşacak büyük verinin nasıl işleneceği bu sorunlara örnek teşkil etmektedir. 2.1 ifadesinde görüleceği üzere, IoT paradigması geniş bir alanı kapsamaktadır ve bu paradigmanın içerdiği alt kümelerin her birinde, gelecekte IoT'un yaygınlaşması sonucunda ortaya çıkacak güvenlik, veri



Şekil 2.2: IoT Mimarisi

paylaşımı ve enerji problemlerine çözümler aranmaktadır.

2.2 IoT Mimarisi ve Özellikleri

Günümüzde internete bağlanabilen birçok nesne farklı işlemci gücü ve kapasiteyle gelebilir. Ancak IoT ile, nesneler farklı olsa da amaçları aynıdır ve IoT mimarisi bu amaç etrafında evrilmiştir. IoT ile nesneler çevrelerindeki veriyi işleyip, bunun sonucunda otomatik olarak bir aksiyon almaları hedeflenmektedir. IoT'un çalışma prensibi şu şekildedir;

- ***Akıllı nesne, çevresindeki sıcaklık, yön, hareket, titreşim, hızlanma, nem, havadaki kimyasal değişimler gibi verileri toplar***.
- ***Çevreden toplanan veriler işlenerek nesnenin bu veriler ışığında bir aksiyon alması sağlanır***.
- ***Nesne, çevresi ve alınan aksiyonun sonuçları ile ilgili bilgiyi yöneticiye aktarır, yönetinin işini kolaylaştıracak çeşitli servisler sağlar***.

IoT'un işleyişi sıradan gibi gözükse de, toplanan datanın büyüklüğü ve internete bağlanan nesnelerin fazlalığı sebebiyle çok daha fazla oranda internet trafiği oluşturacaktır. Bu sebeple, önerilen herhangi bir IoT mimarisinin bu zorlukların altından kalkacak şekilde tasarlanması gerekmektedir. Nesnelerin kendi aralarında haberleşmesi, stabil çalışması ve etki alanının genişletilmesi gibi konular da tasarlanacak sistemin mimarisinde dikkate alınmak zorundadır. 2.3 ifadesinde IoT mimarisi genel hatlarıyla görülebilir.

IoT mimarisi katmansal olarak beş katmandan oluşmaktadır [15]. Bunlar incelenecek olursa;

- ***Algılama Katmanı(Perception Layer)*** Bu katman fiziksel objeler ve sensör ağlarından oluşmaktadır. Sensör ağları, IoT mimarisinin en kritik bileşenlerinden biridir. Bir sensör ağının içindeki sensörler tek tip olabileceği gibi, farklı verileri toplamaya odaklanan sensörler de bir arada kullanılabilir. Farklı sensör ağları arasında iletişim kurulabilir. Bir ağda bulunan sensörler kablolu veya kablosuz olabilir. Sensörler, algılama yönetime göre RFID(Radio-Frequency Identification), barkod veya kızılötesi olabilir. Sensörün tipine göre, konum, sıcaklık, yön, titreşim, hız, nem gibi farklı tipte veriler toplanabilir. Sensörlere ek olarak veriler toplandıktan sonra ağ katmanına iletilmektedir.

- ***Ağ Katmanı(Network Layer)*** Bu katmanın görevi, sensörlerden gelen veriyi veri işleme sistemine aktarmaktır. Aktarım kablolu veya kablosuz olabilir. Sensörün tipine göre 3G, UMTS, Wifi, Bluetooth, kızılötesi, ZigBee gibi çeşitli yöntemler kullanılabilir. Ağ katmanı, toplanan veriyi ara katmana aktarır.

- ***Ara Katman(Middleware Layer)*** IoT üzerinde çalışan aygıtlar farklı tipte veriler toplayıp, farklı servislere sahip olabilirler. Bu nedenle, farklı aygıtlardan gelen servislerin yönetilmesi ve gelen verilerin veritabanına kaydedilmesi gerekmektedir. Bu temel gereksinimlere ek olarak, bu katmanın sahip olması gereken en önemli özellik, farklı sensörlerden gelen farklı tipte verileri tek bir potada eritebiliyor olması, uygulamadan ve kendisini besleyen sensör ve ağ katmanının mimarisinden bağımsız olarak, entegre edildiği sistemde sorunsuz biçimde çalışabilmesidir [16]. Ayrıca, güvenlik, yüksek çevrimiçi çalışma süresi gibi sorunları da çözmesi bu katmandan beklenmektedir. Veriyi işleyip bulunan sonuçlar doğrultusunda bir çıktı üretir.

- ***Uygulama Katmanı(Application Layer)*** Bu katman, ilk üç katman sonucundaki işlenmiş veriyi yöneten ve ara katmanın ürettiği çıktı ile nasıl bir aksiyon alınacağına karar veren katmandır. Çeşitli endrüstri uygulamaları bu katmanda olabilir.

- ***İş Katmanı(Business Layer)*** Bu katman, IoT sisteminin genel yönetiminden ve uygulama katmanından alınan verilerle çeşitli iş modelleri oluşturup bunların yönetiminden sorumludur. Bir IoT sisteminin başarılı olması için iyi bir iş modeli üzerinden yürütülmesi gerekmektedir.

Bir IoT sistemini oluşturan çeşitli karakteristik özellikler bulunmaktadır. Bunlar aşağıdaki maddelerde incelenmiştir.

- ***Cihazların Çeşitliliği:*** Sensör ağlarında bulunan sensörlerin ve IoT altyapısını oluşturan cihazların birbirinden farklı yapısı nedeniyle, bunların yönetimi sağlayacak bir mimari ve iletişim protokolleri olması gerekmektedir [16].

- ***Ölçeklenebilirlik:*** Günlük yaşantıda kullanılan her objenin dünya çapında bir bilgi ağına bağlanması sonucunda, bu cihazların isimlendirilmesi, adreslenmesi, cihazların aralarında haberleşmesi, ortaya çıkan çok büyük miktarda verinin işe yarar bir hale getirilip yönetilmesi gibi birçok üzerinde düşünülmesi gereken nokta çıkacaktır. IoT'u baz alan bir sistemin bu nedenlerden dolayı ölçeklenebilir olması ve bahsedilen noktaları göz önünde bulundurması gerekmektedir [17].

- ***Verimlilik:*** IoT sonucunda oluşturulan sensör ağlarının, mimaride bulunan ara katman, yazılım gibi bileşenlerin kendi aralarında haberleşmesi ve veri üretmesi sırasında harcanan enerjinin minimum seviyede olması gerekmektedir. IoT, ölçeklenebilir olduğundan, milyonlarca sensörün içinde bulunduğu bir sistemin güç tüketimi sınırlandırılmak zorundadır.

- ***Zeka:*** Sensör ağları tarafından toplanan verinin üzerinde çıkarım yapılarak gizli verilere ulaşılmasıdır. Toplanan veriden çeşitli modellerin çıkarılabilmesidir.

- ***Mimari:*** 2.3 ifadesinde görülebileceği gibi, IoT bir çok katmandan oluşmaktadır. Bu katmanların da kendi içlerinde birçok alt katmanı olabilir. Ancak temelinde, tüm mimarilar olay odaklı(event-driven) veya zaman odaklıdır(time driven) [18]. Olay odaklı mimarilerde, sensörler bir olay olduğu zaman veri üretirler. Zaman odaklı mimarilerde ise, sensör düzenli olarak veri üretir. Sıcaklık sensörü düzenli veri üreten sensöre örnek olarak verilebilir. IoT sistemleri genellikle olay odaklı olarak çalışırlar [19].

- ***Karmaşık Sistem:*** IoT, otomatik olarak çalışan birçok sensör sistemi içermektedir. Mevcut durumda dünya üzerinde milyonlarca sensör bulunmaktadır ve bu sensörler birbirlerinden farklı olabilirler. Örneğin, bazılarının hafızası ve işlem gücü daha düşükken, bazı sensör ağları daha fazla işlem gerçekleştirebilecek güçtedir.

- ***Nesne Sayısı:*** 2020 yılına kadar Internete bağlı 50-100 milyar civarı nesne olması beklenmektedir. İnternete daha çok nesne bağlandıkça, bunun doğal bir sonucu olarak bu nesneler arasında olan etkileşim sayısı da ciddi oranda artacaktır. Dolayısıyla, bir IoT sistemi oldukça fazla nesneye sahip olabilir.

- ***Zaman:*** IoT milyarlarca eş zamanlı olayı işleyebilir. Bu nedenle IoT sisteminde gerçek zamanlı veri işleme olması zorunludur.

- ***Organize Olabilme:*** Nesnelerin bulunduğu konumu bilmesi, çevrelerindeki diğer nesnelerle olan iletişimlerinde ve olaylar karşısında aksiyon almalarında oldukça kritiktir [20]. Burada önemli nokta, iletişim kuran ve aksiyon alan cihazla-

rın minimum insan müdahalesiyle, dinamik olarak çevrelerinde gelişen olaylara karşı otomatik bir tepki oluşturabilmesidir.

- ***Semantik birlikte çalışabilirlik ve veri yönetimi:*** Sensörlerin farklı veriler üretebilmesi sebebiyle, bu üretilen verilerin standart bir formatta ve dilde üretilmesi, herhangi bir IoT uygulamasının, herhangi bir sensör verisi üzerinde yeni çıkarımlar yaparak, farklı bilgilere ulaşabilmesi için gereklidir.

- ***Gömülü güvenlik:*** Sensör ağları ve diğer IoT mimari bileşenleri arasındaki iletişim güvenli olması gerekmektedir. Bu gereksinim, IoT teknolojisinin geniş kitleler tarafından benimsenmesi için gereklidir. Bu nedenle oluşturulacak herhangi bir IoT sisteminde, güvenlik gözardı edilmemelidir.

- ***Servis olarak sunulabilirlik:*** Günümüzde bulut bilişim oldukça popülerdir. Dolayısıyla, yazılan platformların, yazılımların, altyapının bulut üzerinde bir servis olarak sunulması ve gerektiğinde başka servislerle kolayca iletişime geçebilmesi, IoT gibi ciddi bir altyapı gereksinimi olan teknoloji için oldukça kritiktir. IoT temelini nesnelerin iletişiminden aldığından, internet üzerinden herhangi bir platforma servis olarak ulaşabilmek, bu platformun özelliklerini kullanabilmek, IoT sistemlerinin yaygınlaşmasını kolaylaştırır.

2.3 IoT ile İlgili Alanlar

IoT kavramını daha iyi anlayabilmek için, IoT ile ilgili alanlar hakkında bilgi sahibi olmak gerekmektedir. IoT, bu alanlarda yapılan çalışmalarla yükselmekte ve temelini almaktadır. Bunlar alt başlıklarda incelenecektir.

2.3.1 Yaygın Hesaplama(Pervasive Computing)

Yaygın hesaplama(YH), birbirine bağlı sensor ve bilgisayarların çevrelerini anlayabilmeleri için birbirleriyle haberleşmesidir. Ancak burada kritik nokta, YH özelliğine sahip bir ortamın, insanlarla doğal biçimde entegre olması, insanların hareket alanını kısıtlamamasıdır. YH dört farklı araştırma alanını ilgilendirmektedir. Bunlardan ilki, akıllı alanların efektif biçimde kullanılmasıdır. Akıllı alanı tanımlamak gerekirse, bina ile bilgisayar sistemlerini birleştiren ve belirli sınırları olan alan denebilir. Örnek olarak, alanın ışık ve sıcaklık seviyelerinin, ortamda bulunanlara göre ayarlanması gösterilebilir. Bir diğer karakteristik ise, kullanıcılar ile doğal biçimde entegre olması ve kullanıcıların yaşam alanına minimum müdahale ile çalışmasıdır. Üçüncü bir özellik ise, akıllı alanın sadece ilgili olduğu kullanıcı ile iletişimde bulunmasıdır [21]. Aksi

takdirde, gereğinden fazla veri ve enerji kullanımı gerçekleşebilir. Son olarak, konumundan bağımsız olarak, akıllı alanların aynı teknolojik seviyede olması gerekmektedir. Ancak günümüzde oldukça zordur. Herhangi bir alanın teknolojik alt yapısı, dünya üzerindeki farklı bir alanın teknolojik altyapısından daha gelişmiş veya kötü olabilir. Bu durum, yaygın hesaplamanın önemli özelliğinden biri olan görünmezliği sekteye uğratmaktadır.

YH, IoT'un temellerini oluşturmaktadır. IoT ile benzer olarak, akıllı alanlar, birbirleriyle ve kullanıcılarla etkileşime geçen ve ortam şartlarına göre değişik davranışlarda bulunan aygıtlar, YH ve IoT'un yapıtaşını oluşturmaktadır.

2.3.2 Ortam Bilgisi(Ambient Intelligance)

1990'lı yılların sonuna doğru, araştırmacılar ***ortam bilgisi*** isimli bir kavramı ortaya atmışlardır [5]. Ortam bilgisi(OB), küçük ve gözle görünmeyen boyuttaki elektronik cihazlarla akıllı alanlar oluşturabilmek için gereksinimleri tanımlamaktadır [22]. Yakın gelecekte, OB teknolojisi kullanılarak insanların yapmak istediklerini önceden sezinleyerek bunlara otomatik olarak destek olan aygıtlar geliştirilmesi beklenmektedir. Günümüzde hali hazırda, bir alana kullanıcı girdiği zaman bu alan otomatik olarak odanın sıcaklığını ve ışık seviyesini ayarlayabilmektedir. OB ise, alanda bulunan gelişmiş iletişim ağı kullanılarak, daha iyi eğitim, engelli insanlar için daha rahat yaşama alanları sunmak ve genel olarak insanların hayatını basitleştirerek kolaylaştırmak hedefine sahiptir.

Günümüzde OB, dağıtık zeka, veri iletişimi, yazılım ve donanım tasarımı, bilgisayar görüsü, konuşma tanımlama, robotik gibi birçok farklı dalın çalışmalarıyla geliştirilmektedir. Dağıtık zeka ile akıllı alanda bulunan bir çok zeki iş birimi OB üzerinden geliştirilmiş sistemleri kontrol etmekte kullanılabilir. Çevredeki değişikliklere göre ortamda bulunan kullanıcıya destek olabilir. Yazılım ve donanım tasarımı ise, YH ile anlatılan ve kullanıcının teknolojiyi minimal derecede hissetmesini sağlayan teknolojilerin akıllı alana entegre edilmesiyle ilgilidir. Bunların bir bütün halinde çalışması için ise, ortamda bulunan sensörler veya elektronik aygıtlar vasıtasıyla bilgi toplanması gerekmektedir. Bunlar üzerinde veri madenciliği ve makine öğrenmesi uygulanarak, o anki duruma uygun çeşitli modeller oluşturulması, bunlar üzerinden kullanıcıya dinamik olarak bir çıktı verilmesi mümkün olmaktadır.

2.3.3 Nesnelerin Ağı(Web of Things)

IoT kavramı, her ne kadar nesnelerin birbirleriyle etkileşimini ele alsa da, bunun hakkında herhangi bir teknolojik altyapı öne sürmemektedir. Özünde, nesneleri internet vasıtasıyla birbirlerine bağlamaktan bahsetmektedir. Nesnelerin Ağı(NA) ise, nesnelerin birbirine bağlanma işleminin nasıl olacağı hakkında bir yaklaşım öne sürmektedir [23]. NA yaklaşımında, kişiler istemci(client), nesneler ise sunucu(server) olarak ele alınmaktadır. Dolayısıyla, nesneler içinde gömülü bir web sunucusu olmak zorundadır. İstemci, nesne üzerindeki bu gömülü sunucuya erişim sağlayabilir. Sunucuya, kişinin sadece aktif olarak kullandığı istemci nesnelerden erişim sağlanabilir. Böylelikle standart bir web sunucudan farklı olarak, milyonlarca farklı lokasyondan eş zamanlı gelecek istekler olmayacaktır. Bu da, sunucu görevindeki nesnenin o kişiye özel olmasını sağlar.

Günümüzde HTTP tabanlı çeşitli bağlantı yöntemleri olsa da, SOAP gibi, gömülü sistemde çalışması zor olan ve daha karmaşık protokellerin yerine REST mimarisi tercih edilmektedir [24].

REST bir dizayn konsepti olup, spesifik bir teknoloji kümesini kapsamamaktadır. Bir uygulamanın kullanılacak herhangi bir komponenti REST kaynağı(resource) olarak adlandırılabilir. Buna göre [25], REST mimarisini beş maddeyle açıklayabiliriz;

- ***Kaynak Tanımlama(Resource Identification)*** Bir REST endpointine gelen istek sonucunda dönen cevap, o kaynakla alakalı linkleri bulmak için kullanılabilir. Bu linkler takip edilerek kaynakla ilgili bilgilere ulaşılabilir.

- ***Tek Arayüz(Uniform Interface)*** REST, HTTP tabanlı bir protokol olduğundan dolayı, belli bir metod kümesi kullanılarak kaynaklara ulaşılabilir. Bu işlem kullanılan platformdan bağımsız olduğundan, örneğin farklı programlama dilleriyle yazılmış sunucu ve istemciler veya farklı aygıtlar, aynı HTTP metodları üzerinden birbirleriyle iletişim sağlayabilirler. Bu metodlar GET, PUT, POST ve DELETE'dir. GET üzerinden kaynak ile ilgili bir bilgi alınabilir. GET isteği ***idempotent*** özelliktedir. Bunun anlamı, aynı istek defalarca atılsa bile dönülen sonuç aynıdır, server tarafında herhangi bir değişikliğe sebep olmaz. PUT, bir kaynak güncellemek istendiği zaman kullanılan bir istektir. Örneğin, hali hazırda var olan kaynağın spesifik bir özelliği güncellenmek istendiği zaman PUT isteği kullanılabilir. DELETE, herhangi bir kaynak silinmek istendiği zaman kullanılabilir. Spesifik bir sensörü silmek buna örnek olabilir. POST ise, sunucu üzerinde yeni bir kaynak oluşturulmak istendiği zaman kullanılan bir istektir. Örneğin, sisteme yeni bir aygıt eklenmek isteniyorsa POST kullanılabilir.

- ***Anlaşılır Mesaj Yapısı(Self-Describing Messages)*** Dünyaca kabul görmüş ve yaygın olarak kullanılan JSON formatı, okunabilir ve kolayca pars edilebilir olduğundan dolayı, mesajlar daha rahat işlenebilir.

- ***Link Yapısı(Hypermedia Driving Application State)*** REST cevabı olarak dönen linkler sayesinde, istemci, sunucuya ait tüm linkleri bilmeden sadece belirli bir link üzerinden istek atabilir ve gelen cevaplardaki yeni linkleri takip ederek sunucudan istediği veriyi alabilir.

- ***Belleksiz(Stateless)*** HTTP isteklerinde, herhangi bir durum bilgisi tutulmadığından, istek at, cevabı al ve unut şeklindedir. Dolayısıyla herhangi iki istek birbirinden tamamen bağımsızdır.

2.3.4 Nesnelerin Semantik Ağı(Semantic Web of Things)

Günümüzde sensorler altyapı, mobil aletler ve çeşitli elektronik aletlerde yaygın olarak kullanılmaktadır. Ancak, farklı alanlarda kullanılan sensörler farklı türde veri üretebilecekleri için, birçok çeşit, birçok farklı formatta veri ortaya çıkmaktadır. Bunun sonucu olarak, bu verilerin sınıflandırılması, üzerinde veri madenciliği yapılması, çeşitli paternler çıkarılması oldukça zor olmaktadır. Bu sebeple, dünya üzerindeki tüm sensörlerin ortak bir şekilde kullanabileceği bir semantik bir format geliştirme gerekliliği ortaya çıkmıştır [26]. Bu semantik format, herhangi bir sensor verisi okunmak istendiğinde, o sensöre göre özelleştirilmiş gömülü bir sistemi bilmeden rahatlıkla buna olanak sağlamalıdır. Ayrıca bu semantik veri, üzerinde çeşitli çıkarımlar yapılabilmesini desteklemeli, sistemin durumunun anlaşılmasını sağlayacak biçimde sorgu atılabilmesine olanak sağlamalıdır. Bu gereksinimler ışığında ortaya çıkan veri çeşitlerinden biri Bağlı Sensor Verisi-BSV-(Linked Sensor Data) olmuştur. Bu yöntemin alt kümesi olan "Resource Description Framework"(RDF)'e göre, veri ***subject-predicate-object*** olacak biçimde üçlü halde tutulmaktadır. Şu şekilde bir örnek verilebilir, ***sensor1 is-in Dikmen and Dikmen is-in Ankara***. Buradan şu şekilde bir çıkarım yapılabilir, ***sensor1 is-in Ankara***. Bir diğer avantajı ise bu veri yapısının sorgulanabilir bir formatta olmasıdır.

2.3.5 Veri Madenciliği(Data Mining)

Veri Madenciliği(VM), büyük veri setleri üzerinde çeşitli paternler ve algoritmalar kullanarak verideki gizli bilgileri elde etmek için kullanılan bir araçtır. VM için bilgi keşfi, bilgi çıkarılması, veri/patern analizi, veri arkeolojisi, veri taraması, bilgi hasatı gibi çe-

şitli tanımlamalar kullanılmaktadır. Standart bir veri madenciliği süreci üç adımdan oluşmaktadır; [27]

- ***Veri hazırlığı:*** Bu adımda veri VM için hazırlanır. Verinin çeşitli gereksiz alanlardan temizlenmesi, veri başka veri kaynaklarıyla entegre etme ve verinin belli kısımlarının VM sistemine entegre edilmesi gibi alt adımlardan oluşur
- ***Veri Madenciliği:*** İlk adımda temizlenen ve VM için uygun hale getirilen veri üzerinde, çeşitli algoritmalar uygulanarak paternler keşfetmek ve değerlendirmek için VM uygulanır.
- ***Veri Sunumu:*** VM sonucunda elde edilen verinin kullanıcıya sunulduğu aşamadır.

IoT, doğası gereği oldukça büyük verilerle çalışmaktadır. Dünya üzerindeki her aygıtın bir veri kaynağı olabileceği düşünülürse, ortaya çıkan büyük verinin anlamlaştırılması ve bu veriden işe yarar çıkarımlar yapılması gerekmektedir. VM bu noktada devreye girerek, aygıtlardan gelen çeşitli veriler üzerinden anlamlı çıkarımlar yaparak, ortamdaki aygıtların daha doğru şekilde çalışmasına katkıda bulunur. IoT verisi büyük veri olarak nitelendirilebilir. Bu nedenle büyük veriye has çeşitli karakteristik özellikleri vardır. Bunlar;

- Zettabayt seviyesinde veri kümeleri içerebilir
- Çok çeşitli veri kaynakları bulunmaktadır ve her büyük veri üreten aygıt farklı formatlarda veri üretebilir.
- VM yapılacak veri kaynakları, boyutları dolayısıyla oldukça kompleks yapıdadır ve verideki gizli paternlerin elde edilmesi daha zordur.

Bu karakteristik özellikler dolayısıyla, IoT daha efektif çalışabilmek için VM'ye ihtiyaç duymaktadır ve bu alanla doğrudan ilişkilidir. VM alanında IoT ile çalışabilecek çeşitli büyük veri madenciliği alt yapısı sunan açık kaynak projeler bulunmaktadır. Bunlardan öne çıkanları aşağıda belirtilmiştir.

Apache Mahout [2], makine öğrenmesi ve VM algoritması sunan platformdur. R programlama dili [3], istatistiksel hesaplama ve görselleştirme için kullanılmaktadır. MOA

[2]https://mahout.apache.org/
[3]https://www.r-project.org/about.html

projesi [4] gerçek zamanlı VM yapabilme kapasitesine sahiptir. Pegasus [5], Hadoop platformu için madencilik yapabilme kapasitesine sahip bir araçtır. Bu araçların hepsi IoT'un ihtiyaç duyduğu büyük veriyi temizlemek, analiz etmek ve çeşitli paternler bulmak için kullanılabilir.

2.3.6 Radyo Frekansı Tanımlama Sistemi(Radio Frequency Identification System)

Radyo frekansı tanımlama sistemi(RFTS), bilgisayarların ve makinelerin çevrelerindeki objeleri tanımalarını ve spesifik bir hedefin radyo dalgalarıyla kontrol edilmesini sağlayan bir teknolojidir. Teknoloji ilk defa 1945 yılında ortaya çıkmıştır ve Sovyetler Birliği tarafından aktif olarak kullanılmıştır [28].

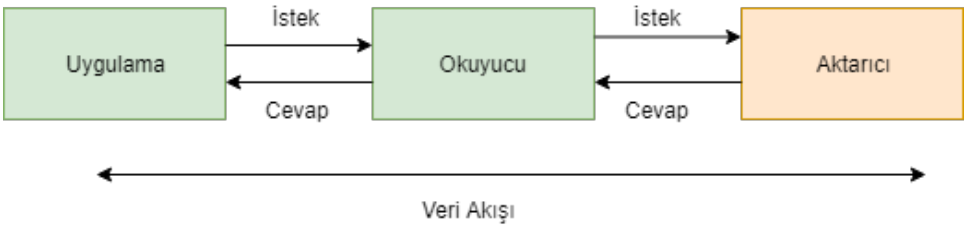
RFTS sistemi aktarıcı(transmitter/tags) ve okuyucu(receiver)'dan oluşmaktadır. Aktarıcı ufak bir mikroçip olup, bir obje ayırt edilmek isteniyorsa, objenin üzerine takılır. Daha sonra RFTS okuyucusu ile, mikroçip takılan obje radyo dalgaları vasıtasıyla iletişim kurabilir. Dünya üzerinde, takip edilmek ve iletişim kurulmak istenen objelere takıldıktan sonra, okuyucular vasıtasıyla bu objeler gerçek zamanlı olarak ve lokasyondan bağımsız olarak takip edilebilir, tanımlanabilir. Bu sebeple IoT gücünü RFTS sisteminden almaktadır. 2.3 ifadesinde görülebileceği üzere, RFTS mimarisi uygulama, okuyucu ve aktarıcı olmak üzere üç ana parçadan oluşmaktadır [28].

Aktarıcı, tanımlanmak veya sayılmak istenen objelerin üzerine eklenir. Aktif veya pasif olabilir. Aktif aktarıcılar batarya içerirken, pasif olanlarında böyle bir ihtiyaç yoktur. Aktif aktarıcılar kendi aralarında haberleşebilecekleri gibi, okuyucularla iletişim başlatma özelliğine sahiptirler. Pasif aktarıcılar ise okuyucu tarafından işlevsel hale getirilirler.

Okuyucu, radyo frekansı arayüz modülünden ve kontrol biriminden oluşmaktadır. Görevi, aktarıcıları aktif hale getirmek, aktarıcılarla iletişime geçmek, uygulama ve aktarıcılar arasında veri aktarımı yapmaktır.

Uygulama, aynı zamanda veri işleme sistemi olarak adlandırılmaktadır ve veritabanı içerebilir. Görevi, tüm okuyucu ve aktarıcıları çalışır hale getirmektir. Görülebileceği üzere, RFTS oldukça esnek ve güvenilir bir iletişim yöntemi sunmaktadır.

[4] https://github.com/Waikato/moa/blob/master/moa/.project
[5] http://www.cs.cmu.edu/ pegasus/



Şekil 2.3: Radyo Frekansı Tanımlama Sistemi

2.3.7 Kablosuz Sensör Ağları(Wireless Sensor Networks)

Günümüzde bir ortamı akıllı hale getirebilmek için, bu ortamın gerçek dünya verisinden haberdar olması gerekmektedir. Akıllı bir ortam, kullanıcıya anlamlı bir çıktı üretilmek için oldukça fazla ve çeşitli veriye ihtiyaç duymaktadır. Veri toplama işi de sensörler kullanılarak yapılmaktadır. Toplanan verinin oldukça fazla ve çeşitli olması nedeniyle, veri toplama işinin kablolu sensör sistemleri tarafından yapılması imkansıza yakındır. Bu noktada devreye kablosuz sensör ağları girmektedir ve ihtiyaç duyulan hızlı ve kurulumu kolay altyapıyı sağlamaktadır [29]. KSA standartları güç tüketimlerine göre belirlenmiş olup, bu standartlara örnek olarak IEEE 802.15.4, IEEE 802.15.3, ZigBee, WirelessHART, IETF ve 6LoWPAN verilebilir [5][30].

KSA'nın karmaşık ağ ve veri yapısını daha anlaşılır hale getirebilmek için, çeşitli semantik yaklaşımlar ortaya atılmıştır. Bu sebeple semantik sensör ağları(SSA) ortaya çıkmıştır. Sensörlerin çıktılarının semantik bir biçimde ifade edilmesiyle, verinin yönetimi, sorgulanması, veri üzerinde arama yapılması, veri üzerinden çeşitli çıkarımlar yapılarak yeni bilgilerin elde edilmesi kolaylaşmıştır. SSA ile, sensörleri fonksiyonlarına, çıktılarına ve sağladıkları alan bilgisine göre sınıflandırmak ve yönetmek kolaylaşmıştır [5].

W3C Semantik Sensör İnkübator grubu, sensörleri ve bu sensörlerin birbirleri arasındaki ilişkiyi açıklayabilmek için SSN ontolojisini oluşturmuşlardır. Ontoloji, bir alanı(domaini) açıklamak için kullanılan bir kelimedir. SSN ontolojisi ile bir sistemdeki tüm sensörlerin birbirleriyşe ilişkisi, bu sensörlerin ölçümleri, ölçüm yöntemleri, menzilleri ve bu menzillerdeki performansları açıklanabilir. Bunlara ek olarak, sistemde kullanılan ölçüm birimleri, sensör lokasyonları, tipleri, özellikleri sistemden çıkartılabilir. SSN ontolojisi, ilgili sistemin ihtiyacına göre kısmen kullanılabilir [31].

2.3.8 Bilgi Merkezli Ağ(Information Centric Networking)

Bilgi Merkezli Ağ(BMA), ip adres modelini baz alan internet mimarisine göre farklı bir mimari ortaya koymaktadır. BMA yaklaşımına göre, veri sunucu lokasyonundan ve uygulamadan bağımsızdır. Veriye ulaşılmak istendiği zaman, benzersiz(unique) anahtar kelimelerle ulaşılabilir. BMA, IoT ile yapısı gereği oldukça uyumludur, IoT verilerine BMA anahtar kelimeleriyle, standart ip istek cevap mekanızması kullanılmadan ulaşılabilir. BMA'nın diğer avantajları da enerji tasarruflu, mobil, güvenliği yüksek ve ölçeklenebilir olmasıdır [32][5]. Bunlar IoT'un ihtiyaç duyduğu gereksinimlerle örtüşmektedir. BMA henüz geliştirilme aşamasındadır ve geleceğin internet mimarisi olarak kabul edilmektedir.

2.3.9 Bağlam Farkındalığı(Context Awareness)

Bağlam, bir varlığın(entity) durumunu açıklayabilmek için kullanabilecek herhangi bir bilgi olarak açıklanabilir. Bu noktada varlık; insan, yer veya bir obje olabilir [33]. Eğer herhangi bir bilgi, bir durumu tanımlayabilmek için kullanılabiliyorsa, bu bilgi bağlamdır. Buna örnek olarak, bir kişinin konumu verilebilir. Eğer kişi Türkiye'de yaşıyorsa, bu bilgiden yola çıkarak Türk lirası kullandığı çıkarımı yapılabilir. Bu nedenle konum bu örnekte bir bağlamdır.

Bir sistemin bağlam farkındalığına(BF) sahip olması için, sistemin bir bağlamı kullanarak bir varlığa, varlıkla ilgili bir bilgi sunabilmesi gerekir. BF'ye sahip uygulamalar, **kim, nerede, ne zaman, ne yapıyor** sorularıyla varlığı değerlendirir ve bu değerlendirmeler ışığında, varlığın içinde bulunduğu durumun neden gerçekleştiğini anlamaya çalışır. Sistemin sorduğu soruların cevabı için baz alabileceği temel bağlamlar konum, kimlik, zaman ve aktivitedir. Bu bağlamlar doğrudan sorulara cevap üretilmesini sağladığından, temel bağlamlar olarak adlandırılırlar [33]. IoT sisteminde, kullanıcının o anki durumuna göre bir çıktı üretilip bunun kullanıcıya sunulması hedeflendiğinden, bir IoT sisteminin BF'ye sahip olması gerekmektedir. BF'ye örnek olarak, hastalığı için düzenli ilaç alması gereken bir insanın, ilacı azaldığı zaman, ilacın bulunduğu nesnenin bunun azlığını ve bu azlığın ilacı kullanan kişiyle ilişkisini algılaması ve buna göre kullanıcıyı uyarması verilebilir.

2.3.10 Büyük Veri Analizi(Big Data Analytics)

Büyük Veri Analizi(BVA), günümüzde veri depolama ve veri işleme teknolojileri geliştikçe daha fazla ön plana çıkmaya başlamaktadır. Büyük verileri efektif olarak iş-

leme gerekliliğinin sonucunda Map Reduce[34] ve Hadoop[35] gibi teknolojiler ortaya çıkmıştır. Bu teknolojiler bilişim alanında ortaya çıkmasına rağmen, sağlık, enerji gibi farklı alanlarda da büyük veri işleme ihtiyacı gittikçe artmaktadır. Veri boyutunun artması çeşitli zorlukları da beraberinde getirmektedir. Bunlar hacim(volume), hız(velocity) ve çeşitlilik(variety) olmak üzere üç farklı kategoriye ayrılabilir [36];

- ***Hacim:*** Veriyi depolama, işleme ve işlenen veriye kısa sürede erişmeyi ifade etmektedir. Bir veri setinin büyük veri olabilmesi için spesifik bir alt limit olmasa da, yüzlerce terabayt veya daha fazla veri büyük veri olarak kabül edilmektedir. Hadoop ve MapReduce gibi teknolojiler, paralel işleme sayesinde büyük veri setlerini işleme problemini çözebilmektedir. Hadoop özelinde, mevcut teknolojik altyapı veriyi işlemede yetersiz kaldığında sadece yeni bilgisayarlar ekleyerek, işleme süreci hızlandırılabilir. Yeni eklenen bilgisayarlar için ekstra bir programlama yapılmasına gerek yoktur, Hadoop sisteme eklenen yeni kaynakları otomatik olarak ele alabilir.

- ***Hız:*** Verinin boyutu kadar, verinin işleneceği sisteme giriş miktarı da önem taşımaktadır. Gelen veri, minimum gecikmeyle işlenmek zorundadır. NoSQL veri tabanlarının yazma hızı yüksek olduğundan, gelen verinin sistemde tutulması NoSQL altyapısıyla sağlanabilir. Storm[6] ve S4[7] gibi altyapı projeleriyle büyük veri için istenen işleme performansı sağlanabilir.

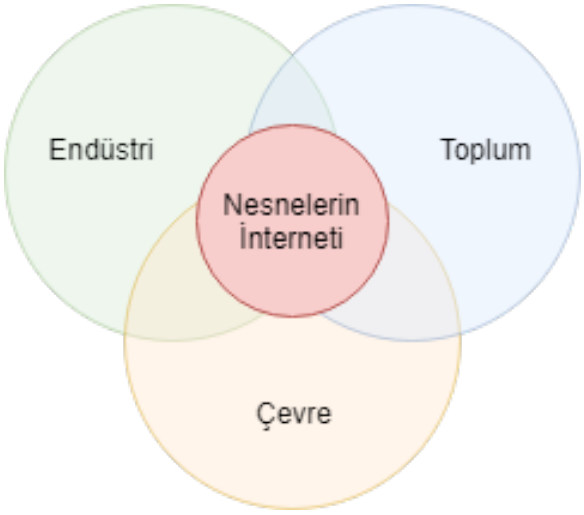
- ***Çeşitlilik:*** İnternet üzerinde birçok erişilebilir büyük veri kaynağı bulunmaktadır. Bunlarla ilgili temel sorun, bu kaynakların farklı yapılarda olması ve bunu kullanmak isteyen kullanıcının, ihtiyacına göre bu verileri işlemesi gerektiğidir. Farklı veri kaynaklarından gelen verilerin, ortak bir paydada buluşturulup kullanılabilmesi büyük veriyle ilgili bir başka zorluktur.

Akıllı şehirler oluşturmak için kurulan sensör altyapıları sebebiyle, üretilen veri miktarı hızla artmaktadır. Bunun sonucunda IoT, işlevsel olabilmek için büyük veri analizinde ortaya çıkan zorlukların çözülmesine doğrudan bağımlıdır.

2.4 IoT'un Potansiyel Uygulama Alanları

İnternet teknolojilerinin yaygınlaşmasıyla, çevredeki her nesneyi internete bağlayarak akıllı hale getirmeyi amaçlayan IoT'un da market potansiyeli ciddi bir artış göstermektedir. IoT'un donanımsal ayağı olan sensörlerin 2015 pazar büyüklüğü, 101.9 milyar

[6]http://storm.apache.org/
[7]http://incubator.apache.org/projects/s4.html



Şekil 2.4: Nesnelerin İnterneti Uygulama Alanları

dolar seviyesine ulaşmıştır. Araştırmacılar, 2021 yılına kadar 190.6 milyar dolarlık bir pazar büyüklüğüne ulaşması beklenmektedir [37] [5]. IoT'un ve onu destekleyen teknolojilerin gelişmesiyle birlikte, IoT'u kullanan birçok farklı uygulama alanları ortaya çıkmıştır.

IoT'un birçok uygulama alanı bulunsa da, bunlar 2.4 ifadesinde görülebileceği üzere üç temel kategoriye ayrılabilir [9];

- ***Endüstri:*** Şirketler, kuruluşlar ve diğer kurumlar arasındaki finansal veya ticari işlemlerin yer aldığı faaliyetler bu kategoride incelenebilir. Bunlara örnek olarak lojistik, imalat, servis sektörü, bankacılık verilebilir.
- ***Çevre:*** Çevrenin korunması, gözetlenmesi ve doğal kaynakların daha efektif kullanımını sağlayan faaliyetler bütünüdür. Tarım, hayvancılık, geri dönüşüm, çevre yönetim sistemleri, enerji yönetimi bu kategoride yer alır.
- ***Toplum:*** Toplumun ve şehirlerin gelişimini sağlayan faaliyetler bu kategoriye dahil edilebilir. Şehirde yaşayanlar için verilen hizmetler bu kategoriye örnek olarak verilebilir.

2.4.1 Havacılık Endüstrisi

Bu alanda IoT kullanılarak, ürünlerin güvenliği ve işlevselliği artırılabilir, ek olarak sahte ürünlerin önüne geçilebilir. Havacılık endüstrisinin en önemli problemlerinden biri, onaylanmayan parçaların(Suspected Unapproved Parts-SUP-) kullanılma olasılığıdır. Onaylanmayan parçalar, ilgili parçanın sahip olması gereken güvenlik ve kalite

gereksinimlerini sağlamayan parçalar olarak açıklanabilir. Bu parçaların sebep olacağı kazaları engellemek için, kullanılan materyallerin detaylı analizinin yapılması gerekmektedir. IoT'un temellerinden birini oluşturan RFTS sistemi kullanılarak, üretilen her parça RFTS aktarıcılarıyla donatılabilir. Bir parça herhangi bir hava aracına monte edilmeden önce, bu aktarıcılar okuyucular vasıtasıyla ilgili parçanın durumunu kontrol edip, bu kontrollerden çıkan sonuca göre parçanın uçağa takılıp takılmamasına karar verilebilir. Bu yolla, uçakların güvenliği ciddi oranda artırılabilir [38].

2.4.2 Otomobil Endüstrisi

Günümüzde otomobil, tren, otobüs gibi araçlar gelişmiş sensörlerle donatılmaktadır. Bu sensörler sayesinde aracın çeşitli fonksiyonları izlenebilmekte ve herhangi bir sorun durumunda kullanıcıya önceden uyarı verilmektedir. RFTS kullanılarak araçların içerdikleri parçalar hakkında seri numarası, tipi, ürün kodu ve konum gibi detaylı bilgiler araçlar üzerinde tutulabilmekte ve bu veriye gerçek zamanlı erişim olduğundan, aracın bakımı, hatalı araçların geri çağrılması gibi işlemler daha efektif biçimde yürütülebilmektedir. Yakın gelecekte trafik yönetimi ve araç güvenliği ile ilgili servislerin de doğrudan IoT üzerinden sunulması beklenmektedir [38].

2.4.3 Telekomünikasyon Endüstrisi

IoT ile telekomünikasyon alanındaki birçok farklı teknoloji birleştirilerek kullanılabilir ve bunlar üzerinden yeni servisler sunulabilir. Bunun örneği olarak GSM, NFC(Near Field Communication), Bluetooth, WLAN gibi teknolojiler gösterilebilir. Telefonu aktarıcı(tag) ile entegre edip, farklı uygulamaların aynı SIM kartı paylaşması sağlanabilir. Buna ek olarak NFC vasıtasıyla, birbirine yakın objelerin veri aktarması sağlanabilir. Telefon üzerindeki aktarıcı sayesinde NFC okuyucu olarak kullanılıp, elde ettiği bilgiyi uzaktaki bir sunucuya aktarabilir [38].

2.4.4 Sağlık Endüstrisi

IoT gelecekte sağlık alanında da yaygın olarak kullanıma uygundur. RFTS özelliklerine sahip bir cihaz yardımıyla, hastanın durumu takip edilebilir ve uygun ilaç hastaya iletilebilir. Bunun avantajı, hasta anlık olarak takip edileceğinden, herhangi bir geç müdahale durumunda ortaya çıkacak sıkıntıların en aza indirilecek olmasıdır. Vücuda yerleştirilebilen cihazlar sayesinde, hastadan elde edilen bilgiler doğrultusunda, acil durumlarda anlık müdahalelerle hastanın yaşamı kurtarılabilir. Diyabet, kanser, kalp

hastalıkları, kriz gibi durumlarda anlık müdahale oldukça önemlidir [38].

2.4.5 Bağımsız Yaşam

Günümüzde, özellikle şehirleşmenin getirdiği etkilerden bir tanesi, insanların çeşitli rutinlere sahip olmasıdır. Bunlara örnek olarak, çalışan bir yetişkinin her sabah işe belli bir saatte gidip, belli bir saatte geri dönmesi, yemeğini yemesi ve çeşitli aktivitelerde bulunması olarak gösterilebilir. Çevrede bulunan akıllı cihazlar vasıtasıyla, her bir bireyin günlük rutini takip edilip, herhangi anomali, hastalık, kaza olması durumunda gerekli birimlere otomatik uyarı gidebilir. Bu özellikle belli bir yaşın üzerindeki insanlarda, beklenmedik ölümlerin önüne geçilmesi yönünde oldukça yararlı olacaktır [38].

2.4.6 Eczacılık

Eczacılık endüstrisinde ilaçların güvenilir ve kullanılabilir durumda olması oldukça önemlidir. İlaçlara yerleştirilecek akıllı etiketler sayesinde, ilaçlar üretiminden son kullanıcıya ulaşana kadar gerçek zamanlı olarak takip edilebilir. Böylelikle, eğer herhangi bir ilacın son kullanma tarihi geçtiyse veya ilacın prospektüsünde yazana aykırı bir koşulda saklanıyorsa, bunun önüne geçilebilir. Sahte ilaçlar eczacılık alanında ciddi bir sorun oluşturmaktadır [39]. IoT teknolojilerinin entegrasyonu sayesinde, aynı havacılıkta olduğu gibi sahte malzeme içeren ilaçların da önüne geçilebilir. Bunlara ek olarak, akıllı etiketler sayesinde, ilacı kullanan kişiye son kullanma tarihi, alınması gereken doz ve ne zaman alınacağı gibi önemli bilgiler verilebilir [38].

2.4.7 Perakende, Lojistik ve Tedarik Zinciri Yönetimi

IoT, tedarik zincirinin yönetiminde birçok avantaj sağlayabilir. RFTS ile donaltılmış ekipmanlar ve akıllı raflar ile, ürünlerin gerçek zamanlı takibi yapılıp çeşitli optimizasyonlar geliştirilebilir [40]. Ürünlerin ne zaman stoğa girdiği, stokların gerçek zamanlı takibi, hırsızlık varsa bunun anlaşılması gibi durumlarda oldukça işlevsel olabilir. IoT teknolojilerinin entegrasyonu sayesinde, perakende mağazada ciddi bir kar artışı sağlanabilir. Araştırmalara göre, satışların %3.9'u, stoklar tükendikten sonra aynı ürünü talep eden ancak ürün olmadığından alamayan tüketicilerden kaynaklanmaktır [41]. Bunlara ek olarak, IoT ile mağazalardan toplanan bilgiler işlenerek, tedarik zincirinde çeşitli düzenlemeler yapılabilir. Eğer üreticiler hangi mağaza ne kadar satmış, hangi ürün daha popüler gibi verilere erişim sağlarsa, buna göre ellerindeki stokları daha

doğru lokasyona ve daha isabetli miktarlarda yönlendirebilirler. Her bir lokasyonun ve tedarik zincirinin karbon üretimi takip edilerek, çevreyi daha çok kirleten süreçler tespit edilebilir ve buna göre bir önlem alınabilir. Ayrıca, IoT sayesinde, biometrik veri ile daha hızlı ödeme, alınan ürünün kullanıcıyla uyumu, kullanıcının alışkanlıklarına göre ürün önerme yapılabilir [38].

2.4.8 İmalat Endüstrisi

Üretim sürecindeki araçların ve ürünlerin takibi, araçları akıllı etiketlerle donatarak yapılabilir. Böylelikle tüm üretim bandıyla alakalı veriler elde edilip, üretimin başlangıcından bitişine kadar tüm üretim döngüsü takip edilebilir. Bu veriler ışığında düzenlemeler yapılıp, daha rafine bir üretim süreci tasarlanabilir [38].

2.4.9 Süreç Endüstrisi

Petrol ve gaz endüstrisinin birçok fabrikasında, operasyonların yönetilmesi, konteynerlerin izlenmesi, kullanılan ekipmanların takip edilmesi, IoT altyapısıyla harmanlanmış sensörlerin kullanılmasıyla yapılmaktadır. Bu fabrikalarda üretilen ürün doğası gereği tehlike içerdiğinden, en küçük bir ihmal durumunda ciddi kazalar olabilmektedir. İngiltere'de kimyasal ve petrokimyasal sektörlerde yapılan araştırmalar sonucunda, meydana gelen felaketlerin, sürece hakim olmamak, depolama, işleme ve kimyasal süreçlerin iyi yönetilmemesi gibi sebeplerden kaynaklandığı görülmektedir [42]. IoT kullanılarak, tehlikeli maddeler içeren konteynerlar kablosuz sensör teknolojisiyle izlenebilir ve ortaya çıkabilecek olası bir tehlikeli duruma önceden müdahale edilebilir [38].

2.4.10 Ulaşım Endüstrisi

IoT, yolcuların eşyalarını ve dünya çapında yapılan kargoculuk işlemlerini takip etmek için kullanılabilir. Daha güvenli bir taşıma politikasının geliştirilmesine katkıda bulunabilir. Bunlara ek olarak, trafiğin tıkandığı noktalarda, bu tıkanıklık yolcuların telefonundan takip edilebilir ve daha efektif bir trafik akışı elde edilebilir. Konteynerlerin kendilerini tarayarak ağırlıklarını otomatik olarak ölçmesi sayesinde, ulaşım şirketleri daha verimli bir paketleme süreci yürütebilirler. Ayrıca, havalimanlarında yolcuların bagajlarının otomatik olarak takibi ve sıralanması yapılarak, daha güvenli ve efektif bir sistem kurulabilir [38].

2.4.11 Çevre

IoT teknolojisi çevre izleme uygulamalarına rahatlıkla entegre edilebilir. Gerçek zamanlı veri izleme ve veriyi üreten cihazların birbirleriyle haberleşebilmesi sayesinde, insan veya hayvan hayatına tehdit oluşturan anomaliler tespit edilip izlenebilir. Bir diğer kullanım senaryosu ise; volkanik alanlar, okyanus veya insan hayatına elverişli olmayan alanlarda kurulacak bir sensör alt yapısıyla, bu bölgelerde izleme yapılabilir ve olası tehlikelerde(volkan patlaması gibi) önceden uyarı veren bir sistem geliştirilebilir. IoT, çevre güvenliğinin sağlanmasında da oldukça önemli bir rol oynayabilir. Ormanlık alanlara kurulacak IoT sistemiyle, yangından çok daha önce, sensörler vasıtasıyla bölgedeki anomaliler tespit edilip, yangın önceden önlenebilir. IoT bu alanlarla sınırlı olmayıp, deprem, tsunami ve diğer doğal afetlerde de bölgeye gelen kurtarma ekiplerinin koordineli çalışmasında önemli bir rol oynayabilir [17].

2.4.12 Güvenlik ve Gözetleme

Günümüzde güvenlik, şirketlerde, alışveriş merkezlerinde, fabrikalarda, araba park yerlerinde ve birçok halka açık alanda oldukça önemli bir konuma gelmektedir. Güvenlik her ne kadar önemli de olsa, yüksek derecede güvenlik insanların yaşam alanlarını sınırlayabilir ve özel hayata fazlaca müdahale edebilir. IoT ile, insanların hayatını zorlaştırmadan, daha görünmez ama mevcut güvenlik sistemleri kadar etkili güvenlik altyapıları kurulabilir. Çevredeki değişimleri algılayabilen sensörlerle, bir alan içindeki tehlikeye yol açabilecek kimyasallar tespit edilebilir. İnsanların davranışını analiz eden sensörlerle, şüpheli biçimde davranan kişiler tespit edilebilir. Ayrıca, IoT sistemi kameralara göre daha ucuz bir alternatif olacağından, geniş çaplı kamera sistemlerinin kurulması gerekliliği ortadan kalkabilir. Bunlara ek olarak, kamera sistemleri azalacağından, insanlar arasında daha fazla kabul gören bir teknoloji olacaktır [17].

2.4.13 Akıllı Ev/Akıllı Bina

Evlerin ve binaların IoT teknolojileriyle entegre edilmesi sayesinde, gereksiz elektrik ve su kullanımları azalabilir, ısınma daha efektif biçimde sağlanabilir. Bunun en bariz avantajı, hane başına düşen enerji tüketiminin azalması ve bu vasıtayla karbon salınımı düşürülerek çevreye daha az zarar verilmesidir. IoT teknolojilerinin binalara entegre edilmesindeki en önemli parça sensörlerdir. Bina içine yerleştirilecek sensörlerle, binanın kaynak tüketimi ve binadaki mevcut kullanıcıların kaynağa ihtiyacı tespit edilebilir. Bu veriler ışığında yeteri kadar kaynak ilgili kullanıcıya aktarılabilir [17].

2.4.14 Akıllı Şehir

IoT'un akıllı şehirlerde kullanılması, oldukça gelişmiş iletişim yöntemlerinin şehir altyapısına entegre edilmesi anlamına gelmektedir. Günümüzde bir şehrin en önemli sorunlarından biri trafiktir. IoT altyapısını efektif olarak kullanan bir şehirde, her bir aracın gideceği lokasyona özel olarak ve trafikteki diğer araçların da lokasyonları gözetilerek en verimli rota oluşturulabilir. Park yeri sorunu yaşanmaması için, sürücü boş park alanlarına yönlendirilebilir. Bu senaryoda, araçlar da bir nesne olarak IoT ekosisteminde yer almaktadır. Bunlara ek olarak, havadaki kirlilik oranı sensörler vasıtasıyla tespit edilerek, bu veriler bir aksiyon alınabilmesi için ilgili kurumlara iletilebilir. Ayrıca sensörler ile trafik kurallarını ihlal eden araçlar hakkında bilgi otomatik olarak güvenlik güçlerine iletilip, ihlalı yapan kişi tespit edilebilir [17].

2.4.15 Hayvancılık

Tarımcılık alanında kullanılan hayvanların takibi ve düzenlemesi IoT teknolojilerinin kullanımını zorunlu kılmaktadır. IoT teknolojileri kullanılarak, hayvanlar arasında çıkan salgın hastalıklar önceden tespit edilip önlemler alınabilir ve hayvanların sağlık durumları(aşılı olup olmadıkları gibi) daha efektif biçimde takip edilebilir. Bazı ülkelerde hayvancılık için çeşitli destekler verilmektedir, ancak bu destekler istismara açık durumdadır. Hayvan sayısı mevcut durumdan fazla gösterilerek çeşitli sahtekarlıklar yapılabilir. Bunun önüne geçmek için yine IoT'dan faydalanılabilir, hayvanların sayımı daha doğru yapılabilir [38].

2.4.16 Eğlence Endüstrisi

IoT teknolojilerinin yaygınlaşması ile, kullanıcıların bulunduğu yerlere göre anlık haber alınması mümkün olacaktır. Bu toplanan veriler ışığında, eğer kullanıcının bulunduğu ortamda multimedya özelliğine sahip bir aygıt varsa, bu aygıttan kullanıcının zevkine özel etkinlik gösterilebilir. Eğer kullanıcı gösterilen içerikle ilgilenirse, kullanıcı bir URI adresine yönlendirilip, daha fazla bilgi alması sağlanabilir [38].

2.4.17 Sigorta Endüstrisi

Araba sigortalarında eğer müşteri arabasına, arabanın hızlanmasını, o anki hızını vb. parametreleri kaydeden bir elektronik kaydedici takmaya ikna olursa, sigorta masrafları daha ucuz olmaktadır [43]. Bu aynı zamanda sigorta şirketine de avantaj sağlamak-

tadır. Müşterilerin araçlarından alınan veriler sayesinde, şirket kazadan önce en uygun aksiyonları alıp, daha fazla tasarruf sağlayabilir [38].

2.4.18 Geri Dönüşüm

IoT teknolojileri kullanılarak, araçların karbon salınımı takip edilebilir, kullanılan teknolojik aletlerin parça parça takibi yapılabilir ve bunların vakti geldiğinde geri dönüşümü sağlanabilir. Böylelikle, elektronik atık oluşturulmasının önüne geçilebilir. RFTS teknolojisi ile bilgisayar, telefon ve diğer elektronik cihazların her bir alt bileşenlerinin takibi yapılabilir [38].

2.5 IoT Araştırma Alanları

IoT, hızlı gelişen bir alan olmakla beraber, gelişimi önünde çeşitli engeller bulunmaktadır. IoT'un daha fazla yaygınlaşabilmesi ve efektif çalışabilmesi için çeşitli alanlarda yeni teknolojik gelişmelere ihtiyacı bulunmaktadır. Bu alanlara örnek olarak kimlik saptama, mimari, iletişim, ağ, yazılım, arama motoru, donanım, ağ yönetimi, enerji, güvenlik ve standardizasyon verilebilir [9]. Bu bölümde IoT'un ihtiyaç duyduğu alanlar kısaca açıklanmıştır.

2.5.1 Kimlik Saptama Teknolojileri

Bir IoT sistemi içinde milyonlarca nesne barındırabilir. Bu nesnelerin her birinin benzersiz bir kimliği olması gerekmektedir. Bu kimliğe göre çeşitli kimlik doğrulama mekanizmaları çalıştırılabilir. Bu nedenle, milyonlarca nesnenin olduğu bir alanda, kimlik yönetimi yapabilecek teknolojilerin ve altyapıların geliştirilmesi gerekmektedir. IoT platformları tasarlanırken, içlerinde bulundurdukları nesnelerin kimliklerini doğru yönetmek zorundadır [9].

2.5.2 IoT Mimari Teknolojileri

IoT sistemleri, insan, servis, yazılım, akıllı objeler gibi heterojen sistemler arasındaki birlikte çalışabilirliği en üst düzeye çıkarabilmek için açık bir mimariye sahip olmalıdır. Tasarlanan mimari, sınırları belli olan data modelleri, arayüzleri, mevcut protokolleri ve günümüzde yaygın olarak kullanılan web servisi, xml gibi teknolojileri kapsayacak biçimde geliştirilmelidir. Böylelikle günümüzde kullanılan işletim sistemleri ve programlama dilleriyle daha kolay entegre edilebilir ve daha yaygın kullanı-

labilir. Bunlara ek olarak mimarinin katmanlarının sınırları belli olmalıdır. Böylelikle herhangi bir kullanıcı, spesifik bir IoT çözümüne bağlı kalmadan mimariyi kendi sistemine rahatça entegre edilebilir.

IoT sistemi yapısı gereyi büyük veri ile çalışacağından, IoT mimarisinin bu veriyi filtreleyecek, veri üzerinde çıkarım yapabilecek, semantik arama yapılmasına imkan sağlayacak, makine öğrenmesi altyapısı sunabilecek ve veriyi merkezi işlemek yerine, dağıtık olarak işleyecek kabiliyette olması gerekmektedir. Verinin bulut ortamında saklanabilmesi ve bağlantının olmadığı durumlar da sistemin belli ölçüde operasyona devam edebilmesi, hızlı bir sorgu sistemi için veriyi hafızada saklayabilme ve asenkron çalışabilme kabiliyeti de iyi tasarlanmış bir mimarinin içinde olmalıdır [9].

2.5.3 İletişim Teknolojileri

Günümüzde milyonlarca aygıt, mevcut internet altyapısını ve teknolojilerini kullanarak internete bağlı olarak çalışmaktadır. IoT sistemlerinin gelişmesiyle bu sayı ciddi oranda artacağından, internet alanında yeni teknolojik araştırmaların yapılması ve mevcut protokollerin, mimarilerin, aygıtların IoT altyapısını kaldıracak biçimde geliştirilmesi gerekmektedir. Aşağıda IoT'un ihtiyaç duyduğu olası iletişim yöntemleri belirtilmiştir;

- Nesnelerin birbirleriyle ve internetle iletişimi
- Sensörlerin veri toplamak ve verileri aktarmak için ihtiyaç duyacağı iletişim
- Dağıtık veri saklama sistemleriyle iletişim
- IoT sisteminin bulunduğu ortamdaki canlılarla kurduğu iletişim
- Veri madenciliği sonucunda elde edilen verilerin işlenebilmesi için ilgili servislerle kurulacak iletişim
- Objelerin mevcut konumunu algılayabilmek için kullanılacak iletişim

Yukarıdaki maddelerde de görüleceği üzere, IoT sistemleri yaygınlaştıkça mevcut iletişim altyapısı üzerindeki yük de ciddi oranda artacaktır. Bu yükün altından kalkabilmek için internet teknolojilerinde de ciddi bir yatırım gerekmektedir. Bunlara ek olarak, geliştirilecek altyapının aynı zamanda verimli çalışması, yeni ve sistemdeki yük arttığı zaman kullanılamayacak duruma gelmeyen, yüksek performanslı algoritmaların ve protokollerin geliştirilmesi gerekmektedir [9].

2.5.4 Ağ Teknolojileri

IoT sistemleri yaygınlaştıkça, bu sistemlerin işlevselliğinin de korunabilmesi için, sistemin içinde bulunan sensör ağları gibi bileşenlerin de gelişmesi gerekmektedir. Bu gelişimin odak noktası olarak, olabildiğince verimli, kolay kurulabilir ve kullanılabilir, düşük maliyetli olması gösterilebilir. Bu kriterler göz önüne alındığında, kablosuz sensör ağları düşük güç tüketimi ve düşük izleme maliyetleriyle ön plana çıkmaktadır. Kablosuz sensör ağlarının bir diğer avantajı ise, ölçeklenebilir olmasıdır. Kablolamanın getirdiği kurulum karmaşıklığı ortadan kalktığı için, ihtiyaç halinde yeni sensör ağları rahatlıkla sisteme entegre edilebilir. Bunlara ek olarak, IoT için geliştirilecek ağ teknolojilerinde anonim veri akışı, IP(Internet Protocol)'den daha gelişmiş teknolojilerin araştırılması, kimlik ve şifreleme sistemlerinin IoT ile uygun olacak biçimde geliştirilmesi ihtiyaç duyulan araştırma alanlarıdır [9].

2.5.5 Yazılım, Servis ve Algoritma Gelişimi

IoT'un yaygınlaşabilmesi için teknolojik yatırımların yanı sıra, yazılımsal ve algoritmik anlamda da gelişim gerekmektedir. Yazılım, verilerin toplanması, işlenmesi, çıkarım yapılması, kullanıcıya bir arayüz vasıtasıyla sunulması, iletişim altyapılarının verimli biçimde çalışabilmesi, artan yükün otomatik olarak dağıtılması, verilerin efektif biçimde saklanabilmesi gibi, IoT mimarisinin her katmanında kendine yer bulmaktadır.

Yazılıma ek olarak, sistemin sunduğu bir işlevin karmaşıklığını ve detaylarını kullanıcıya yansıtmadan işlevi sunabilen servisler de önemlidir. Sistem, çeşitli servisler vasıtasıyla, bunları harmanlayarak, kullanıcıyla sistem arasında bir soyutlama katmanı oluşturularak verimli biçimde kullanılabilir. IoT sistemlerinin de karmaşık ve heterojen bir yapıda olduğu ve zamanla bu yapının daha da karmaşıklaştığı düşünülürse, sistemin işlevselliğini devam ettirebilmek için sağlanan servislerin de basit ve ilgili fonksiyonu sorunsuz biçimde sunabilmesi gerekmektedir. Yazılım alanında IoT'la ilgili araştırma alanları şu şekilde özetlenebilir [9];

- Sistemle ilgili yeni servislerin yazılması, farklı servislerin birleştirilerek kullanıcıya sunulabilmesi
- Semantik olarak sistemlerin anlaşabilmesi, ortak dilden konuşabilmesi
- Veri paylaşımı ve verinin verimli biçimde iletimini, işlenmesini sağlayacak algoritmaların geliştirilmesi

- İnsan makine etkileşiminin yapılabilmesi için gerekli yazılım altyapısının oluşturulması
- Sisteme yeni bir bileşen eklendiğinde, sistemin artan karmaşıklığı kendi içinde yönetebilecek bir yönetim sistemine sahip olması
- Yeni eklenen bileşenlere göre otomatik konfigürasyon ve optimizasyon yapılabilmesi
- Verimli ve IoT sistem bileşenlerinde çalışabilecek seviyede bir işletim sistemi yazılması
- Sistem içinde düzgün çalışmayan bileşenlerin otomatik tespitinin yapılıp, bunların verebileceği zarara göre otomatik olarak çözümler üretilebilmesi
- Sistemde bulunan kaynakların, ihtiyaca göre otomatik biçimde dağıtılabilmesi için yeni ve efektif algoritmaların geliştirilmesi
- Sistemdeki bileşenlerin yönetimi ve veri madenciliği için yeni matematiksel modellerin ve algoritmaların üretilmesi

2.5.6 Donanımsal Teknolojiler

Yazılımlar ne kadar verimli olursa olsun, bu yazılımları çalıştıracak donanımlar zayıfsa veya yüksek güç tüketiyorsa, yazılımın potansiyeli kullanılamaz duruma gelebilir. Bu nedenle, IoT mimarisi geliştirilirken, IoT'u oluşturan donanımların yeterince güçlü ve enerji tüketimi düşük olması bir gerekliliktir. Donanım aynı zamanda, üstünde çalıştırılan IoT uygulamasına göre kendini otomatik olarak entegre edebilmeli, uygulama bazında farklı iletişim ve protokolleri destekleyecek biçimde çalışabilmelidir. Donanım alanında IoT sistemlerinin efektif olabilmesi için nanoteknoloji, sensör teknolojileri, daha efektif çalışan mikro çipler ve iletişim aygıtları, nanoelektronik, düşük maliyetli ve yüksek verimli güvenlik aygıtları, düşük maliyetli üretim teknikleri alanlarında geliştirmeler yapılması gerekmektedir [9].

2.5.7 Keşif ve Arama Motoru Teknolojileri

Bir IoT sistemi birçok dağıtık sensör ve kaynak bulundurmaktadır. Bu kaynakların efektif biçimde taranabilmesi ve bulunabilmesi için arama motorunun yüksek kabiliyetli olması gerekmektedir. Arama motoru sadece insanlar tarafından değil, aynı zamanda sistemdeki nesneler tarafından da yeni servisleri keşfetmek ve bilgi toplamak

için kullanılacaktır. Bu nedenle nesne-nesne ve nesne-insan iletişimde kritik bir rol oynamaktadırlar. Arama motorunun verimli olabilmesi için, verilerin semantik olarak ifade edilebilmesi ve arama moturunun bu semantik veriyi insan müdahalesi olmadan yorumlayabilmesi gerekmektedir. IoT'un ihtiyaç duyduğu düzeyde bir arama motoru üretilmek için; aygıt ve dağıtık veri kümelerinin efektif biçimde keşfedilebilmesi, lokalizasyon, verilerin semantik olarak aranabilmesi, dünya çapında ortak olarak kullanılabilecek bir kimlik doğruma mekanizması kurulması gerekmektedir [9].

2.5.8 Ağ Yönetim Teknolojileri

IoT sistemi içindeki alt sistemlerin yönetiminin efektif yapılabilmesi için, bu sistemleri izleyen ve veri akışındaki bir tıkanıklık durumunu yönetip, veriyi alternatif rotalardan iletebilecek bir yönetim sistemi gerekmektedir. Bu yönetim sistemi, üzerinde çalıştığı IoT sisteminden bağımsız olarak, çalışan prosesleri aygıttan ve protokollerden bağımsız biçimde otomatik olarak tanıyabilmeli, bunları izleyebilmeli, sorunları ayırt edip bunlar için çözüm yöntemlerini kısa sürede sunabilmelidir [9].

2.5.9 Güç ve Enerji Depolama Teknolojileri

IoT sistemindeki bir nesne, kablosuz iletişim, çalışması için gerekli elektronik bileşenlerden oluştuğundan belli bir miktar enerji tüketmektedir. RFTS gibi teknolojiler harici bir güç kaynağına ihtiyaç duymasa da, nesnenin izlenebilmesi, aktif iletişim kurulabilmesi, nesnenin durumunun gözlenebilmesi için ek enerji ihtiyacı bulunmaktadır. Bu nesnenin içinde depolayabilmesi ve depoladığı enerjiyi uzun süre kullanabilmesi gerekmektedir. Nesnelerin boyutu konusunda bir standart olmadığından, günümüzde en büyük problem, her nesneye güç verebilecek ve uzun süre dayanabilecek batarya teknolojilerinin geliştirilmesidir. IoT'un efektif olarak uzun süre müdahale olmadan çalışabilmesi için mikro enerji alanında çeşitli çalışmalar yapılmaktadır. Bu alan mikro batarya, mikro yakıt hücreleri, termoelektrik sistem ve mikro soğutucular, güneş pili, enerji depolamak için materyal çalışmaları, enerji saklama şemaları gibi konuları kapsamaktadır [9].

2.5.10 Güvenlik ve Gizlilik Teknolojileri

IoT ekosistemine dahil olan nesnelerin sürekli bilgi üretmesi ve diğer nesnelerle iletişim halinde olması çeşitli güvenlik endişelerini ortaya çıkarmaktadır. Paylaşılmaması istenmeyen bir verinin nesne tarafından işlenmemesi, verinin bütünlüğü, güvenli ak-

tarımı gibi sorunlar, IoT'un yaygınlaşması ve verimli kullanılması için çözülmek zorundadır. Verinin izinsiz kullanımının önüne geçmek ve milyonlarca ve birbirinden farklı iletişim protokolleri kullanma ihtimali olan nesnelerin, güvenlik sorunları ortaya çıkarmamaları için çeşitli araştırma alanlarında çalışılması gerekmektedir [9];

- Şifreleme ve veri koruma algoritmalarının, her nesnede ve düşük güç tüketimiyle çalışabilecek yapıda geliştirilmesi
- Merkezi olmayan bir kimlik doğrulama sistemi için yeni modeller geliştirilmesi
- Kişilerin özel hayatını ihlal etmemek ve özel bilgileri paylaşmamak için nesne kullanımına uygun yeni teknolojiler geliştirilmesi
- Veri erişim kurallarının tanımlanması, nesnelerin erişeceği verilerin yönetilebilmesi
- Nesnelerin verileri otomatik olarak sınıflandırıp, gizlilik ihlali yapabilecek verileri paylaşmayacak şekilde programlanabilmesi

2.5.11 Standardizasyon

IoT birçok farklı veri kaynağı arasında iletişimi sağlayacaktır. Bu veri kaynakları farklı formatlarda veri üretiyor olabilir. Bu verilerin aynı arayüz üzerinden ve ortak bir dille konuşması gerekmektedir. Bu noktada devreme semantik standartlar girmektedir. Farklı veri kaynaklarını ortak bir semantik dil vasıtasıyla aynı noktaya getirip, IoT sistemi üzerindeki her nesnenin aynı dili kullanması sağlanmalıdır. Dolayısıyla bu alanda üzerinde çalışılması gereken nokta, veri üretiminin ve bu verinin izlenmesinin, okunmasının, yorumlanmasının ortak bir dille olacak şekilde yapılmasıdır, Kısacası, IoT sistemlerinin iletişim mekanizmasının ortak bir noktada buluşturulması bu çalışmaların temel amacıdır [9].

2.6 IoT Platformları

Günümüzde birçok IoT platformu aktif olarak kullanılmaktadır. Bu bölümde [5] ve [44] makalelerinden yararlanılarak, çeşitli platformlar 2.1 tablosunda gösterilmektedir. Tezde sunulan platforma yakın olması için, açık kaynak, cihaz ve veri yönetimi, REST mimarisini destekleyip desteklemediği tabloda öncelikli olarak sunulmuştur. Platformların tamamına yakını REST desteklemektedir. Platformların yaygın olarak kullanılabilmesi için, yeni teknolojilerin yanısıra, hali hazırda kullanılmakta olan web servis

teknolojilerinin kullanılması önemli bir noktadır. Tablo 2.1'de belirtilen platformlar arasında, Microsoft Azure ve Ibm Watson platformları, diğer platformlara göre daha fazla miktarda depolama, sanal makine kurma, gerçek zamanlı veri izleyip analiz etme gibi ek özellikler de sunmaktadır.

2.1'da sunulan platformlardan Arkessa, hali hazırda M2M(Makineden Makineye) bağlantı konusunda uzmanlaşmış bir platform olup, IoT ekosisteminde bulunan cihazların birbirlerine GSM, GPRS, 3G ve 4G gibi mobil ağlar üzerinden bağlanmasına imkan tanımaktadır. Bunlara ek olarak, mobil ağların olmadığı durumlarda uydu üzerinden de bağlantı yapılmasına imkan tanımaktadır. Platform daha çok profesyonel şirketleri hedeflediğinden dolayı, sunduğu hizmetlerin büyük bir kısmı ücretlidir.

Carriots platformu, akıllı bina, akıllı enerji, akıllı ürün ve akıllı şehir alanları üzerine yoğunlaşmaktadır. Akıllı binada bulunan IoT cihazlarının yönetimini kolaylaştıran ve otomatikleştiren platform, aynı zamanda akıllı enerji ile enerji üreten platformların izlenmesi ve güç yönetimini de yapabilmektedir. Akıllı ürünler ile kullanıcıların tüketim alışkanlıklarına dair bilgileri toplayabilen ve bu veriler üzerinde büyük veri analizi yapabilen Carriots, bulgularla ilgili raporlar üretebilmektedir. Bu veriler ışığında, daha verimli bir pazarlama süreci ve daha isabetli bir ürün grubunun oluşturulmasına imkan sağlamaktadır. Bunlara ek olarak, Carriots platformunu kullanan cihazların uzaktan bakımı yapılabilmektedir. Carriots ücretli bir platform olup, açık kaynak olmadığından üzerinde herhangi bir geliştirme yapılamamaktadır.

Everyware platformu, iletişimi mail, yazışma ve toplantılar üzerinden yönetmeyi amaçlayan bir IoT platformudur. Platform, kullanıcıları sisteme kaydettikten sonra, bu kullanıcıların tek bir merkezden haberleşmesini, toplantı ayarlamasını, çeşitli kampanyalar düzenlemesini ve birbirlerine eğer ödeme yapacaklarsa, bunu yazışma üzerinden yapabilmelerine olanak sağlamaktadır. Platform, API entegrasyonu sunduğundan, başka bir istemci bu API'yi kullanarak platform yönetimini yapabilmektedir veya bu API'ye uygun biçimde yazılım geliştirebilmektedir. Everyware platformu ücretli olup, açık kaynak desteği sunmamaktadır.

Evrythng platformu, gerçek zamanlı olarak farklı kaynaklardan gelen verileri alabilme ve bu veriler üzerinde gerçek zamanlı veri analizi yapabilme özelliğine sahip bir platformdur. Platform aynı zamanda bir kullanıcı arayüzü sunmaktadır. Dinamik veri özelliği sayesinde, kullanılan ürüne özel veri modelleri geliştirilip kullanılabilmektedir. Bunlara ek olarak platform, kayıtlı ürünler üzerinde izin yönetimi sunmakta ve bu ürünlerin API çağrıları ile yönetilebilmesine olanak vermektedir. Platform açık kaynak değildir ve kullanımı ücretlidir.

Microsoft Azure platformu, gerçek zamanlı veri analizi, Azure SQL veri tabanı servisi, sanal makineler, yapay zeka ve makine öğrenmesi servisleri sunan çok amaçlı bir platformdur. Azure IoT özelinde, IoT Hub isimli bir servis sunmaktadır. Bu servis ile milyarlarca cihazı birbirine güvenli bir biçimde bağlamak, durumlarını takip etmek ve açık kaynak yazılım geliştirme kitleri sayesinde geliştirme yapabilmek mümkündür. IoT Central servisi ile, herhangi bir bulut programlama deneyimine sahip olmadan, IoT cihazlarının haberleşmesi ve izlenmesi yapılabilmektedir. Azure platformu ücretlidir.

Exosite platformu, IoT sisteminde bulunan cihazların gerçek zamanlı olarak durumlarının takip edilmesine, bulguların kullanıcıya grafiksel olarak sunulmasına imkan veren bir IoT platformudur. Platform, ilk adım olan veri toplamadan itibaren, bu verilerin iletimi, saklanması, dağıtılması ve ilgili cihazlar tarafından kullanılmasına kadar geçen süreci yönetebilme kapasitesine sahiptir. Platform ayrıca bulut desteği sunduğundan, IoT uygulamalarının kapasite sıkıntısı olmadan büyümesine ve yönetilmesine olanak sağlamaktadır.

GroveStreams platformu, farklı kaynaklardan gelen verileri toplayabilen ve bunları büyük veri bulut sistemlerinde saklama kapasitesine sahiptir. Toplanan verilerin analiz edilmesine ve bu analizin sonuçlarının görselleştirilmesine imkan vermektedir. Açık kaynak ve ücretsiz olarak sunulan platform, ücretli ve daha fazla veri toplama ve işleme kapasitesine sahip çeşitli paketler de sunmaktadır.

Linksmart platformu, tamamen açık kaynak olan ve MQTT, REST gibi standartları destekleyen bir IoT platformudur. Platform, gerçek zamanlı veri akışını dinleyebilme ve bu veriler üzerinde veri madenciliği yapabilme kapasitesine sahiptir. Bunlara ek olarak, veri akışının görselleştirilmesi için Grafana yazılımını kullanmaktadır. Verilerin saklanabilmesi için InfluxDB isimli zaman serisi verilerinin saklanabileceği ve verileri üzerinde analiz yapabilen bir veri tabanı kullanılabilmektedir. Platform ayrıca yazılımların daha kolay yönetilmesini sağlayan Docker[8] ile uyumlu olarak tasarlanmıştır.

OpenIoT platformu, IoT sisteminde bulunan, sensörlerin tipinden bağımsız olarak bunlardan veri almaya olanak sağlayan, çeşitli güvenlik ve bulut hesaplama özellikleri sunan açık kaynak bir IoT platformudur. Platform akıllı kampüs ve akıllı şehir projelerinde kullanılmaktadır.

OpenMTC platformu, farklı alanlarda, farklı teknik altyapılara sahip IoT cihazlarının birlikte sorunsuz biçimde çalışmasını hedefleyen ve makine-makine iletişim standartına uygun olarak geliştirilen bir IoT platformudur. Temel özellikleri açık kaynak ol-

[8] https://www.docker.com/

ması ve IoT cihazlarınınn üreticilerinden bağımsız olarak hepsiyle uyumlu olmasıdır.

OpenSense platformu, dünya üzerindeki çeşitli noktalardaki sensör bilgilerini toplayan ve bunları ücretsiz kullanıma açan bir platformdur. Platform, REST API desteği ile gelmektedir ve Swagger[9] üzerinden sensör verilerine erişim imkanı sağlamaktadır. Sensör verilerine ek olarak, sensörün ölçümünü yaptığı özelliğe göre de sorgu atılabilmektedir. Platform, sensörün sunduğu verinin çeşitli kriterlere göre filtrelenmesine izin vermektedir.

RealTime platformu, kendi içinde barındırdığı modüller vasıtasıyla çeşitli IoT cihazlarının entegrasyonunu sağlayan ve bunların yönetilip, izlenmesine imkan veren bir IoT platformudur. Bu modüller, birbirlerine bağlanmak isteyen cihazlara yerleşik yazılım(embedded software) veya ayrı modül olarak entegre edilebilir. Platform aynı zamanda verinin görselleştirilmesine de imkan sağlamaktadır.

SensorCloud platformu, çeşitli veri kaynaklarından gerçek zamanlı veri akışına destek verdiği gibi, verinin Excel programı üzerinden de yüklenmesine imkan sağlayan bir API sunmaktadır. Ayrıca veriler üzerinde çeşitli alarmlar oluşturulabilmektedir. Bu alarmlar mail veya telefon mesajı yoluyla alınabilmektedir. Platform aynı zamanda sisteme gelen verilerin grafiksel olarak gösterimine imkan sağlamaktadır. Bunlara ek olarak veri filtreleme imkanı da sunmaktadır.

Skyspark platformu, birçok veri giriş yöntemini desteklemektedir. Veri düzenli olarak bir web servisinden veya Excel programı üzerinden yüklenebilir. Ayrıca platform, SQL veri tabanı üzerinden doğrudan veri çekebilir. Platform çektiği verileri saklamak için kendi üzerinde Folio isminde bir veri tabanı kullanmaktadır. Veriyi semantik olarak etiketleme özelliğine sahiptir. Platform ücretli olarak sunulmaktadır.

The thing system platformu, birçok farklı alanda farklı açık kaynak IoT çözümlerine erişilmesine olanak sağlayan ve bunlara REST arayüzü sunan bir platformdur. Platformda bulunan modüller ücretsizdir ve birçok farklı tipte sensörün yönetimine imkan sağlamaktadır.

ThingSquare platformu, IEEE 802.15.4e, TCP/IP, 6lowpan, TLS/SSL gibi birçok protokolü destekleyen ve düzenli olarak platform kullanıcılarını etkilemeden platform üzerinde güncellemelere izin veren bir IoT platformudur. Platform yazılıma ek olarak, güç tüketimi düşük sensörler gibi donanımsal ürünler de sunmaktadır. Platform ücretlidir.

Xively platformu, Google Cloud platformunun bir parçasını oluşturmaktadır. Veriyi

[9]https://swagger.io/

Çizelge 2.1: IoT Platformları

İsim	*Açık Kaynak*	*Cihaz Yönetimi*	*Veri Yönetimi*	*REST*
Arkessa	Evet	Evet	Evet	Hayır
Carriots	Hayır	Evet	Evet	Evet
Everyware	Hayır	Evet	Evet	Evet
EvryThng	Hayır	Evet	Evet	Evet
Microsoft Azure	Hayır	Evet	Evet	Evet
Exosite	Evet	Evet	Evet	Evet
GroveStreams	Evet	Evet	Evet	Evet
LinkSmart	Evet	Hayır	Evet	Hayır
OpenIoT	Evet	Hayır	Evet	Hayır
OpenMTC	Evet	Evet	Evet	Evet
Open.Sen.se	Hayır	Hayır	Evet	Evet
realTime.io	Hayır	Evet	Evet	Evet
SensorCloud	Hayır	Hayır	Evet	Evet
SkySpark	Hayır	Hayır	Evet	Evet
The thing system	Evet	Evet	Hayır	Evet
ThingSpeak	Evet	Evet	Evet	Evet
ThingSquare	Hayır	Evet	Evet	Evet
ThingWorx	Hayır	Evet	Evet	Evet
Xively	Hayır	Evet	Evet	Evet
AllJoyn	Evet	Evet	Evet	Evet
Ibm Watson	Hayır	Evet	Evet	Evet
Ninja Blocks	Evet	Evet	Hayır	Evet
Pentaho	Hayır	Evet	Evet	Evet

ilişkilendirme, saklama ve analiz etme gibi özellikleri bulunan platform, bu özellikleri hem bulut üzerinde hem de IoT cihazlarda gerçekleştirme kapasitesine sahiptir.

AllJoyn platformu, farklı cihazların iletişimi kolaylaştırmayı hedefleyen ve otomotiv, sağlık, akıllı ev ve güvenlik gibi birçok farklı alanda çalışabilen bir IoT platformudur. Platformun tüm kodları açık kaynak olup, yazılımcılar bu platformu kullanarak kendi IoT cihazları için ara katman yazabilirler.

IBM Watson platformu, makine öğrenmesi, dil işleme ve sınıflandırma, konuşmanın yazıya dökülmesi, görsel tanıma, dil çevirici gibi birçok alt dala ayrılan bir platformdur. Platform, cihazların yönetimini REST API üzerinden gerçekleştirmektedir. Bunlara ek olarak, veriler üzerinde çeşitli alarm mekanizmaları ve büyük veri analizine olanak sağlamaktadır.

2.1 IoT platformları tablosunda görülebileceği üzere, platformların çoğu açık kaynak değildir. Açık kaynak olan seçeneklerde ise REST gibi yaygın bir mimari desteği veya veri yönetimi bulunmayan platformlar bulunmaktadır. Bir sonraki bölümde tanıtılacak

olan MISIoT, kapsamlı bir veri yönetimi sistemiyle gelmekte ve kullanıcıların platform üzerinden, platform verisiyle kendilerine ait herhangi bir betiği yönetmesine olanak sağlamaktadır. Ayrıca açık kaynak olan platform, diğer geliştiricilerin de katkı sağlamasına ve olası problemlerin daha hızlı saptanıp, çözüme ulaştırılmasına imkan tanımaktadır.

3. MISIOT PLATFORMU

MISIoT platformu, 3.1 şeklinde görülebileceği üzere çok katmanlı bir kavramsal tasarıma sahiptir. Bu tasarımda veri toplama katmanı(VTK), çıkar çevir yükle(ÇÇY), veri saklama katmanı(VSK), makine öğrenmesi ve analiz(MÖA) ve grafiksel arayüz katmanı(GAK) olmak üzere beş katman bulunmaktadır.

İlk katman olan VTK katmanı, dış dünyadan gelen verinin platforma alınmasından sorumludur. Veri sensör dünyasından doğrudan veya başka bir programlama dilinde yazılmış bir emülatörden gelebilir. MISIoT Verinin geliş formatı olarak JSON'ı desteklemektedir ve gelen kaynaktan bağımsız olarak, JSON geldiği sürece bu veriyi alıp saklayabilir. Bu katman veriyi sadece almaktan ve ÇÇY katmanına iletmekten sorumludur, veriyi işlemek için herhangi bir aksiyon alınmaz. Örneğin bir hava durumu verisi şu şekilde gelebilir;

```
{
    windspeed: 200,
    windgust: 10,
    temperature: 20
}
```

İkinci katman ÇÇY katmanı, verinin sınıflandırılmasından ve veri tabanına kaydedilmesinden sorumludur. Platforma gelen veri soket, rest veya CSV(Comma Separated Value) üzerinden çoklu yükleme(bulkloading) şeklinde olabilir. Bu katman, verinin geliş kaynağından bağımsız olarak veriyi sınıflandırır, veri kaynağına özel bir işlem yapmaz. Böylelikle veri işleneceği zaman, belli bir sınıf ve düzende olacağından, daha kolay işlenir.

Üçüncü katman VSK katmanı, verinin belli bir yapıda tutulmasından ve kullanılacağı zaman getirilmesinden sorumludur. Bu katmanda 3.4.5 bölümünde detaylıca açıklanan MongoDB kullanılmaktadır.

Dördüncü katman MÖA katmanı olup, veri tabanında saklanan veriler üzerinde çeşitli analizler sonucu modeller üretmekten ve bu modelleri kullanarak analiz yapmaktan sorumludur. Aynı zamanda bu katman, veri üzerinde model oluşturmadan temizleyip(boş alanların temizlenmesi gibi) veriyi makine öğrenmesi için daha uygun hale getirir.

Veri tabanına kayıt etme haricinde bu katmanların hepsiyle iletişim halinde olan ve bunları yöneten son katman GAK katmanıdır. Bu katman, kullanıcıya veri bağlantılarını yönetme, toplu veri yükleme ve makine öğrenmesi analiz sonuçlarını görme, sınıflandırılan veriler üzerinde yeni görevler tanımlama, veri portlarını takip etme imkanı sunar.

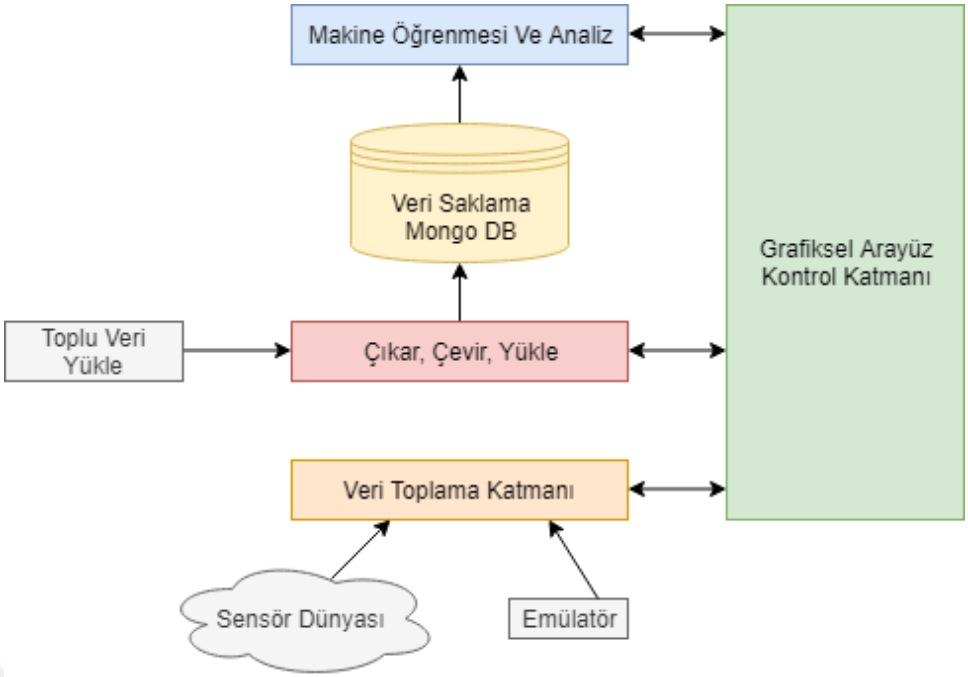
Platform modüler bir yapıda tasarlanmıştır ve arayüz, sunucu, öğrenme modülü olmak üzere üç modül bulunmaktadır. Bu modüller birbirlerinden bağımsız çalışmaktadır ve haberleşmek için REST mimarisinden faydalanmaktadırlar. Bu nedenle ayrı sunucular üzerinde çalışabilirler. Modüllerin veri aktarımı yapabilmesi için, istek atacakları URI'i(Uniform Resource Identifier) bilmeleri yeterlidir.

Arayüz modülü, platformun daha kolay yönetilmesi amacıyla yazılmış olup, sunucu modülüyle haberleşebilir. Arayüz üzerinden herhangi bir sorgu yapılacağı veya öğrenme isteği oluşturulacağı zaman, bu işlem sunucu modülü üzerinden gerçekleştirilir. Arayüz modülünün doğrudan öğrenme modülü ile iletişimi yoktur. Kavramsal tasarımda GAK katmanına karşılık gelmektedir.

Sunucu modülü, tüm platformun yönetildiği ve diğer iki modülün doğrudan iletişim kurabileceği merkezi bir modüldür. Çalışmak için arayüz modülüne ihtiyaç duymamaktadır. Sunucu modülü tasarlanırken aynı zamanda REST API olarak tasarlanmış, dolayısıyla arayüzden yapılan her işlemin, arayüz olmadan da konsoldan komutlarla veya POSTMAN[10] gibi REST istekleri atılmasına olanak sağlayan bir programdan veya herhangi bir programlama dilinde yazılmış ve rest isteği atabilen bir uygulamadan iletişim kurularak yönetilebilir. Veri tabanıyla iletişimi olan tek modüldür. Bu modül kavramsal tasarımda VTK, ÇÇY, VSK katmanlarına karşılık gelmektedir.

Öğrenme modülü, LSTM(Long-short term memory) algoritması üzerinden modeller oluşturan, modüle gelen veriler üzerinden anomali bulan ve modelin başarısını değerlendirebilme kapasitesine sahip bir modüldür. Çok iş parçacıklı bir yapıda tasarlanmış olup, sunucu modülünden aldığı zaman serisini, paralel biçimde LSTM algoritmasını kullanarak analiz etme kapasitesine sahiptir. Bu modüle gelen her istek farklı bir iş parçacığına atanır ve işlem sonucu, daha sonra kullanılmak üzere bir ***hashmapde*** saklanır. Öğrenme modülü MÖA katmanında bulunmaktadır.

[10]https://www.getpostman.com/



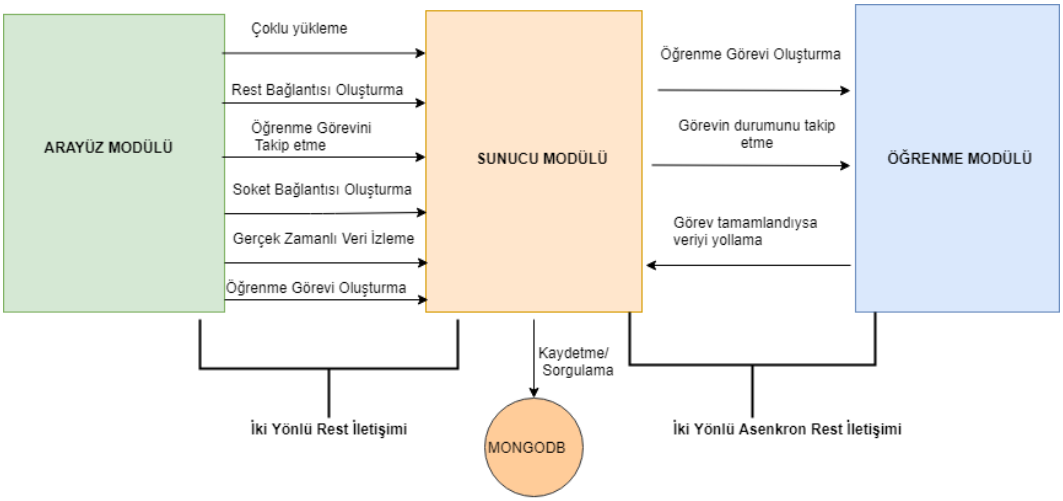
Şekil 3.1: MISIoT Kavramsal Tasarım

3.1 Sunucu Modülü

Sunucu modülü, tüm platformun yönetiminden sorumlu olan ana modüldür ve birçok görevi bulunmaktadır. Bunlar özetlenecek olursa;

- Farklı kaynaklardan verilerin toplanması ve toplanan verilerin sınıflandırılıp veri tabanına kaydedilmesi
- Öğrenme görevlerinin oluşturulması ve asenkron biçimde yönetilmesi
- Canlı olarak bir porttaki veri akışının izlenmesi
- Soket veya REST bağlantılarının yönetilmesi
- Grafiksel arayüzden gelen sorgu isteklerine cevap verilmesi
- Toplu veri yüklemesinin yönetilmesi

Bu bölümde sunucu modülünün yapısı ve yukarıda belirtilen görevlerin nasıl gerçekleştirildiği detaylı bir biçimde anlatılacaktır.



Şekil 3.2: MISIoT Modül Mimarileri

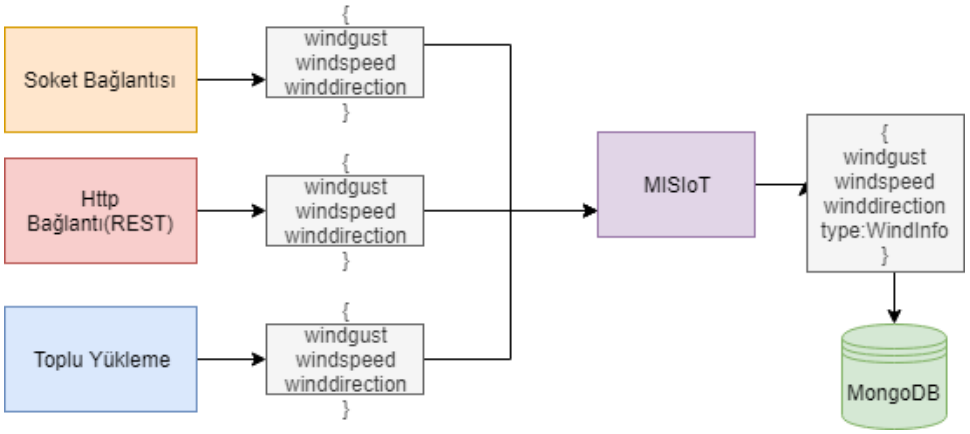
3.1.1 Konfigürasyon

Platform çalıştırılırken her seferinde bir konfigürasyon dosyasından, platforma herhangi bir kaynaktan gelen veriyi nasıl veri tabanına kaydedilmesi gerektiği bilgisi alınır. Konfigürasyon dosyasından bir örnek gösterilecek olursa;

```
<collection>
  <collectionName>Weather</collectionName>
    <types>
      <type>
        <typeName>WindInfo></typeName>
        <columns>windGust, windDirection, windspeed</columns>
      </type>
      <type>
        <typeName>AirInfo></typeName>
        <columns>temperature, windSpeed</columns>
      </type>
    </types>
<collection>
```

Bu dosyada öncelikle gelen verinin kaydedileceği MongoDB koleksiyon(collection) ismi veriliyor, daha sonra bu koleksiyon altındaki dökümanların içerebileceği olası kolonlar ve bu kolonlarla eşlesmesi durumunda sınıflandırılacağı tip(type) ismi belirleniyor. Bu örneğe göre, eğer platforma herhangi bir veri kaynağından şu şekilde bir verinin geldiği varsayılırsa;

```
{
```



Şekil 3.3: MISIoT gelen verinin konfigürasyon dosyasına göre sınıflandırılması

```
windGust:50,
windDirection:45,
windSpeed:100
}
```

gelen verideki en çok WindInfo tipine ait kolonlarla kesiştiğinden dolayı, veriye *__type:WindInfo* şeklinde yeni bir alan eklenip, veri tabanında bu şekilde saklanıyor. Verinin sınıflandırma sonrası son hali;

```
{
__type:WindInfo
windGust:50,
windDirection:45,
windSpeed:100
}
```

Verinin sınıflandırılması için gelen kolonlarla konfigürasyon dosyasında bulunan kolonların birebir eşleşmesi gerekmemektedir. Yukarıdaki örnekte verilen konfigürasyon dosyasına göre, platformun gelen veriyi sınıflandırabileceği, WindInfo ve AirInfo olmak üzere iki seçenek bulunmaktadır. Platform, bu tiplerin içerebileceği olası kolonlarla gelen verinin ne kadar kesiştiğini hesaplayarak, en fazla kesişen tipi o verinin tipi olarak belirliyor. Yukarıda verilen örnek veri, WindInfo ile %100, AirInfo tipi ile %33 eşleşmekte, bu nedenle tipi WindInfo olarak belirlenmektedir. Daha sonra bu veri, dinamik olarak oluşturulan **Weather** isimli koleksiyona kaydedilmektedir. Konfigürasyon dosyasında istenilen sayıda koleksiyon ve tip tanımlaması yapılabilir. 3.3 ifadesinde yukarıda anlatılan senaryo gösterilmiştir.

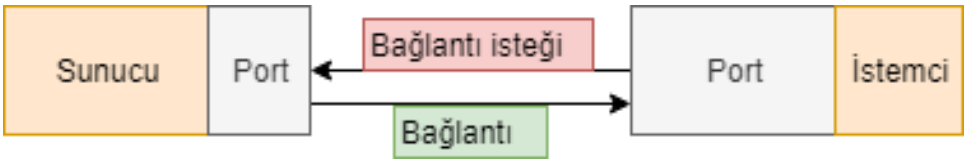
Her veriye belli bir tip vermenin çeşitli avantajları bulunmaktadır. Bunlardan ilki, daha

rahat sorgulanabilir bir veri tabanı yapısı sunmasıdır. Platforma gelen her verinin platform tarafından belirnenen bir tipi olacağından, kullanıcı daha sonra arayüz modülü üzerinden sadece belirli bir tipe sahip verilerle sorgulama yapabilir veya öğrenme görevi oluştururken, sadece belirli tipteki verileri kullanabilir. Bir diğer avantaj ise, konfigürasyon vasıtasıyla kullanıcı, istediği tipte verileri otomatik olarak istediği MongoDB koleksiyonuna kaydedebilir, veri tabanı kullanıcının kendi isteği doğrultusunda şekillendirilebilir.

3.1.2 Soket ve REST Bağlantı Oluşturulması

Soket, istemci ve sunucu arasında iletişimi sağlayan ve belirli bir IP ve port numarasına bağlı olan bir yapıdır ve REST'in temelini oluşturan HTTP bağlantısına göre daha alt seviye çalışmaktadır. HTTP, yine soket üzerinde çalışan ancak daha üst seviye soyutlamaya sahip olan bir mimaridir. Bu iki teknoloji oldukça yaygın olarak kullanıldığından, MISIoT'da ikisi de desteklenmektedir.

MISIoT, üzerinde çalıştığı işletim sisteminin izin verdiği ölçüde üzerinde yeni bağlantı noktaları açılmasını desteklemektedir. Günümüzde yaygın olan REST mimarisinin temelini oluşturan HTTP bağlantı veya soket bağlantı, işletim sistemi tarafından kullanılmayan herhangi bir portta oluşturulabilir. 3.4 ifadesinde örnek bir bağlantı görülebilir. REST mimarisi ve avantajları hakkında detaylı bilgi 2.3.3 bölümünde verilmiştir.



Şekil 3.4: Sunucu-İstemci Bağlantısı

Bu portlar üzerinden, bağlantı tipinden bağımsız olarak veri akışı sağlanabilir. Her iki bağlantı üzerinden gelen veri, 3.1.1 bölümünde anlatıldığı şekilde sınıflandırılır ve veri tabanına kaydedilir. Kullanıcı ihtiyacına göre bağlantı tipini oluşturduktan ve bu bağlantı üzerinden gelebilecek olası alanları konfigürasyon dosyasında belirttikten sonra, sistem bağlantı tipinden bağımsız olarak gelen veriyi sınıflandırabilir.

3.1.3 Toplu Veri Girişi

Soket ve rest bağlantılarına ek olarak, platforma CSV formatında herhangi bir dosya yüklenebilir. Dosyadaki satırların her biri bir JSON objesine karşılık gelecek biçimde dönüştürülür, 3.3'da gösterildiği şekilde, belirli bir konfigürasyona göre sınıflandırıl-

dıktan sonra, veri tabanına kaydedilir. Platform, çoklu dosya yüklemesini de desteklemektedir. Platforma REST üzerinden gönderilen istekte, belirtilen dosya lokasyonları okunur, bu dosya lokasyonlarındaki CSV uzantılı dosyalar işlenerek sisteme aktarılır.

3.1.4 Asenkron Mimari

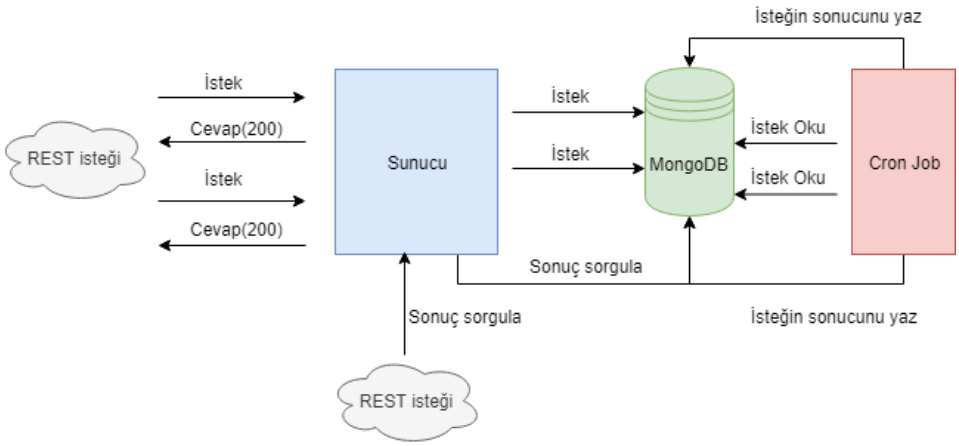
MISIoT, birçok farklı kaynaktan veri alabileceğinden dolayı, bu verileri hızlı işlemesi ve akıcı bir mekanizmada çalışması oldukça önemlidir. Bu sebeple, platform modüler ve asenkron yapıda tasarlanmıştır. Bu bölümde anlatılan sunucu modülü, diğer iki modül olan arayüz modülü ve öğrenme modülünden bağımsızdır. Bu modüllerle REST istekleri vasıtasıyla haberleşir. Sunucu modülünde yapılan işlemler zaman alabileceğinden, modüle atılan isteklerin senkron biçimde işlenip, isteği atan modülün veya programın bekletilmesi, kullanıcı deneyimini baltalamaktadır. Özellikle, bir makine öğrenmesi kodunun çalışması günler alabilecek bir işlem olduğundan, arayüzden veya POSTMAN gibi REST isteği atabilen bir programdan atılan isteklerin bunların cevaplarını beklemesine imkan yoktur.

3.5 ifadesinde görülen asenkron mimaride, sunucu modülüne herhangi bir kaynaktan gelen bir görev oluşturma isteğine anında isteğin kaydedildiğine dair ***200*** gibi olumlu bir cevap dönülür. Daha sonra modül, kaydedilen bu istekleri bir ***cron zamanlayıcı***[11](cron job) vasıtasıyla sunucunun durumuna göre ileriki bir zamanda işleyerek, sonucu veri tabanına kaydeder. Bu sonucun çıkması günler alabilir, ancak hali hazırda isteği atan tarafa bir cevap dönüldüğünden, istek sahibi farklı işlerle uğraşabilir, sunucu tarafından senkron olarak bir cevap beklemediğinden kitlenmez. Bundan sonraki aşamada, kullanıcı oluşturduğu talebin sonucunu görmek için, platform tarafından sağlanan farklı bir REST endpoindine sorgu atarak, sonuçları eğer işlem bittiyse görebilir. MISIoT'un genelinde kurulan bu asenkron yapı sayesinde, modüller birbirini beklemeden, akıcı biçimde çalışabilmektedir. MISIoT'daki her modül için geçerli olan bu yapı, hali hazırda 3.4.1 bölümünde anlatılan Node.js'de de kullanılmaktadır.

3.1.5 Görev Yönetimi

MISIoT, **CustomTaskWithFrameworkData, CustomTaskWithoutFrameworkData, LSTM** olmak üzere üç farklı görev tipini desteklemektedir. 3.1.4 bölümünde anlatıldığı gibi, bu üç görev tipi de asenkron olarak çalışmaktadır ve birbirlerinden bağımsız çalışabildiklerinden, istenilen sayıda görev tanımlanabilir ve bu görevlerin ilerleme durumları platform üzerinden takip edilebilir.

[11]https://en.wikipedia.org/wiki/Cron



Şekil 3.5: Sunucu modülü asenkron mimari

İlk görev çeşiti LSTM görevidir. 3.3 bölümünde anlatılan LSTM algoritması, çalışmak için **lstm unit, drop out, dense, loss, optimizer, look back, epoch, batch size, verbose** gibi birçok parametreye ihtiyaç duymaktadır. Bu nedenle arayüzden veya herhangi bir REST istemcisinden istek oluşturulurken, bu parametrelerin verilmesi gerekmektedir. Bunlara ek olarak, LSTM algoritmasının uygulanmak istediği kolon ismi verilir. Platform, bu kolonu içeren verileri bir dizide toplar ve görev oluşturulurken, veri tabanına bu bilgi de kaydedilir. Dolayısıyla cron zamanlayıcı, hali hazırda bir dizi haline getirilmiş zaman serisini öğrenme modülüne yollar ve bu seri üzerinde LSTM algoritması uygulanıp çeşitli çıktılar tekrar sunucu modülü tarafından kullanılmak üzere saklanır.

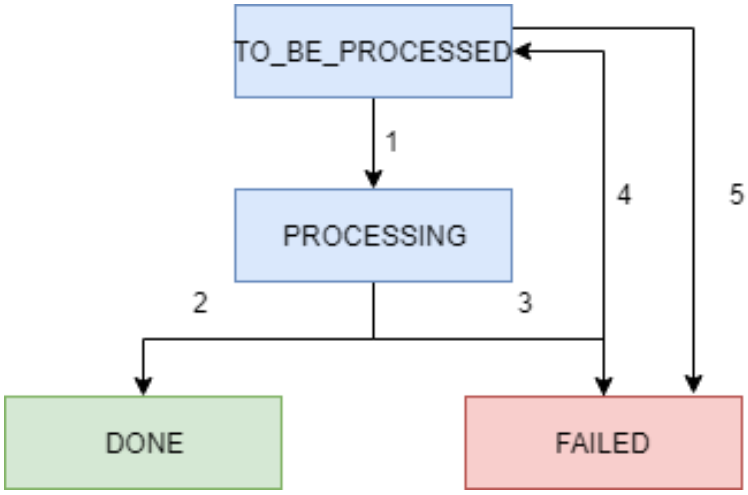
İkinci ve üçüncü görev çeşitleri olan **CustomTaskWithFrameworkData, CustomTaskWithoutFrameworkData** ise yine benzer bir yapıda çalışmaktadır. Bu iki görev, öğrenme modülüne ihtiyaç duymadığından dolayı daha az karmaşık bir cron zamanlayıcı yapısı kullanmaktadır. **CustomTaskWithFrameworkData** görevinde, görev sisteme kaydedilirken, çeşitli sorgu kriterleriyle beraber kaydedilir. Aynı zamanda kullanıcı, platform verisiyle çalıştırmak istediği betiğin bulunduğu dizini de(path) REST isteği atarken isteğe ekler. Kullanıcı platform üzerinde bulunan verileri zaman, büyüklük veya veride bulunan spesifik bir kolonu dahil edecek şekilde sorgu kriterleriyle belirtebilir. Daha sonra cron zamanlayıcı çalıştığında, bu sorgu kriterlerini kullanarak veri tabanı için bir sorgu oluşturur ve sorgu sonuçlarını JSON objeleri haline çevirip, bunları bir dizi içinde tutar. Daha sonra görev oluşturulurken belirtilen Python betiklerini bu JSON dizisini argüman olarak verip çalıştırır. Bu betiğin çıktısı veri tabanına kaydedilir, böylelikle daha sonra kullanıcı görmek istediği zaman arayüzden veya doğrudan bir REST istemcisinden attığı isteklerle sonuçları görebilir. Kullanıcıya ait herhangi bir Python betiği, platformda yüklü bulunan veri setinin spesifik bir kısmı kullanılarak çalıştırılabilir. **CustomTaskWithoutFrameworkData** görevi ise, platforma ait herhangi

bir veriyi kullanmadan, kullanıcı tarafından belirtilen dizindeki betiği doğrudan çalıştırır ve sonucu yine platformda saklar.

Platform, her oluşturulan görev için **to_be_processed, processing, done, failed** gibi durum kodları tutmaktadır ve kaydın durumuna göre bu kodlar güncellenmektedir. Platform üzerinde görev tanımlandığı anda, bu görev veri tabanına **to_be_processed** olarak kaydedilir. Cron zamanlayıcı çalıştığı zaman, durumu **to_be_processed** olan kayıtları veri tabanından okur ve görev tipine özel aksiyonları gerçekleştirmeye başlar. LSTM haricindeki **CustomTaskWithFrameworkData, CustomTaskWithoutFrameworkData** görevlerinde, veri tabanındaki **to_be_processed** durumundaki görevler işlenmeye başlanır başlanmaz, durumları **processing** olarak güncellenir. Yapılan işlem bittiği zaman, görevin durumu **done** olarak güncellenir. Eğer süreçte bir hata çıkarsa, durum **failed**'a çekilir. Bu iki görev için sadece sunucu modülünün kullanılması yeterlidir, tüm süreç arayüze ihtiyaç olmaksızın REST istekleriyle yönetilebilir. Kullanım kolaylığı için arayüz modülünden de bu iki görev tanımlanıp, durumları, çalıştırıldıkları parametreler ve ürettikleri sonuçlar görülebilir.

LSTM özelinde, işin içine öğrenme modülü girdiğinden cron zamanlayıcı daha farklı çalışmaktadır. Öğrenme görevi oluşturulduğunda, yine ilk durum **to_be_processed** olarak kaydedilir. Ancak durum diğer iki görev çeşidinde olduğu gibi direkt **processinge** çekilmez. Bir başka REST modülü olan öğrenme modülüne istek atılır, bu modülden olumlu cevap dönülürse(200), görevin durumu **processinge** çekilir. Eğer dönülmezse görev **failed** olarak işaretlenir ve daha sonra ekrandan veya herhangi bir istemciden gelecek bir REST isteği ile tekrar **to_be_processed** olarak işaretlenir, böylelikle cron zamanlayıcı tarafından bir sonraki döngüde tekrar denenir. Eğer görev başarılı bir şekilde **processinge** çekildiyse, cron zamanlayıcı ilgili görevin MongoDB kayıt id'si ile, zamanlayıcı her çalıştığında öğrenme modülüne sorgu atıp, görevin durumunu sorgular. Öğrenme modülü işlemi bitirdiyse, yine 200 ve beraberinde görevin sonuçlarını içeren bir cevap dönmektedir. Bu durumda sunucu modülündeki cron zamanlayıcı, ilgili görev kaydını gelen verilerle günceller, görevin durumunu **done** olarak belirler ve bir sonraki döngüde bu görev bir daha dikkate alınmaz. Kısacası cron zamanlayıcı, LSTM görevlerinde iki farklı kayıt tipini dikkate almaktadır. **to_be_processed** kayıtlar için öğrenme modülünde süreci başlatır, **processing** durumundaki kayıtların ise, bitip bitmediğini sorgular, bittiyse gelen verilerle ilgili görev kaydını günceller.

Platformun asenkron çalışma mekanizması ve kümeleme kullanılarak yazılması sebebiyle, istenildiği kadar Python betiği aynı anda çalıştırılabilir, sonuçlar birbirinden bağımsız olarak elde edilecek ve saklanacaktır. Bu yapı sayesinde, LSTM gibi doğrudan domain spesifik bir algoritma kullanılabileceği gibi, kullanıcı kendi istediği herhangi bir betiği, platformda bulunan verilerle çalıştırabilmekte ve bunun sonucunu yönete-



Şekil 3.6: Lstm durum şeması

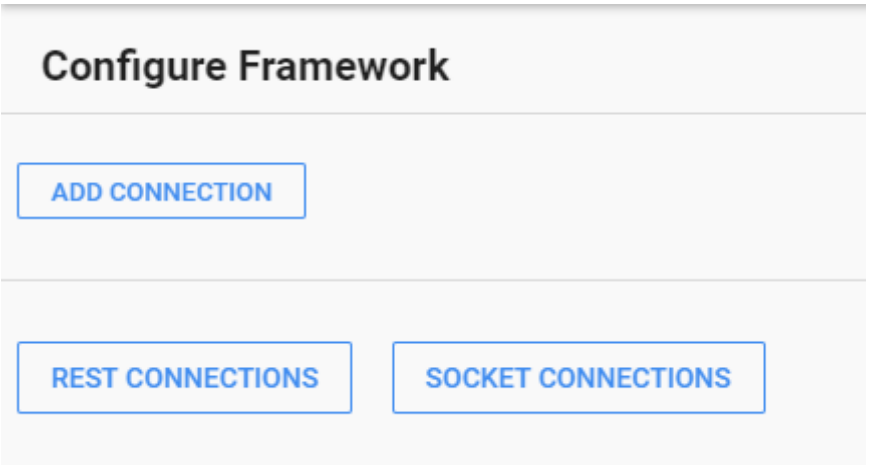
1: Öğrenme modülü 200 dönerse
2: Öğrenme modülü 200 ve ilgili göreve ait verileri dönerse
3: Öğrenme modülüne atılan isteklere uzun süre cevap gelmezse
4: Başarısız durumdaki görevlerin tekrar TO_BE_PROCESSED'e çekilip işlenebilir olması
5: Öğrenme modülüne atılan isteklere uzun süre cevap gelmezse

bilmektedir.

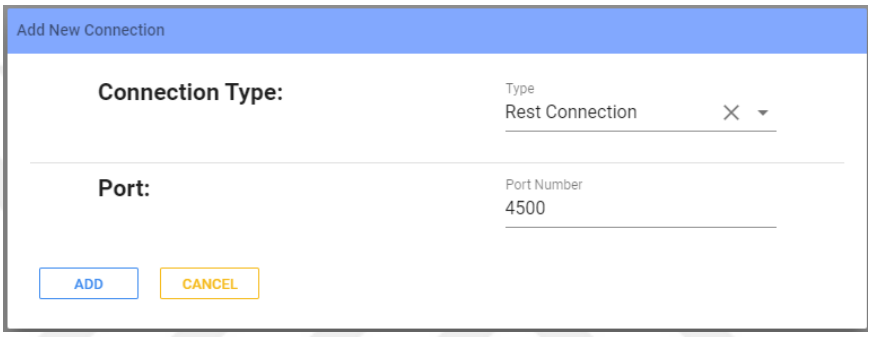
3.2 Arayüz Modülü

MISIoT, sunucu modülü üzerinden herhangi bir REST istemci ile yönetilebilmektedir. Ancak REST istemci üzerinden isteği oluşturmak, gerekli alanları doldurmak ve süreci takip etmek kullanıcı dostu değildir. Bu nedenle REST istemci ile yapılan her işlemin daha kolay yapılabilmesi için, arayüz modülü yazılmıştır. Arayüz modülü üzerinden yapılabilecek işlemler;

- ***Konfigürasyon:*** Aktif soket ve rest bağlantıları takip edilebilir, açılıp kapatılabilir
- ***Çoklu yükleme:*** Platforma CSV formatında toplu veri yüklemesi yapılabilir.
- ***LSTM Öğrenme Görevi:*** LSTM analizi için belli bir veri seti üzerinde öğrenme görevi oluşturulabilir.
- ***Farklı betikler çalıştırma:*** Platform kullanılarak, herhangi bir Python betiği, platformda bulunan veri seti ile asenkron biçimde çalıştırılabilir.



Şekil 3.7: Konfigürasyon Genel Görünüm

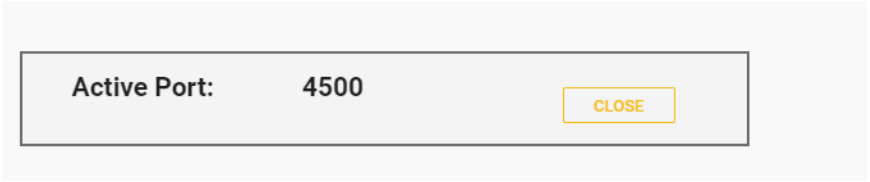


Şekil 3.8: Soket ve REST bağlantısı ekleme

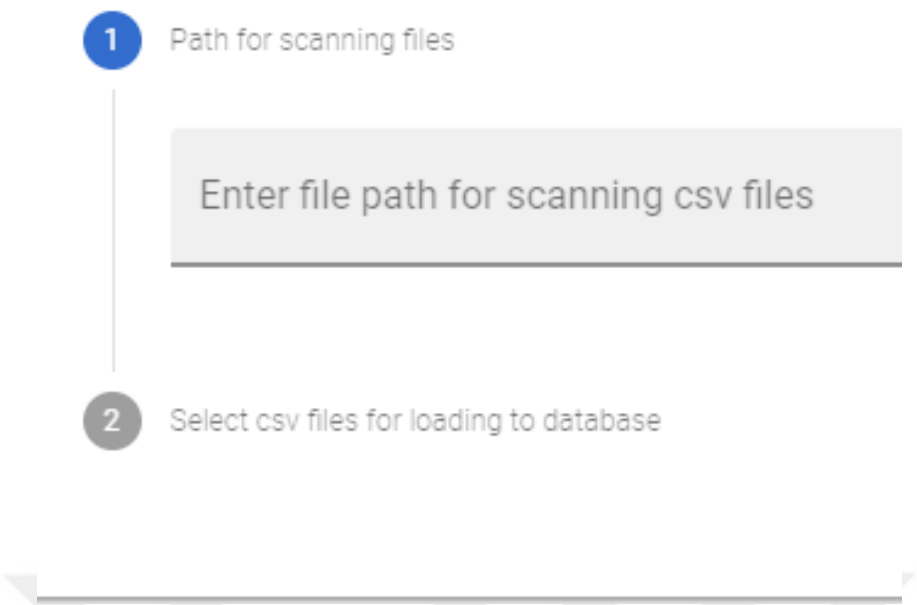
- ***Oluşturulan görevlerin takibi:*** Platform üzerinde oluşturulan her görevin durumu ve tamamlandıysa sonucu takip edilebilir.
- ***Grafiksel analiz:*** Platform üzerinde aktif tüm portların mevcut veri akışı izlenebilir.

3.2.1 Konfigürasyon

MISIoT, işletim sistemi veya başka bir program tarafından kullanılmayan herhangi bir portta soket veya REST bağlantısı oluşturabilir. Bu işlem her ne kadar doğrudan sunucu modülüne REST isteği gönderilerek yapılabilse de, arayüz üzerinden de yapılabilir.



Şekil 3.9: Aktif bağlantıları görüntüleme



Şekil 3.10: Bulkloading genel görünüm

3.7 ifadesinde konfigürasyon ekranının genel görüntüsü gösterilmektedir. **Add Connection** butonundan yeni bağlantılar eklenebilir. Aktif REST ve soket bağlantılarını görüntülemek için **Rest Connections** ve **Socket Connections** butonları kullanılabilir.

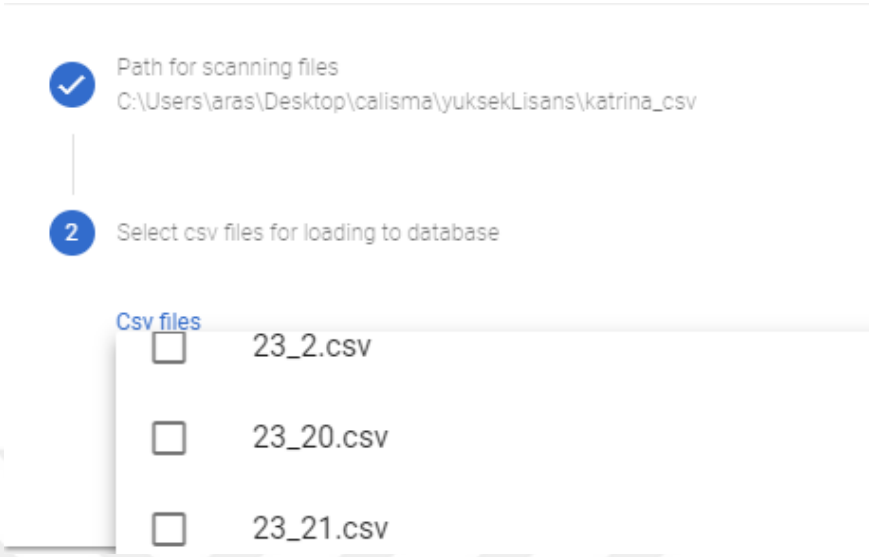
3.8 ifadesinde nasıl yeni bağlantı eklenebileceği görülmektedir. **Connection Type** eklenmek istenen bağlantı türünün seçilmesine olanak sağlar. Bağlantı türü olarak soket veya REST bağlantısı seçilebilir. Bağlantı türü seçildikten sonra **Port** ile bağlantının hangi port üzerinde açılacağı belirlenir. Seçilen port hali hazırda başka bir program tarafından kullanılıyorsa bağlantı açılamaz.

3.9 ifadesinde, bağlantı oluşturulduktan sonra bu bağlantıların nasıl görüntülenebileceği görülmektedir. **Active Port** ile bağlantının kurulu olduğu port görülebilir ve **Close** ile o bağlantı kapatılabilir.

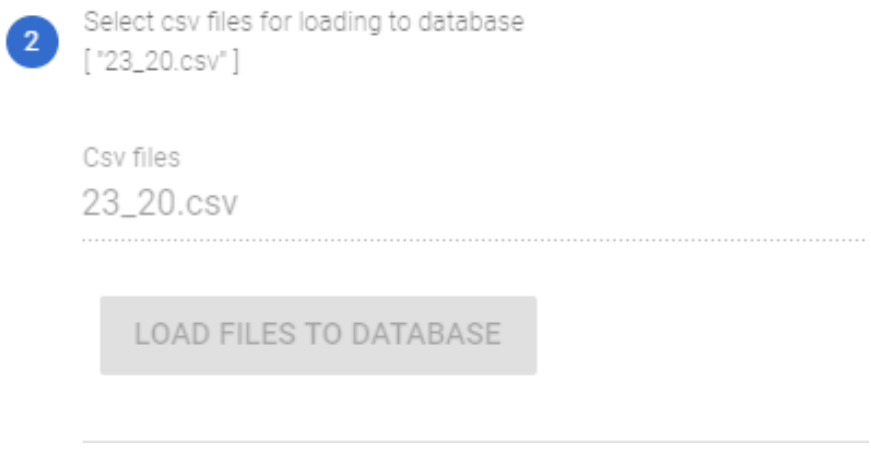
3.2.2 Çoklu Yükleme

CSV günümüzde oldukça yaygın kullanılan bir dosya formatı olmakla birlikte, büyük veri setlerinde yaygın olarak kullanılmaktadır. Bu nedenle MISIoT, soket ve REST teknolojilerine ek olarak CSV formatında dosya yüklenmesini hem herhangi bir REST istemciden hem de arayüz üzerinden desteklemektedir.

3.10 ifadesinde çoklu yükleme ekranının genel görüntüsü gösterilmektedir. Bu ekran sıralı çalışmaktadır. Kullanıcıdan ilk adımda yüklemek istediği CSV dosyalarının bu-



Şekil 3.11: Dizin girilmesi ve dizin altındaki dosyaların bulunması



Şekil 3.12: Seçilen dosyaların sisteme yüklenmesi

lunduğu dizini girmesi beklenir. Girilen dizin Windows için **C:\Users\aras\Desktop** benzeri veya Unix dosya sistemine özgü bir format olabilir. Kullanıcı dizini girdikten sonra kutunun sağındaki **>** butonuna basarak ikinci adıma geçer.

3.11 ifadesinde kullanıcı dizini girdikten sonra, ikinci adımda o dizin altındaki CSV dosyalarının gösterimi bulunmaktadır. Bu adım çoklu seçim desteklediğinden, kullanıcı istediği kadar CSV dosyasını seçebilir. Dosyalar seçildikten sonra, kutunun hemen altında çıkan **Load Files To Database** butonuna basılarak dosyalar sisteme yüklenebilir.

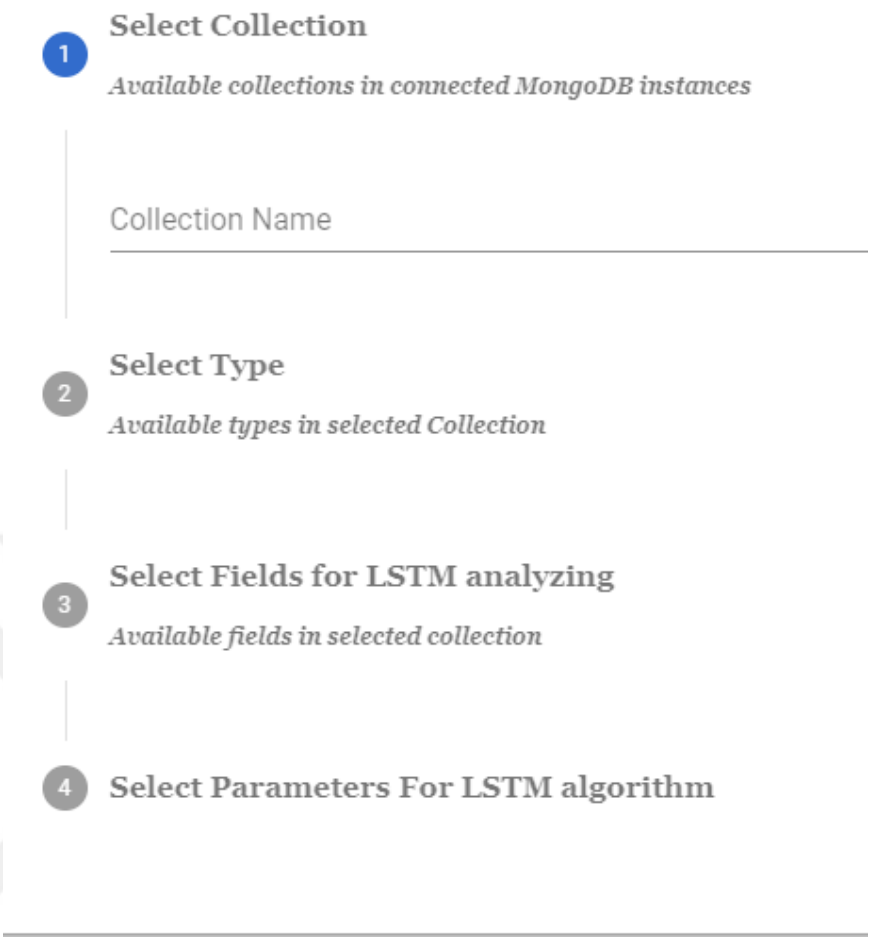
3.12 ifadesinde ise son adımda dosyalar sisteme yüklenirken takip edilmesine olanak veren durum yuvarlağı gösterilmektedir. Verilen örnekte kullanıcı bir dosya seçtiğinden, **Total Files: 1** olarak gösterilmektedir. **Remaining** ise, yüklenecek kaç dosya kaldığını göstermektedir. Dosyalar yüklendikten sonra, kullanıcı **Load New Files** butonuna basarak tekrar ilk adımdan ve farklı bir dizin kullanarak dosya yükleme işlemine devam edebilir.

Çoklu yükleme, sistem kaynaklarının verimli kullanılması için dosya başına bir istek olacak biçimde tasarlanmıştır. Örneğin, kullanıcı 200 CSV dosyası seçtiyse, her dosyanın ismini ve dizinini içeren 200 ayrı REST isteği sunucu modülüne gitmektedir. Sunucu modülüne gelen REST istekleri herhangi bir istemciden olabileceğinden dolayı, istemci tarafında zaman aşımı oluşmaması için isteklerin işlenip hızlıca cevaplanması gerekmektedir. Dosya başına REST isteği bu sebepten dolayı atılmaktadır.

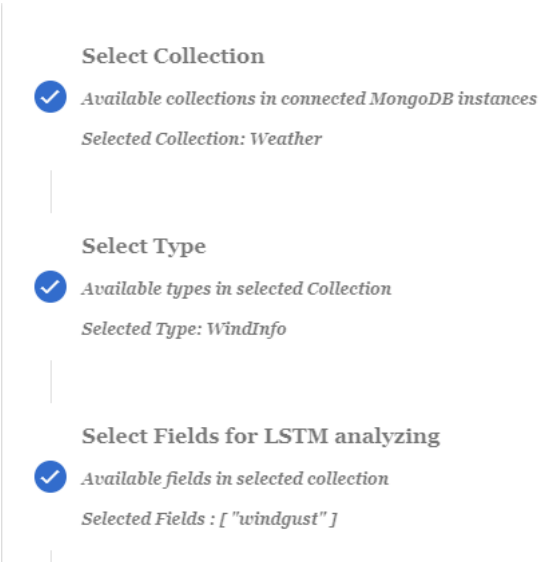
3.2.3 LSTM Görevi Oluşturma

MISIoT'un desteklediği görev tiplerinden biri olan LSTM öğrenme görevini oluşturmak için sırasıyla koleksiyon, koleksiyonun içerdiği tip, o tipe bağlı kolonlar ve en son olarak LSTM parametrelerinin girildiği dört adımlı bir ekran tasarlanmıştır. 3.1 bölümünde detaylıca anlatılan platformun konfigürasyon yapısıyla uyumlu olarak, kullanıcı ilk üç adımda algoritmanın üzerinde çalışacağı veri setini seçmektedir, son adımda ise LSTM algoritmasının spesifik parametlerini belirleyip, görevi oluşturmaktadır.

3.13 ifadesinde başlangıç durumu görülmektedir. Kullanıcı ilk adımda sistemde yüklü olan collectionlardan birini seçer.İkinci adımda bu koleksiyonun içerdiği tipler seçilebilir olmaktadır. Bu tiplerden birini seçtikten sonra, üçüncü adımda bu tipe bağlı kolonlar arasından seçim yapar. Kolonlar çoklu seçilebilmektedir, her seçilen kolon için ayrı bir LSTM görevi oluşturulmaktadır. 3.14 ifadesinde kullanıcı sırasıyla **Weather** koleksiyonunu, **WindInfo** tipini ve bu tipe bağlı **windgust** kolonunu seçmiştir. 3.15 ifadesinde ise dördüncü adım görülmektedir. Bu adımda kullanıcı LSTM'e özel

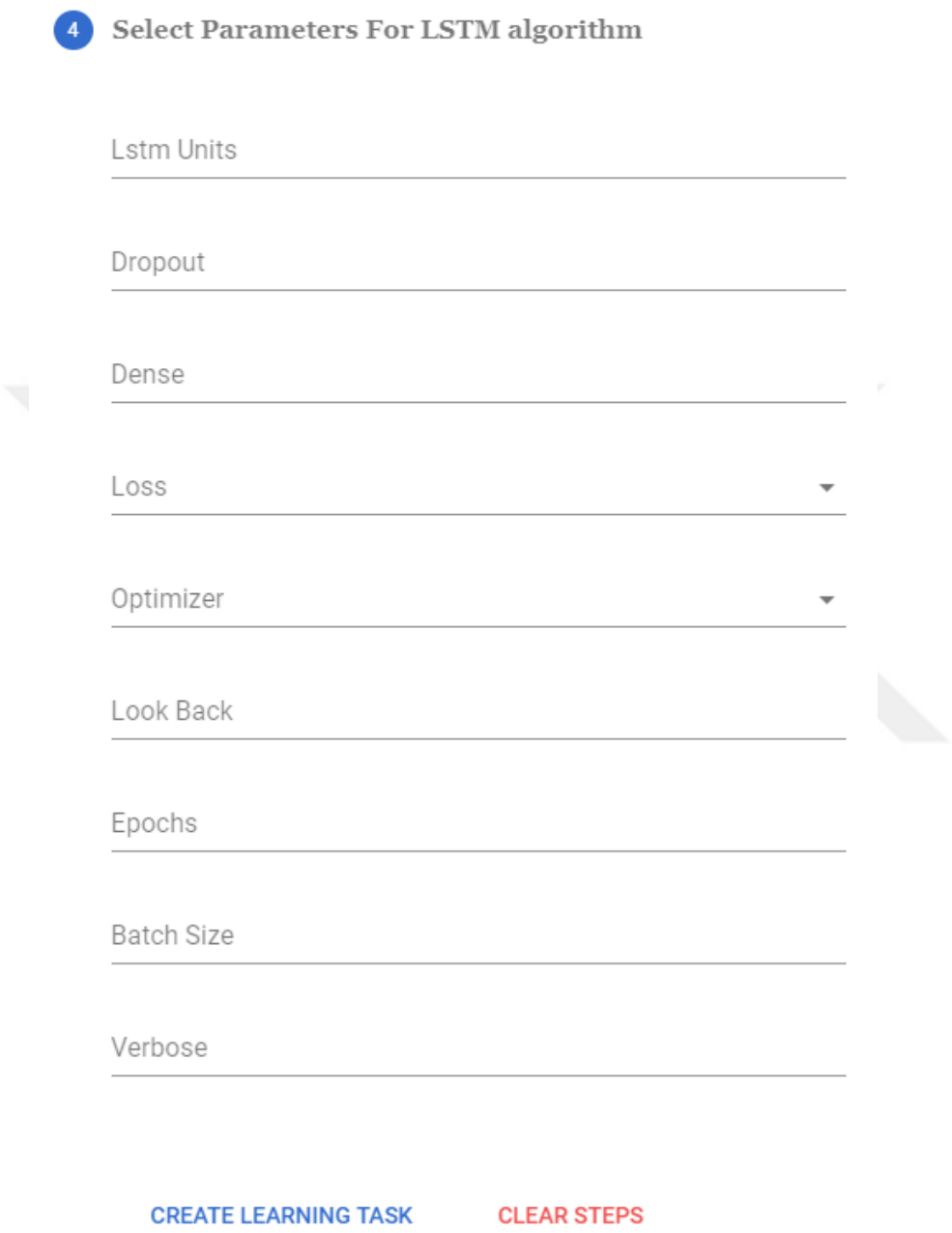


Şekil 3.13: LSTM oluşturma genel görünüm



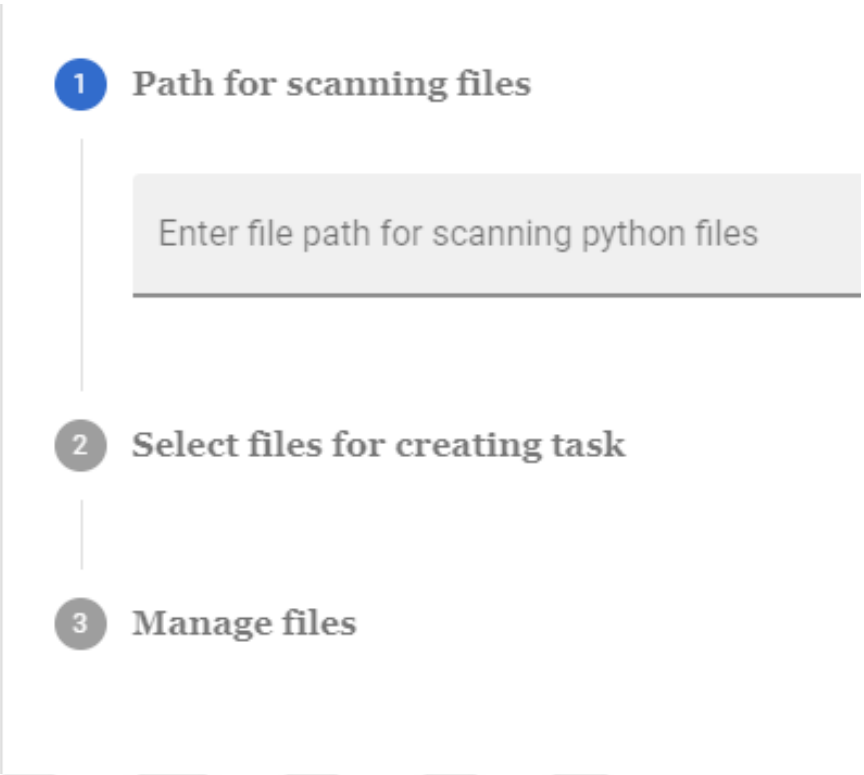
Şekil 3.14: Collection, type ve field bilgilerinin girilmesi

parametreleri girebilir, boş bıraktığı alanlara sistem tarafından önceden belirlenen parametreler verilmektedir. Kullanıcı, parametreleri belirledikten sonra **Create Learning Task** butonuna basarak öğrenme görevini oluşturabilir veya **Clear Steps** butonuna basarak birinci adıma dönebilir.



Şekil 3.15: LSTM parametrelerinin girilmesi ve görev oluşturulması

Görev sunucu modülünde kaydedilirken, kullanıcı tarafından seçilen kolonlar kendi aralarında gruplanır, kolonlara karşılık gelen değerlerden bir dizi oluşturulur ve veri tabanında bu şekilde saklanır. LSTM algoritması zaman serisine ihtiyaç duyduğundan böyle bir yöntem tercih edilmiştir.



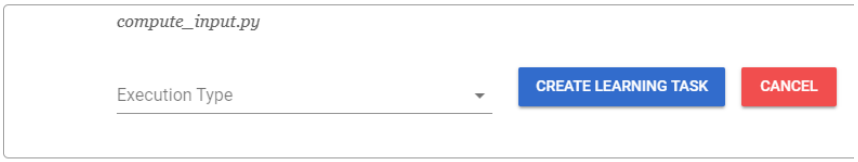
Şekil 3.16: LSTM dışı görev oluşturma genel görünüm

3.2.4 LSTM Dışı Görevler Oluşturma

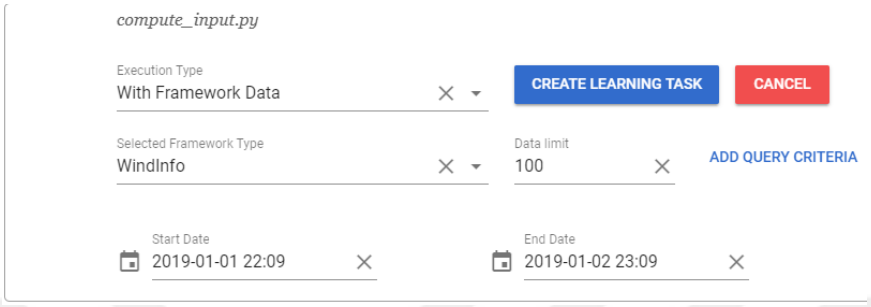
MISIoT'da zaman serisi şeklinde gelen verinin analizi için spesifik olarak LSTM algoritması kullanılsa da, kullanıcıların kendi yazdıkları Python betiklerinin de çalıştırılmasına imkan verilmiştir. Kullanıcı, belirlediği bir dizin altındaki her Python betiği üzerinden, LSTM'de olduğu gibi asenkron olarak ve platform verisini kullanarak görev tanımlayabilir. MISIoT aynı zamanda, betiğin herhangi bir platform verisi kullanılmadan doğrudan asenkron olarak çalıştırılmasına da imkan sağlamaktadır.

Bu sayfada LSTM ve çoklu yükleme sayfalarında olduğu gibi adımlı bir tasarım yapılmıştır ve üç adım bulunmaktadır. 3.16 ifadesinde sayfa ilk açıldığında kullanıcıdan Python betiklerinin bulunduğu bir dizin girmesi beklenmektedir. Bu dizin girildikten sonra kutunun sağındaki **>** butonuna basıldığında, bir sonraki adıma geçilir. Bu aksiyon alındığında sunucu modülüne dizin REST isteği ile iletilmekte ve o dizin altında bulunan **.py** uzantılı dosyalar, ikinci adımda kullanılmak üzere cevap olarak alınmaktadırlar.

İkinci adımda kullanıcı, ilk adımda seçtiği dizinde bulunan Python betikleri arasından istediklerini seçebilir.Üçüncü adımda kullanıcının seçtiği her betik için, o betik özelinde çeşitli parametreleri ayarlayabileceği bir panel oluşturulmaktadır. 3.1 bölümünde anlatıldığı üzere, platform LSTM haricinde **CustomTaskWithFrameworkData, Cus-**



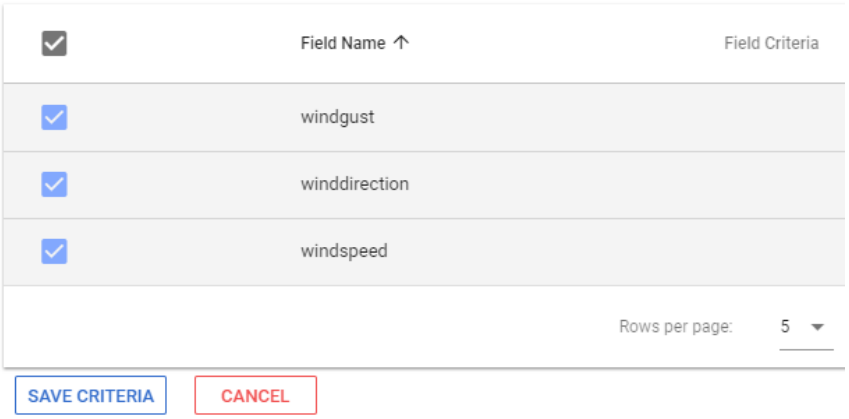
Şekil 3.17: Dizin girildikten ve görev tanımlanmak istenilen dosyalar seçildikten sonraki durum



Şekil 3.18: With Framework Data görev tanımlaması

tomTaskWithoutFrameworkData olmak üzere iki farklı görev tipini desteklemektedir. İkinci adımda seçilen herhangi bir betik, üçüncü adımda bu görev tiplerinden biri kullanılarak çalıştırılabilir.

3.17 ifadesinde ikinci adımda örnek olarak **compute_input.py** ve **compute_input2.py** betiklerinin seçildiği görülmektedir. Bu seçimler yapıldıktan sonra üçüncü adımda her betik için **execution type** seçmeli kutusunu kullanarak görev tipini seçebileceği bir panel oluşturulmaktadır. 3.18 ifadesinde verilen örnekte **compute_input.py** için **With Framework Data** seçilmiştir. Bu görev tipinde, kullanıcı, ilgili betiği platform verisini kullanarak görev tanımı yapabilir. Platform verisinden çeşitli kriterler vererek ihtiyacına göre bir veri seti seçtikten sonra, bu veri setini parametre olarak çalıştırmak



Şekil 3.19: With Framework Data görev tipi için detaylı sorgu kriteri girilmesi

istediği Python betiğine verebilir. **Selected Framework Type** ile kullanıcı, platformda bulunan ve kullanmak istediği tipi seçmektedir. 3.1 bölümünde tip yapısı ile ilgili bilgi bulunabilir. **Data Limit** ile, oluşturulacak sorgu kriterlerinde veri tabanından gelen sonuçlara üst limit eklenebilir. **Start Date** ve **End Date** kullanılarak, oluşturulacak görev için spesifik bir tarih aralığındaki veri seti kullanılabilir. 3.19 ifadesinde kullanıcı **Add Query Criteria** butonuna bastıktan sonra açılan bir pencere gösterilmektedir. Bu pencerede seçilen tiplere ait kolonlar düzenlenebilir, bunlardan parametrede yer alması istenmeyen kolonlar çıkartılabilir veya **Field Criteria** alanı kullanılarak filtreleme yapılabilir. Örneğin, 3.19 ifadesinde **windspeed** alanı 30'dan küçük olan alanlar için bir kriter oluşturulmak isteniyorsa, yanına **<30** yazılabilir, bu operatör sunucu modülünde MongoDB'de karşılığı olan bir operatöre dönüştürülmektedir. Benzer şekilde **>, <=, >=, =, !=** gibi operatörler de daha spesifik bir filtreleme için kullanılabilir.

Şekil 3.20: Without Framework Data görev tanımlaması

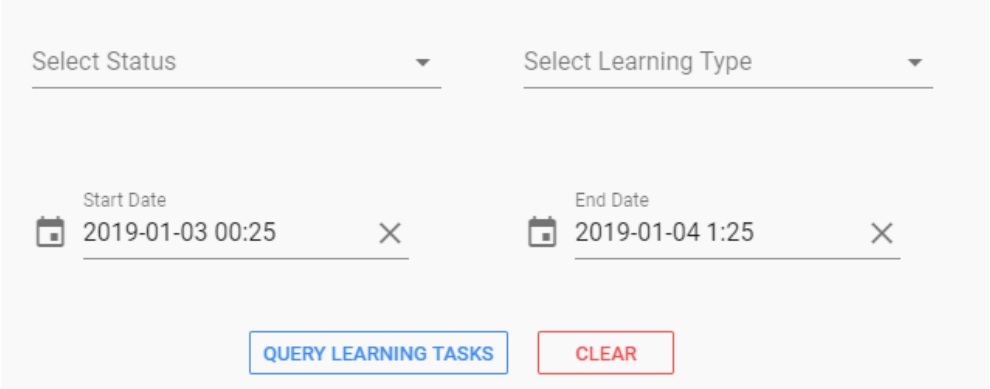
3.20 ifadesinde, diğer görev tipi olan **Without Framework Data** görülmektedir. Bu örnek için **compute_input2.py** betiği kullanılmıştır. Şekilde görüldüğü üzere bu görev tipinde herhangi bir sorgu kriteri bulunmamaktadır. Bu görev tipiyle platform verisi kullanılmadan, platformun asenkron görev yapısı kullanılarak herhangi bir betik üzerinde görev tanımlanabilir ve bunların sonuçları takip edilebilir.

Create Learning Task butonuna basılarak, görevler oluşturulabilir. Görevler oluşturulduktan sonra bunların takibi **Active Tasks** sayfasından yapılabilmektedir.

3.2.5 Görevlerin Takip Edilmesi

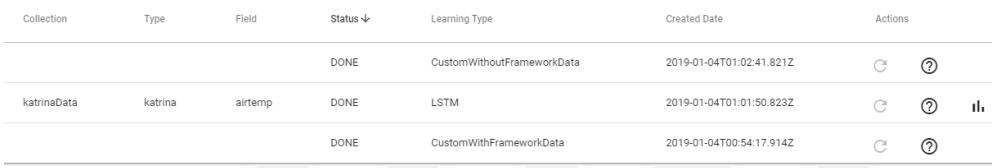
MISIoT, kullanıcıların oluşturdukları öğrenme görevleri için, görevlerin durumunu, hangi parametreler ile oluşturulduklarını, eğer tamamlandıysa sonuçlarını takip edebilmelerini sağlayan bir görev yönetim sayfası sunmaktadır. Platformun arayüz üzerinden kullanılabilen diğer kısımlarında olduğu gibi, herhangi bir REST istemciyle görevler arayüz modülüne ihtiyaç duymaksızın sorgulanabilir.

3.21 ifadesinde, kullanıcı **ActiveTasks** sayfasına girdiğinde karşılaştığı sorgulama kriterleri görülmektedir. **Select Status** ile to_be_processed, processing, done, failed durumlarından biri seçilerek görev o anki durumuna göre sorgulanabilir. **Select Learning**



Şekil 3.21: Görev yönetim ekranı genel görünüm

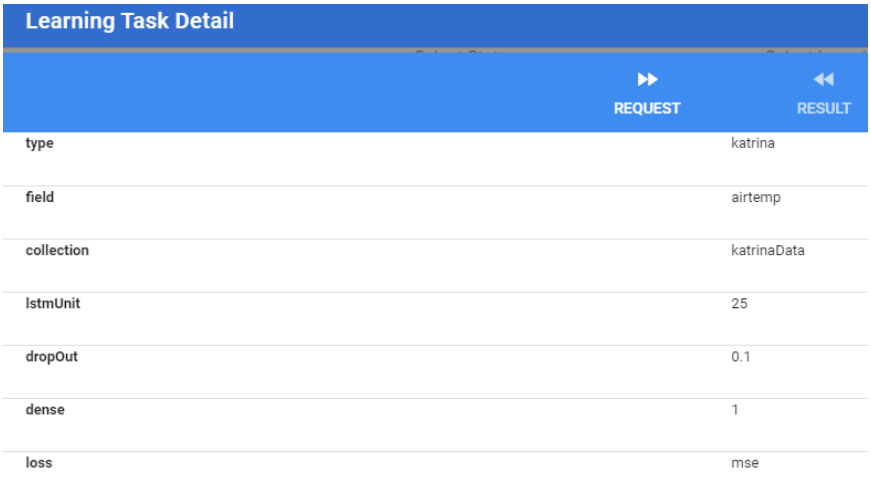
Type ile, platformun desteklediği ve 3.1.5 bölümünde bahsedilen görev tiplerinden biri seçilerek sorgu kriteri olarak eklenebilir. **Start Date ve End Date** seçenekleri ile spesifik bir zaman aralığındaki görevler sorgulanabilir.



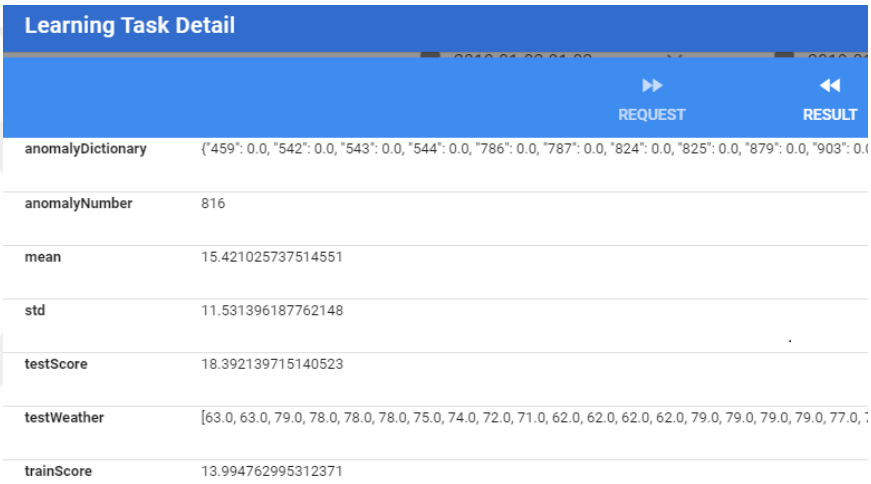
Şekil 3.22: Sorgulama sonucu gösterilen tablo

3.22 ifadesinde kullanıcı **Query Learning Tasks** butonuna basınca gelen sorgu sonuçları görülmektedir. Tüm görev çeşitlerinden birer örnek görebilmek amacıyla, sorgulama öncesi üç görev tipinden birer öğrenme görevi oluşturulmuştur. Görev tiplerini bulmak için atılan sorguda, arayüzün performanslı çalışması için sayfalama yöntemi kullanılmıştır. Teorik olarak milyonlarca eski öğrenme görevi olabileceğinden, kullanıcının her sorgusunda tüm verileri getirmek, sunucu üzerinde gereksiz bir yük oluşturabilir. Bunu engellemek için kullanıcının o an bulunduğu tablo sayfasına göre bir sorgu aralığı oluşturulup, veri tabanı bu şekilde sorgulanmaktadır. Örneğin, kullanıcı sorgu sonuçlarının bulunduğu tabloda üçüncü sayfadaysa ve sayfa başına 8 kayıt gösteriliyorsa, veri tabanı sorgusu da sadece 16 ile 24. kayıtları çekecek şekilde atılmaktadır. Bu şekilde milyonlarca kayıt olsa bile sadece belli bir kısmı gösterildiğinden, performans sorunları yaşanmamaktadır.

3.22 ifadesinde kullanıcının görevi oluşturduğu tarih, görevin durumu ve eğer mevcutsa hangi tip ve kolonlar kullanılarak oluşturulduğu görülmektedir. **Learning Type** kolonunda, görevin tipi okunabilir. **Actions** kolonundaki **?** butonu ile, ilgili görev hakkındaki detaylı bilgiler görülebilir. Görevin hangi parametrelerle çalıştırıldığı, hangi dosya üzerinde oluşturulduğu bilgisi ve görev sonucunda elde edilen çıktı bu bölümde gösterilmektedir. LSTM özelinde **?** butonunun yanında yer alan grafik simgesine tık-



Şekil 3.23: LSTM detaylı bilgi ekranı istek sekmesi



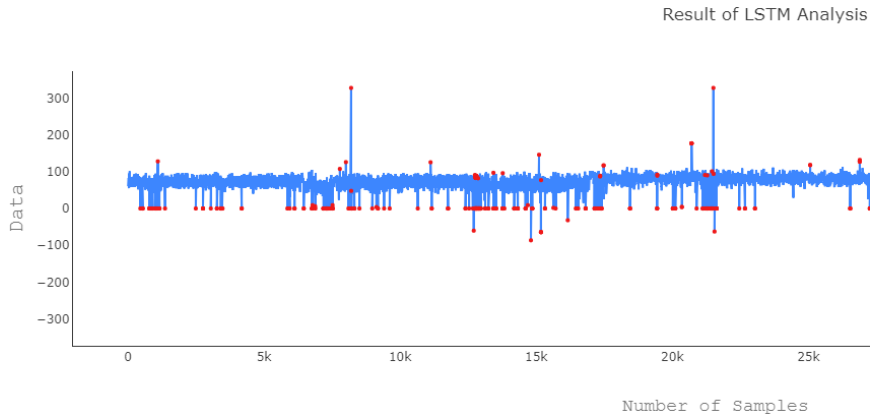
Şekil 3.24: LSTM detaylı bilgi ekranı cevap sekmesi

lanarak, tamamlanan LSTM görevleri için grafiksel analiz de yapılabilir.

3.23 ifadesinde LSTM görevine ait detay ekranı görülmektedir. LSTM özelinde, görevin oluşturulduğu parametreler ve hangi veri tabanı kriterleri kullanılarak oluşturulduğu bu sekmede görülebilir. 3.24 ifadesinde ise, oluşturulan LSTM görevi tamamlandıktan sonra, öğrenme modülünden elde edilen sonuçlar görülmektedir. LSTM spesifik **anomali sayısı, ortalama, standart sapma, öğrenme ve test skorları** gibi veriler görülebilmektedir.

3.25 ifadesinde, LSTM öğrenme görevi sonucunda yapılan ve 3.3 bölümünde anlatılan anomali analizinin grafiksel gösterimi bulunmaktadır. Veri setindeki anomali noktaları kırmızı ile gösterilmekte olup, yakınlaştırma, uzaklaştırma, grafiği PNG formatında kaydetme gibi işlevler grafik ekranında yapılabilmektedir.

3.26 ifadesinde **CustomTaskWithFrameworkData** görev tipindeki **Request** ekranı



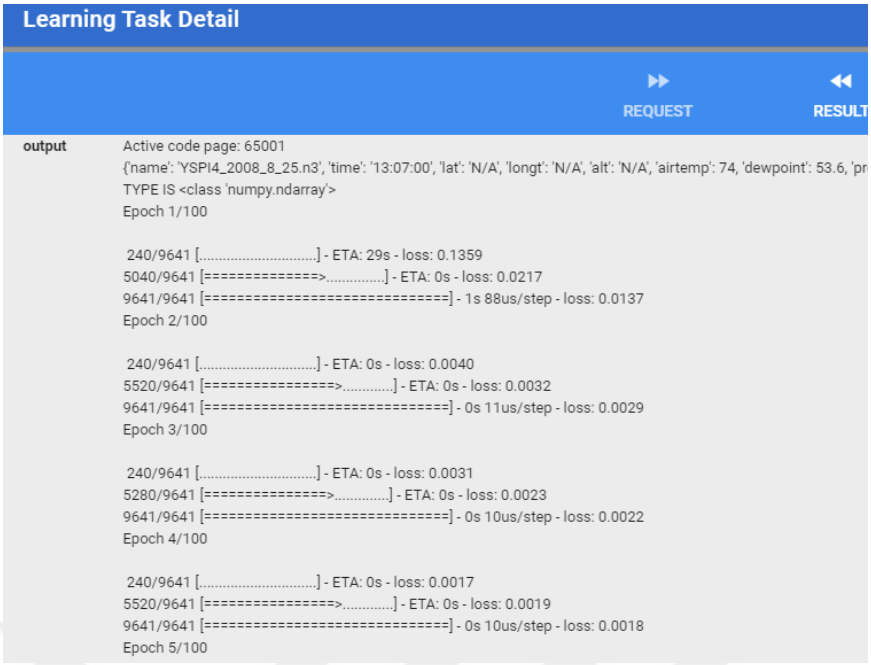
Şekil 3.25: LSTM grafiksel analiz

Learning Task Detail

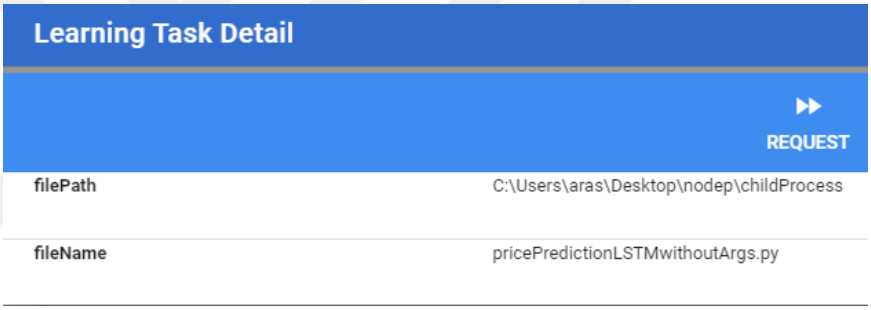
REQUEST

filePath	C:\Users\aras\Desktop\nodep\
fileName	pricePredictionLSTM.py
queryFields	{ "startDate": "2018-12-01T01:19 "endDate": "2019-01-04T02:19: "type": "katrina" }
typeSpecificFields	{ "name": "", "time": "", "lat": "", "longt": "", "alt": "", "airtemp": "", "dewpoint": "", "precip": "", "pressure": "", "relativeh": "", "winddirection": "", "windgust": "", "windspeed": "", "visibility": "" }
queryLimit	0

Şekil 3.26: CustomTaskWithFrameworkData bilgi ekranı istek sekmesi



Şekil 3.27: CustomTaskWithFrameworkData bilgi ekranı cevap sekmesi

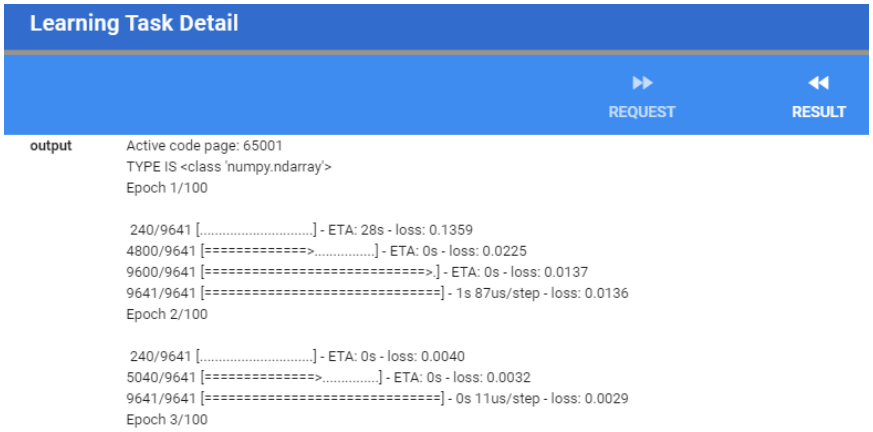


Şekil 3.28: CustomTaskWithoutFrameworkData bilgi ekranı istek sekmesi

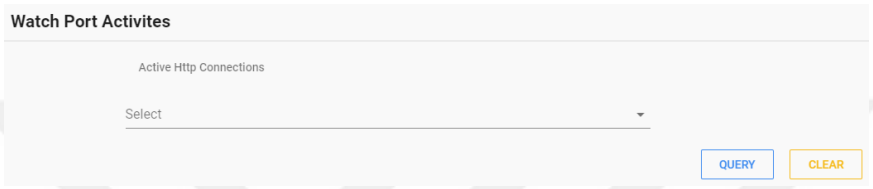
görülmektedir. Bu ekranda görevin oluşturulduğu Python betiğinin ismi, lokasyonu ve bu betiğin hangi parametrelerle çalıştırılacağı, kaç tane parametre geçildiği gibi veriler detaylı olarak görülebilmektedir. 3.27 ifadesinde ise betiğin çalışması sonucu oluşturduğu çıktının bir kısmı gösterilmektedir. Betiğin içine örnek olarak gösterilmek amaçlı, gönderilen parametrenin bir bölümünü konsola yazdıran bir kod eklenmiştir, çıktının üst bölümündeki JSON objeler sunucu tarafından, kullanıcının kriterleri sonucunda betiğe aktarılmıştır. Kullanıcı kendi betiğinde basit bir modifikasyonla bu parametreleri kullanabilir ve makine öğrenmesini bu parametreler üzerinden yapabilir.

3.28 ifadesinde **CustomTaskWithoutFrameworkData** görev tipindeki **Request** ekranı görülmektedir. Bu görev tipi platforma ait veriyi kullanmadığından, doğrudan asenkron olarak çalıştırılmaktadır. Bu nedenle sadece ilgili Python betiğinin konumu görülmektedir. 3.29 ifadesinde yine çalışan betiğin sonucu görülebilmektedir.

CustomTaskWithFrameworkData ve CustomTaskWithoutFrameworkData görev



Şekil 3.29: CustomTaskWithoutFrameworkData bilgi ekranı cevap sekmesi



Şekil 3.30: Grafiksek analiz ekranı genel görünüm

tiplerinde, betik herhangi bir sebepten dolayı çalışmadıysa, alınan hata mesajı **Response** sekmesinde detaylı olarak görülebilir.

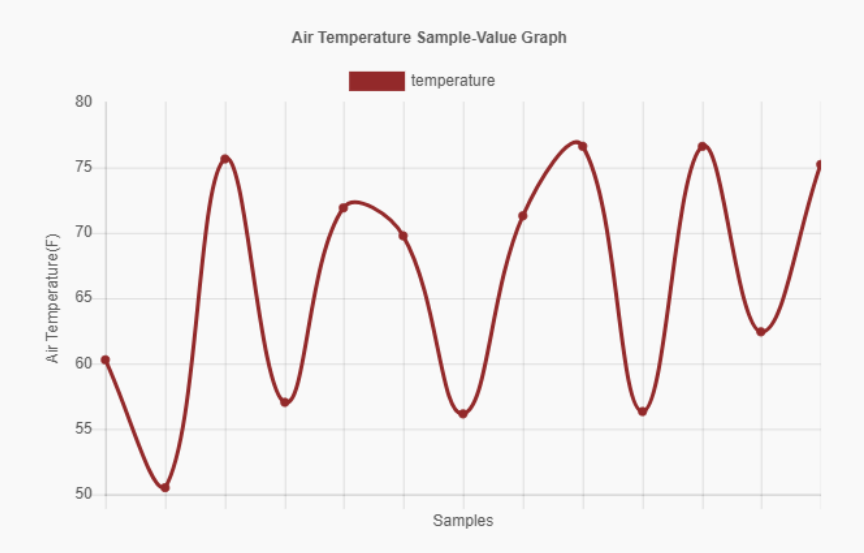
3.2.6 Portların Grafiksel Takibi

MISIoT, kendisi üzerinde açılan portlar üzerindeki veri akışlarını grafiksel analiz ekranı üzerinden gösterebilmektedir. Açılan soket ve REST bağlantıları üzerindeki veri akışları ve bu bağlantılara gelen verilerin tipleri gerçek zamanlı olarak takip edilebilir.

3.30 ifadesinde ekranın genel görünümü görülebilir. **Active Http Connection** ve **Active Socket Connection** kutularından, veri akışı izlenmek istenen portlar seçilebilir. İstenilen portlar seçildikten sonra, **Query** butonuna basılırsa, bu portlardaki mevcut veri akışı gözlenebilir. 3.31 ifadesinde, 4000 portundaki dinamik veri akışı görülmektedir. Bu porta gelen veride bulunan **temperature** alanı dinamik olarak ve her saniyede yenilenecek biçimde gösterilmektedir.

3.3 Öğrenme Modülü

MISIoT, kullanıcıların kendi betiklerini çalıştırmalarına imkan sağlamasına ek olarak, bir öğrenme modülü sunmaktadır. Bu modül kullanılarak, zaman serisi şeklinde verilen bir veri üzerinde LSTM(Long Short Term Memory) algoritması ile anomali analizi ya-



Şekil 3.31: Veri akışının gerçek zamanlı izlenmesi

pılabilir. Yapılan analizin sonuçları arayüz modülü kullanılarak veya doğrudan sunucu modülüne REST isteği atılarak görülebilir.

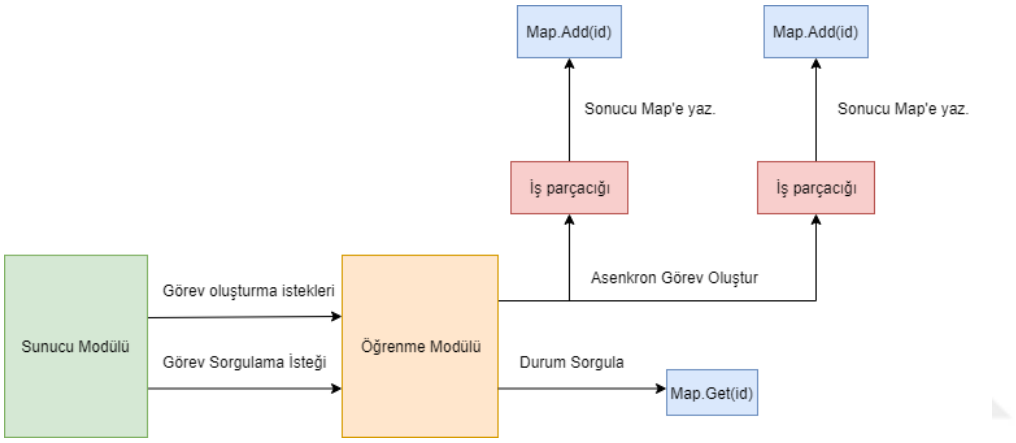
3.3.1 Mimarisi

Makine öğrenmesi yapılan veri setinin işlenmesi, bu veri seti üzerinde model oluşturulması potansiyel olarak fazla zaman alabileceğinden, bu modül asenkron olarak tasarlanmıştır. Sunucu modülü, LSTM üzerinden bir öğrenme isteği tanımladığında, bu istek veri tabanına kaydedilir ve kaydın id'si veri tabanında oluşur. 3.1.4 bölümünde bahsedilen cron zamanlayıcı çalıştığında, **to_be_processed** durumundaki kayıtları sorgulayıp, ilgili kaydı öğrenme modülüne gönderir. Sunucu modülünden gelen istek içinde, öğrenme görevinin içerdiği ve LSTM algoritmasının çalıştırılması için gerekli olan spesifik alanlar ve görevin veri tabanı kayıt id'si bulunmaktadır. Bunlara ek olarak, sunucu modülünde kullanıcının seçimleri doğrultusunda oluşturulan zaman serisi de istek içinde bulunmaktadır. LSTM spesifik alanlara örnek olarak **lstmUnit, dropOut, dense, lookBack, loss, optimizer, epoch, batchSize, verbose** gösterilebilir.

Sunucu modülünden alınan id kullanılarak, gelen istek, paralel işleme olanak verebilmek için yeni bir iş parçacığına atanır ve isteğin id'si daha sonra kullanılmak üzere Map yapısında **completed** isimli, görevin bitip bitmediğinin anlaşılmasını sağlayan bir değişkenle beraber saklanır. Görev oluşturulduktan hemen sonra, sunucu modülüne görevin oluşturma isteğinin alındığını ve görevin başarıyla oluşturulduğunu belirtmek için **200** mesajı dönülür. Sunucu modülü kendi tarafında, 200 cevabını aldıktan sonra görevin durumunu **processing**'e çekecektir. Görev tamamlandıktan sonra, daha önce

kaydedilen id ile map üzerinden kayıt tekrar bulunur. Standart sapma, ortalama, anomali noktaları, eğitim ve test skorlarının bulunduğu bir sonuç objesi bu id'ye karşılık gelen değere atanıp, **completed** değişkeni **true** olarak güncellenir. Sunucu modülü, oluşturulan görevin bitip bitmediğini **completed** üzerinden anlayabilir.

Sunucu modülü, 3.1 bölümünde anlatıldığı üzere, oluşturduğu görevin takibi için yine cron zamanlayıcı yapısını kullanmaktadır. Cron zamanlayıcı her çalıştığında, durumu **processing** olan kayıtları id'leri ile sorgular. Öğrenme modülü, eğer ilgili id'ye karşılık gelen kaydın **completed** değişkeni **true** ise, aynı kayıt içinde bulunan sonuç objesini sunucu modülüne cevap olarak döner. Sunucu modülü, sonucun olduğu bir cevap alırsa kendi tarafında kaydın durumunu **done**'a çekecektir. Bu statüye sahip kayıtlar daha sonra ekranda anomali ve diğer sonuçların gösterimi için kullanılabilir.



Şekil 3.32: Öğrenme modülü genel mimari

Öğrenme modülü, üzerinde çalıştığı sistemin gücüyle orantılı olarak, çoklu iş parçacığı yapısında çalışacak biçimde tasarlanmıştır. Dolayısıyla sunucu modülünden gelen her görev tanımlama isteği için ayrı bir iş parçacığı atanmaktadır ve bunlar birbirinden bağımsız, paralel biçimde çalışabilmektedirler. Her bir görevin mapde farklı bir id'si bulunduğundan, diğer parçacıkların çalışmasından bağımsız olarak, işi biten kayıt sorgulanabilir ve kullanılabilir. 3.32 ifadesinde öğrenme modülünün yapısı görülebilir.

3.3.2 Anomali Analizi

MISIoT öğrenme modülünde anomali analizi için LSTM algoritması kullanılmaktadır. LSTM Recurrent Neural Network(RNN)'ün bir özel versiyonu olup, zaman serisi halinde gelen veri üzerinde çalışmaktadır. RNN algoritmalarının temelinde, önceki verileri hatırlayıp, yeni gelen veriyi bunlara dayanarak tahmin etmek yatmaktadır. LSTM modeli çeşitli LSTM üniteleri içermektedir. Bu üniteler giriş, çıkış ve unutma kapısından oluşmakta olup, veri akışını kontrol etmektedir. Her bir hücre istenilen değerleri

belirli bir süre tutabilmektedir. LSTM hücreleri bir araya gelerek neural network katmanlarını oluşturmaktadır. MISIoT özelinde, herhangi bir zaman serisi üzerinde bu yöntem uygulanabilmektedir. 4 bölümünde, Katrina kasırgasına ait sıcaklık verileri üzerinde bu algoritma uygulanmış olup, eski sıcaklık verilerine dayanarak yenileri tahmin edilmiş ve tahmin verisi ile orijinali arasındaki farktan anomali analizi yapılmıştır.

LSTM kullanılarak tahmin edilen veri seti ile orijinal sonuçların karşılaştırılıp, anomali analizi yapılabilmesi için **üç sigma limitinden** yararlanılmıştır. Bu yöntemin **standart sapma(standart deviation) ve ortalama(mean)** gibi çeşitli alt kavramları bulunmaktadır. Standart sapma, aynı zamanda sigma olarak tanımlanmakta olup, bir veri setindeki değişkenliği(varyans) ölçmek için kullanılmaktadır, varyansın karekköüdür. Ortalama ise veri setinin ortalama değerini ifade eder. Sigma kullanılarak, bir veri setinde verilerin ortalama değere göre ne kadar değişim gösterdiği anlaşılabilir. Üç sigma limitinde ise verilerin, veri setinin ortalama değerinden maksimum üç sigma uzaklıktaki bir alan içine düşüp düşmediğine bakılır. Dolayısıyla, herhangi bir veri, ortalamaya üç sigma değerinden daha uzaktaysa, bu veri anomali olarak sınıflandırılabilir.

Bu kavramların daha iyi anlaşılabilmesi ve MISIoT'daki kullanım amacına uygun olarak, MISIoT'da kullanılan Katrina kasırgasına ait veri setinden ufak bir parça üzerinde gösterilecektir. Bu örnekte veri setinde orijinal veri ve LSTM algoritması tarafından tahmin edilen değerlerin üzerinden bir anomali analizi yapılacaktır.

- LSTM tarafından tahmin edilen ve orijinal veri çift olarak verilmektedir: **(55.8, 75), (22.4, 46.4), (63.6, 41), (52.9, 59), (48.5, 62.6)**
- İlk adımda bu çiftlerin farklarından yeni bir seri oluşturulur. Oluşturulan seri **19.2, 24, 22.6, 6.1, 14.1**'dir.
- İkinci adımda bu seri üzerinden ortalama değer hesaplanır. Bu seri için bu değer **17.2**'dır.
- Üçüncü adımda, ikinci adımdaki ortalama kullanılarak, serinin varyansı hesaplanır. Setin her bir elemanının ortalamayla farkının karesi bulunur, bunlar toplandıktan sonra setteki eleman sayısına bölünerek varyans hesaplanır. Bu veri seti için değer **53.05**'dir.
- Dördüncü adımda Varyansın karekökü alınarak sigma bulunur. Bu değer **7.28**'dur.
- Üç sigma limitini hesaplamak için; $\sigma * 3 + ortalama$ formülü kullanılabilir. Bu değer örnekte verilen veri seti için **39.04**'dir.

Bu örnekte oluşturulan fark serisindeki herhangi bir değer 39.04'den büyük olduğundan bir anomali bulunmamaktadır. Eğer veri setinde bu değerden büyük bir değer olsaydı, anomali olarak gösterilecekti. MISIoT, anomali hesaplarken bu yaklaşımı kullanmaktadır. Zaman serisi üzerinde LSTM çalıştırılması sonucu elde edilen tahmin verileri, orijinal verilerle karşılaştırılarak üç sigma limiti bulunmaktadır. Daha sonraki adımda tahmin edilen veriyle orijinal olanlar karşılaştırılıp, limitin aşılıp aşılmadığına bakılmaktadır. Eğer limit aşılırsa ilgili nokta anomali olarak işaretlenmektedir.

3.4 Kullanılan Teknolojiler

Platform yazılırken, ihtiyaca göre birçok farklı teknoloji birlikte kullanılmıştır ve modüler bir mimari tercih edilmiştir. Bu bölümde kullanılan teknolojiler detaylıca anlatılacak, avantajları ve neden kullanıldıkları hakkında bilgi verilecektir.

3.4.1 Node.js

Node.js[12], asenkron ve olay odaklı(event driven) çalışan bir Javascript platformudur ve büyük ölçekli, aynı anda birçok isteği işleyebilecek kapasitededir. Bloklayıcı IO(Giriş-Çıkış) ve yoklama(polling) yöntemlerine alternatif olarak geliştirilmiştir ve olay döngüsü(event loop) yapısını kullanmaktadır. Node.js'in temel aldığı bazı prensipler bulunmaktadır, bunlar;

- ***Küçüklük:*** Node.js, mümkün olan minimum fonksiyonaliteyi sağlayıp, geri kalan işlevleri başka modüllere bırakmaktadır. Bu yaklaşım sayesinde, platformu kullanan programcılar, daha rahat biçimde platform üzerinde denemeler yapma veya platforma yeni özellikler kazandırma imkanı bulmaktadırlar. Böylelikle, platformun gelişimi daha hızlı olmaktadır. Ayrıca platform üzerinde sadece gerekli olan fonksiyonalitelerin sağlanmasıyla, bakımı ve geliştirilmesi daha kolay olmaktadır.
- ***Küçük Modüller:*** Unix'in temel prensiplerinden biri, olabildiğince küçük ve spesifik olarak bir problemi çözmeyi hedefleyen programları temel almasıdır. Node.js'in modül sistemi de, bu prensip göz önüne alınarak inşa edilmiştir. Modüller, program özelinde kullanılabilen ve spesifik bir işlevi platform'a kazandırmayı hedefleyen iş parçacıkları olarak tanımlanabilir. Modül sisteminin en önemli özelliklerinden biri, hem kod büyüklüğü olarak hem de çözmeyi hedefledikleri problem olarak küçük boyutta olmalarıdır. Modüllerin bağımlılık so-

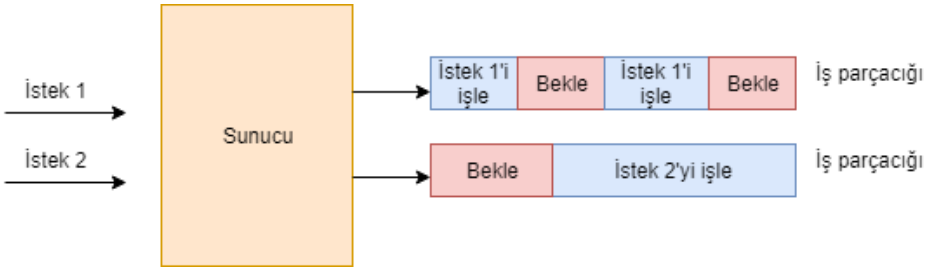
[12]https://nodejs.org/en/about/

runları(dependency) yaşamamaları için, her birinin ayrı bağımlılıkları vardır, bu sistemle beraber yüzlerce modul ve bağımlılık içeren bir projede bile, herhangi bir bağımlılık karmaşası olmadan kod yazılabilir. Bu yapının bir diğer avantajı, bağımlılık sorunları olmadığından, sadece tek bir fonksiyonu olan, küçük ve anlaşılır modüller yazılabilir.

- ***Tek İşlevi Olan Modüller:*** Tek sorumluluk(single responsibility) prensibi, günümüzde gittikçe yaygınlaşmakta olan bir yazılım prensibidir. Bu prensibe göre, herhangi bir fonksiyonun, kütüphanenin, hatta eğer mikroservis mimarisi kullanılıyorsa programın, yalnızca tek bir sorumluluğu olması ve bunu kusursuz bir biçimde yerine getirmesi gerekmektedir. Node.js'in modül sistemi de bu yapının sağlanmasını kolaylaştırmaktadır. Yazılan çoğu modülün spesifik bir işlevi vardır, spesifik bir eksikliği gidermek için yazılmıştır ve genelde modülün tek bir başlangıç noktası bulunmaktadır. Bu yaklaşım sayesinde, yazılımcı kullanmayacağı birçok özelliği programına eklemek zorunda kalmaz. Node modülleri, ek özellikler eklenmeye uygun değildir, modüller iç özelliklerin değiştirilmesinin engelleyecek biçimde tasarlanırlar. Böylelikle, modülün kullanımı ve bakımının basit olması sağlanır.

- ***Basitlik:*** Birçok özelliği bir arada bulunduran, karmaşık bir yazılım yerine, daha basit tasarımlı bir yazılım yapmak, yazılımın daha hızlı geliştirilmesi, daha hızlı biçimde kullanıma hazır hale getirilmesi, daha az kaynakla geliştirilmesi ve daha kolay bakımının yapılmasına olanak sağlar. Javascript programlama dili de, kapama(closure), fonksiyon ve nesne değişmezleri(object literals) ile karmaşık sınıf tasarımları yerine basitliğe olanak sağlamaktadır.

Günümüzde programlama için kullanılan çeşitli IO yöntemleri bulunmaktadır. Bunlardan ilki, **standart bloklayıcı IO programlamadır**. Bu programlama yöntemine göre, sisteme herhangi bir IO isteği yapıldığı zaman, isteği yapmaktan sorumlu iş parçacığı(thread), veriyi okuyana kadar program durur. Buna örnek olarak bir soketten veri okuyup, daha sonra bu veriyi kullanmak gösterilebilir. Soketten veri okuma işlemi bitmeden veri kullanılamayacağından, program bloklanır. Bu yöntemle programlanan uygulamalarda, tek iş parçacığının aynı anda birden fazla isteğe bakması mümkün değildir, bu nedenle farklı istekleri karşılayabilmek için çoklu iş parçacığı kullanımı gerekmektedir. Ancak bu yaklaşımdaki sorun, iş parçacığının sadece gelen istekle değil, herhangi bir IO operasyonu ile bloklanabileceği, bu sürede başka bir iş yapamayacağı ve bunlara ek olarak maliyetinin fazla olmasıdır. 3.34 ifadesinde görülebileceği üzere, eğer yapılan istek, bir veri kaynağından veri okumayı gerektiriyorsa, isteğin işlendiği iş parçacığı bloklanmış durumdadır ve yeni istek kabul edememektedir.



Şekil 3.33: Bloklayıcı IO Programlama

İkinci IO yöntemi **bloklamayan IO programlamadır**. Bu yaklaşıma göre, veriyi okumak için bir istek yapıldıktan sonra, bu isteğin cevabı beklenmez. Eğer istek yapıldığında okunmak istenen veri hazır değilse, karşı sistemden bunu belirten bir hazır cevap döner. Bu cevap istemci tarafında bilindiğinden, buna göre bir aksiyon alınır. Bu işlem bir döngü içinde yapılmaktadır ve veri gelene kadar sistem sürekli olarak karşı tarafa verinin durumunu sormak durumundadır. Buna yoklama denmektedir. Bu yöntemin ilk yönteme göre avantajı, iş parçacığı, veri okumak için yaptığı isteğin sonucunu beklemek zorunda olmadığından, herhangi bir bloklama olmaz ve kendisine gelen isteklere aktif olarak cevap verebilir. Bu yöntemin bariz bir dezavantajı ise, veri olmadığı halde sürekli sorduğundan, sistemin işlemci, bellek gibi kaynak kullanımını yükseltmesidir. MISIoT özelinde, öğrenme modülüne yollanan her istekte yeni bir model oluşturulduğundan ve modellerin oluşturulması onlarca dakika alabileceğinden, eğer bu yöntem kullanılmış olsaydı, isteğin sonucu gelene kadar oldukça yüksek bir kaynak kullanımı yapılmış olacak ve platformun verimliliği ciddi oranda düşmüş olacaktı.

Bu iki IO yöntemi yeterince performanslı olmadığından Node.js, daha performanslı olan **olay çoklayıcı(event demultiplexing)** yöntemi kullanmaktadır. Gelen IO istekleri, kuyruk yapısı kullanılarak tutulmakta ve bu isteklerin durumları izlenmektedir. Eğer herhangi bir isteğin cevabı geldiyse, bu istekten gelen veri herhangi bir bekleme olmadan işlenmektedir. Bu yöntemle işleme kısmı herhangi bir bekleme olmadan yapılırken, beklenen nokta isteklerin izlendiği noktadır.

Algoritma 1'da görüleceği üzere, izlenmek istenen veri kaynakları en başta belirlenip, daha sonra bir döngü içinde olay çoklayıcı tarafından izlenmektedir. Bu işlem senkron olarak yapılır, dolayısıyla program **while** kısmında, izlenen kaynaklardan birine veri gelene kadar beklemektedir. Veri geldiği anda, döngüdeki **watch** method çağrısı, ilgili veri kaynaklarının sonuçlarını döner ve döngü içine girilir. Burada kritik nokta, bloklayıcı IO programlamadan farklı olarak, döngü içinde hali hazırda veri olduğundan, **herhangi bir bloklanma olmaz**, kaynaklardaki veriler kullanılır ve program tekrar veri kaynaklarını izleme moduna geçer. Bu döngüye olay döngüsü(event loop) denmektedir ve Node.js'in temelini bu programlama yöntemi oluşturur. Bu yapının avantajı, birden

Algoritma 1 Olay Çoklayıcı

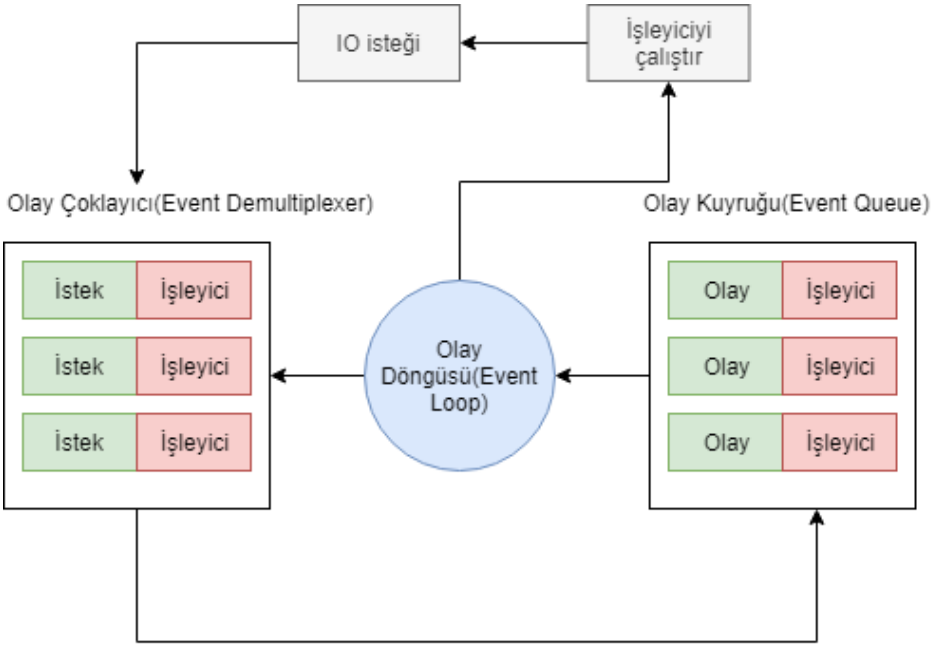
```
1:  function EVENTLOOP()
2:      dataSourceA, dataSourceB;
3:      list.add(dataSourceA);
4:      list.add(dataSourceB);
5:      while(events = multiplexer.watch(list))
6:        foreach(event in events)
7:          data = event.dataSource.read();
8:          if(data == END)
9:            multiplixer.unwatch(event.dataSource);
10:         else
11:           useData(data);
```

fazla iş parçacığı kullanmak yerine, tek iş parçacığı kullanarak, işleri zamana yaymaktır. Bu yöntemle ayrıca çoklu iş parçacığı kullanımının getirdiği parçacıklar arası yarış problemi(race condition) gibi sorunlarla karşılaşılmaz, işlemci daha efektif kullanılır.

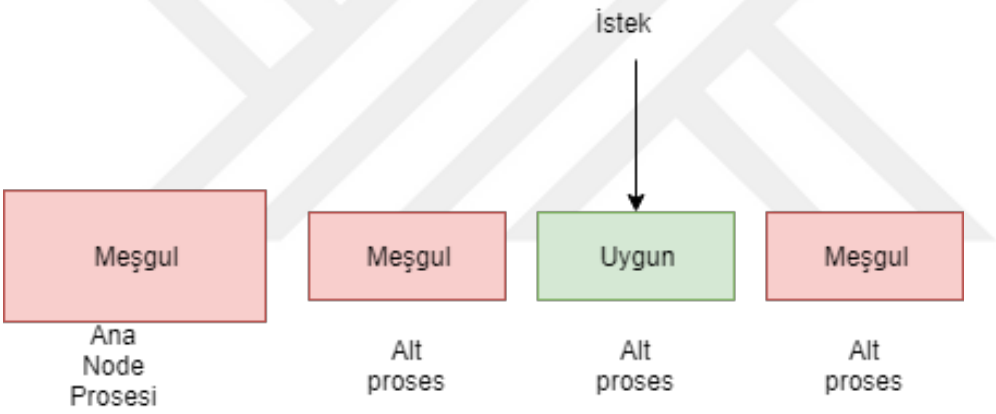
Node.js, olay çoklayıcı baz alarak daha gelişmiş bir algoritma kullanmaktadır. Bu algoritmada, olay çoklayıcıya ek olarak bir de işleyici(handler) kullanılmaktadır. Gelen istek olay çoklayıcıya aktarılırken, bu istek için aynı zamanda bir işleyici tanımlanır. Bu işlem bloklayıcı değildir, istek aktarılır aktarılmaz uygulama devam eder. İstekteki IO işlemi gerçekleştirildikten sonra bu işleyici çalışmaktadır. İstek çoklayıcıya aktarıldıktan sonra, tamamlandığında olay kuyruğuna aktarılır. Bu noktada olay döngüsü, kuyruktaki isteklerin üzerinden geçer ve her birinin işleyicisini çalıştırır. Bu işleyiciler sırasında yeni IO istekleri oluşabilir, bu durumda bu istekler tekrar olay çoklayıcıya aktarılır. Olay çoklayıcıda daha fazla istek kalmadığında, Node.js uygulaması kapanacaktır.

Node.js'in genel mimarisi asenkron istekleri işlemek için oldukça uygun olduğundan, MISIoT yazılırken tercih edilmiştir. Platform aynı anda birçok eş zamanlı isteği işlemek zorunda olduğundan, çalıştığı sistemin kaynaklarını efektif kullanan ve hızlı biçimde cevap dönebilen Node.js tercih edilmiştir.

Tek iş parçacığı ve olay dögüsü kullanımı, iş parçacıklarına özgü yarış durumunu engelleyip daha kolay bir programlama deneyimi sunsa da, eğer işleyicinin yaptığı iş çok uzun sürüyorsa bu durum yine olay döngüsünde tıkanmaya ve Node.js'in yeni istekleri alamamasına sebep olmaktadır. Bu durumun önüne geçebilmek için Node.js, küme(cluster) oluşturmayı desteklemektedir. Küme yapısı, Node.js'in çoklu çekirdeğe sahip bilgisayarların işlemci gücünden faydalanabilmek için kullanmakta olduğu bir yapıdır. Uygulama başlangıcında, uygulamanın üzerinde çalıştığı işlemcinin çekirdek sayısına göre kümeleme ile birbirinin aynısı Node prosesleri açıp, uygulamaya gelen



Şekil 3.34: Node.js genel mimari



Şekil 3.35: Core i5 6500 üzerinde örnek kümeleme

isteğin bu proseslerden en uygununa yönlendirilmesi sağlanabilir. Bunu örneklemek gerekirse Intel Core i5 6500[13] 4 çekirdek 4 iş parçacığından oluşmaktadır. Kümeleme yöntemi kullanılarak yazılmış bir Node.js uygulamasında, bu işlemciye uygun olarak 3 alt Node prosesi açılabilir. 1 proses de ana proses olarak çalışmaktadır, toplamda 4 proses üzerinden gelen istekler karşılanabilir. Şekil 3.35'te bir istek geldiğinde kümeleme yöntemiyle yazılmış bir Node.js uygulamasının nasıl davranacağı görülebilir. Çalıştırılan uygulama, ana proses üzerinden işlemci çekirdek sayısına göre alt prosesleri oluşturduktan sonra, uygulama gelen istekler için hazır duruma geçer. Daha sonra gelen istek, o anda boş olan bir prosese aktarılarak, uygulamanın bloklanmadan çalışması sağlanılır.

[13]https://ark.intel.com/products/88184/Intel-Core-i5-6500-Processor-6M-Cache-up-to-3-60-GHz-

MISIoT yazılırken kümeleme tekniği kullanılarak yazılmıştır. Böylelikle sisteme eş zamanlı gelebilecek sensör verilerinin minimum zaman kaybıyla işlenmesi sağlanmıştır. Ayrıca bu yaklaşım sayesinde, uygulama ne kadar fazla çekirdeği olan bir işlemci üzerinde çalıştırılırsa, o kadar hızlı olmaktadır, bu durum da yazılan platformun Node.js'in temel dayanak noktalarından biri olan ölçeklenebilirlik(scalability) kavramıyla uyumlu olmasını sağlamaktadır.

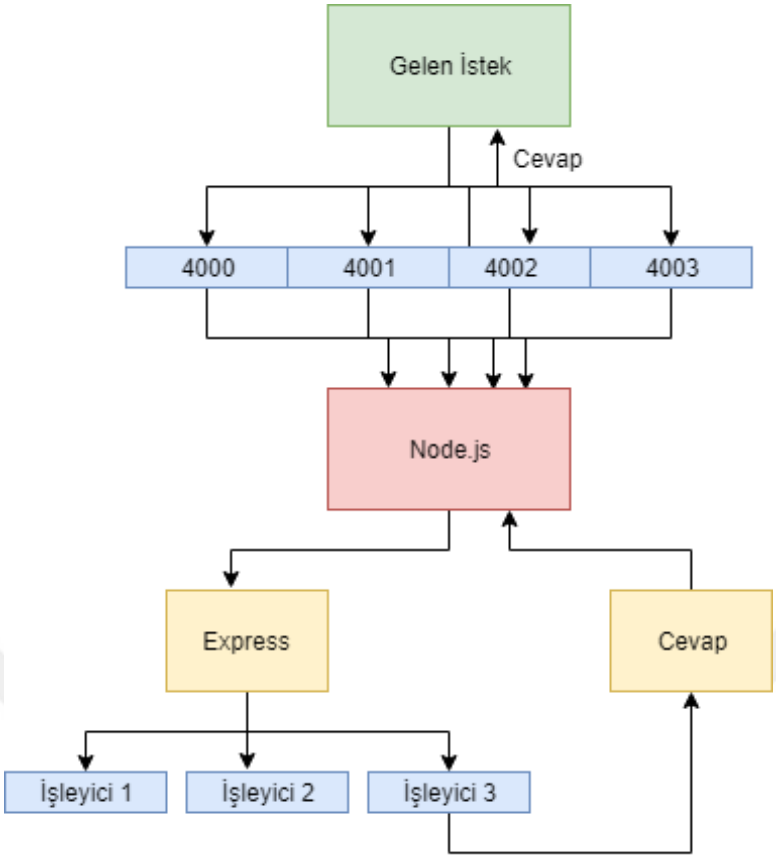
3.4.2 Express.js

Express.js[14], Node.js için yazılmış, küçük bir web platformudur. Web ve mobil platformlar için çeşitli iletişim özellikleri sağlamaktadır. Şekil 3.36'de görüleceği üzere Express.js, Node.js'e ek bir katman sağlamakta, gelen istek önce Node.js üzerinden Express yönlendiricilerinden(router) ilgili olanına aktarılıp, daha sonra oluşturulan cevap Node.js üzerinden istemciye gönderilmektedir. Express.js buna ek olarak, ara katman desteği de sunmaktadır. 3.36 ifadesinde gelen istek doğrudan Express.js yönlendiricisine düşmektedir ancak ara katman sayesinde, bu istek ilgili yönlendiriciye düşmeden önce bir ara katmanda yakalanabilir ve burada ön validasyon yapılıp, eğer istek geçersiz ise yönlendirilmeden sonlandırılması sağlanabilir. Örneğin kimlik doğrulama işlemleri ara katmanlarda yapılıp, eğer istemci yetkili değilse istek kesilebilir ve sunucu kaynakları gereksiz yere meşgul edilmemiş olur. Bunlara ek olarak Express.js, isteği daha anlamlı hale getiren Node.js modüllerinin ara katmanda kullanımına da olanak vermektedir. MISIoT'da REST arayüzü oluşturmak ve istekleri yönetmek için Express.js kütüphanesinden yararlanılmıştır.

3.4.3 Npm

Npm[15] bir paket yönetim sistemidir ve Node.js ile beraber kullanılmaktadır. Node.js tasarlanırken minimalist bir yaklaşım güdüldüğünden, eksik olan özellikler Npm paketleriyle beraber herhangi bir uygulamaya eklenebilir. Npm'in getirdiği bir diğer avantaj ise, her uygulamada kullanılan paketlerin tutulduğu package.json dosyası sayesinde, **npm install** komutu ile bu paketlerin otomatik olarak yüklenebiliyor olmasıdır. Böylelikle, proje aktarılırken projeyle beraber bağımlılıkların verilmesine gerek kalmamaktadır, **package.json** üzerinden ilgili bilgisayarda tüm bağımlılıklar yüklenebilir. MISIoT yazılırken, tüm paket yönetimi Npm üzerinden yapılmıştır.

[14]https://expressjs.com/
[15]https://www.npmjs.com/



Şekil 3.36: Express ve Node.js arasındaki ilişki

3.4.4 Javascript

Javascript(JS), yüksek seviye(high level) ve yorumlanmış(interpreted) programlama dili olup, Github istatistiklerine göre dünyada en büyük kod tabanına sahip olan programlama dillerinden biridir[16]. Yorumlanmış dilin anlamı, Java ve C gibi çalışmadan önce derlenme zorunluluğunun olmamasıdır. Yüksek seviye dil ise, doğrudan bilgisayarla iletişim kurmak yerine, arada büyük bir soyutlama katmanının üzerinden çalışmasıdır.

MISIoT'un, onlarca farklı kaynaktan veri toplaması ve bunları eş zamanlı olarak işleyebilmesi gerekmektedir. Bu işlem senkron biçimde düşük performansta çalışacağından, asenkron bir yapıya ihtiyaç duymaktadır. JS'nin kapama(closure) özelliği, Node.js'in asenkron olay odaklı yapısını daha da güçlendirmektedir. Kapama özelliği sayesinde, fonksiyon çağrısı bitip metod yığın(stack)'den çıkarılsa bile, fonksiyonun içindeki başka bir iç fonksiyon, dıştaki fonksiyona ait değişkenlere erişebilir. Kapamaların neden güçlü bir özellik olduğu MISIoT'dan alınan bir örnekle daha iyi anlaşılabilir;

```
function executePythonShell(requestId) {
```

[16]https://blog.github.com/2018-11-15-state-of-the-octoverse-top-programming-languages/

```
    let py = spawn(`python.exe`, {
        shell: true
    });
    let dataString = '';
    py.stdout.on('data', function (data) {
        dataString += data.toString();
    });
    py.stdout.on('end', async function () {
        await LearningModel.update(requestId, {
            result: {output: dataString, error},
            status: 'DONE'
        });
    });
}
```

Burada 3.4.1 ile detaylıca açıklanan Node.js asenkron olay mimarisi ile JS kapama özelliğinin birlikte kullanımı sayesinde, fonksiyonun farklı istekler tarafından defalarca çağrılsa bile her bir isteği asenkron olarak işlediği görülmektedir. Burada **executePythonShell** fonksiyonu bir Python prosesi açmaktadır. Açtığı Python prosesinden gelen veriyi **data** isimli dinleyicide dinlemekte ve gelen veriyi bir değişkene kaydetmektedir. Benzer şekilde, **end** dinleyicisi, Python prosesi bittikten sonra elde edilen son sonucu veritabanına kaydetmektedir. Burada önemli nokta, iki dinleyici de dış fonksiyonu bloklamamaktadır, dış fonksiyon çağrısı 1 milisaniyeden kısa bir sürede tamamlanmakta, ancak arka planda veri geldikçe, iç fonksiyonlar metoddan bağımsız olarak tekrar tetiklenip, dıştaki fonksiyona ait **dataString** değişkenini kullanıp güncelleyebilip, veri tabanına kayıt yaparken dış fonksiyona ait bir parametreyi kullanabilmektedirler.

MISIoT yazılırken JS'in seçilmesinin çeşitli nedenleri bulunmaktadır. Bunlar incelenecek olursa;

- Arayüz modülünde kullanılan Vue.js platformu JS üzerinde çalışmaktadır. Sunucu tarafında çalışan Node.js de JS kullanılarak programlanabilir. Bunun getirdiği en büyük avantaj, JS bilgisine sahip bir yazılımcının hem arayüz hem de sunucu tarafını kodlayabiliyor olmasıdır ve bunun da zaman avantajı bulunmaktadır.
- JS yaygın bir dil olduğundan dolayı, teknik bir sorunla karşılaşıldığında sorunun çözümüne ulaşmak, dilin yaygınlığı dolayısıyla oldukça kolaydır.

- Proje açık kaynak olacağından, programlama dilinin yaygınlığı sebebiyle başka yazılımcılardan destek gelme ihtimali daha fazladır.

- JS, aynı anda birçok programlama yönteminin kullanımına izin vermektedir. Dil yapısı gereği nesne yönelimli ve fonksiyonel programlama yöntemlerini desteklemektedir. Dolayısıyla tasarım konusunda daha esnek bir yol izlenmesine olanak sağlamaktadır.

- Node.js ile beraber, minimalist yaklaşıma olanak sağlaması bir diğer avantajıdır. JS yazım konusunda esnek bir dil olduğundan, minimum konfigurasyon ve kodlama ile yapılmak istenen işin yapılmasına olanak sağlamaktadır. Örnek olarak, Java'daki gibi her sınıf için bir tanımlama yapmak gerekmemektedir, sadece fonksiyonlarla kod yazılabilir.

- Asenkron işlem için iş parçacığı yönetimi yapılmasına gerek olmaması, dilin asenkron işlemi kapamalar ile yapısı gereği desteklemesi, bunun Node.js'in olay odaklı mimarisi ile harmanlanarak kullanılabilmesi en önemli avantajlarından biridir.

3.4.5 MongoDB

MongoDb[17], çoklu platform desteğine sahip ve döküman odaklı bir NoSQL(Not only SQL) veri tabanıdır. Bu yapıda veri, SQL'de olduğu gibi tablolarda saklanmaz, SQL sorguları üzerinden sorgulanmaz. NOSQL veritabanlarının avantajı, SQL'deki herhangi bir ilişkisel model gerekmediğinden büyük verinin üzerinde çalışılması için daha uygun bir yapıda olmasıdır.

NOSQL veri tabanları dağıtıktır ve birçok farklı serverde paralel biçimde veri işlenmesine uygun şekilde dizayn edilmiştir. Piyasada MongoDB, Hypertable[18], CouchDB[19], Cassandra[20], RavenDB[21], Couchbase[22]. NOSQL veri tabanları anahtar-değer(key-value), kolon odaklı(column-oriented) veya döküman odaklı olabilir. Mongo NOSQL'in ortaya çıkmasında iki ana faktör vardır;

- Dünya genelinde kullanıcılar, sistemler, sensörler gibi veri kaynaklarından üretilen verinin oldukça büyük miktarlarda olması.

[17] https://www.mongodb.com/
[18] http://www.hypertable.org/
[19] http://couchdb.apache.org/
[20] http://cassandra.apache.org/
[21] https://ravendb.net/
[22] https://www.couchbase.com/

Şekil 3.37: CAP teoremini oluşturan üç bileşen. Bu üç özelliği aynı anda tamamen sağlayan bir veri tabanı bulunmamaktadır.

- Üretilen verilerin birbirine bağımlılığının ve karmaşıklığının artması, birçok farklı sistemden birçok farklı formatta veri üretilmesi.

Üretilen verinin çeşitliliği arttıkça bunu klasik ilişkisel veri tabanlarında modellemek daha zor hale gelmektedir. Bu nedenle NOSQL ortaya çıkmış ve yaygınlaşmıştır.

Ayrık veri tabanları CAP(Consistency, Availability, Partitioning) teoremine göre, teorimi oluşturan bileşenlerin aynı anda sadece ikisine sahip olabilir. Veri tabanları arasında bir seçim yapılırken bu bileşenlerde ağırlıklı olarak ihtiyaç olan ikiliyi sağlayan bir veri tabanını seçmek gerekmektedir. Hiçbir veri tabanı bu üç bileşeni de tam anlamıyla sağlayamaz. Bileşenler incelenecek olursa;

- ***Tutarlılık(Consistency)*** Bu maddeye göre, dağıtık veri sisteminde bulunan her düğüm(node), sisteme bir okuma isteği geldiği anda aynı cevabı dönebilecek durumda olmaktadır. Eğer düğümlerden birinde veri güncellenirken bir hata meydana gelirse, işlem diğer düğümlerde de geri alınmaktadır. Bu durum sistemin tutarlılığını ve istek bazında aynı cevabın dönmesini garanti etse de, sistemin isteklere cevap verme sıklığını düşürmektedir.

- ***Bulunurluk(Availability)*** Bu maddeye göre, sisteme gelen her istek başarılı veya başarısız bir cevap almaktadır. Bunun anlamı, dağıtık bir sistemin %100 aktif olma garantisi vermesidir. Sistemdeki herhangi bir düğümün durumundan bağımsız olarak, her istemci isteğine bir cevap olur. Bu yaklaşım sistemin işlevselliğini arttırsa da, iki istemci aynı isteği gönderdiği farklı düğümlerden, farklı

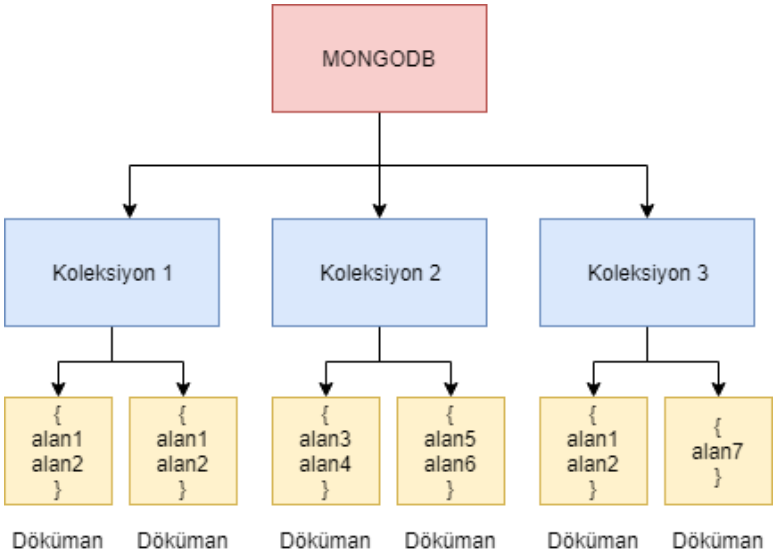
cevaplar alabilmektedir. Bu nedenle, gelme sıklığı yoğun olan gerçek zamanlı akan bir veri kümesi analiz ediliyorsa, düğümler arası farklılıklar olacağından, bulunurluk maddesinin yüksek olduğu bir sistem çok uygun değildir.

- ***Bölünme(Partitioning)*** Bu maddeye göre, çoklu düğümün bulunduğu bir dağıtık sistemde, eğer düğümlerden birinde bir hata olursa diğer düğümlerin isteklere cevap vermeye devam etmesi gerekmektedir. Bu madde sistemin devamlılığı için çok kritik olmakla beraber, dağıtık bir sistemde belli bir ölçüde de olsa mutlaka bulunması gerekmektedir.

MisIOT için seçilen ve bir NOSQL veri tabanı olan MongoDB, CAP'da bulunan üç maddeyi de belli ölçülerde sağlamaktadır. Dağıtık bir sistemde eğer tek bir MongoDB bağlantısı kullanılırsa ve **majority** gibi bir okuma/yazma seviyesi[23] seçilirse, sistem belli bir performans kaybı karşılığında tutarlı çalışmaktadır. Bulunurluk maddesi, MongoDB'de **Replica-Sets**[24] yapısı ile sağlanmakta olup, ana MongoDB veri tabanı, ikincil başka veri tabanlarıyla senkronize olup çalışabilmektedir. Böylelikle, veri tabanlarından birinde bir sorun olduğunda, ikincil veri tabanları üzerinden veri okuma yazma işlemleri yapılabilmektedir. MongoDB bir diğer madde olan bölünme özelliğini de karşılamaktadır. İkincil veri tabanları kendi aralarında bir ana veri tabanı belirleyip, okuma yazma işlemleri bu ana veri tabanı üzerinden gerçekleştirilir ve yazma işleminde ikincil veri tabanı güncellenir.

MongoDB CAP kriterlerini belli ölçülerde karşıladığından, MISIOT'da kullanılmıştır. Bir diğer kullanım sebebi ise, döküman odaklı çalışmasıdır. Bir IoT sisteminde, sisteme farklı alanlara sahip veriler gelebilir. Örneğin, A sensöründen gelen veride nem bilgisi varken, B sensörü sadece rüzgar hızı içerebilir. Dolayısıyla sisteme gelen verinin tipi ve içerdiği alanlar belli olmadığından, bunları herhangi bir ilişkisel veri tabanlarında modellemek oldukça zordur. Belli bir ölçüde modellense bile, sisteme daha sonra farklı tipte bir veri geldiğinde, tabloda ilgili veriyi karşılayacak kolonlar olmadığından verinin kaydedilebilmesi için tasarımın tekrar değiştirilmesi gerekmektedir. Döküman odaklı NOSQL veri tabanı olan MongoDB'de, her gelen veri bir dökümana karşılık gelmektedir ve bu dökümanlar birbirinden farklı alanlar içerebilir. Oluşturulan dökümanlar bir koleksiyon altında gruplanır. Şekil 3.38'de örnek bir MongoDB yapısı görülebilir.

[23] https://docs.mongodb.com/manual/reference/read-concern/read-concern-levels
[24] https://docs.mongodb.com/manual/replication/



Şekil 3.38: MongoDB örnek mimari

3.4.6 Mongoose

Mongoose[25], MongoDB üzerinde çalışan ve veri tabanına şema özelliği, veriyi kaydetmeden önce validasyon, sorgu inşa etme gibi ek özellikler ekleyen bir platformdur. MISIoT'da MongoDB'ye ek olarak kullanılmıştır. MongoDB'nin karmaşık yapısını bir soyut katman üzerinden ve kullanımı kolay olarak kullanıcıya sunmaktadır.

3.4.7 Vue.js

Vue.js[26], arayüz yazmak için kullanılan JS tabanlı bir platformdur. Tek sayfa uygulama(TSU) mimarisiyle yazılmıştır. TSU tamamen tarayıcı içinde çalışır ve kullanıcı sayfalar arasında gezinirken sayfanın yeniden yüklenmesine ihtiyaç duyulmaz. Bu mimaride yazılan uygulamalara Gmail, Google Maps, Facebook, Github örnek verilebilir. Kullanıcı siteye ilk girdiği anda sunucu tarafından sitenin içeriği(html, css ve JS kodları) istemciye gönderilir. Bu noktadan sonra, kullanıcı sayfalar arası gezindiğinde sadece ilgili JS kodları çalışır ve sunucu tarafına herhangi bir istek atılmaz. Sunucu ve istemci arasında sadece ilgili sayfada gösterilecek veri alışverişi yapılır. MISIOT'da bu mimariyi kullanan Vue.js'in tercih edilmesinin çeşitli sebepleri vardır, bunlar;

- Uygulamanın sunucu tarafından ayrı olması aynı zamanda kodların da ayrı olması anlamına gelmektedir. Sunucu ve arayüz kodu birbiriyle iç içe olmadığından, sunucu tarafında kullanılan teknoloji arayüze bağımlı değildir, ihtiyaca göre

[25]https://mongoosejs.com/
[26]https://vuejs.org/

tamamen farklı bir programlama dilinde, farklı teknolojiler kullanarak yazılabilir.

- Vue.js, JS tabanlı bir platformdur. Platformun JS tabanlı olması, dilin yaygınlığı sebebiyle herhangi bir sorunla karşılaşıldığında, çözüme daha kolay ulaşılmasına olanak sağlamaktadır. Sunucu tarafında kullanılan platform Node.js de JS tabanlı olduğundan, sunucu ve istemci tarafında aynı programlama dilinin kullanılması geliştirme maliyetini düşürmektedir. Ayrıca, modern tarayıcılarda TSU'larda hata yakalama ve trafiği izlemeye olanak sağlayan araçlar bulunmaktadır.

3.4.8 Python

Python[27], JS gibi yüksek seviye ve yorumlanmış bir programlama dilidir. Dil geniş kütüphane, kullanıcı desteği ve daha kullanıcı dostu yapısı sayesinde, web programlama, veri analizi, yapay zeka, REST servisi, makine öğrenmesi gibi çok farklı alanlarda kullanılabilir. Ayrıca 3.4.4 bölümünde belirtildiği üzere, Github istatistiklerine göre en çok kullanılan dillerden bir tanesidir. Dilin yaygın olması ve makine öğrenmesi ile ilgili çok fazla kütüphaneye sahip olmasından dolayı, MISIoT'un öğrenme modülünde tercih sebebi olmuştur. MISIoT ayrıca kullanıcılara istedikleri bir Python betiğini platformun üzerinde bulunan verilerle çalıştırma imkanı da sunmaktadır.

3.4.9 Flask

Flask[28], mikro web platformu olarak tanımlanmaktadır ve uygulamaya REST arayüzü eklemek için kullanılmaktadır. Flask'ın mikro olarak görülmesinin sebebi, platformun temel seviye fonksiyonları kullanıcıya sunması ve geri kalan entegrasyonlar için(örn. veri tabanı entegrasyonu) diğer kütüphanelerin Flask'la beraber kullanımına izin vermesidir. Node.js'de olduğu gibi, bunun getirdiği avantaj platformun daha anlaşılır olması, sadece ihtiyaç duyulan kısımların entegre edilerek kullanılabilmesidir. Bu da daha düşük bir geliştirme zamanına imkan vermekte ve kodda daha az hata çıkma olasılığını arttırmaktadır. MISIoT'da öğrenme modülünde kullanılmış olup, sunucu modülünden gelen istekleri karşılamak ve sunucu modülüne istek göndermek için kullanılmıştır.

[27] https://www.python.org/
[28] http://flask.pocoo.org/

3.4.10 Conda

Conda[29], Npm gibi bir paket yönetim sistemidir ve Python modüllerinin yönetimi için kullanılmaktadır. Conda ile farklı Python ortamları, ihtiyaca göre aktif edilebilmekte, çeşitli Python versiyonları arasında geçiş yapılabilmektedir. Conda, arayüze sahip paket yönetim sistemi Anaconda[30] ile beraber gelmektedir. Tercihe bağlı olarak paketler, Conda veya Anaconda üzerinden yüklenebilir. MISIoT yazılırken Windows ortamında geliştirme yapıldığından dolayı ortam ve paket bağımlılıklarında minimum sıkıntıyla karşılaşmak için, Anaconda'nın arayüz programı kullanılmış, bu program üzerinden ilgili Python modülünün kullandığı kütüphaneler oluşturulan ortama kurulup aktif edilmiştir. Her ortamın kendine ait bir Python.exe dosyası vardır ve bu dosya üzerinden çalıştırılan kodlar, ortama ait bağımlılıkları kullanmaktadır. MISIoT'da Python kodlarının çalışmasının istendiği ortamda bulunan Python.exe üzerinden istenen kodlar çalıştırılmaktadır.

3.4.11 Webpack

Webpack[31], projedeki modülleri canlı ortama(production ready) hazır hale getirmek için kullanılan, modüllerin ve testlerin yönetimine olanak sağlayan bir paket yönetim sistemidir. JS kodu yazılarak konfigüre edilebilir. Webpack ile modüller belli bir sırada çalıştırılabilir, program bağımlılıkları yönetilebilir ve karmaşık komutlara kısa takma isimler ayarlanıp, uygulama bu isimler üzerinden çalıştırılabilir. MISIoT özelinde, arayüz uygulaması çalıştırılırken Webpack'den yararlanılmıştır. Arayüz modülünde kullanılan Vue.js tek bir dosya üzerinden çalışmaktadır, Webpack istenilen dizinde Vue.js kodlarını bir araya getirip, canlı ortama konulmaya hazır tek bir dosya halinde sunabilir. Vue.js, Angular.js gibi tek sayfa(single page) uygulamaların yaygınlaşması ve Webpack'in bu uygulamaların yönetimini kolaylaştırması sebebiyle, MISIoT yazılırken tercih edilmiştir.

3.4.12 Xml

Xml(Extensible Markup Language)[32], farklı sistemler arasındaki veri iletişimi standart bir hale getirmek için kullanılan bir dildir. Her sistemin kendine özel bir veri yapısı olabileceğinden, farklı sistemleri konuşturabilmek için, farklı veri kümelerini ortak bir

[29]https://conda.io/docs/
[30]https://anaconda.org/
[31]https://webpack.js.org/
[32]https://en.wikipedia.org/wiki/XML

formatta ifade etmek olarak da açıklanabilir. SOAP protokolünde aktif olarak kullanılmaktadır. MISIoT özelinde sadece kullanıcıya bir konfigürasyon dosyası sağlama amacıyla kullanılmıştır, sistemler arası iletişimde Xml tabanlı SOAP yerine REST tercih edildiğinden sistemler arası iletişim amacıyla kullanılmamıştır.

Bir sonraki bölüm olan **örnek kullanım senaryoları** bölümünde, MISIoT kullanılarak oluşturulan çeşitli kullanım senaryoları anlatılmaktadır. Bu senaryolar, platformun **MISIoT** bölümünde anlatılan modüllerinin hepsi kullanılacak şekilde tasarlanmış olup, kullanıcıya tüm platformu tanıtmayı hedeflemektedir. Senaryolarda, platformun konfigüre edilme aşamasından başlanılıp, 3.1.5 bölümünde anlatılan görevlerin oluşturulması ve takip edilmesi detaylı olarak açıklanmıştır.

4. ÖRNEK KULLANIM SENARYOLARI

Bu bölümde, 3 bölümünde anlatılan platformun desteklediği görev tipleri için örnek kullanım senaryolarından bahsedilmektedir. Her üç görev tipi için de aynı konfigürasyon dosyası ve veri kümesi kullanılmaktadır.

4.1 Platformun Konfigüre Edilmesi ve Veri Yüklenmesi

MISIoT, 3 bölümünde anlatıldığı üzere, soket, REST ve doğrudan CSV üzerinden veri alabilmektedir. Gelen verilerin platformda doğru biçimde sınıflandırılıp saklanabilmesi için, konfigürasyon dosyası kullanımı gerekmektedir. Her üç senaryoda da Katrina kasırgasına ait veri seti[33] kullanıldığı için, bu veri setinin içerdiği kolonlar konfigürasyon dosyasına eklenmiştir. 4.1 ifadesinde senaryolar için yapılan örnek konfigürasyon görülebilir.

```
<collection>
    <collectionName>katrinaData</collectionName>
    <types>
        <type>
            <typeName>katrina</typeName>
            <columns>name,time,lat,longt,alt,airTemp,dewPoint,precip,pressure,relativeH,windDirection,windGust,windSpeed,visibility</columns>
        </type>
    </types>
</collection>
```

Şekil 4.1: Katrina veri seti için yapılan konfigürasyon

Name	Time	Lat	Longt	Alt	AirTemp	DewPoint	Precip	Pressure	RelativeH	WindDire	WindGust	WindSpee
3CLO3_20	0:05:00	46.22	-124	20	N/A	N/A	N/A	N/A	N/A	320	N/A	9
3CLO3_20	0:20:00	46.22	-124	20	N/A	N/A	N/A	N/A	N/A	330	N/A	12
3CLO3_20	0:35:00	46.22	-124	20	N/A	N/A	N/A	N/A	N/A	330	N/A	9
3CLO3_20	0:50:00	46.22	-124	20	N/A	N/A	N/A	N/A	N/A	320	N/A	10
4UT01_20	0:00:00	40.82944	-111.882	5350	72.9	31.8	0	N/A	22	90	16	10
4UT01_20	0:05:00	40.82944	-111.882	5350	65.1	41.4	0	N/A	42	90	8	5
4UT01_20	0:10:00	40.82944	-111.882	5350	64.7	37.8	0	N/A	37	112	10	6
4UT01_20	0:15:00	40.82944	-111.882	5350	65.7	37.3	0	N/A	35	112	11	6
4UT01_20	0:20:00	40.82944	-111.882	5350	66.6	37.3	0	N/A	34	112	11	8

Şekil 4.2: Katrina veri seti

4.1 ifadesinde konfigürasyona göre, platform **name, time, lat, longt, alt, airTemp, dewPoint, precip, pressure, relativeH, windDirection, windGust, windSpeed, visibility** alanlarının hepsini veya bir kısmını içeren bir veri geldiği zaman, bu verinin

[33] http://wiki.knoesis.org/index.php/Main_Page

tip bilgisini katrina olarak belirleyip, gelen verinin içine bu bilgiyi entegre ederek veri tabanına kaydedecektir.

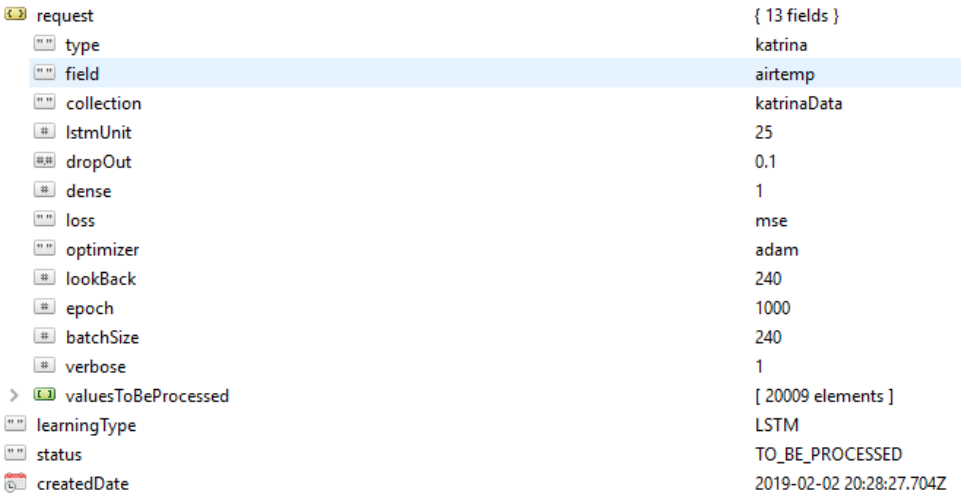
Bir sonraki aşamada, 3.2.2 bölümünde anlatılan CSV üzerinden toplu veri yükleme yöntemiyle, platforma arayüz üzerinden 20009 kayıt içeren veri yüklenmiştir. 4.2 ifadesinde yüklenen verinin örnek bir parçası görülebilir. Konfigürasyon dosyası yazıldıktan ve veri yüklendikten sonra platform her üç senaryo için de kullanıma hazır hale gelmiştir.


```
{
    "_id" : ObjectId("5c55cbb243831f2c580fca22"),
    "__type" : "katrina",
    "name" : "4UT01_2005_8_24.n3",
    "time" : "0:00:00",
    "lat" : 40.82944,
    "longt" : -111.88222,
    "alt" : 5350,
    "airtemp" : 72.9,
    "dewpoint" : 31.8,
    "precip" : 0,
    "pressure" : "N/A",
    "relativehumidity" : 22,
    "winddirection" : 90,
    "windgust" : 16,
    "windspeed" : 10,
    "visibility" : "N/A",
    "createdDate" : ISODate("2019-02-02T19:56:18.739Z"),
    "__v" : 0
}
```

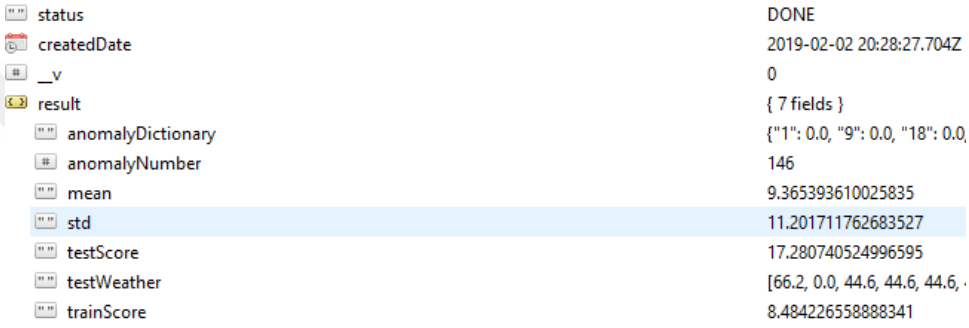

Şekil 4.3: Ekrandan yüklenen Katrina verisinin MongoDB'de gösterimi

Yapılan konfigürasyonda, **name**, **time**, **lat**, **longt**, **alt**, **airTemp**, **dewPoint**, **precip**, **pressure**, **relativeH**, **windDirection**, **windGust**, **windSpeed**, **visibility** alanlarına sahip her bir kayıt, veri tabanına kaydedilirken tip olarak **katrina** alanı eklenerek ve eğer hali hazırda mevcut değilse, **katrinaData** isimli bir koleksiyon oluşturularak bunun altına kaydedilmektedir. 4.3 ifadesinde, arayüz modülünden yüklenen veriye ait bir kaydın, veri tabanındaki gösterimi görülmektedir. Yapılan konfigürasyon dolayısıyla, kayda **__type:katrina** isimli bir alan sunucu modülü tarafından eklenmiştir.

İlk senaryo LSTM senaryosu olup, bu senaryoda öğrenme modülü kullanılmaktadır. Öğrenme modülü LSTM üzerinden anomali analizi yaptığından, doğrudan kullanılmıştır. İkinci senaryo, platforma yüklenen Katrina verisini kullanarak yine Python dilinde, öğrenme modülü kullanılmadan yazılmış LSTM betiğinin sunucu modülü üzerinden görev tanımlanmasıyla gerçekleşmektedir. Üçüncü senaryoda ise, yine platformdan



Şekil 4.4: Oluşturulan LSTM görevinin işlenmeden önce veri tabanındaki kaydı



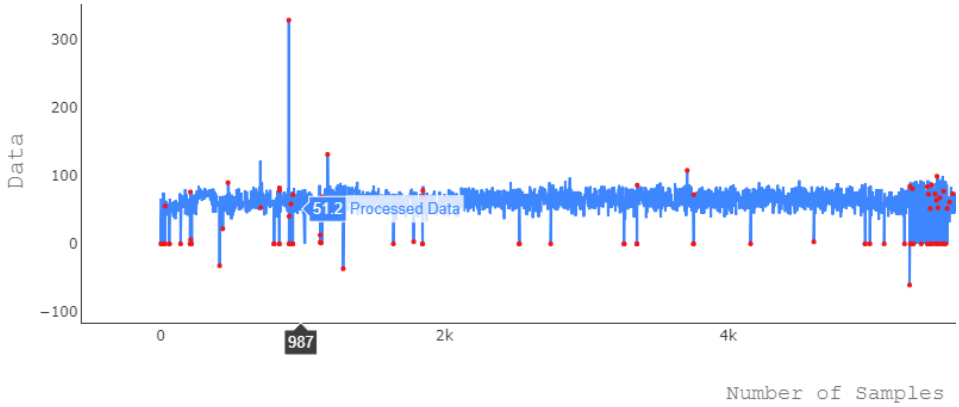
Şekil 4.5: LSTM görevi tamamlandıktan sonra veri tabanı kaydı

bağımsız olarak yazılan LSTM betiğinin, platform verisi kullanılmadan, platform üzerinden doğrudan çalıştırılmasıyla gerçekleştirilmektedir.

4.2 Öğrenme Modülü Kullanılan LSTM Senaryosu

Bu senaryoda, konfigürasyonlar yapıldıktan ve veri yüklendikten sonra ilk adım, LSTM öğrenme görevi oluşturmaktır. Bunun için menüde **Create Learning Task'e** tıklanarak LSTM oluşturma ekranına gelinir. Bu ekran 3.2 bölümünde anlatıldığı gibi, sırasıyla koleksiyon, tip ve bu tipin içerdiği kolonlar seçilmektedir. Koleksiyon olarak **katrinaData**, tip olarak **katrina** ve kolon olarak **airTemp** seçildikten sonra, varsayılan parametreler kullanılarak LSTM görevi oluşturulur.

4.4 ifadesinde, LSTM görevi oluşturulduktan hemen sonra kaydın veri tabanı durumu görülmektedir. **request** alanı içinde LSTM algoritmasının çalışması için gereken parametre ayarları, öğrenme görevinin tipi ve kaydın o anki durumu görülebilir. Kayıt **TO_BE_PROCESSED** durumunda olduğundan, bir sonraki cron zamanlayıcı döngü-



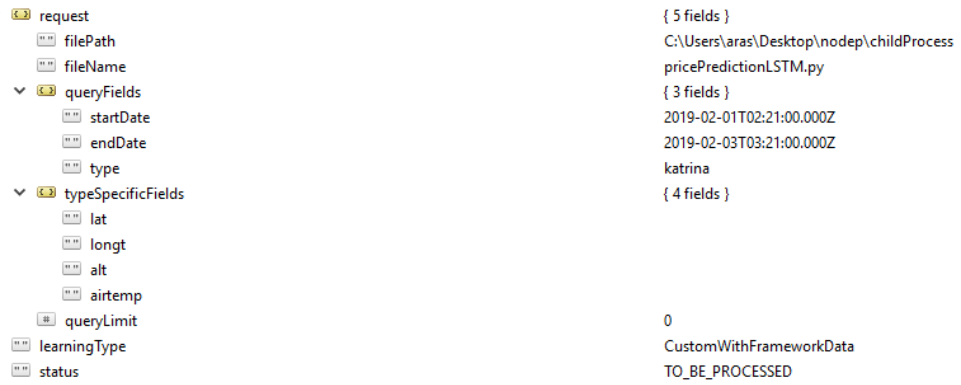
Şekil 4.6: Anomali analizinin kırmızı noktalarla grafiksel gösterimi

sünde ele alınacak ve çalıştırılmak üzere öğrenme modülüne gönderilecektir. Öğrenme görevinin oluşturulduğu kolon **airtemp** kolonu olduğundan, görev oluşturulurken bu kolona ve **katrina** tipine sahip tüm kayıtlar üzerinden geçilerek, bunlar bir seri halinde **valuesToBeProcessed** alanına yazılır. 4.5 ifadesinde ise LSTM görevi tamamlandıktan sonra güncellenen kayıt görülmektedir. Kayda **result** isimli bir alan eklenmiş olup, bu alanda ortalama değer, standart sapma, test ve eğitim sonuçları ve veri setinde anomali noktalarının olduğu yerler bulunmaktadır. Bu alanlar daha sonra arayüz modülünde veriyi görselleştirmek için kullanılmaktadır. Bunlarla beraber, öğrenme görevinin tamamlandığının göstergesi olarak, **status** alanı **DONE** olarak güncellenmiştir. Böylelikle bir sonraki döngüde, sunucu üzerinde çalışan cron zamanlayıcı bu kaydı dikkate almayacaktır.

Senaryonun devamında, ekrandan sonuçlar ve LSTM üzerinden yapılan anomali analizinin grafiksel gösterimi görülebilir. Bu örnekte, sistemde bulunan ve **airtemp** kolonuna sahip 20009 kaydın, N/A olmayan ve bir değer içeren 19280 tanesi öğrenme modülünde işlenmiş, bunların yarısından bir model oluşturulup, kalan kısmı üzerinde bu model kullanılarak tahmin yapılmıştır. Dolayısıyla 4.6 ifadesinde 9640 kayıt üzerinde yapılan anomali analizi görülmektedir. Bu analiz sonucunda 146 anomali noktası bulunmuş olup, bulunan bu noktalar, modelin tahmin yaptığı test verisi üzerinde kırmızı olarak işaretlenmiştir.

4.3 Platform Verisiyle Çalıştırılan LSTM Betiği

Bu senaryoda, kullanıcı tarafından yazılan herhangi bir betik, platform verileri kullanılarak asenkron biçimde çalıştırılabilir ve arayüz modülü vasıtasıyla sonuçları takip edilebilir. Senaryoda, yine LSTM kullanarak anomali analizi yapan bir Python betiği yazılmış, bu Python betiğinin platform tarafından JSON objesi olarak gönderilen veri-



Şekil 4.7: Platform verisiyle oluşturulan görevin veri tabanı kaydı

Şekil 4.8: Platform verisiyle oluşturulan görevin bittikten sonraki veri tabanı kaydı

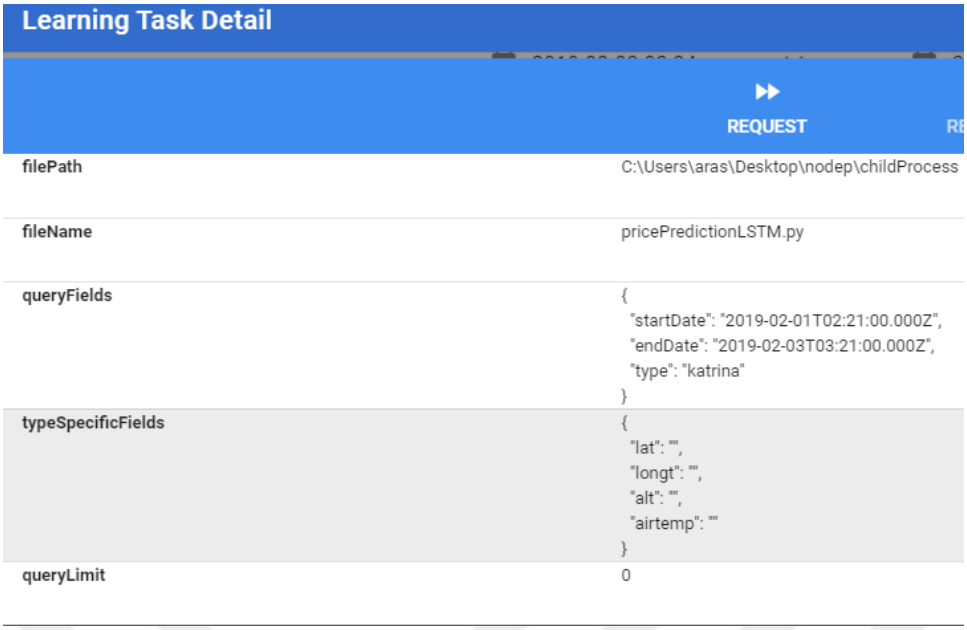
leri kullanabilmesi için şu şekilde bir ekleme yapılmıştır:

```
incomingLines = sys.stdin.readlines()
lines = json.loads(lines[0])
```

Bu kod parçası kullanılarak, platform tarafından betik çalıştırılırken gönderilen veriler Python dizisine çevrilip, herhangi bir işlemde kullanılabilir.

Bu senaryoda ilk adım, menüden **Create Custom Task** seçeneğini seçerek, burada görev oluşturmaktır. Görev oluşturulurken üçüncü adım olan **Manage Files** bölümünde **execution type** olarak **With Framework Data** seçilir. Bu seçim yapıldıktan sonra gelen kutularda **framework type** olarak **Katrina** seçilir. Bu seçenekler platformda oluşturulan verilerin tip bilgisine göre gelmektedir. Tip seçildikten sonra, **Add Query Criteria** butonu görünür olmaktadır. Bu butona tıklayınca gelen pencerede, bu tipin içerdiği alanlar görülebilir. İlgili betik için oluşturulacak parametrelerde olması istenen alanlar işaretlenir, kriter verilmek istenirse >,<, = işaretleri kullanılarak kriterler verilir ve **Save Criteria** butonuna basılarak değişiklikler kaydedilir. Verinin oluşturulacağı zaman aralığı **Start Date** ve **End Date** alanları kullanılarak sınırlandırılabilir. Eğer kullanılacak veriye limit getirilmek isteniyorsa **Data Limit** verilebilir. **Create Learning Task** butonuna basılarak görev oluşturma işlemi tamamlanır.

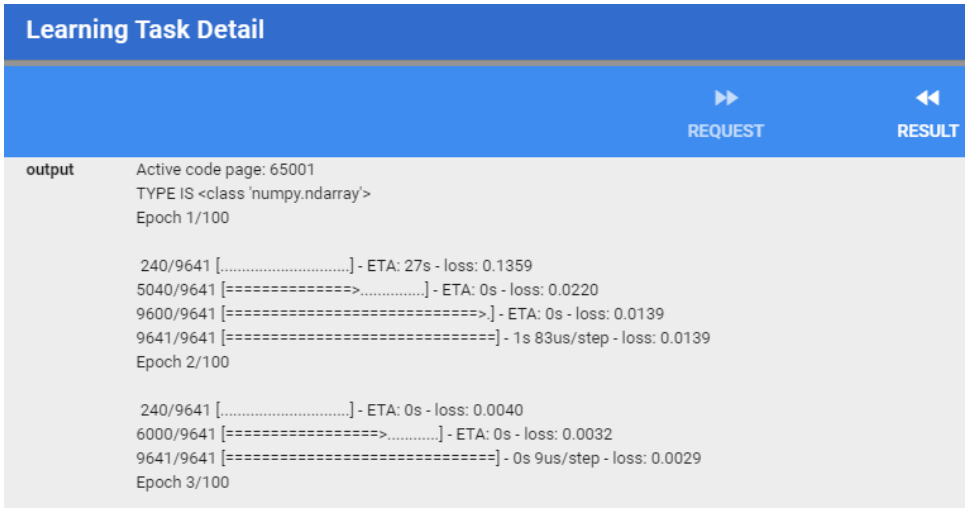
4.7 ifadesinde, görev oluşturulduktan sonra oluşan veri tabanı kaydı görülmektedir. Bu kayıtta **request** alanında dosyanın konumuna ek olarak, kullanıcının görev oluştururken kullandığı kolonlar, kısıtlamalar ve veriyi limitleyen **queryLimit** alanı görül-



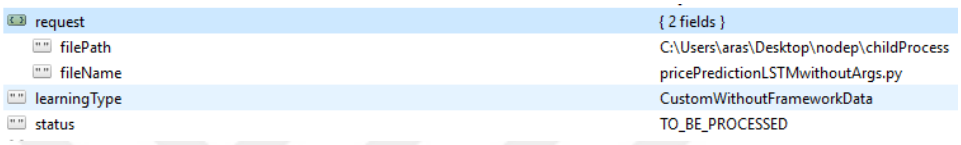
Şekil 4.9: Oluşturulan göreve ait sunucuya giden istek kaydı

mektedir. Bu örnekte yüklü olan Katrina verilerinden sadece **lat, long, alt ve airtemp** kolonları parametre olarak kullanılmak üzere verilmiştir. Daha sonra cron zamanlayıcı çalışıp, bu verilen parametrelere göre bir veri tabanı sorgusu oluşturur ve verileri çektikten sonra bunları JSON objesi haline getirip çalıştırılan Python betiğine parametre olarak verir. 4.8 ifadesinde betik tamamlandıktan sonraki durum görülmektedir. Veri tabanı kaydına **result** isimli bir alan eklenmiş olup, bu alanın altında bulunan **output** alanında çalıştırılan betiğin konsol çıktısı görülmektedir. **error** alanı ise, betik sırasında oluşan hataların yazıldığı bir başka alandır.

4.9 ifadesinde çalıştırılan Python betiğinin konumu ve hangi parametrelerle çalıştırıldığı görülebilmektedir. **Query Fields** bölümünde, veri tabanında bulunan tiplerden bağımsız olarak her tip için ortak olan, verinin veri tabanına yüklendiği tarih aralığına göre sorgulanabilmesini sağlayan **startDate** ve **endDate** alanları bulunmaktadır. **typeSpecificFields** alanında ise, verinin oluşturulduğu kriterler görülmektedir. Bu kriterlere göre Python betiğine verilen parametrelerde sadece **lat, longt, alt, airtemp** kolonlarının bulunması istenmektedir. Bir diğer alan olan **queryLimit** alanında ise, 0 olduğundan dolayı, veri tabanından gelen sonuç olduğu gibi parametre oluşturmak için kullanılmakta ve herhangi bir veri limitlemesi olmamaktadır. 4.10 ifadesinde ise, görevin tamamlanması sonucunda betiğin ürettiği çıktılar görülebilmektedir.



Şekil 4.10: Oluşturulan görev bittikten sonra sunucudan gelen cevap



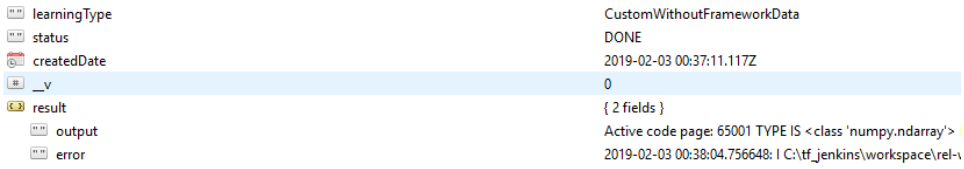
Şekil 4.11: Platform verisi olmadan oluşturulan görevin veri tabanı kaydı

4.4 Platform Verisi Olmadan Çalıştırılan LSTM Betiği

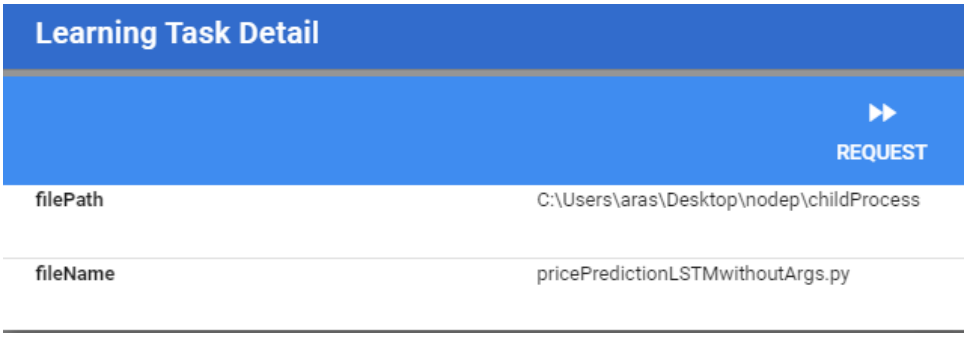
Bu senaryoda, kullanıcı, kendi yazdığı herhangi bir betiği platform üzerinden asenkron olarak ve platformda yüklü olan verileri kullanmadan çalıştırabilir. Senaryo için, öğrenme modülünde kullanılan veriler bir dosyaya konulmuştur. Öğrenme modülünden bağımsız yazılan ve Matplotlib[34] kütüphanesini kullanarak, arayüz modülündeki grafiksek analize benzer bir analiz yapan bir Python betiği yazılmıştır. Daha sonra menüden **Create Custom Task** sekmesi kullanılarak ve **Without Framework Data** seçeneği kullanılarak, platform üzerinden ilgili betik için bir görev oluşturulmuştur.

4.11 ifadesinde görev oluşturulduktan hemen sonra oluşan veri tabanı kaydı görülmektedir. LSTM görevinden farklı olarak, **request** parametresi sadece dosyanın ismi ve bulunduğu konumu içermektedir. Sunucu modülü üzerinde çalışan cron zamanlayıcı, bu bilgileri kullanarak dosyaya ulaşır ve doğrudan çalıştırır. 4.12 ifadesinde görev tamamlandıktan sonra güncellenen veri tabanı kaydı görülmektedir. Veri tabanı kaydına **result** isimli yeni bir alan eklenmiştir ve bu alanın alt sekmesi **output** üzerinden betiğin konsola bastığı tüm çıktı takip edilebilir. **error** alanında ise, betik çalışırken eğer bir hata alındıysa bu görülebilir. Arayüz modülü bu bilgileri kullanarak kullanıcıya betiğin durumu ile ilgili bilgi verir.

[34]https://matplotlib.org/



Şekil 4.12: Görev tamamlandıktan sonra oluşan veri tabanı kaydı



Şekil 4.13: Oluşturulan göreve ait sunucuya giden istek

4.13 ve 4.14 şekillerinde oluşturulan göreve ait bilgilerin arayüz modülünden görüntülenmesi gösterilmiştir. Menüde **Active Tasks** bölümünden, görevin tanımlandığı zaman aralıkları sorgulandıktan sonra, göreve ait istek ve cevap görülebilmektedir. **Request** sekmesinde görevin tanımlandığı betiğin lokasyonu, **Response** sekmesinde ise bu betiğin ürettiği çıktılar görülebilmektedir.

4.15 ifadesinde, betiğin anomali analizi üzerinden ve Matploblib kullanarak ürettiği grafiksel analiz görülebilir. Platform, betiğin ürettiği konsol çıktılarına arayüz modülü üzerinden erişim imkanı sağlarken, aynı zamanda ürettiği diğer çıktıları da gösterebilmektedir.

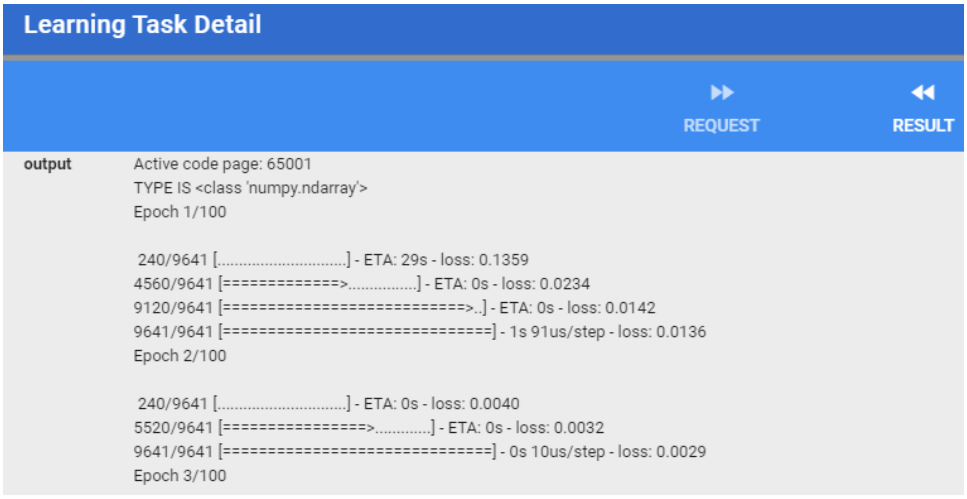
4.5 K-Means Kullanan Betiğin Platform Üzerinden Çalıştırılması

MISIoT, herhangi bir Python betiğini çalıştırabilme özelliğine sahiptir. Dolayısıyla, bu senaryoda [45]'da kümeleme yaparak Katrina verilerini yorumlayan ve bunlardan bir patern oluşturmaya çalışan makine öğrenmesi betiği çalıştırılmıştır.

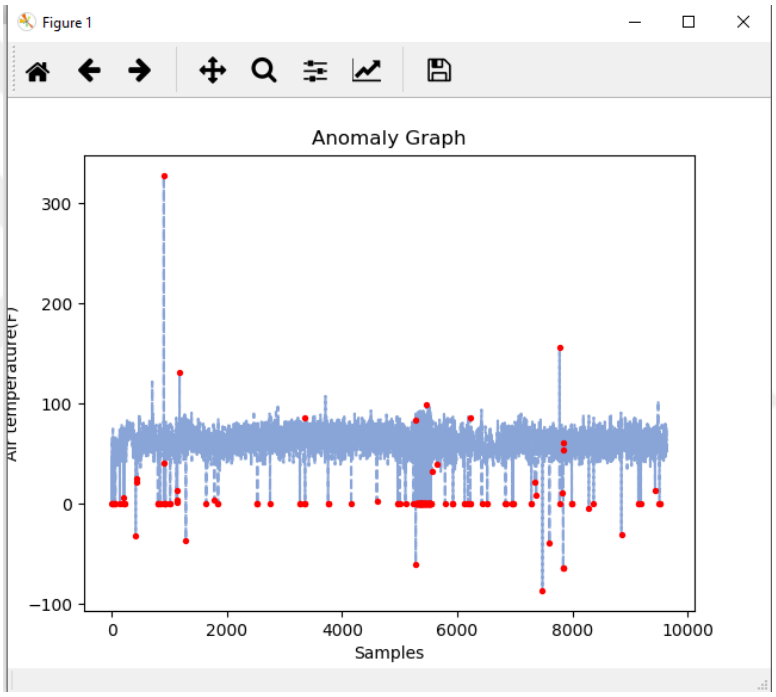
Betik çalıştırılırken herhangi bir platform verisi kullanılmamış, Github üzerinde bulunan betik [35] **Without Framework Data** seçeneği ile asenkron olarak platformdan çalıştırılmıştır.

4.16 ve 4.17 şekillerinde çalıştırılan betiğe ait istek ve sonuçlar görülmektedir. **Request** sekmesinde betiğin lokasyonu ve **Result** sekmesinde betiğin ürettiği çıktı görülmekte-

[35]https://github.com/omerbsezer/IoTWeatherSensorsAnalysis/tree/master/weatherIoTSensors



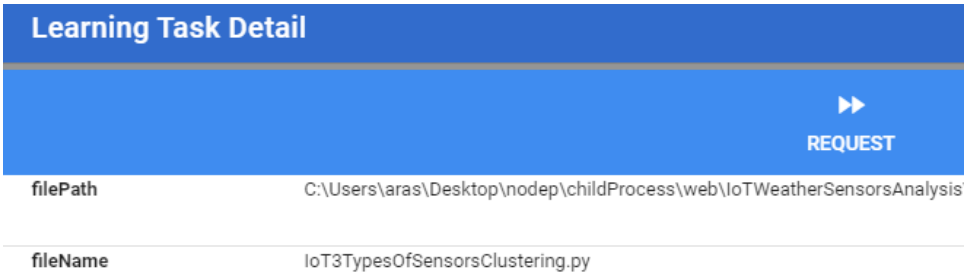
Şekil 4.14: Oluşturulan görev bittikten sonra sunucunun oluşturduğu cevap



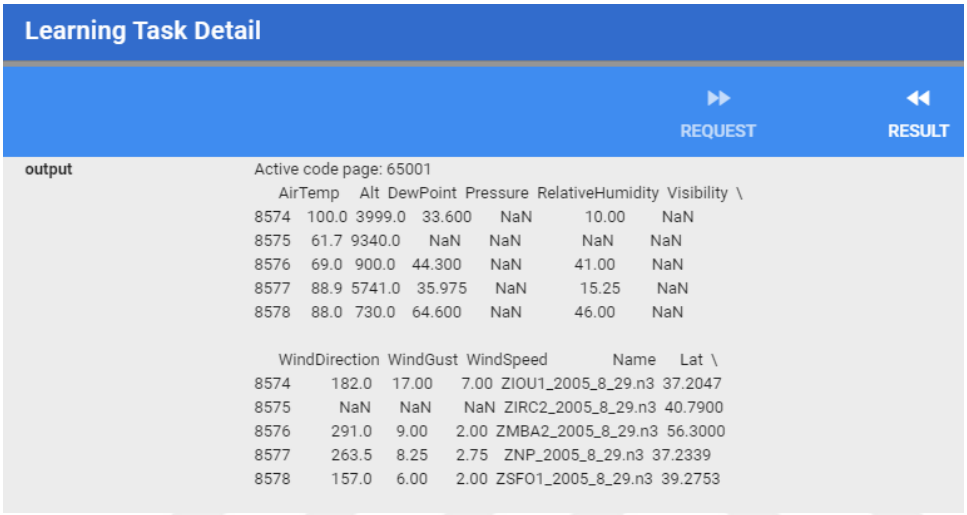
Şekil 4.15: Görev tanımlanan betiğin oluşturduğu grafiksel analiz

dir.

MISIoT, betiği doğrudan sistemde kurulu Python ortamı üzerinde çalıştırdığından, eğer betik grafiksel çıktı üretiyorsa bunların gösterilmesine olanak sağlamaktadır. 4.18 ifadesinde Katrina veri setinde bulunan **airTemp, relativeHumidity ve windspeed** özelliklerini kullanarak kümeleme yapan betiğin ürettiği grafiksel çıktı görülmektedir.



Şekil 4.16: Çalıştırılan betiğe ait istek

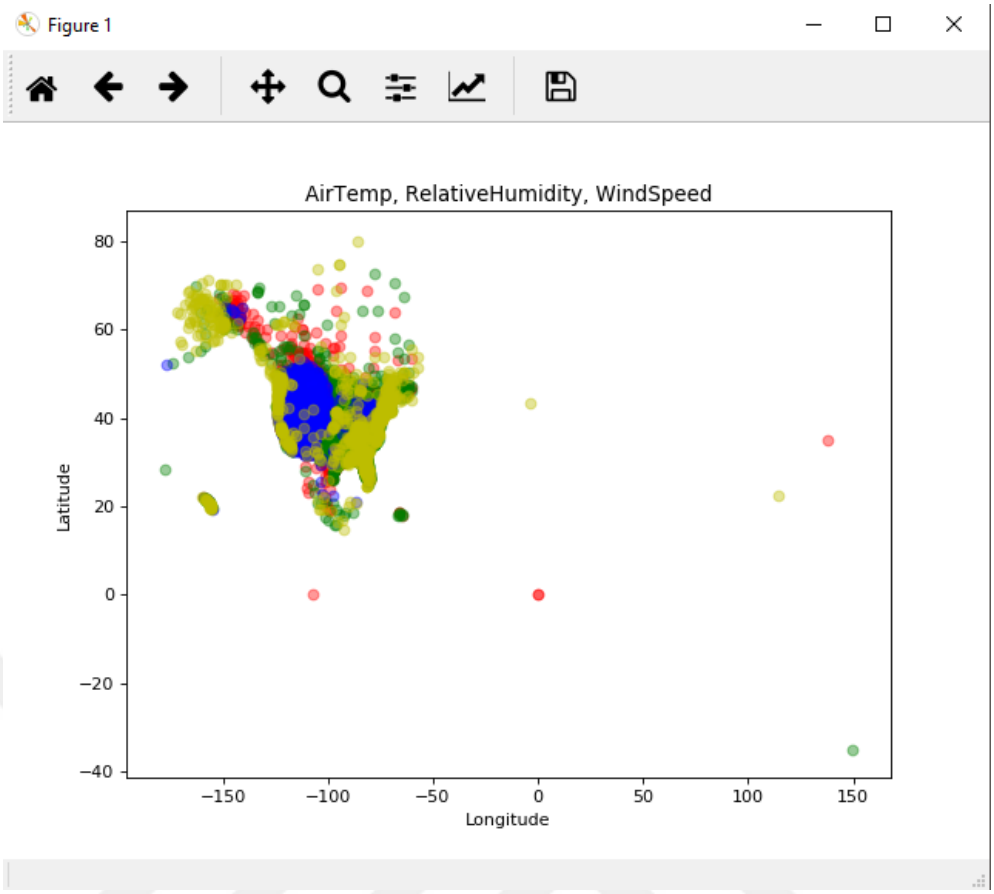


Şekil 4.17: Çalıştırılan betiğe ait sonuç

4.6 Değerlendirme

Örnek kullanım senaryoları bölümünde, MISIoT'un desteklediği görev tipleri ve bunların olası kullanım senaryolarından bahsedilmiştir. Platform bu senaryolara bağımlı olmayıp, farklı betiklerin kullanıldığı birçok farklı senaryo da üretilebilir. Platform tarafından sağlanan öğrenme modülüne alternatif bir modül yazılıp, sunucu modülüyle entegre biçimde çalıştırılabilir.

Farklı kullanım senaryolarına örnek olarak, LSTM veya K-Means dışında herhangi bir farklı algoritmayla yazılmış bir betik kullanılabilir. MISIoT, sistemde kullanıcı tarafından verilen Python dizinini kullanmaktadır. Dolayısıyla, eğer ilgili betiğin çalıştırıldığı Python ortamı MISIoT'a konfigürasyon olarak verilirse, bu ortamda sorunsuz çalışan herhangi bir betik, yazıldığı algoritmadan ve ürettiği çıktıdan bağımsız olarak çalışacaktır. Platform verisi kullanılmak istenirse, MISIoT bu platform verisini JSON halinde ilgili betiği çalıştırırken argüman olarak vermektedir. Kullanıcı bu veriyi kullanacak şekilde ufak modifikasyonlarla betiği yine platform üzerinden istediği veri kümesiyle çalıştırılabilir hale getirir.



Şekil 4.18: Betiğin ürettiği grafiksel çıktı

Platformun arayüz modülü kullanım senaryolarından bağımsız olarak tasarlanmıştır ve sunucu modülü tarafından kaydedilen herhangi bir veriyi kullanıcıya arayüz üzerinden sunabilmektedir. Dolayısıyla, platform üzerinden çalıştırılan herhangi bir betiğin ürettiği çıktılar ve hatalar, betiğin kullandığı algoritmadan bağımsız olarak arayüzden görülebilmektedir. Öğrenme modülü özelinde anomali analizi sonucu üretilen noktalar grafiksel olarak da görülebilmektedir. Modül kullanılmadan üretilen betikler için, kullanıcı arayüz modülünü modifiye ederek betiğin ürettiği çıktılardan kendi grafiklerini çizdirebilir.

Öğrenme modülünü içeren kullanım senaryosunda, platformun sağladığı modül kullanılmıştır ancak çalışma yapısındaki esneklikten dolayı, kullanıcı herhangi bir programlama dilinde bir modül yazıp, sunucu modülüyle etkileşimli olarak çalıştırabilir. Sunucu modülünün asenkron çalışma prensibi veri tabanı idsi üzerinden olmaktadır, dolayısıyla öğrenme modülünde olduğu bu id kullanılarak sonuç takibi yapan ve bu sonucu REST mimarisi üzerinden istemciye sunabilen herhangi bir program, sunucu modülüyle uyumlu çalışabilir.

5. PLANLANAN GELİŞTİRMELER

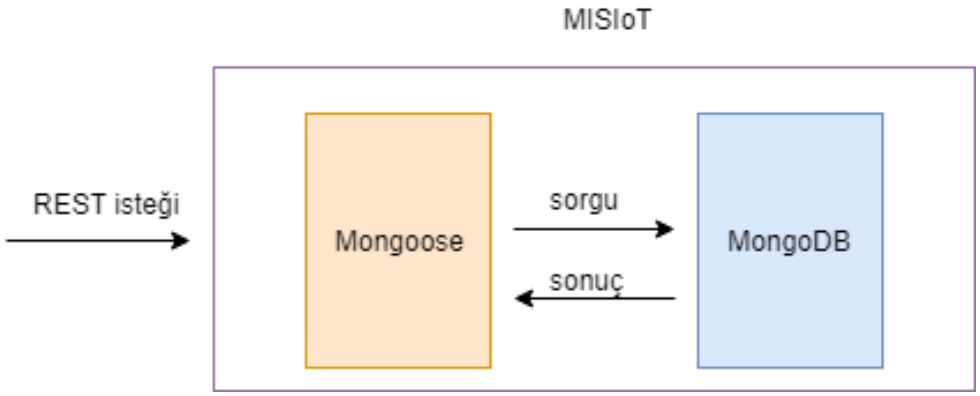
MISIoT'un daha performanslı ve işlevsel olması için planlanan çeşitli geliştirmeler bulunmaktadır. Bu geliştirmelerle platform, daha güncel programlama pratikleri ve teknolojileri kullanarak, diğer geliştiriciler tarafından daha rahat biçimde katkı sağlanabilir bir yapıya kavuşturulacaktır.

Planlanan geliştirmelerden ilki, makine öğrenmesinde Python'a alternatif olarak farklı dillerin de desteklenmesidir. Python, makine öğrenmesi ve yapay zeka alanında dünya üzerinde kullanılan en yaygın dillerden biridir. Dolayısıyla bu alanda birçok makine öğrenmesi kütüphanesi, bu dilde yazılmakta ve geliştirilmektedir. MISIoT da bu sebeple makine öğrenmesi görevlerini Python üzerinden gerçekleştirmektedir. Platformun desteklediği LSTM görevi oluşturma veya herhangi bir betiği platform verisiyle çalıştırma görevinin oluşturulabilmesi Python kullanılmaktadır. Ancak günümüzde Java, R, Scala, C/C++, Javascript diller de yaygın olarak kullanılmaktadır [36]. Dolayısıyla platformun görev oluşturulurken bu dilleri destekleyecek yapıda olması, platformun daha yaygın kullanılmasına olanak sağlayacaktır.

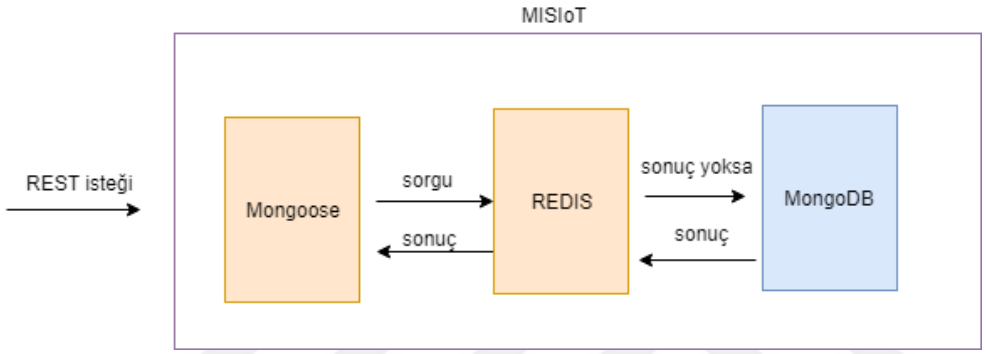
İkinci geliştirme olarak, bir anahtar-değer(key-value) üzerinden çalışan veri tabanı yapısı kurulması hedeflenmektedir. Piyasada bu yapıyı sunan en yaygın platform Redistir[37]. Redis kullanılarak sorgu sonuçları önbellekte tutulabilir ve MISIoT'a veri tabanı ihtiyacı duyan bir istek geldiğinde, veri tabanı sorgulanmak yerine bu önbellekten istenilen sonuç çok daha hızlı biçimde verilebilir.

5.1 ve 5.2 şekillerinde MISIoT'un mevcut veri tabanı mimarisi ve Redis ile beraber tasarlanan veri tabanı mimarisi görülmektedir. İlk yapıda, platforma gelen istekte, eğer veri tabanına sorgu atma ihtiyacı varsa bu her durumda yapılmaktadır. Örneğin, A sorgusu sonucunda her zaman B sonucu gelse bile, bu sorgu tekrarlanmaktadır. Bunun dezavantajı, sonuç değişmemesine rağmen sorgunun tekrarlanması sonucu daha fazla gecikme ile istemciye cevap dönülmesidir. Redis yapısında ise, A sorgusu yapıldığında, bunun sonucu Redis üzerinde saklanmakta, eğer aynı sorgu tekrar geliyorsa, veri tabanına gitmek yerine doğrudan Redis önbelleği üzerinden cevap dönülmektedir. Böylelikle istemci çok daha hızlı bir biçimde sorgu sonucunu görebilmektedir.

[36] https://github.blog/2018-11-15-state-of-the-octoverse-top-programming-languages/
[37] https://redis.io/



Şekil 5.1: Mevcut MISIoT veri tabanı mimarisi

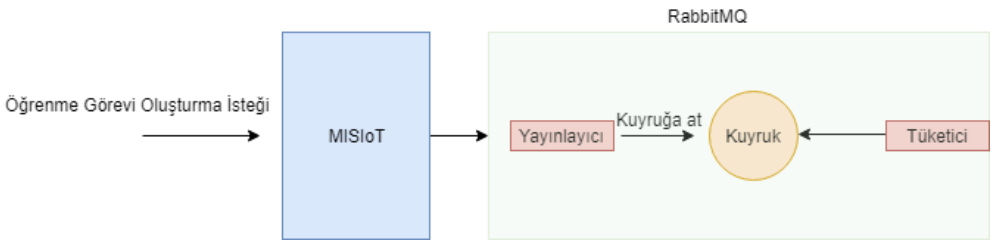


Şekil 5.2: Redisle beraber MISIoT veri tabanı mimarisi

Üçüncü geliştirme ise, cron zamanlayıcı yerine RabbitMQ[38] gibi bir mesaj kuyruğu sistemi kullanılmasıdır. 3.1.5 bölümünde anlatıldığı gibi, MISIoT'da mevcut yapıda kullanılan cron zamanlayıcı ile, belli aralıklarla bir job çalışmakta ve platformda oluşturulan görev kayıtlarını işlemektedir. Bu zamanlayıcı belli aralıklarla çalıştığından, platformda binlerce öğrenme görevi işlenmeye hazır halde beklese bile, zamanı gelmeden bu görevler çalışmamaktadır. Düşük yük durumlarında bu bir sorun olmasa bile, platform üzerinde saniyede yüzlerce öğrenme görevi oluşturulduğu durumda, zamanlayıcı ve zamanlayıcının çalıştığı sistem üzerinde büyük bir yük oluşturacaktır. Bunun önüne geçebilmek ve platformun daha ölçeklenebilir olmasını sağlamak için, mesaj kuyruğu sistemi kurulması planlanmaktadır.

5.3 ifadesinde MISIoT ile RabbitMQ'nun entegre edilmesi sonucunda kullanılabilecek mimari gösterilmektedir. Mevcut mimaride platforma öğrenme isteği geldiğinde, bu ilk olarak veri tabanına kaydedilmekte, daha sonra cron zamanlayıcı çalıştığı zaman bu isteği işlemektedir. RabbitMQ kullanıldığı zaman, gelen istek doğrudan bir kuyruk yapısına aktarılıp(publisher), bu kuyruktan okuyan başka bir modül tarafından(consumer)

[38]https://www.rabbitmq.com/



Şekil 5.3: RabbitMQ ve MISIoT'un birlikte kullanılması

işlenmektedir. Bu mimariyle istekler cron zamanlayıcının çalışmasına bağlı olarak değil, oluşturuldukları anda işlenmektedirler. Bu şekilde platformun daha performanslı ve fazla yük karşısında sistem kaynaklarını yormayan bir yapıda çalışması sağlanmış olur.

Dördüncü geliştirme olarak, öğrenme modülünün daha efektif çalışması için modelleri kaydetmesi ve bunların tekrar kullanılabilir olması planlanmaktadır. Mevcut yapıda öğrenme modülü her LSTM öğrenme isteğinde, verilerin bir kısmıyla önce modeli oluşturmakta, daha sonra geri kalan veri üzerinde bu modeli kullanarak anomali analizi yapmaktadır. Buna alternatif olarak, modellerin sunucu modülü üzerinde saklanıp, saklanan modeller üzerinden farklı veri setleriyle analiz yapılabilmesi sağlanacaktır. Bu geliştirmeyle beraber her seferinde model oluşturmadan kaynaklanan zaman kaybı giderilecek, hem de başarılı modellerin tekrar kullanımına imkan sağlanacaktır.

Beşinci geliştirme olarak platform üzerinde kural sisteminin getirilmesi planlanmaktadır. MISIoT, gerçek zamanlı veriler üzerinde çalışabildiğinden ve anomali analizi yapma özelliğine sahip olduğundan dolayı, sisteme gelen veriler üzerinde çeşitli kurallar tanımlanıp, gelen verilerin gerçek zamanlı sınıflandırılarak anomali olup olmadıkları belirlenebilir.

6. SONUÇ

Bu tez kapsamında büyük verinin saklanabileceği ve bu veri kullanılarak zaman serisi üzerinde analiz yapabilen, Python dilinde yazılmış herhangi bir makine öğrenmesi betiğini platform verisi kullanılarak çalıştırılmasına imkan veren bir IoT platformu anlatılmıştır. Tez kapsamında, arayüz, sunucu ve öğrenme olmak üzere birbirinden bağımsız üç ayrı modül olarak tasarlanan MISIoT platformunun modülleri detaylı olarak açıklanmıştır. Arayüz modülü kullanılarak, REST isteğini oluşturmaya gerek kalmayacak biçimde, platform daha kolay biçimde yönetilebilir. Arayüz modülü üzerinden çoklu veri yükleme, LSTM görevi oluşturma veya kullanıcının yazdığı herhangi bir Python betiği üzerinden görev oluşturma, portlardaki veri akışını takip etme, soket veya REST bağlantısı oluşturma işlemleri, sade tasarlanan arayüz sayesinde yapılabilir.

Verinin saklandığı ve diğer modüllerle iletişimden sorumlu olan sunucu modülü kullanılarak, arayüz modülüne hiç ihtiyaç duymadan, sadece REST istekleri kullanılarak platform yönetilebilir. Veri tabanı bu modül tarafından yönetilmektedir, performanslı çalışması için cron zamanlayıcı kullanılarak asenkron bir mimaride tasarlanan bu modül, arayüz modülünün kullanımının mümkün olmadığı durumlarda, REST istemcisi yazılabilen herhangi bir programlama dilinde yazılmış farklı bir programdan yönetilebilir. Alternatif olarak Postman gibi HTTP isteği atılmasına olanak sağlayan programlar kullanılarak sunucu modülü kullanılabilir. Sunucu modülü platformun en temel modülüdür, tüm görev yönetimi bu modül üzerinden yapılmaktadır. Platform verisi kullanılarak ve herhangi bir Python betiği üzerinde oluşturulabilen öğrenme görevleri diğer modüller olmadan, sadece bu modülle yapılabilir. LSTM görevi için öğrenme modülü sunulmaktadır.

Arayüz ve sunucu modülüne ek olarak, opsiyonel olan öğrenme modülü ise, Python dilinde yazılmış olup, sunucu modülünden kendisine gelen veriler üzerinde LSTM analizi yapabilmekte ve bu veri üzerindeki anomali noktalarını üç sigma limitine göre hesaplamaktadır. Bu modül çok iş parçacıklı ve asenkron olarak tasarlanmıştır, dolayısıyla aynı anda çoklu LSTM öğrenme görevlerini gerçekleştirebilir. Öğrenme modülü platformun sunduğu LSTM analizi için kullanılabilir.

KAYNAKLAR

[1] **Malhotra**, **Pankaj**, **Vig**, **Lovekesh**, **Shroff**, **Gautam**, and **Agarwal**, **Puneet** (2015). "Long short term memory networks for anomaly detection in time series". In: *Proceedings*. Presses universitaires de Louvain, p. 89.

[2] **Chauhan**, **Sucheta** and **Vig**, **Lovekesh** (2015). "Anomaly detection in ECG time signals via deep long short-term memory networks". In: *Data Science and Advanced Analytics (DSAA), 2015. 36678 2015. IEEE International Conference on*. IEEE, pp. 1–7.

[3] **Taylor**, **Adrian**, **Leblanc**, **Sylvain**, and **Japkowicz**, **Nathalie** (2016). "Anomaly detection in automobile control network data with long short-term memory networks". In: *Data Science and Advanced Analytics (DSAA), 2016 IEEE International Conference on*. IEEE, pp. 130–139.

[4] **Perera**, **Charith**, **Zaslavsky**, **Arkady**, **Compton**, **Michael**, **Christen**, **Peter**, and **Georgakopoulos**, **Dimitrios** (2013). "Semantic-Driven Configuration of Internet of Things Middleware". In: *9th International Conference on Semantics, Knowledge and Grids*. IEEE, pp. 66–73.

[5] **Sezer**, **Omer Berat**, **Dogdu**, **Erdogan**, and **Ozbayoglu**, **Ahmet Murat** (2018). "Context-aware computing, learning, and big data in Internet of Things: a survey". In: *IEEE Internet of Things Journal* 5.1, pp. 1–27.

[6] **Zaslavsky**, **Arkady**, **Perera**, **Charith**, and **Georgakopoulos**, **Dimitrios** (2012). "Sensing as a Service and Big Data". In: *Proceedings of the International Conference on Advances in Cloud Computing*, pp. 21–29.

[7] **Borges Neto**, **João**, **Silva**, **Thiago**, **Assunção**, **Renato**, **Mini**, **Raquel**, and **Loureiro**, **Antonio** (2015). "Sensing in the Collaborative Internet of Things". In: *Sensors* 15.3, pp. 6607–6632.

[8] **Atzori**, **Luigi**, **Iera**, **Antonio**, and **Morabito**, **Giacomo** (2010). "The internet of things: A survey". In: *Computer networks* 54.15, pp. 2787–2805.

[9] **Sundmaeker**, **Harald**, **Guillemin**, **Patrick**, **Friess**, **Peter**, and **Woelfflé**, **Sylvie** (2010). "Vision and challenges for realising the Internet of Things". In: *Cluster of European Research Projects on the Internet of Things, European Commision* 3.3, pp. 34–36.

[10] **Bélissent**, **Jennifer** (2010). "Getting clever about smart cities: New opportunities require new business models". In: *Cambridge, Massachusetts, USA*.

[11] **Gubbi**, **Jayavardhana**, **Buyya**, **Rajkumar**, **Marusic**, **Slaven**, and **Palaniswami**, **Marimuthu** (2013). "Internet of Things (IoT): A vision, architectural elements, and future directions". In: *Future generation computer systems* 29.7, pp. 1645–1660.

[12] **Perera**, **Charith**, **Liu**, **C H I Harold**, **Jayawardena**, **Srimal**, and **Chen**, **Min** (2014a). "A Survey on Internet of Things From Industrial Market Perspective". In: *IEEE Access* 2.2014, pp. 1660–1679.

[13] **Perera**, **Charith**, **Liu**, **Chi Harold**, and **Jayawardena**, **Srimal** (2015). "The Emerging Internet of Things Marketplace From an Industrial Perspective: A Survey". In: *IEEE Transactions on Emerging Topics in Computing* 3.4, pp. 585–598.

[14] **Mineraud**, **Julien**, **Mazhelis**, **Oleksiy**, **Su**, **Xiang**, and **Tarkoma**, **Sasu** (2015). "A gap analysis of Internet-of-Things platforms". In: *Computer Communications* 89-90.9, pp. 5–16.

[15] **Khan**, **Rafiullah**, **Khan**, **Sarmad Ullah**, **Zaheer**, **Rifaqat**, and **Khan**, **Shahid** (2012). "Future internet: the internet of things architecture, possible applications and key challenges". In: *Frontiers of Information Technology (FIT), 2012 10th International Conference on*. IEEE, pp. 257–260.

[16] **Perera**, **Charith**, **Zaslavsky**, **Arkady**, **Christen**, **Peter**, and **Georgakopoulos**, **Dimitrios** (2014b). "Context aware computing for the internet of things: A survey". In: *IEEE communications surveys & tutorials* 16.1, pp. 414–454.

[17] **Miorandi**, **Daniele**, **Sicari**, **Sabrina**, **De Pellegrini**, **Francesco**, and **Chlamtac**, **Imrich** (2012). "Internet of things: Vision, applications and research challenges". In: *Ad hoc networks* 10.7, pp. 1497–1516.

[18] **Alam**, **Sarfraz**, **Chowdhury**, **Mohammad MR**, and **Noll**, **Josef** (2010). "Senaas: An event-driven sensor virtualization approach for internet of things cloud". In: *Networked Embedded Systems for Enterprise Applications (NESEA), 2010 IEEE International Conference on*. IEEE, pp. 1–6.

[19] **Patel**, **Pankesh**, **Jardosh**, **Sunil**, **Chaudhary**, **Sanjay**, and **Ranjan**, **Prabhat** (2009). "Context aware middleware architecture for wireless sensor network". In: *2009 IEEE International Conference on Services Computing*. IEEE, pp. 532–535.

[20] **Botts**, **Mike**, **Percivall**, **George**, **Reed**, **Carl**, and **Davidson**, **John** (2008). "OGC® sensor web enablement: Overview and high level architecture". In: *GeoSensor networks*. Springer, pp. 175–190.

[21] **Satyanarayanan**, **Mahadev** (2001). "Pervasive computing: Vision and challenges". In: *IEEE Personal communications* 8.4, pp. 10–17.

[22] **Remagnino**, **Paolo**, **Hagras**, **H**, **Monekosso**, **N**, and **Velastin**, **S** (2005). "Ambient intelligence". In: *Ambient intelligence*. Springer, pp. 1–14.

[23] **Duquennoy**, **Simon**, **Grimaud**, **Gilles**, and **Vandewalle**, **Jean-Jacques** (2009). "The web of things: interconnecting devices with high usability and performance". In: *ICESS 2009*.

[24] **Guinard**, **Dominique**, **Trifa**, **Vlad**, **Mattern**, **Friedemann**, and **Wilde**, **Erik** (2011). "From the internet of things to the web of things: Resource-oriented architecture and best practices". In: *Architecting the Internet of things*. Springer, pp. 97–129.

[25] **Guinard**, **Dominique**, **Trifa**, **Vlad**, and **Wilde**, **Erik** (2010). "A resource oriented architecture for the web of things". In: *Internet of Things (IOT), 2010*. IEEE, pp. 1–8.

[26] **Pfisterer**, **Dennis**, **Romer**, **Kay**, **Bimschas**, **Daniel**, **Kleine**, **Oliver**, **Mietz**, **Richard**, **Truong**, **Cuong**, **Hasemann**, **Henning**, **Kröller**, **Alexander**, **Pagel**, **Max**, **Hauswirth**, **Manfred**, **Karnstedt**, **Marcel**, **Leggieri**, **Myriam**, **Passant**, **Alexandre**, and **Richardson**, **Ray** (2011). "SPITFIRE: toward a semantic web of things". In: *IEEE Communications Magazine* 49.11, pp. 40–48.

[27] **Chen**, **Feng**, **Deng**, **Pan**, **Wan**, **Jiafu**, **Zhang**, **Daqiang**, **Vasilakos**, **Athanasios V**, and **Rong**, **Xiaohui** (2015). "Data mining for the internet of things: literature review and challenges". In: *International Journal of Distributed Sensor Networks* 11.8, p. 431047.

[28] **Jia**, **Xiaolin**, **Feng**, **Quanyuan**, **Fan**, **Taihua**, and **Lei**, **Quanshui** (2012). "RFID technology and its applications in Internet of Things (IoT)". In: *Consumer Electronics, Communications and Networks (CECNet), 2012 2nd International Conference on*. IEEE, pp. 1282–1285.

[29] **Lewis**, **Frank L** (2004). "Wireless sensor networks". In: *Smart environments: technologies, protocols, and applications*, pp. 11–46.

[30] **Akyildiz**, **Ian F**, **Su**, **Weilian**, **Sankarasubramaniam**, **Yogesh**, and **Cayirci**, **Erdal** (2002). "A survey on sensor networks". In: *IEEE communications magazine* 40.8, pp. 102–114.

[31] **Compton**, **Michael**, **Barnaghi**, **Payam**, **Bermudez**, **Luis**, **GarcíA-Castro**, **RaúL**, **Corcho**, **Oscar**, **Cox**, **Simon**, **Graybeal**, **John**, **Hauswirth**, **Manfred**, **Henson**, **Cory**, **Herzog**, **Arthur**, **Huang**, **Vincent**, **Janowicz**, **Krzysztof**, **Kelsey**, **W. David**, **Phuoc**, **Danh Le**, **Lefort**, **Laurent**, **Leggieri**, **Myriam**, **Neuhaus**, **Holger**, **Nikolov**, **Andriy**, **Page**, **Kevin**, **Passant**, **Alexandre**, **Sheth**, **Amit**, and **Taylor**, **Kerry** (2012). "The SSN ontology of the W3C semantic sensor network incubator group". In: *Web semantics: science, services and agents on the World Wide Web* 17, pp. 25–32.

[32] **Amadeo**, **Marica**, **Campolo**, **Claudia**, **Quevedo**, **Jose**, **Corujo**, **Daniel**, **Molinaro**, **Antonella**, **Iera**, **Antonio**, **Aguiar**, **Rui L**, and **Vasilakos**, **Athanasios V** (2016). "Information-centric networking for the internet of things: challenges and opportunities". In: *IEEE Network* 30.2, pp. 92–100.

[33] **Abowd**, **Gregory D**, **Dey**, **Anind K**, **Brown**, **Peter J**, **Davies**, **Nigel**, **Smith**, **Mark**, and **Steggles**, **Pete** (1999). "Towards a better understanding of context and context-awareness". In: *International symposium on handheld and ubiquitous computing*. Springer, pp. 304–307.

[34] **Dean**, **Jeffrey** and **Ghemawat**, **Sanjay** (2008). "MapReduce: simplified data processing on large clusters". In: *Communications of the ACM* 51.1, pp. 107–113.

[35] **White**, **Tom** (2012). *Hadoop: The definitive guide*. " O'Reilly Media, Inc.".

[36] **Strohbach**, **Martin**, **Ziekow**, **Holger**, **Gazis**, **Vangelis**, and **Akiva**, **Navot** (2015). "Towards a big data analytics framework for IoT and smart city applications". In: *Modeling and processing for next-generation big-data technologies*. Springer, pp. 257–282.

[37] **Zaslavsky**, **Arkady**, **Perera**, **Charith**, and **Georgakopoulos**, **Dimitrios** (2013). "Sensing as a service and big data". In: *arXiv preprint arXiv:1301.0159*.

[38] **Bandyopadhyay**, **Debasis** and **Sen**, **Jaydip** (2011). "Internet of things: Applications and challenges in technology and standardization". In: *Wireless Personal Communications* 58.1, pp. 49–69.

[39] **Kelesidis**, **Theodore**, **Kelesidis**, **Iosif**, **Rafailidis**, **Petros I**, and **Falagas**, **Matthew E** (2007). "Counterfeit or substandard antimicrobial drugs: a review of the scientific evidence". In: *Journal of Antimicrobial Chemotherapy* 60.2, pp. 214–236.

[40] **Hardgrave**, **Bill C**, **Waller**, **Matthew**, and **Miller**, **Robert** (2006). *RFID's impact on out of stocks: A sales velocity analysis*. Tech. rep. Working Paper University of Arkansas.

[41] **Gruen**, **Thomas W**, **Corsten**, **Daniel S**, and **Bharadwaj**, **Sundar** (2002). *Retail out-of-stocks: A worldwide examination of extent, causes and consumer responses*. Grocery Manufacturers of America Washington, DC.

[42] **Fewtrell**, **P** and **Hirst**, **IL** (1998). "A review of high-cost chemical/petrochemical accidents since Flixborough 1974". In: *Loss Prevention Bulletin*.

[43] **Coroama**, **Vlad** (2006). "The smart tachograph–individual accounting of traffic costs and its implications". In: *International Conference on Pervasive Computing*. Springer, pp. 135–152.

[44] **Mineraud**, **Julien**, **Mazhelis**, **Oleksiy**, **Su**, **Xiang**, and **Tarkoma**, **Sasu** (2016). "A gap analysis of Internet-of-Things platforms". In: *Computer Communications* 89, pp. 5–16.

[45] **Onal**, **Aras Can**, **Sezer**, **Omer Berat**, **Ozbayoglu**, **Murat**, and **Dogdu**, **Erdogan** (2017). "Weather data analysis and sensor fault detection using an extended IoT framework with semantics, big data, and machine learning". In: *Big Data (Big Data), 2017 IEEE International Conference on*. IEEE, pp. 2037–2046.

ÖZGEÇMİŞ

Ad-Soyad : Aras Can ÖNAL

Uyruğu : T.C.

Doğum Tarihi ve Yeri : 15.04.1992 Keçiören / Ankara

E-posta : aras.onal@gmail.com

ÖĞRENİM DURUMU:

- **Lisans** : 2015, TOBB Ekonomi ve Teknoloji Üniversitesi Bilgisayar Mühendisliği, 3.05.

MESLEKİ DENEYİM

Yıl	Yer	Görev
2016 - 2019	Innova	Yazılım Uzmanı
2015 - 2016	TOBB Ekonomi ve Teknoloji Üniversitesi	Burslu Yüksek Lisans Öğrencisi

YABANCI DİL: İngilizce

TEZDEN TÜRETİLEN YAYINLAR:

- **Onal, Aras Can**, Omer Berat Sezer, Murat Ozbayoglu, and Erdogan Dogdu. "MIS-IoT: Modular Intelligent Server Based Internet of Things Framework with Big Data and Machine Learning." In 2018 IEEE International Conference on Big Data (Big Data), pp. 2270-2279. IEEE, 2018.
- **Onal, Aras Can**, Omer Berat Sezer, Murat Ozbayoglu, and Erdogan Dogdu. "Weather data analysis and sensor fault detection using an extended iot framework with semantics, big data, and machine learning." In 2017 IEEE International Conference on Big Data (Big Data), pp. 2037-2046. IEEE, 2017.
- Sezer, Omer Berat, Erdogan Dogdu, Murat Ozbayoglu, and **Aras Onal**. "An extended iot framework with semantics, big data, and analytics." In 2016 IEEE International Conference on Big Data (Big Data), pp. 1849-1856. IEEE, 2016.