**MAM Projesi Teknik Dokümanı**

Proje, bulunduğu ortamın sıcaklık, nem, basınç ve gürültü değerlerini ölçen bir cihazın ölçüm verilerini ethernet yoluyla gönderdiği sunucu yazılımının oluşturulmasını içermektedir. Yazılım Node JS REST API olarak yazılmıştır ve MongoDB veri tabanı kullanılmıştır. Taşınabilirliği kolay hale getirmek amacıyla “Dockerize” edilmiştir.

Veriler cihaz tarafından TCP protokolüyle iletilmektedir. Cihaza sunucunun IP ve port bilgisi, “IPUpdateApp” programı yardımıyla girildikten sonra cihaz bu IP ve belirtilen porta bilgileri istenilen periyotta gönderir. Yazılım iki sunucu içerir: TCP Sunucu, Express Sunucu. TCP Sunucusu gelen veriyi veri tabanına kaydeder, Express Sunucu ise istemcinin talep ettiği veriyi veri tabanı sorgularını gerçekleştirip filtreleyerek gönderir. Programın detayları aşağıda açıklanmıştır.

**Programın İçeriği ve Çalışması**

* **Ortam Değişkenlerinin Ayarlanması**

Programın ana dizininde (index.js ile aynı dizin) “.env” dosyasında programla ilgili ortam değişkenleri tanımlanmıştır. Program başlatılmadan aşağıda ayrıntıları verilen ortam değişkenleri uygun biçimde ayarlanmalıdır. Programda kullanılan ortam değişkenleri şunlardır:

* + NODE\_ENV: Loglama için yönetilen ortam değişkeni. ‘production’ ise konsola loglama yapılmaz, ‘info’ ve daha düşük seviyedeki loglamalar ‘logFiles’ klasöründeki dosyalara yapılmaya devam eder. Aksi halde konsola loglama yapılır.
  + TCP\_PORT: Cihazın veri gönderdiği port numarasıdır. Varsayılan değer: 3001
  + EXPRESS\_PORT: İstemciden gelen istek için ayrılmış olan port numarasıdır. Varsayılan değer: 3000
  + MONGODB\_CONNECTION\_STRING: Veri tabanına bağlanmak için kullanılan string. Varsayılan değer: 'mongodb://mongodb:27017/mam-app'
  + DAYS\_BEFORE: Veri tabanı kayıtlarının ve log dosyalarının kaç gün öncesine kadar tutulacağını belirten gün sayısıdır. Varsayılan değer: 7
  + TIMEZONE: Ayarlanırsa, veri tabanındaki eski kayıtlar yerel saate göre 00.30’da silinir. Varsayılan değer: ‘Europe/Istanbul’
  + LOG\_FILES\_VOLUME\_PATH: Docker ile başlatılırsa ayarlanmalıdır. Sunucuda hangi klasörde log dosyalarının tutulacağını belirten dosya yoludur. Dosya yolu ‘Absolute Path’ olarak girilmelidir. (Örn: /mam-app/logFiles veya C:\\Users\\ahmet\\mam-app\\logFiles gibi.) Varsayılan değer: “~/mam-app/logFiles”
  + ERROR\_TOLERANCE: Gelen verinin veri tabanına kaydedilmediği ya da verinin JSON formatında gelmediği durumlarda program hata verir ve çalışmasına devam eder. Veri tabanıyla bağlantı bir süre sonra yapılabilir ve program çalışmaya devam edebilir. Uzun süre bağlantı sağlanamazsa verinin kaydedilmesi aşamasında yakalanan hatalar sayılır. Bir başka durumda da veri tabanına bağlantı sağlanmış olabilir fakat gelen veri hatalıdır, JSON formatına uygun değildir. Bu durumda yakalanan hatalar da sayılır. Üst üste en fazla kaç hata yakalanırsa program sonlandırılmalıdır? Bu değer ERROR\_TOLERANCE ortam değişkenine yazılmalıdır. Varsayılan değer: 5
* **Veri Tabanına Bağlanılması**

Program, MongoDB veri tabanı kullanır. Veri tabanına bağlanmak ve CRUD operasyonlarını gerçekleştirmek amacıyla NodeJS “mongoose” kütüphanesini kullanır (bkz. <https://mongoosejs.com/docs/index.html> ).

Veri tabanına bağlanmak için kullanılan “connection string”: “MONGODB\_CONNECTION\_STRING” ortam değişkeni ile tanımlanır. Tanımlanmazsa varsayılan olarak mongodb://mongodb:27017/mam-app” stringi kullanılır.

* + **Detay:** MongoDBiçin connection string formatı:

“mongodb://<kullanıcı-adı>:<şifre>@<database-IP>:<Port>/<db-adı>” şeklindedir. Kullanıcı oluşturulmadığı için kullanıcı adı ve şifre kısmı girilmemiştir. Dockerize kısmında “database-IP”nin neden “mongodb” olarak yazıldığını açıklayacağız. MongoDB veri tabanı 27017 portunu kullanır. Detaylı bilgiye <https://www.mongodb.com/docs/drivers/node/current/fundamentals/connection/connect/> linkinden ulaşabilirsiniz.

“mongoose” nesnesinin “connect” fonksiyonu “Promise” döndürür. Asenkron bir şekilde çalışır. Veri tabanına bağlantı başarılı olursa “promise”, “resolved state” durumuna döner ve “then” isimli “callback” fonksiyonunu çağırır. Burada bağlantı başarılı olduğunda “info log”u gerçekleştirilir (bkz. Logger). Veri tabanına bağlanılamazsa “promise”, “rejected state” durumuna döner ve “catch” isimli “callback” fonksiyonunu çağırır. Veri tabanına bağlanılamazsa programın sonlandırılmasını istediğimiz için “catch” fonksiyonunu kullanmadık. Bu durumda “unhandledPromiseRejection” istisnası fırlatılacak ve winston kütüphanesinde tanımlanmış olan “RejectionHandler” nesnesi tarafından ele alınacaktır (bkz. Logger) ve program sonlandırılacaktır.

* **TCP Sunucusu**

TCP Sunucusunun görevi, tanımlanan “TCP\_PORT” ortam değişkeniyle tanımlanan portu (varsayılan: 3001) dinlemek, veri geldiğinde bunu bir string değişkene kaydetmektir. Ayrıca verinin son geldiği zamanı da ‘chunkHolder.lastEditTime’ değişkeninde saklar.

* + **ChunkHolder Sınıfı:** Gelen verinin sonuna eklendiği ‘chunk’ string değişkeni içerir. Bir de en son güncellendiği zamanı tutan ‘lastEditTime’ değişkenini içerir. “chunk.holder.js” dosyasında bu sınıftan bir nesne (instance) üretilir ve nesne export edilir. ‘İmport’ edilen dosyalarda aynı nesneye ulaşıldığından bu nesne üzerinde birden fazla dosyada birlikte çalışılır.

Sunucu, veriyi ‘chunkHolder.chunk’ string değişkene kaydettikten sonra ‘chunkHolder.lastEditTime’ değişkenine o anki zamanı kaydediyor, “server” nesnesinin “dataAdded” “event”ini -bu event’i biz oluşturduk- “emit” ediyor. Böylece “saveDataToDatabase” fonksiyonun çağrılmasını sağlıyor.

* **saveDataToDatabase() Fonksiyonu**

Cihaz veriyi “json” formatında gönderir. Oluşturduğu her verinin sonuna “;” ayıracını koyar. Veri oluşturulduktan sonra belirtilen periyotlarla sunucuya gönderilir. TCP protokolü, gönderilen paketlerin sırayla gideceğini garanti altına alır fakat bir bütün olarak gönderileceğinin garantisini vermez. Verinin parça parça gönderilmesi durumunda, aktarılan veri sunucuda birleştirilir. Sunucu, ayraç yardımıyla veriyi ayrılarak veri tabanına kaydeder. Fonksiyon, “chunkHolder” nesnesinin “chunk” değişkenine yazılan veriyi “;” ayıracına göre ayırır, veri tabanına kaydeder ve “chunk” değişkeninden, veri tabanına kaydettiği kısmı siler. Bu işlemi “;” ayıracı bulunamayana kadar devam ettirir. Ayraç bulunmadığında ise herhangi bir işlem yapmaz.

* + **Detay:** Fonksiyon asenkron çalışır. Çünkü veri tabanına veriyi kaydetme işlemlerini içerir. NodeJS’teki “async/await” yapısı kullanılmıştır.

(bkz. <https://nodejs.dev/en/learn/modern-asynchronous-javascript-with-async-and-await> )

Verinin veri tabanına kaydedilmesi adımı “await” edilmiştir. Kodun bu kısmı “try/catch” bloğu içerisinde yazılmıştır. Bir hata olması durumunda hata mesajı loglanır, “errCount” değeri bir arttırılır, eğer hata sayısı ERROR\_TOLERANCE’tan fazlaysa bu durum loglanır ve program sonlandırılır, fazla değilse “chunk” değişkeninde herhangi bir değişiklik yapılmadan döngüden çıkılır. Program çalışmaya devam eder.

* + **data.model.js**

“mongoose” kütüphanesinin yardımıyla cihazdan aldığımız veriyi veri tabanına hangi formatta kaydedeceğimizi belirttiğimiz dosyadır. Cihazdan sıcaklık (temp), nem (hum), basınç (pres), ses (vol) bilgileriyle birlikte, cihazın çalışmaya başlamasından bu yana geçen süre (time) bilgisi gelmektedir. MongoDB veri tabanında kabul edilen veri tiplerine uygun olarak temp, hum, pres, vol bilgilerinin “Number” tipinde; time bilgisinin “String” tipinde tutulacağı belirtilmiştir. Veri tabanına kaydedildiğinde kaydetme zamanı bilgisini de tutmak için “createdAt” nesnesi de “Date” tipinde tanımlanmıştır.

* + **Verinin Data Modeline Dönüştürülmesi**

Başarılı bir şekilde alınıp ayraç kullanılarak birbirinden ayrılan veri “json” formatında bir string’dir. “JSON.parse()” metoduyla “javascript object” nesnesine dönüştürülür. Burada “parse” edilmek istenen string uygun formatta değilse metot “SyntaxError” istisnası fırlatır. Bu işlem “try/catch” bloğunda yapıldığı için “catch” bloğu tarafından yakalanır ve hata bilgisi loglanır, “errCount” değeri bir arttırılır, eğer hata sayısı ERROR\_TOLERANCE’tan fazlaysa bu durum loglanır ve program sonlandırılır, fazla değilse “chunk” değişkeninde herhangi bir değişiklik yapılmadan döngüden çıkılır. Program çalışmaya devam eder.

* **Zamanlanmış Görevler**

Belirli periyotlarla yapılması istenen görevlerin yazıldığı kısımdır. “scheduled.tasks.js” dosyasına yazılmıştır.

Planlanan görevlerden biri veri tabanındaki eski kayıtların belirli periyotlarla silinmesidir. Burada amaç, kullanılmayacak olan eski kayıtların silinmesini sağlamak, böylece veri tabanı sorgularının daha hızlı yapılabilmesini sağlamaktır. “TIMEZONE” ortam değişkeniyle yapılacak işlemin yerel saate göre ayarlanması sağlanabilir. Varsayılan olarak “Europe/Istanbul” değeri verilmiştir. “cron zamanlama ifadesi” olarak “ 30 0 \* \* \* ” girilmiştir. Bu da silme işleminin her gün yerel zamanda saat 00.30’da gerçekleşeceği anlamına gelir (Bkz. [https://crontab.guru/#30\_0\_\*\_\*\_\*](https://crontab.guru/#30_0_*_*_*) ). “DAYS\_BEFORE” ortam değişkeni kaç günden önceki verilerin silinmesi gerektiğini belirten değişken olarak ayarlanmıştır. Varsayılan olarak 7 değeri verilmiştir. Bu da bir haftadan daha eski verilerin her gün saat 00.30’da silineceği anlamına gelir. Örneğin bir aydan daha eski verilerin düzenli olarak silinmesini istersek “DAYS\_BEFORE” ortam değişkenine 30 değerini vermemiz gerekir. Veri tabanından veri silme işlemi asenkron gerçekleşmesi gereken bir iş olduğundan “async/await” yapısı kullanılmıştır. İşlem başarılı olmazsa hata bilgisi loglanır, program çalışmaya devam eder.

Planlanan bir diğer görev de her 10. dakikada, cihazdan verinin gelip gelmediğinin kontrol edilmesidir. Kontrol sonrasında 1 dakikadan daha uzun süredir veri gelmemişse ‘warn’ seviyesinde loglama yapar, ne kadar süredir verinin gelmediği loglanır. Program çalışmaya devam eder.

* **EXPRESS Sunucusu**

Express sunucusunun görevi, “EXPRESS\_PORT” ortam değişkeniyle tanımlanan portu (varsayılan: 3000) dinlemek, bu porta gelen istekleri yanıtlamaktır. “/api/data” “endpoint”ine gelen istekler “data” isimli “router”a yönlendirilir. REST API görevi görür. Gelen sorgulara JSON formatında yanıtlar gönderir. Sorgularda bir hata oluşması durumunu yönetmek için “error middleware”i oluşturulmuştur.

* + **“data.js” Router ( /api/data )**

“/api/data” endpoint’ine gelen “HTTP GET” istekleri işlenmektedir. İşleyici (handler) olarak “dataControler” fonksiyonu çağrılır.

* + **“dataController.js”**

Bu fonksiyon, HTTP GET isteğindeki “query string”i “query-to-mongo” kütüphanesi yardımıyla MongoDB sorgusuna dönüştürür. Önce, gelen sorgudaki sayfalandırma (pagination) ile ilgili kısım hariç tutularak istenen verinin sayısını “dataCount” değişkeninde saklar. Sonra, sayfalama (paging) ve diğer kriterler göz önünde bulundurularak veri tabanı sorgusunu gerçekleştirir, gelen veriyi “data” objesinde tutar. Eğer varsa sayfalandırma ile ilgili linkleri oluşturur (ilk sayfa, önceki sayfa, sonraki sayfa ve son sayfa). Yanıt (response) olarak toplam veri sayısını (dataCount), veriyi (data), sayfalandırma linklerini (links) döndürür. Sorgu sırasında bir hatayla karşılaşılması durumunda “next” fonksiyonunu çağırır ve hata nesnesini bu fonksiyona gönderir. Yani kontrolü bir sonraki ara katman yazılımına (middleware) devreder. Bu da “index.js” dosyasında belirtilen “errorMiddleware”dir.

* + - **Sorgu (query)**

Veri tabanı sorgularının doğru gerçekleştirilebilmesi için yapılacak olan sorgularda;

Kriterler için:

* + - * time (string)
      * temp (number)
      * hum (number)
      * pres (number)
      * vol (number)
      * createdAt (Date) parametreleri kullanılmalıdır.

Örnek: “/api/data?temp=>30&hum=<50” sorgusu, sıcaklığı 30’dan büyük ve nem değeri 50’den küçük olan tüm kayıtları getirir.

Seçilecek alanlar için ‘fields=’ yazdıktan sonra virgülle ayrılarak:

* + - * time (string)
      * temp (number)
      * hum (number)
      * pres (number)
      * vol (number)
      * createdAt (Date) parametreleri kullanılmalıdır.

Örnek: “/api/data?fields=time,temp,createdAt” sorgusu, tüm kayıtların ‘cihazın çalışma zamanı, sıcaklık ve kaydedilme zamanı’ alanlarını getirir.

Sıralama için ‘sort=’ yazdıktan sonra virgüllerle ayrılarak:

* + - * time (string)
      * temp (number)
      * hum (number)
      * pres (number)
      * vol (number)
      * createdAt (Date) parametreleri kullanılmalıdır. Tersten sıralanması için ilgili parametrenin önüne ‘–‘ konmalıdır.

Örnek: “/api/data?temp=>30&sort=temp,-createdAt” sorgusu, sıcaklık değeri 30’dan büyük olan kayıtları filtreler, sıcaklık değerlerini artan olarak listeler, aynı sıcaklık değerlerine sahip kayıtların kaydedilme zamanlarına göre yeniden eskiye doğru sıralanmış halini verir.

Sayfalandırma için:

* + - * offset
      * limit parametreleri kullanılmalıdır.

Örnek: “/api/data?temp=>30&fields=time,temp,createdAt&offset=40&limit=20” sorgusu, sıcaklık değeri 30’dan büyük olan kayıtları filtreler, ‘cihazın çalışma zamanı, sıcaklık ve kaydedilme zamanı’ alanlarını getirir, ilk 40 kaydı atlar, 41. kayıttan itibaren 20 kayıt döndürür.

Herhangi bir sorgu sonucunda oluşturulan yanıt (response) JSON formatındadır ve 3 alan içerir: dataCount, data, links.

* + - * dataCount, sayfalama yapılmadan önceki kayıt sayısıdır. Sayfalama yapılmamışsa tüm kayıtların sayısını verir.
      * data, tüm filtreleme işlemleri gerçekleştirildikten sonra elde edilen veriyi içerir. Dizi formatındadır.
      * links, sayfalandırma parametreleri kullanılmamışsa null değerini verir. Kullanılmışsa;
        + “prev”: önceki sayfa linkini,
        + “first”: ilk sayfanın linkini,
        + “next”: sonraki sayfanın linkini,
        + “last”: son sayfanın linkini içerir. Object formatındadır.

Daha detaylı bilgi için kullandığım kütüphanenin web sayfasını ziyaret edebilirsiniz: <https://www.npmjs.com/package/query-to-mongo>

* + **Error Middleware (error.js)**

Express sunucusuna yapılan istekler sonucunda bir hatayla karşılaşılması durumunda istemciye (client) “HTTP 500 Internal Server Error” status kodu ile ‘Something failed.’ mesajını gönderir.

* **Loglama (logger.js)**

Tüm projede, loglama işini kolaylıkla yönetmemize yardımcı olan “winston” kütüphanesinden yararlanılmıştır. Kütüphaneyi projeye dahil ettikten sonra ‘winston.createLogger()’ fonksiyonuyla bir ‘logger’ nesnesi oluşturulur. Fonksiyona parametre olarak ‘options’ objesi verilir.

Winston kütüphanesi, loglama seviyeleri belirlemiştir: error, warn, info, http, verbose, debug, silly. Sırasıyla ilki 0. seviye, sonuncusu 6. seviye olarak derecelendirilmiştir. ‘logger’ nesnesi, fonksiyona gönderilen ‘options’ objesinde ‘level’ etiketiyle belirlenen seviye ve aşağısındakilerin loglanmasını sağlar.

‘transports’ etiketi, loglamanın nereye yapılacağının ayarlandığı kısımdır. Birden fazla yere loglama yapılacaksa bu etikete verilen ‘Transports’ nesneleri bir dizi içerisinde verilir. ‘Transports’ nesneleri, ‘logger.add()’ fonksiyonuyla sonradan da eklenebilir. Fonksiyon, parametre olarak verilen ‘Transports’ nesnesini ekler.

Kullanılan diğer etiket olan ‘exitOnError’ etiketi ise winston tarafından işlenen hata denetimi işlemi sonrasında programın sonlandırılıp sonlandırılmayacağının ayarlandığı etikettir. ‘true’ ise hata mesajı loglandıktan sonra program sonlandırılır. Aksi halde loglama işleminden sonra program çalışmaya devam eder.

Programda dört farklı loglama yeri belirlenmiştir. Dosyalara yapılan loglamalarda, kaydedilen dosyalar “logFiles” klasöründedir.

* + ‘winston-daily-rotate-file’ kütüphanesi eklenerek “YYYY-MM-DD-combined.log” isimleriyle kaydedilen dosyalara loglama yapılması sağlanmıştır. Bu dosyalara ‘info’ seviyesindeki ve daha alt seviyelerdeki loglamalar birlikte yazıldığı için ‘combined’ ismi verilmiştir. Her gün farklı bir dosyaya loglama yapılır. Dosya sayısı ‘maxFiles: ${DAYS\_BEFORE}d’ etiketiyle ayarlanarak DAYS\_BEFORE ortam değişkeninde belirtilen günden daha eski dosyaların silinmesi sağlanmıştır. Dosyaya JSON formatında loglamalar yazılmıştır.
  + ‘winston.transports.File()’ yapıcı fonksiyonuyla ‘error’ seviyesindeki loglamaların ‘error.log’ dosyasına kaydedilmesi sağlanmıştır. Programda herhangi bir hata ile karşılaşıldığında bakılması gereken dosya budur.
  + ‘NODE\_ENV’ ortam değişkeni ‘production’ değerinden farklı bir değere sahip olduğunda konsol ekranına ‘debug’ seviyesi ve altındaki loglamalar yazılmaktadır.
  + “index.js” dosyasında yazılan ‘exception’ ve ‘rejection’ işleyicileri (handler) tarafından yapılan loglamalar ise ‘exceptions.log’ dosyasına yazılmaktadır. Bu kısımda ele alınan hatalarda program sonlandırılır. Programın sonlandırıldığı durumlarda neden sonlandığını anlamak için bakılması gereken dosya budur.

Loglama ile ilgili detaylı bilgilere, ‘winston’ kütüphanesi için <https://www.npmjs.com/package/winston> adresinden, ‘winston-daily-rotate-file’ kütüphanesi için <https://www.npmjs.com/package/winston-daily-rotate-file> adresinden ulaşabilirsiniz.

* **SIGTERM Signal Handling**

“index.js” dosyasının sonunda aşağıdaki kod yazılarak programın sıfır çıkış koduyla sonlanması sağlanmıştır. Docker compose kullanılarak çalıştırıldığında, ‘docker compose down’ komutu ‘SIGTERM’ sinyali gönderir. Bu sinyal geldiğinde tcp sunucu ve express sunucu sonlandırılır. Hata varsa 1 çıkış kodu ile, yoksa sıfır çıkış kodu ile program sonlandırılır.

*process.on('SIGTERM', () => {*

*let tcpErr, expressErr;*

*console.info('SIGTERM signal received.');*

*console.log('Closing tcp server.');*

*server.close((err) => {*

*console.log('Http server closed.');*

*tcpErr = err;*

*});*

*console.log('Closing express server.');*

*expressServer.close((err) => {*

*console.log('Http server closed.');*

*expressErr = err;*

*});*

*const exitCode = (tcpErr || expressErr);*

*process.exit(exitCode ? 1 : 0);*

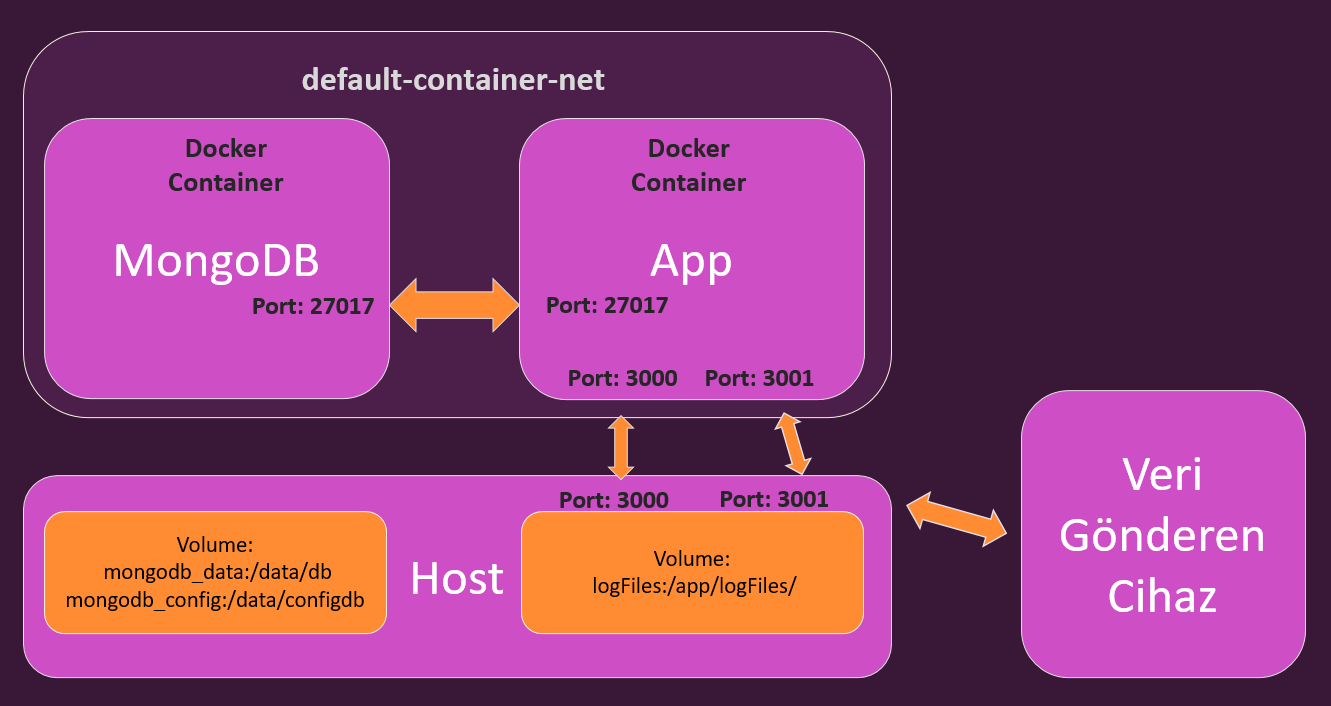
*});*

**Programın Dockerize Edilmesi**

* **Docker**

Docker, açık kaynak kodlu bir ‘container’ teknolojisidir. Docker, aynı işletim sistemi üzerinde, yüzlerce hatta binlerce birbirinden izole ve bağımsız containerlar sayesinde sanallaştırma sağlayan bir teknolojidir. Docker Daemon, Hypervisor’ün Docker’daki karşılığıdır. Bütün CPU ve RAM vb. gibi işletim sistemine ait işlerin yapıldığı bölümdür. Container, Docker Daemon tarafından Linux çekirdeği içerisinde birbirinden izole olarak çalıştırılan process’lerin her birine verilen isimdir. Virtual Machine (Sanal Makina) analojisinde Docker’ı Hypervisor’e benzetirsek fiziksel sunucu üzerinde halihazırda koşturulmakta olan her bir işletim sisteminin (sanal sunucunun) Docker’daki karşılığı Container’dır. Containerlar layer halindeki Image’lardan oluşur. Docker Image ise containerlara kurulacak ve run edilecek olan uygulamaların veya OS’lerin image dosyalarıdır. Örnek verecek olursak mysql, mongodb, redis, ubuntu, mariadb, … gibi yüzlercesi mevcuttur. Docker hakkında detaylı bilgiye ulaşmak için <https://www.docker.com/> sitesini ziyaret edebilirsiniz.

Uygulamada Docker teknolojisinden yararlanmak istememizin sebebi, uygulamanın taşınabilir olmasını ve tek komutla çalışabilmesini sağlamaktır. Programın nasıl ‘Dockerize’ edildiğini anlatacağız. Programın kurulumu ve kullanılmasını sonraki bölümde anlatacağız.

* **Program Şeması**

*Şekil 1: Docker Container’ları Şeması*

Program, iki adet Docker Container içermektedir: MongoDB ve Mam-App.

MongoDB container’ı ‘mongo’ official image kullanılarak oluşturulacaktır (<https://hub.docker.com/_/mongo>). MongoDB veri tabanı 27017 portundan haberleşir. Veri tabanına sadece programımız tarafından erişileceği için ve ‘Docker Compose’ bu container’lar arasında varsayılan olarak bir ağ oluşturacağı için MongoDB container’ında portlarla ilgili bir ayar yapılmayacaktır. MongoDB container kapatıldığında veriler de kaybolur. Verilerin kaybolmasını önlemek için, MongoDB’nin verileri kaydettiği klasörlerin ‘Named Volumes’ yardımıyla host makinede de kaydedilmesini sağlamamız gerekir. Bunun için “/data/db” klasörünü “mongodb\_data” ismiyle, “/data/config” klasörünü “mongodb\_config” ismiyle kaydedilir. Böylece container kapatılıp tekrar başlatıldığında kaydedilmiş olan verilere ulaşabilir.

Mam-App container’ı, yukarıda detayları açıklanan programı içerir. Önce programın “Docker Image”ı oluşturulur. Bunun için “Dockerfile” isminde özel bir dosya oluşturulur. Bu dosyanın içerdiği kodları inceleyelim:

* **FROM node:slim**

Bu kod, oluşturulacak Docker Image’ın ‘node:slim’ official image’ının baz alınacağını söyler. (Bkz. <https://hub.docker.com/_/node> ) Bu image’ın kullanılmasının sebebi ‘node:latest’ image’ından daha az yer kaplaması ve ‘node:slim’ image’ında programımızın çalışabiliyor olmasıdır.

* **WORKDIR /app**

Bu kod, çalışılan dosya yolunun ‘/app’ olarak belirlenmesini sağlar. Bir linux terminalinde ‘cd /app’ komutunu çalıştırmışız gibi davranır.

* **COPY package.json .**

Bu kod, host bilgisayardaki package.json dosyasını workdir olan /app klasörüne kopyalar.

* **RUN npm install**

Bu kod, ‘npm install’ komutunu çalıştırarak projede kullanılan kütüphanelerin /app/node\_modules klasörüne indirilmesini sağlar.

* **COPY . .**

Bu kod, Dockerfile’ın bulunduğu klasördeki tüm dosyaları /app klasörüne kopyalar. Eğer “.dockerignore” dosyası varsa burada ismi belirtilen dosyaları kopyalamaz.

**Not:** Dosyaların kopyalanmasının yukarıdaki sırayla yapılmasının sebebi, kullanılan kütüphanelerin bulunduğu /app/node\_modules klasörünün önce oluşturulmasının istenmesidir. Image oluşturulurken ‘npm install’ kodunun çalıştırılmasıyla gerekli kütüphane dosyaları indirilir ve bu işlem internet hızına bağlı olarak uzun sürebilir. Dockerfile satır satır çalıştırılır ve çalıştırılan her komut sonrasında ‘cache’lenir. Dockerize ettiğimiz programın kaynak kodlarında bir değişiklik yaptığımızda image build edilirken sadece bu değişiklikler işlenir ve her seferinde, kullanılan kütüphanelerin indirilmesine gerek kalmaz. Bu da bize zaman kazandırır.

* **CMD [“node”, “index.js”]**

Bu kod, Docker image’dan container oluşturulduktan sonra çalıştırılmasını istediğimiz kodu belirttiğimiz yerdir. Container, yukarıdaki adımlar işletilerek oluşturulur, sonra “node index.js” kodu çalıştırılır. Bir linux terminalinde yazılmış gibi davranır. “node” image’ından başlattığımız için “node” kodu çalıştırılabilir. Dosyalarımız /app klasörüne kopyalandığı için “index.js” dosyasına da erişilebilir ve program çalıştırılmış olur.

Programın doğru çalışabilmesi için “mongo” container’ının da başlatılmış olması gerekir. Host bilgisayarda ayrı ayrı “docker run <image adı>” şeklinde kodların çalıştırılması yerine “docker-compose.yaml” dosyası oluşturulur ve tek bir komutla container’ları ayağa kaldırılabilir. Şimdi de “docker-compose.yaml” dosyasını inceleyelim.

Dosya “.yaml” dosya formatına uygun olarak yazılır (Bkz. <https://en.wikipedia.org/wiki/YAML>). Bu dosyada “version”, “services”, “volumes” üst seviye elemanları yazılmıştır.

* **version: ‘3.8’**

“version” docker compose dosyasının hangi versiyona göre yazıldığının belirtildiği elemandır.

* **services:**

**mongodb:**

**image: mongo**

**container\_name: mongodb**

**volumes:**

**- mongodb\_data:/data/db**

**- mongodb\_config:/data/configdb**

**mam-app:**

**build: ./**

**ports:**

**- ${EXPRESS\_PORT}:${EXPRESS\_PORT}**

**- ${TCP\_PORT}:${TCP\_PORT}**

**volumes:**

**- ${****LOG\_FILES\_VOLUME\_PATH}:/app/logFiles**

“services” elemanının alt seviye elemanları, başlatılacak container’lardır. Uygulamamızda “mongodb” ve “mam-app” isimli iki servis başlatılacaktır.

“mongodb” servisinde container adı, docker image’ı ve verileri kaydedeceği volume’ler alt eleman olarak burada belirtilmiştir. Resmi “mongo” image’ı kullanılmıştır. Host bilgisayarda bu image yoksa docker hub’dan indirilir. Container’lar arasındaki ağı kullanabilmek için container’a bir isim verilmiştir. <https://hub.docker.com/_/mongo> web sitesindeki dokümanda veri tabanının verileri kaydettiği klasörlerin “/data/db” ve “/data/configdb” olduğu belirtilmiş. Bu klasörlere birer Docker Volume atanmıştır.

“mam-app” servisinde, aynı klasörde bulunan Dockerfile yardımıyla uygulamanın Docker image’ının oluşturulması için “build: ./” elemanı eklenmiştir. Host bilgisayarın TCP\_PORT numaralı portuna gelen verinin mam-app container’ının TCP\_PORT numaralı portuna aktarılması ve EXPRESS\_PORT numaralı portuna gelen http isteklerinin, container’ın EXPRESS\_PORT numaralı portuna aktarılması işlemleri “ports:” elemanının eklenmesiyle sağlanmıştır. Uygulama, log dosyaları oluşturmaktadır. Bu dosyaların saklanması için container’daki LOG\_FILES\_VOLUME\_PATH klasörüne bir Docker Volume eklenmiştir.

**Not: “**docker-compose.yaml” dosyasında, hazırlanmış olan “ahmetcetinkou/mam-app” image’ını kullanmak için aşağıda belirtilen değişiklikler yapılmalıdır:

**mam-app:**

**image: ahmetcetinkou/mam-app**

**ports:**

**- ${EXPRESS\_PORT}:${EXPRESS\_PORT}**

**- ${TCP\_PORT}:${TCP\_PORT}**

**volumes:**

**- ${LOG\_FILES\_VOLUME\_PATH}:/app/logFiles**

**environment:**

**- NODE\_ENV=${NODE\_ENV}**

**- TCP\_PORT=${TCP\_PORT}**

**- EXPRESS\_PORT=${EXPRESS\_PORT}**

**- MONGODB\_CONNECTION\_STRING=${MONGODB\_CONNECTION\_STRING}**

**- DAYS\_BEFORE=${DAYS\_BEFORE}**

**- TIMEZONE=${TIMEZONE}**

**- ERROR\_TOLERANCE=${ERROR\_TOLERANCE}**

**- LOG\_FILES\_VOLUME\_PATH=${LOG\_FILES\_VOLUME\_PATH}**

“environment:” altında, varsayılan değerleri kullanmak istemediğiniz ve ‘docker-compose.yaml’ dosyası ile aynı dizinde olan .env içindeki ortam değişkenlerini eklemeniz gerekir. “.env” dosyasında bulunan tüm değişkenler eklenmelidir.

* **volumes:**

**mongodb\_data:**

**mongodb\_config:**

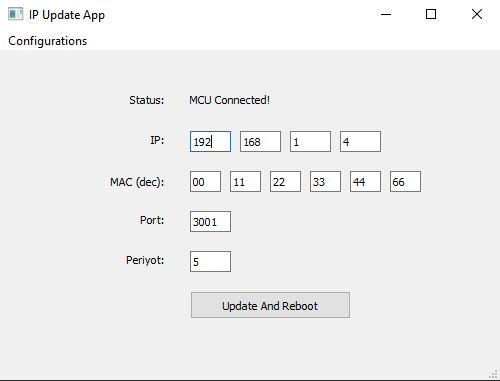
“volumes” elemanında, tüm servislerde adreslenen Named Docker Volume’ler alt eleman olarak eklenmiştir.

**Programın Diğer Cihazlarda Çalıştırılması İçin Gereksinimler**

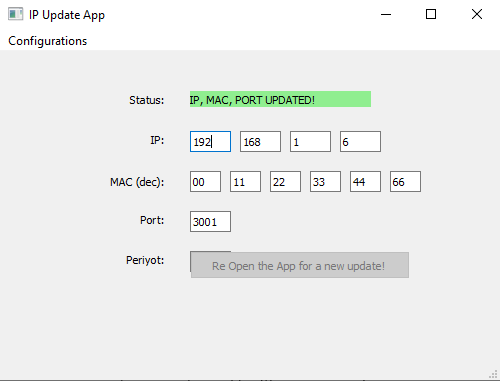
Programın çalıştırılacağı sunucuya docker ve docker compose kurulmalıdır (bkz. <https://docs.docker.com/get-docker/> ). Projenin içeriği ve çalışması bölümünde belirtilen ortam değişkenleri, docker-compose.yaml dosyasıyla aynı dizinde bulunacak olan ‘.env’ dosyasında, kullanım amacına göre oluşturulmalıdır. Sunucuya, docker-compose.yaml dosyası ve proje dosyaları aynı klasör içerisinde olacak biçimde kaydedilmelidir.

**Programın Çalıştırılma Adımları**

1. Cihaz ile sunucunun birbiriyle haberleşebilmesi için aynı ağa bağlayınız.
2. Sunucunun IP adresini ve cihazdan gelen veriler için ayıracağınız port numarasını belirleyiniz.
3. Cihazdan kaç saniyede bir veri alacağınızı kararlaştırınız.
4. Cihazı usb ile bir bilgisayara bağlayınız. Bilgisayarınızdan “IPUpdateApp” programını çalıştırınız. Programın görseli aşağıdadır. Cihaz bilgisayara bağlanmışsa “Status: MCU Connected!” yazmalıdır. Aksi halde bağlantıları kontrol ediniz.



Şekil 2: IPUpdateApp Uygulamasının Arayüzü

1. IP alanına sunucunun IP numarasını, Port alanına cihazdan gelen veriler için ayıracağınız port numarasını, Periyot alanına cihazdan kaç saniyede bir veri almak istiyorsanız o değeri yazınız. “Update and Reboot” butonuna tıklayınız. İşlem başarılı olduğunda aşağıdaki şekilde gösterildiği gibi “Status: IP, MAC, PORT UPDATED!” yazısı görülecektir.

Şekil 3: IPUpdateApp uygulamasında işlem başarılı ise karşılaşılan ekran.

1. Cihazı bilgisayardan çıkarınız ve ölçüm yapılacak bölgede uygun bir yere koyunuz. Cihazın adaptörünü prize takınız. Ethernet girişinden modeme bağlayınız.
2. Aşağıdaki adımları gerçekleştirmeden önce programın kurulacağı sunucuda Docker daemon’ın başladığından emin olunuz.
3. Eğer uygulamanın çalıştırılacak olduğu sunucu ‘www’ ağına bağlanabiliyorsa sadece ‘docker-compose.yaml’ ve ‘.env’ dosyalarını sunucuda uygun bir klasöre kopyalayınız (‘docker-compose.yaml’ dosyasında mam-app servisi altında ‘image: ahmetcetinkou/mam-app’ yazıldığına emin olunuz). 10. maddeden devam ediniz.
4. Eğer uygulamanın çalıştırılacak olduğu sunucu ‘www’ ağına bağlanamıyorsa, internete bağlanabilen ve docker yüklü olan herhangi bir bilgisayara ‘ahmetcetinkou/mam-app’ ve ‘mongo’ image’larını indiriniz:
   * 1. docker pull ahmetcetinkou/mam-app
     2. docker pull mongo

İndirdiğiniz image’ları aşağıdaki kodları çalıştırarak ‘.tar’ uzantılı sıkıştırılmış dosyaları oluşturunuz:

* + 1. docker save -o <dosya yolu>/ahmetcetinkou-mam\_app.tar ahmetcetinkou/mam-app
    2. docker save -o <dosya yolu>/mongo.tar mongo

‘ahmetcetinkou-mam\_app.tar’ , ‘mongo.tar’ , ‘docker-compose.yaml’ ve ‘.env’ dosyalarını, programı çalıştıracağınız sunucuda uygun bir klasöre taşıyınız.

Aşağıdaki komutları çalıştırarak docker image’larını sunucuya yükleyiniz.

* + 1. docker load < ahmetcetinkou-mam\_app.tar
    2. docker load < mongo

1. Ortam değişkenlerini kullanım amacınıza uygun olarak, ‘docker-compose.yaml’ dosyasıyla aynı dizinde olan ‘.env’ dosyasında ayarlayınız.
2. “docker-compose.yaml” dosyasının bulunduğu dizinde terminal ekranında aşağıdaki kodu çalıştırınız:

* docker compose up

1. Programı sonlandırmak istediğinizde aşağıdaki komutu başka bir terminal ekranından çalıştırabilirsiniz.

* docker compose down

1. Programda bir hata ile karşılaşılırsa ya da log dosyaları incelenmek istenirse, LOG\_FILES\_VOLUME\_PATH ortam değişkeniyle ayarlanan dosya yoluna bakılabilir.