**GAZİ ÜNİVERSİTESİ MÜHENDİSLİK FAKÜLTESİ**

**BİLGİSAYAR MÜHENDİSLİĞİ**


# BM495 BİLGİSAYAR PROJESİ I

## Yazılım Projelerine Yönelik Proje Yönetim Yazılımı Geliştirilmesi

## Software Design Description (SDD) Dokümanı

Mert UYĞUR - 201180753

Mücahid Bilal KESKİN - 191180759

Kelime sayısı: 2843

# İÇİNDEKİLER

# 1. KAPSAM

## 1.1 Tanım

Yazılım projelerinin yönetimini kolaylaştırmayı amaçlayan “Yazılım Projelerine Yönelik Proje Yönetim Yazılımı”, yazılım proje takımlarının ihtiyaç duyduğu birçok şeyi bünyesinde içeren ve günümüzdeki proje yönetim yazılımlarındaki karmaşıklığı da gidererek, takımlara projelerini geliştirilmesi için gerek duyulan her fonksiyonu da sağlayan bir sistemdir.

## 1.2 Sisteme Genel Bakış

Sistemin bütün testlerden geçip gerçek dünyaya açıldıktan sonra amacı, yazılım geliştirme takımlarının ihtiyaç duyduğu çoğu aracı ve fonksiyonu içinde barındırarak, takımlara projelerini rahatça yönetebileceği bir sistem sunmak olacaktır.

Bu hedefle sistem, en üst seviye ayrımı “organizasyon” seviyesinde yapmaktadır. Gerçek dünyaya uygun olacak biçimde bu birim gerçek dünyada “şirket” kavramına karşılık olarak görülebilir. Yine bu birimin altında gerçek dünyaya uygun olarak, her şirketin birden fazla ürünü, projesi olabileceği gibi, “organizasyon” biriminin altında “proje” birimleri oluşturulacaktır. Yapılacak işin en genel kavramı bu “proje” birimine karşılık gelmektedir. Bu işi gerçekleştirecek olan kullanıcılar, geliştiriciler ise bu “proje” birimlerine dahil olacaktır. Takip edilecek ve sistem hakkında genel bilginin toplanacağı birim “proje” birimi olacaktır. Takımların bu “proje” birimleri içerisinde güçlü bir iletişim ağında geliştirme yapmaları sağlanacaktır.

## 1.3 Dökümana Genel Bakış

Bu Software Design Description dokümanında “Yazılım Projelerine Yönelik Proje Yönetim Yazılımı”nın nasıl tasarlanıp, nasıl uygulanacağına değinilecektir. Bu doküman içerisinde diyagramlar, tablolar, tasarımlar gösterilip uygulama hakkında daha tasarımsal detaylara öncelik verilecektir.

# 2. İLGİLİ DÖKÜMANLAR

Aynı projenin Software Requirements Specification (SRS) ve Software Project Management Plan (SPMP) dokümanlarına referans yapılmıştır. Sözü geçen dokümanlar proje ile ilgili kapsayıcı bilgiler barındırmaktadır.

# 3. YKE ÇAPINDA TASARIM KARARLARI

## 3.1 Arayüz

Kullanıcıyı uygulamanın giriş ekranı karşılar. Kullanıcı bu bölümde doğrulandıktan sonra kullanıcının veritabanındaki verisi bir “payload” içerisinde kullanıcı tarafına gönderilir ve gösterilir. Uygulama tek sayfa üzerinde olacaktır. Yani kullanıcı işlem yaptıktan sonra sayfa

yenilenmeyecektir yada arayüzde gösterilen değerler değiştiğinde güncellenmesi için sayfa yenilenmesi gerekmeyecektir. Dinamik bir sayfada tüm işlemler gerçekleştirilecektir.

### 3.2 Sunucu

Gereksinim değişiklikleri olabileceğinden sunucu tarafında mümkün olduğunca ölçeklenebilir bir yapı kurulmaya çalışılmıştır. Bunun için AWS Lambda ve AppSync kullanılacaktır. Örnek olarak yeni bir fonksiyon eklediğimizde yeni bir Lambda fonksiyonu oluşturup bu fonksiyonu AppSync e bağlayarak sistemi devre dışı bırakmadan yeni özellik devreye sokulabilir. Kullanıcı tarafında deneyimin yüksek tutulması için istekler mümkün olduğunca hızlı cevaplanmalıdır. Sunucu tarafında istekler 1000 ms sürmeden cevaplanmalıdır bunun için Lambda fonksiyonunun varsayılan işlem gücü kapasitesi artırılmıştır 3 MB ve üzeri veri içeren isteklerde %50 ye yakın süre kazanılmıştır. Güvenlik için kod içinden erişilebilir API linkleri kullanılmayacaktır bunun yerine bütün bir ekosistem olan Amplify-Cognito-AppSync üç servisi entegre olacak ve istekler Cognito servisi tarafından doğrulanmadan işlenmeyecektir. Detaylı anlatımı dökümanın ileriki bölümlerinde gösterilmiştir.

## 4 YKE'NIN YAPISAL TASARIMI

### 4.1 YKE Bileşenleri

#### 4.1.1 AWS Amplify

Uygulama AWS Amplify üzerine kurulmuştur. Bu servis hem diğer AWS hizmetleri ile bağlantı kurmamızı sağlayan hemde web uygulamasının domain ve hosting ihtiyaçlarını karşılayan bir servistir. Uygulamanın geliştirildiği github üzerinde bulunan repo muz Amplify servisine bağlıdır. Böylece "main branch"de değişiklik yaptığımızda değişiklik otomatik olarak Amplify üzerinde güncellenir. Amplify servisi Cognito kullanıcı havuzuna bağlıdır. Kayıt olma, giriş yapma, çıkış yapma işlemleri Cognito üzerinden gerçekleştirilir.

#### 4.1.2 Cognito User Pool

Bu servis uygulamanın güvenlik kısmı için konfigüre edilmiştir. Uygulamaya yapılan giriş-çıkış-kayıt işlemlerinin bilgisi bu servis üzerinde tutulur. Kullanıcı "şifremi unuttum" seçeneğini seçtiğinde kayıt olurken bildirdiği mail'e bir şifre sıfırlama maili gider. Bunun dışında kullanıcının sistemde yapacağı işlemler için kullanılacak token ler bu servis üzerinde doğrulanır.

#### 4.1.3 GraphQL API

Sistemin ölçeklenebilir ve idame edilebilir olması için gelişmiş bir API türü olan GraphQL API tercih edilmiştir. Bu API, AWS AppSync üzerine yerleştirilmiştir. Kullanıcının uygulama içerisindeki bütün istekleri bu "API"ye ulaşır ardından işlenmek üzere Lambda fonksiyonuna aktarılır. Lambda fonksiyonunda istek işlendikten sonra cevap tekrar API üzerinden kullanıcıya ulaşır.

### 4.1.4 Lambda Fonksiyonu

Lambda fonksiyonları bulut sunucu üzerinde çalıştırabildiğimiz geçici bir bilgisayardır. Çeşitli programlama dillerine desteği vardır. Lambda üzerinde veri işlemleri çalıştırılacağı için bu alanda kullanışlı olan Python dili tercih edilmiştir. İki ayrı Lambda fonksiyonu bulunmaktadır. Birisi verilerin getirilmesi için diğeri ise verilerin değiştirilmesi için kullanılmaktadır.

### 4.1.5 DynamoDB Veritabanı

Kullanıcıların bütün verileri bu veritabanında oluşturulacak tablolarda tutulacaktır. Kullanılan tablolar sırası ile kullanıcı veri tablosu(Tablo 1), organizasyon veri tablosu(Tablo 2), proje veri tablosu(Tablo 3) ve görev veri tablosu(Tablo 4)'dur. Tablolar aşağıda gösterilmiştir.

| **Veri** | **Tip** | **Açıklama** | |
|---|---|---|---|
| Id | Yazı | Kullanıcı "ID"si. | Otomatik olarak oluşturulur. |
| İsim | Yazı | Kullanıcı ismi. | |
| Şifre | Yazı | Kullanıcı şifresi. | Bu değer şifrelenmiş bir biçimde veritabanında tutulmalıdır. |
| Email Adresi | Yazı | Kullanıcı email adresi. | |
| Organizasyonlar | Dizi<Yazı> | Kullanıcı'nın dahil olduğu organizasyonlar. | Bu dizi, organizasyonların id'lerini içeren bir dizidir. Çünkü birden fazla aynı isimde organizasyon bulunabilir. |
| Projeler | Dizi<Yazı> | Kullanıcı'nın dahil olduğu projeler. | Id dizisi. |
| Görevler | Dizi<Yazı> | Kullanıcı'nın atanmış olduğu görevler. | Id dizisi. |

Tablo 1: Kullanıcı veri tablosu.

| **Veri** | **Tip** | **Açıklama** | |
|---|---|---|---|
| Id | Yazı | Organizasyon “ID”si. | Otomatik olarak oluşturulur. |
| İsim | Yazı | Organizasyon ismi. | Organizasyonu tanımlayan bir değer değildir. Birden fazla aynı olabilir. |
| Projeler | Dizi<Yazı> | Organizasyonun sahip olduğu projeler. | |
| Üyeler | Dizi<Yazı> | Kullanıcıların “ID”lerinin dizisini tutar. Bunun yanında organizasyondaki rolünü de belirtir. | |

Tablo 2: Organizasyon veri tablosu.

| **Veri** | **Tip** | **Açıklama** | |
|---|---|---|---|
| ID | Yazı | Proje “ID”si. | |
| İsim | Yazı | Proje ismi. | |
| Açıklama | Yazı | Proje açıklaması. | |
| Üyeler | Dizi<Yazı> | Üye ID’leri. | |

Tablo 3: Proje veri tablosu.

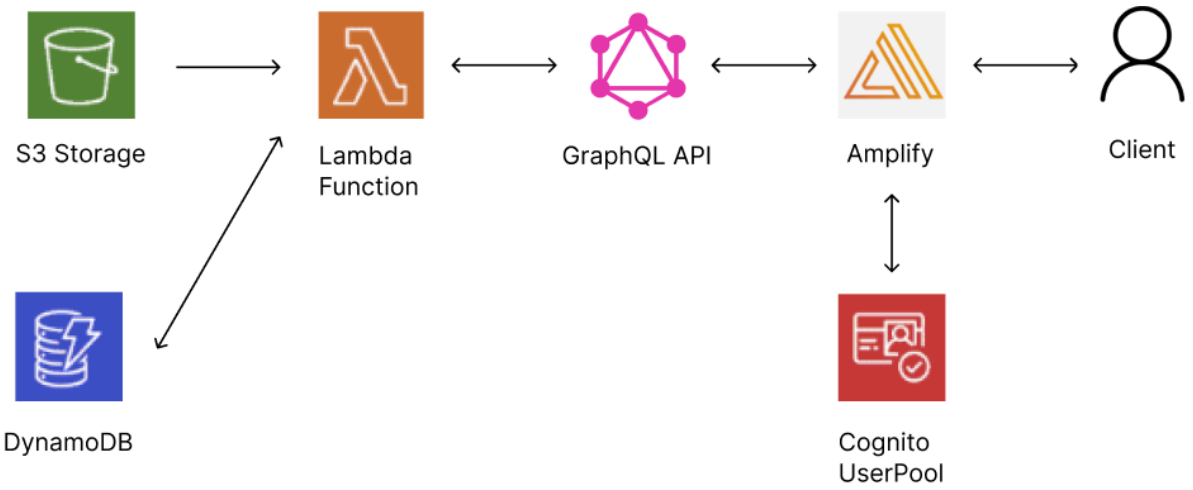
| **Veri** | **Tip** | **Açıklama** | |
|---|---|---|---|
| ID | Yazı | Görev “ID”si. | |
| İsim | Yazı | Görev ismi. | |
| Açıklama | Yazı | Görev açıklaması. | |
| Atanan | Yazı | Atanan üye “ID”si. | |
| Zorluk | Sayı | 1, 2, 3, 4 ve 5 numaralı zorluk dereceleri. | |
| Durum | Yazı | Görevin durumu. | |

Tablo 4: Görev veri tablosu.

### 4.1.6 S3 Depolama Servisi

Depolama servisi kullanıcıların üzerinde daha az değişiklik yapacağı veriler(silinen organizasyonlar, yedek) için kullanılacaktır. Şu an sistemin en az kullanılan elemanıdır. Kapatılmış bir proje, görev veya organizasyonun verilerini DynamoDB veritabanında tutmak çok daha maliyetli olduğundan bu tipteki veriler .json formatında işlenerek depolama servisinde tutulacaktır. Her kullanıcının bu depolama servisinde bir klasörü bulunacaktır ve verileri bu klasörlerde saklanacaktır.

## 4.2 Genel Çalıştırma Kavramı



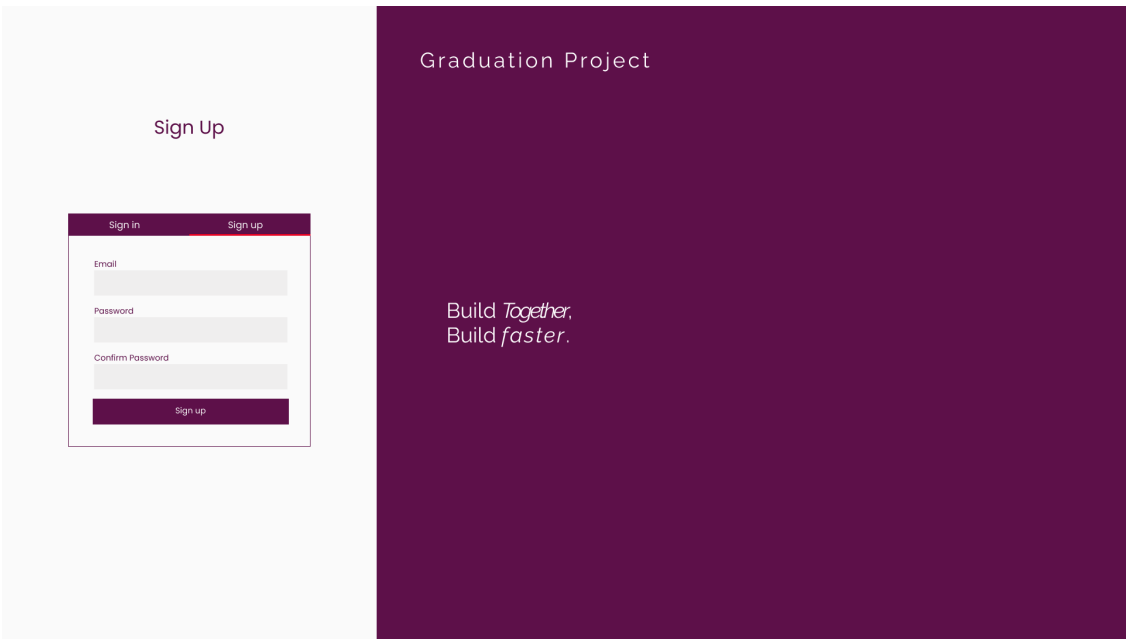

Şekil 4.1

Web uygulaması AWS Amplify üzerine kuruludur bu servis uygulamanın hem hosting hemde domain işlevini yerine getirmektedir ayrıca diğer AWS servisleri ile iletişimi konfigüre edebildiğimiz servistir. Kullanıcının sistem üzerindeki istekleri Amplify üzerinden GraphQL API a ulaşır. İsteğin türüne göre istek işlenmek üzere ilgili Lambda fonksiyonuna aktarılır. Lambda fonksiyonu DynamoDB veritabanı ve S3 depolama servisine bağlıdır. İsteğin içeriğine göre veritabanı veya S3 depolama servisi üzerinde işlem yaparak isteği cevaplar. Uygulamanın genel çalışma şeması Şekil 4.1 de gösterilmiştir. Aşamaların detaylı anlatımı başlık 4.1 de anlatılmıştır.

## 4.3 Arayüz Tasarımı

### 4.3.1 Sistem Kayıt Olma Arayüzü

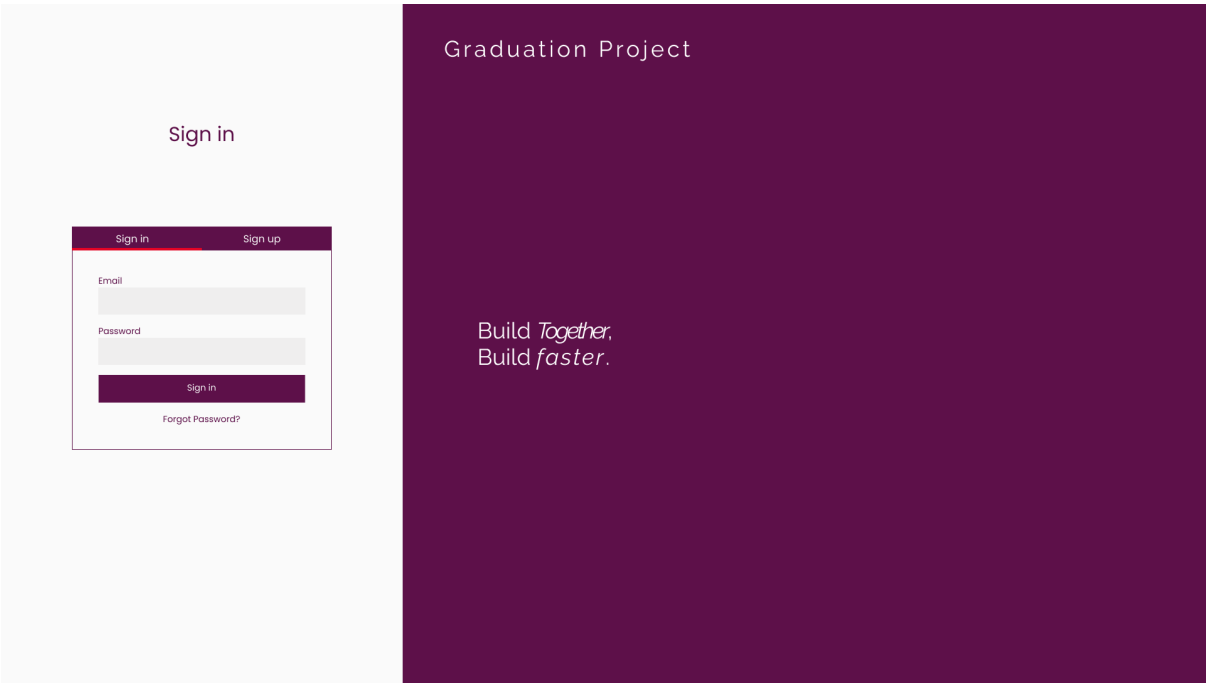
Kullanıcının sistemle ilk iletişimini sağlayacak olan kayıt olma arayüzü, kullanıcıyı tanımlama ve sisteme kayıt etme fonksiyonlarını gerçekleştirecektir. Veritabanı ile birebir iletişimde olacak sistem, buradan alınan kullanıcı bilgilerini DynamoDB veritabanına yazacak, daha sonra bu bilgiler ile gelen kullanıcı tanınacaktır.




Şekil 4.2: Kayıt olma arayüzü

### 4.3.2 Sistem Giriş Yapma Arayüzü

Sistem kayıt olma arayüzü ile birçok benzerlik taşıyan giriş yapma arayüzü, kullanıcıdan aldığı girişler açısından farklılık taşır. Kullanıcının kayıt olma işlemi başarılı bir şekilde gerçekleşti ise, bu bilgiler DynamoDB veritabanına yazılmıştır ve kullanıcı artık sistem tarafından tanınabilir hale gelmiştir. Sistem tarafından tanınabilir hale gelen kullanıcıdan alınacak olan email adresi ve şifre ile birlikte artık kullanıcılar sisteme giriş yapma fonksiyonunu gerçekleştirebilir hale geleceklerdir.




Şekil 4.3: Sistem giriş yapma arayüzü.

Burada, kullanıcıdan istenen email adresi ve şifre, sistem kayıtları ile karşılaştırılacak, eğer bilgiler geçerli ise sistem giriş yapma fonksiyonunu başarılı bir şekilde gerçekleştirdiğini “frontend” tarafına dönecek ve bu bilgi ile birlikte kullanıcıyı tanımlayabileceği “token”ler dönecektir.

### 4.3.3 Sidebar ve Erişebilirlik Arayüzü

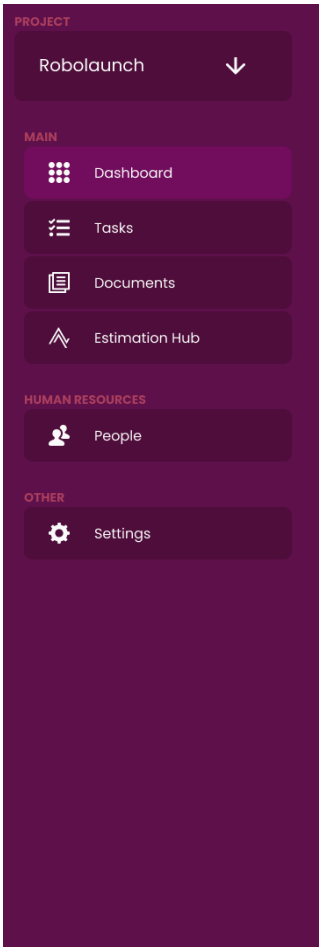
“Sidebar” komponenti erişilebilirliği artırması açısından uygulama için kritik öneme sahiptir. Bu birimin doğru tasarlanması ve kullanıcının sıkça erişmek isteyeceği bilgileri barındırması elzemdir. Bu sebeple farklı sayfalar altında kullanılmak üzere iki adet “sidebar” birimi tasarlanmıştır.

İlki, kullanıcı henüz bir proje sayfasına girmediğinde, ona dahil olduğu tüm projelerin durumunu anlık olarak takip edebileceği sayfanın altında yer alacaktır.


Şekil 4.4: İlk “sidebar”.

Burada kullanıcı dahil olduğu projeleri ve anlık görev ilerlemelerini görebilmektedir. Ayrıca tüm projelerdeki ekip arkadaşlarını da görmek isterse “Human Resources” bölümünün altındaki “People” sayfasına girmelidir. Son olarak şu anda görüntülediği organizasyon ile ilgili ayarları düzenlemek isterse, “organization settings” sayfasına girmelidir. Bu ayar kısmı, sadece organizasyon adminlerine görünür durumda olacaktır.

İkinci “sidebar”, bir proje sayfasının içine girildiğinde görünür olacaktır. Burada ise kullanıcı proje ile ilgili daha detaylı bilgilerin yer aldığı sayfalar arasında geçiş yapabilecektir.




Şekil 4.5: İkinci “sidebar”.

Herhangi bir projenin sayfasına giren kullanıcı, sayfanın sol tarafında bu “sidebar” ile karşılaşacaktır. Burada, içinde bulunduğu proje ismi ile birlikte projenin diğer sayfaları arasında geçiş yapabilecektir. “Dashboard” sayfasında kullanıcı, proje ile ilgili en genel bilgileri görebilecektir. Burada projenin ilerleme durumu, insanlar, ve anlık görevler gibi komponentler bulunacaktır. “Tasks” sayfası sadece görevler ile daha detaylı bilgilerin bulunduğu, kimin hangi görev üzerine çalıştığı bilgileri bulunacaktır. “Documents” sayfası, proje ile ilgili, hazırlanmış dokümanları içermektedir. Her kullanıcı kendi alanı ile ilgili dokümanları burada tutacaktır. “Estimation Hub”, kimin hangi görevi, kaç günde tamamlayacağının yapay zeka tarafından tahmin edildiği kısımdır. Bu sayfa sadece proje ve organizasyon yöneticileri tarafından görülebilir olacaktır. “Human Resources” bölümünün altındaki “People” sayfasında projedeki kullanıcıların eklenip çıkarılması ve görüntülenmesi işlemleri yapılabilecektir. Ekleme ve çıkarma işlemlerini sadece yönetici yetkisine sahip olan kullanıcılar gerçekleştirebilecektir. “Other” bölümü altındaki “Settings” sayfasında proje ile ilgili ayarlar bulunacaktır. Burada proje isminin değiştirilmesi, kullanıcı yetkilerinin atanması gibi işlemler yapılabilecektir.

### 4.3.4 Proje Kartlarının Tasarım Arayüzü

Sistemdeki kullanıcılar dahil oldukları projeler hakkında bilgi sahibi olmak için proje sayfalarına girmek zorunda olmayacaklardır. Birçok projenin aynı sayfadan görüntülenebildiği sayfalarda her proje için hazırlanmış kartlar ile en genel bilgiler edinilebilecektir.


Şekil 4.6: Proje kartları.

Bu kartlar vasıtasıyla kullanıcıyla o proje hakkında bilmek isteyeceği en genel bilgiler sunulacaktır. Ayrıca yan tarafta bulunan ayar kısmı ile de projenin ayarlar sayfasına kısa yoldan gitmesi sağlanacaktır. Bu kart üzerinde kullanıcı, projedeki aktif kullanıcı sayısı, genel ilerleme durumu gibi bilgileri görüntüleyebilecektir.

## 5. YKE DETAYLI PLANI

### 5.1 AWS Amplify Yazılım Birimi

Amazon servisi olan “Amplify” ile iletişimi sağlayacak olan bu birim, amazon servislerini de kullanmamıza imkan sağlayacaktır. Bu servis ile kurulacak bağlantı için, Amazon Web Servisleri’nin kendi hazırladığı kütüphaneler kullanılacaktır. Bu kütüphaneleri kullanarak servislerle iletişim kurulacak, gerekli talimatlar verilecektir.

Amazon servisi ile iletişimi sağlayacak olan kütüphane, “frontend” uygulamasına dahil edilecektir. Bu uygulama “Javascript” dili ile yazılacağından, Amazon’un bu dil için sunduğu kütüphane kullanılacaktır. Bu kütüphane ile “Amplify”a bağlantı gerçekleştirilecek, uygulamanın yönetiminin bir kısmını bu kütüphaneye devretmiş olacağız. Bu servisin bize sunduğu veritabanı hizmeti olan “DynamoDB”, yetkilendirme servisi olan “Cognito” kullanılacak olup, tamamen dinamik olan bu servislerin istediğimiz kısımlarını istediğimiz biçimde değiştirebiliyor olacağız.

“AWS Amplify” servisinin kodun içerisine dahil edilmesini sağlamak için Amazon’un sağladığı kütüphane olan “aws-amplify” kütüphanesi kullanılacaktır.

```
import { API } from "aws-amplify";
```

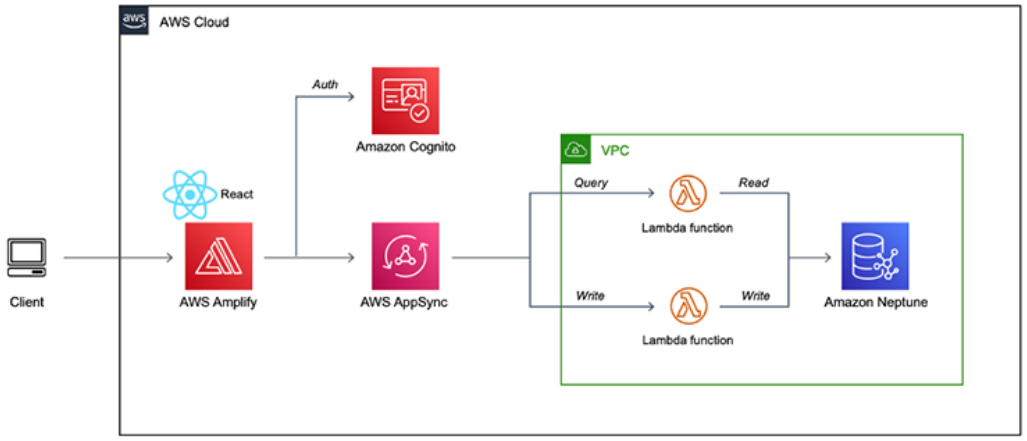
Şekil 5.1: Amplify kütüphanesinin dahil edilmesi.

Uygulamaya dahil edilen kütüphane ile Amplify üzerine kurulu olan uygulama için geliştirdiğimiz  GraphQL API' a istek gönderilebiliyor olunacaktır.

```
const newTodo = await API.graphql({ query: mutations.createTodo, variables:
{input: todoDetails}});
```

Şekil 5.2: "GraphQL API" a istek gönderilmesi.

Bu isteği atmak için gerekli bilgilerin ayar dosyasına girildiğini belirtmekte fayda var. Kütüphaneyi dahil ettikten sonra "Amplify" servisi işleri oldukça kolaylaştırarak tek bir fonksiyon ile iletişimi sağlamayı sağlamıştır. "GraphQL" arayüzü ile iletişim kurarken bu fonksiyon kullanılacak olup, veritabanı ile iletişimde işleri oldukça kolaylaştıracaktır.




Şekil 5.3: Amplify servisinin çalışma mantığı ve diğer servislerle iletişimi.

Amplify servisi, bir üst servis gibi düşünülebilir. Amazon, daha önce yaptığı servisler ile bir uygulama geliştirmeyi kolaylaştıran ayrı bir servis olarak "Amplify"ı geliştirmiştir. Bizim de bu amaçla kullanacağımız "Amplify" servisi, yetkilendirme sistemini kurmamızda, veritabanı işlevi görecek bir yapı oluşturmamızda ihtiyaçlarımızı karşılayacaktır.

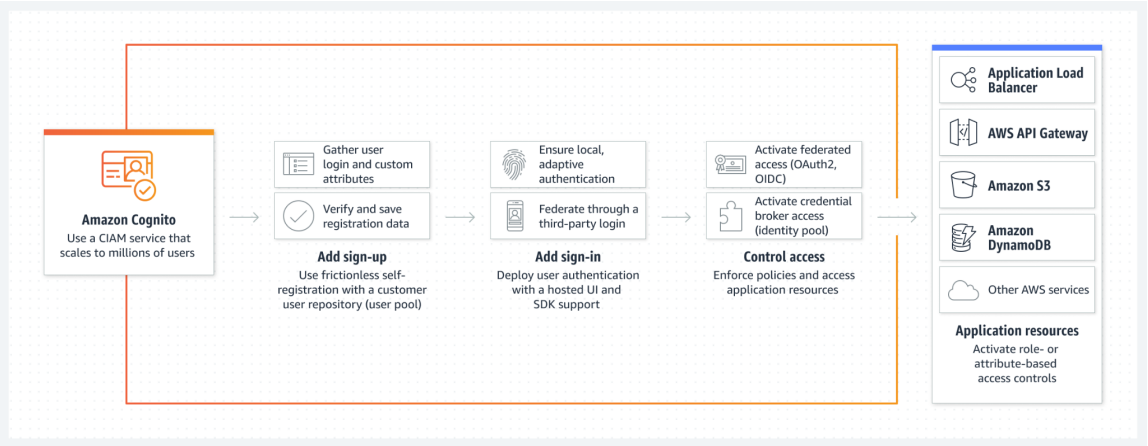
Kodun yazılması kısmında, "Javascript" dilinin fonksiyonel özellikleri kullanılarak, yapılacak her işlem fonksiyonlara ayrılacaktır. Bu fonksiyonlar da kodun içerisinde farklı lokasyonlarda konumlandırılırak kodun okunması ve yönetilmesi daha kolay hale getirilecektir. Bu amaçla, Amazon Web Servisleri üzerinde yapılacak her işlemin hangi kategoriye girdiği belirlenip, kendi dosyasına yerleştirilecektir.

**5.2 AWS Cognito**

Amazon Web Servisleri'nden biri olan "AWS Cognito", uygulama tarafından kullanıcıları saklayıp yetkilendirme işlemlerini gerçekleştiren birim olarak tanımlanabilir. Bu birim daha

çok “Amplify” tarafından kullanılacak olsa da, gereksinim duyulduğunda, herhangi bir yetkilendirme ayarı ayrıca eklenmek istediğinde değiştirilebilir olacaktır.

Uygulamadaki yetkilendirme işlemlerinden sorumlu olan “AWS Cognito” servisi, kayıt olma ve giriş yapma fonksiyonlarını da yönetecektir. Daha sonra kullanıcıların hangi kaynaklara erişip erişemeyeceğini belirleyen bu servis, kaynakları korurken kullanılacak olan önemli servislerden biri olacaktır.




Şekil 5.4: Cognito servisinin çalışma prensibi.

Sistemin içerisinde bulunun kısıtlamalar organizasyon, proje ve görev seviyesinde olacaktır. Bu üç aşamada gerekli duyduğumuz yetki kısıtlamalarını karşılamak için “AWS Cognito” servisi kullanılacaktır. Öncelikle organizasyon kaynağına erişmek için gerekli olan roller belirlenecek, bu roller “AWS Cognito” servisine tanıtılıp, bu role sahip olmayanların kaynaklara erişmesi engellenecektir.

Bu yazılım birimini projeye dahil ederken “client-cognito-identity-provider” kütüphanesi kullanılacaktır.

```
import { CognitoIdentityProviderClient, AddCustomAttributesCommand } from
"@aws-sdk/client-cognito-identity-provider";
```

Şekil 5.6: “AWS Cognito” için gerekli kütüphanenin “Javascript” uygulamasına dahil edilmesi.

Daha sonra bu kütüphanenin içinden çağrılacak olan “CognitoIdentityProviderClient” fonksiyonu ile, “AWS Cognito” ile iletişim kanalımızı oluşturacak olan “client” objesi oluşturulmuş olacaktır.

```
// a client can be shared by different commands.
const client = new CognitoIdentityProviderClient({ region: "REGION" });

const params = {
  /** input parameters */
};
const command = new AddCustomAttributesCommand(params);
```

Şekil 5.7: “AWS Cognito” için “client” objesinin oluşturulması.

Üstteki şekilde belirtilen “command” fonksiyonu ise bu servise gönderilecek olan komutu temsil etmektedir. Bu komut ihtiyaçlara göre farklı biçimlerde kullanılabilmektedir. Böylelikle bu komutu servise göndermek için son bir işlem kalmış olacaktır.

```
// async/await.
try {
  const data = await client.send(command);
  // process data.
} catch (error) {
  // error handling.
} finally {
  // finally.
}
```

Şekil 5.8: Komutun çalıştırılması.

Hata yakalama kapasitesinin artırılması amacıyla bir “try catch” bloğunun içerisine yerleştirilen gönderme komutu ile iletişim kanalı olan “client” kullanılarak istek “AWS Cognito” servisine iletilmiş olur.

### 5.3 DynamoDB

Sistemde veritabanı olarak kullanılan olan Amazon Web Servisleri’nden biri olan “DynamoDB”, sistemin ihtiyaç duyduğu kaynakları saklamakla görevlendirilecektir. Bu kaynakların arasındaki ilişkilerin de belirleneceği bu servis, sistemin bütünü için en kritik öneme sahip komponentlerden biri olacaktır.

Sistemin SRS dokümanında belirtilmiş olan gereksinimleri karşılaması için birçok fonksiyon bu yazılım biriminin dahil olması ile gerçekleştirilecektir. Aşağıda gösterilen her gereksinim için veritabanı ile iletişim kurulacaktır.

| **Gereksinim** | **Öncelik Seviyesi** |
|---|---|
| Kullanıcı başarılı bir biçimde sisteme kayıt olabilmelidir. | Yüksek |
| Kullanıcı başarılı bir biçimde sisteme giriş yapabilmelidir. | Yüksek |
| Kullanıcı organizasyon oluşturabilmelidir. | Yüksek |
| Kullanıcı organizasyon içerisinde proje oluşturabilmelidir. | Yüksek |
| Kullanıcı proje içerisinde görev oluşturabilmelidir. | Yüksek |
| Yetkilendirme fonksiyonları işlevsel olmalıdır. Organizasyon'da yönetici olmayan kişiler proje oluşturamamalıdır. Proje'de yönetici olmayan kişiler projede görev oluşturamamalıdır. | Yüksek |
| Organizasyon yöneticisi organizasyona kullanıcı davet edebilmelidir. | Yüksek |
| Organizasyon yöneticisi projelere kullanıcı ekleyip çıkarabilmelidir. | Yüksek |
| Proje yöneticisi görevleri kullanıcılara atayabilmelidir. | Yüksek |
| Kullanıcılar kendilerine atanan görevlerin durumlarını değiştirebilmelidir. | Yüksek |
| Organizasyon yöneticisi tüm projeler ile ilgili bilgileri tek bir sayfada inceleyebilmelidir. | Yüksek |
| Proje yöneticisi tüm görevleri tek bir sayfadan takip edebilmelidir. | Yüksek |
| Kullanıcıların görevleri tamamlama sürelerine göre performansı hesaplanmalıdır. | Orta |
| Kullanıcıların performansına göre görevlerin ortalama tamamlanma süresi hesaplanmalıdır. | Düşük |

Tablo 5: Öncelik tablosu

"DynamoDB" servisine bağlanmak için projeye dahil edilen kütüphanenin içinden bağlantı fonksiyonları çağrılacaktır. Daha sonra bu fonksiyonlar ile oluşturulan "client" üzerinden istekler "DynamoDB" servisine gönderilecektir.

```
// a client can be shared by different commands.
const client = new DynamoDBClient({ region: "REGION" });

const params = {
  /** input parameters */
};
const command = new BatchExecuteStatementCommand(params);
```

Şekil 5.9: DynamoDB "client" oluşturma.

Daha sonra bu "client" kullanılarak sorgular yapılabilecektir.

```
try {
  const data = await client.send(command);
  // process data.
} catch (error) {
  // error handling.
} finally {
  // finally.
}
```

Şekil 5.10: "DynamoDB" istek gönderme.

Buradaki "try" bloğunun içinde dönen cevap işlenip uygulama içerisinde dilenildiği biçimde kullanılacaktır.

### 5.4 AWS Lambda Fonksiyonları

Uygulamanın backend tarafında işlenecek her veri için Lambda servisinde ve python dilinde yazılacak fonksiyonlar kullanılacaktır. GraphQL API dan Lambda servisine gelen event verisinden alınan istek çeşidine göre istek Lambda servisinde işlenip cevap tekrardan GraphQL API ile cliente ulaştırılacaktır. Lambda fonksiyonunu çalıştıracak bulut üzerindeki bilgisayar için 2GB Ram ve maksimum 15 saniye işlem gücü ayrılmıştır. Bu değerler gereksinimlerin arttığı durumda sistemin hiçbir yapısına müdahale etmeden artırılabilir. Lambda üzerinde DynamoDB de tablo oluşturmak için yazdığımız kodun bir parçası örnek olarak Şekil 5.11' te gösterilmiştir.

```
    table = dynamodb_client.create_table(
    TableName=SystemModulesTableName,
    KeySchema=[
        {
            'AttributeName': 'ID',
            'KeyType': 'HASH'
        },
        {
            'AttributeName': 'Time',
            'KeyType': 'RANGE'
        }
        ],
        AttributeDefinitions=[
            {
                'AttributeName': 'ID',
                'AttributeType': 'N'
            },
            {
                'AttributeName': 'Time',
                'AttributeType': 'S'
            },
        ],
        BillingMode='PAY_PER_REQUEST'
    )
    # Wait until the table exists.
    table.wait_until_exists()

```

Şekil 5.11

API üzerinden gelen bir isteğin işlendiği ve cevandığı bir diğer örnek kod ise Şekil 5.12' te gösterilmiştir.

```
if ev=="/getLogs":
    print(event["queryStringParameters"]["name"])

    for object_summary in my_bucket.objects.filter(Prefix = (event["queryStringParameters"]["name"])):
        count+=1
        response = s3_client.generate_presigned_url('get_object',
                                                    Params={'Bucket': "logbucket71500-staging",
                                                            'Key': object_summary.key},
                                                    ExpiresIn=3600)
        zipLink = s3_client.generate_presigned_url('get_object',
                                                    Params={'Bucket': "logbucket71500-staging",
                                                            'Key': object_summary.key.replace(".json",".zip")},
                                                    ExpiresIn=3600)
        name = object_summary.key
        name = name[(name.rfind("/")+1):]
        date = object_summary.last_modified
        dtimestamp = round(date.timestamp())
        x = {
            "fileName":name,
            "url":response,
            "zipurl":zipLink,
            "date":dtimestamp
        }
        if(".json" in x["fileName"]):
            x["fileName"] = x["fileName"].replace(".json","")
            fileList.append(x)
    return fileList

```

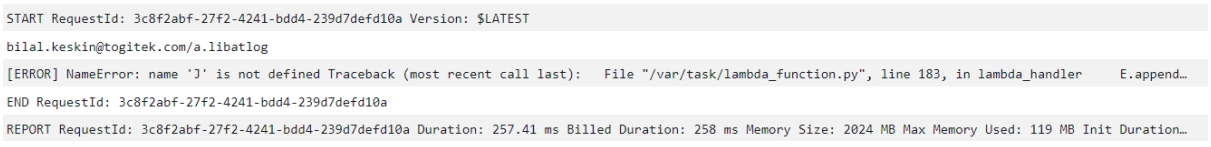
Şekil 5.12

### 5.5 AWS S3 Depolama Servis

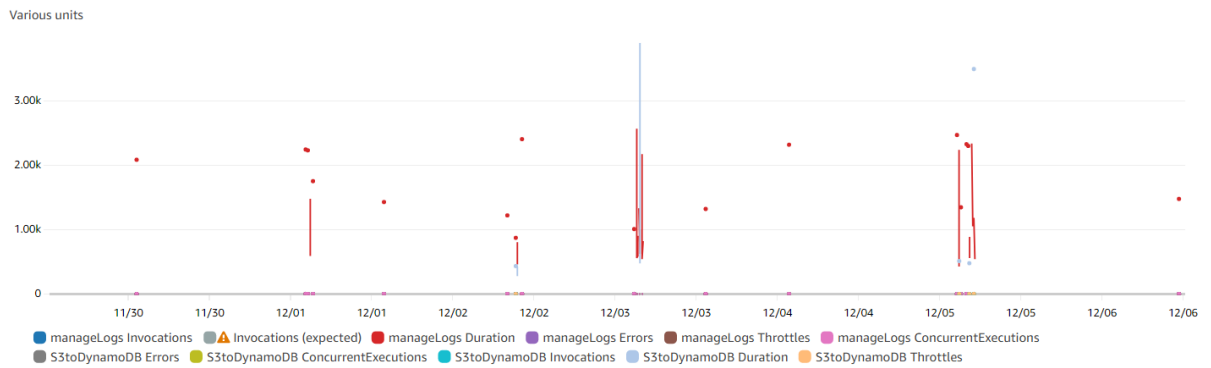
Uygulamanın depolama gereksinimleri için S3 depolama servisinde bir “Bucket” oluşturulmuştur. Bucket nesneleri S3 depolama servisinde bir depolama birimini temsil eder ve bazı fonksiyonları vardır(veri versiyonlama, erişim kontrolü). Uygulama için oluşturulan bucket içerisinde her kullanıcının bir klasörü vardır ve kullanıcının verileri bu klasörler içinde organize edilecektir. Silinen organizasyonların veriside organizasyonu oluşturan kişinin klasöründe organizasyon ismi_organizasyonId şeklinde JSON formatında saklanacaktır. Bucket üzerinde yapılacak işlemler için AWS Lambda kullanılacak ve “boto3” kütüphanesi kullanılacaktır. S3 depolama servisinde saklanacak veriler uygulama geliştirim aşamasında olduğundan, şimdilik sadece veri kaybından korunmak için kullanılacaktır. Gereksinimlerin değişmesi durumunda daha işlevsel hale getirilebilir.

## 6. GEREKSİNİMLERİN İZLENEBİLİRLİĞİ

Uygulamada yapılan bütün işlemler AWS CloudWatch servisine işlemi detayları ile bildirecek şekilde konfigüre edilmiştir. Bunun sayesinde sistemdeki bütün veri akışı, hangi kullanıcının hangi işlemleri yaptığı yada ne kadar zaman aldığı takip edilebilmektedir. Bunun sayesinde her hata oluştuğu anda geliştiriye bildirilir ve hemen müdahale imkanı sunar. Bu yapı gereksinimlerin değişeceği durumlarda öngörü imkanı sağlayacaktır. Örnek olarak bir hata çıktısı Şekil 6.1 ‘ te  ve grafik olarak Lambda fonksiyonlarının çıktıları Şekil 6.2’ te gösterilmiştir.




Şekil 6.1




Şekil 6.2

## 7. NOTLAR

Dokümanda bahsedilen “client” kavramı iki servis arasındaki iletişimi sağlayan birim olarak düşünülebilir.
Dökümanda bahsi geçen GraphQL API AWS AppSync üzerine kurulu olacaktır. GraphQL bir sorgu dilidir.
DynamoDB olarak bahsedilen servis NoSQL bir veritabanıdır.