

VTYS PROJE RAPORU

22. Grup

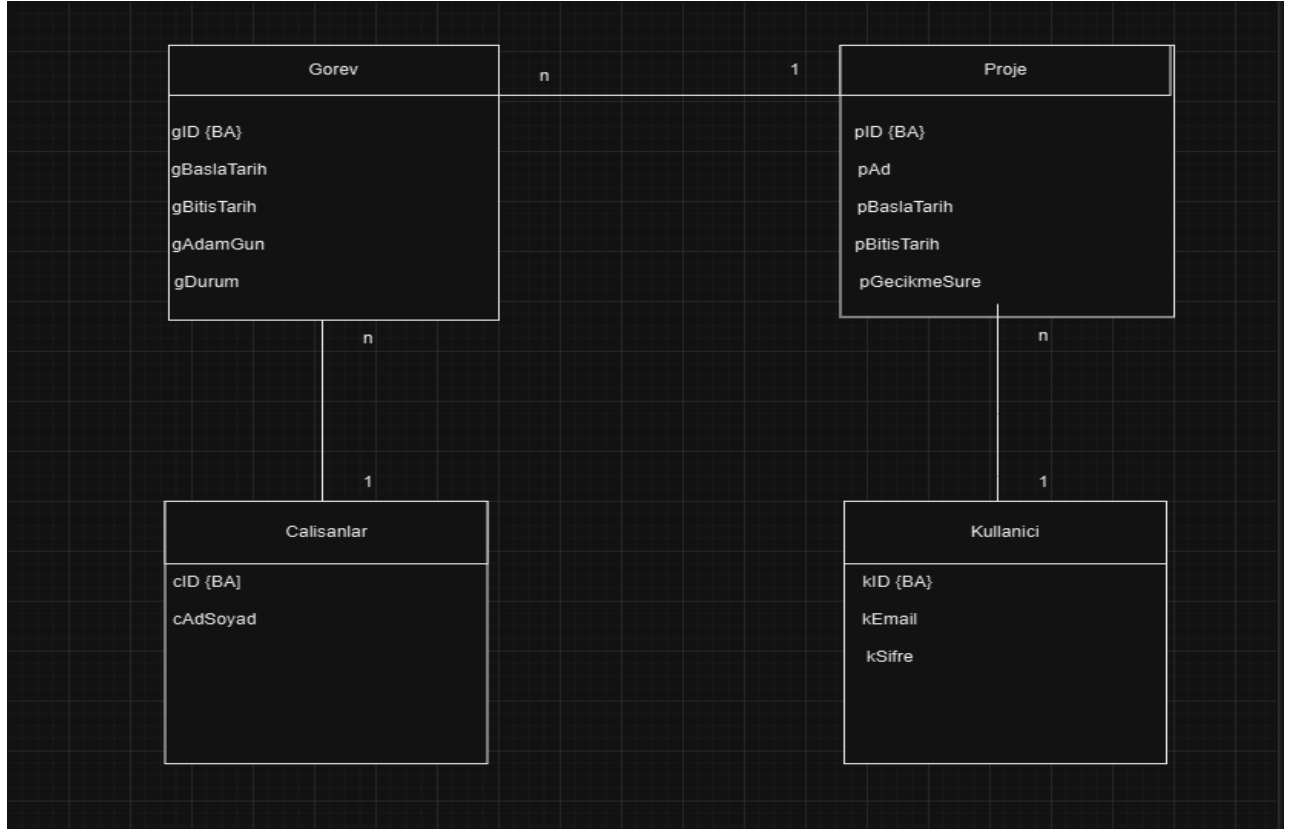
Github linkine [buradan](#) ulaşabilirsiniz.

Youtube linkine [buradan](#) ulaşabilirsiniz

1. Veri Tabanı Tasarımı

Kullanılan veri tabanı yönetim sistemi : **MYSQL**

1.1 Uml Diyagramı



1.2 ER Diyagramı

```
Proje (pID, pAd, pBaslaTarih, pBitisTarih, pGecikmeSure, Kullanici_kID)
Gorev (gID, gBaslaTarih, gBitisTarih, gAdamGun, gDurum, Proje_pID, Calisanlar_cID)
Kullanici (kID, kEmail, kSifre)
Calisanlar (cID, cAdSoyad)
```

1.3 Veri Tabanı Tabloları

1.3.1 Proje Tablosu

[illegible]

1.3.2 Gorev Tablosu

[illegible]

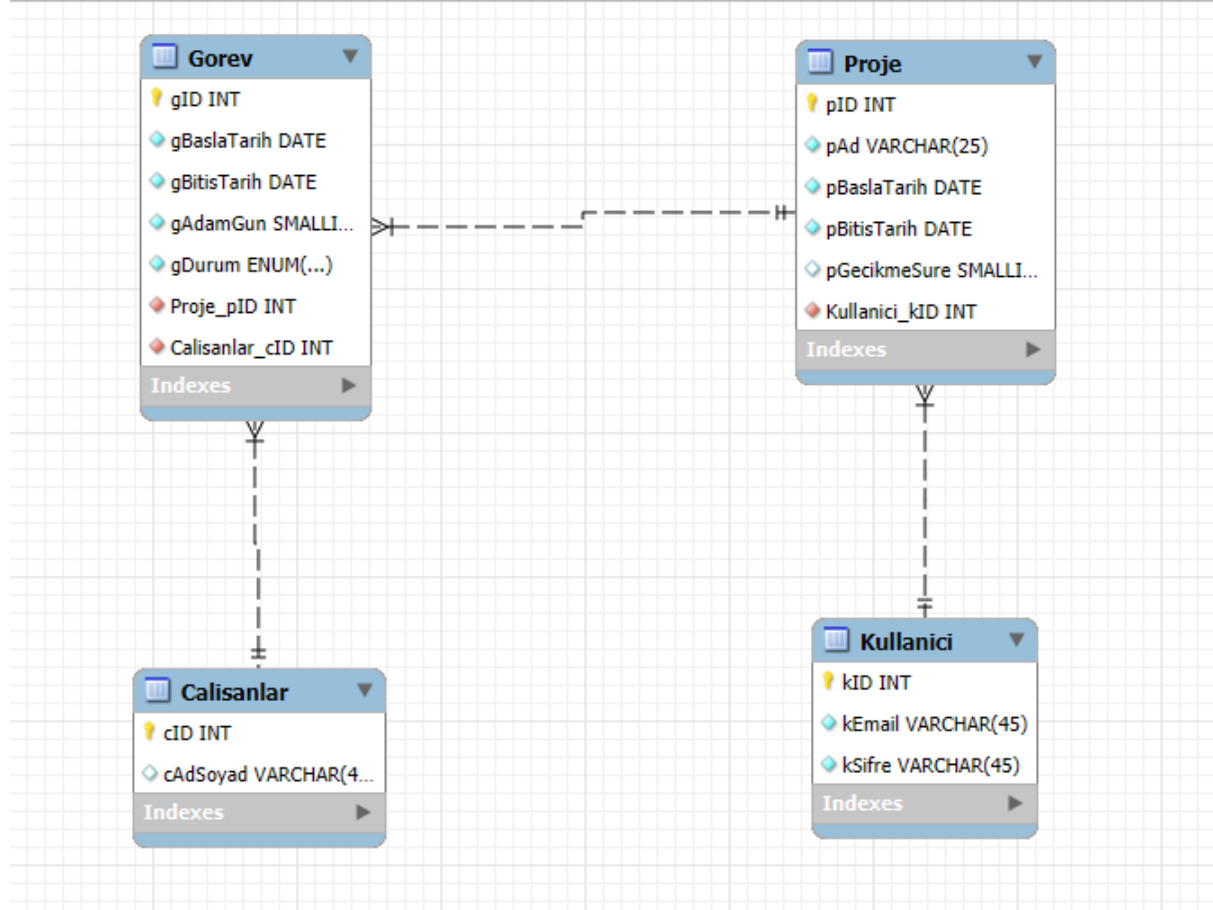
1.3.3 Kullanıcı Tablosu

[illegible]

1.3.4 Çalışanlar Tablosu

[illegible]

1.4 EER Diyagram



Tablolar Arası İlişkiler

Kullanıcı - Proje : 1:N

Bir proje sadece bir kullanıcı tarafından oluşturulabilir fakat bir kullanıcı birden fazla projede bulunabilir.

Proje - Görev : 1:N

Bir projede birden fazla görev bulunabilir fakat bir görev sadece bir projede bulunabilir.

Çalışanlar - Görev : 1:N

Bir görevde sadece bir çalışan bulunabilir fakat bir çalışan birden fazla görevde bulunabilir.

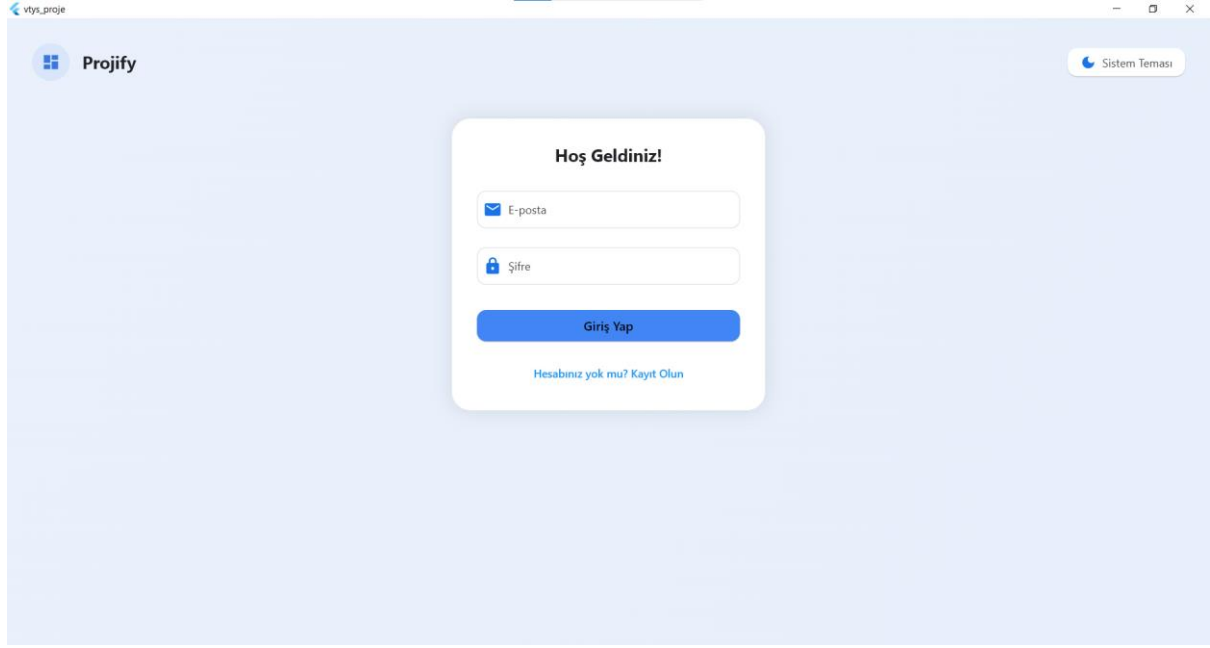
2.

ARAYÜZ

TASARIMI

Kullanılan Diller : **FLUTTER (DART)**

2.1 GİRİŞ EKRANI



Bu arayüzde “Kayıt Olun” ile kullanıcı veritabanına kaydedilir. Giriş yapmak isteyen kullanıcıların bilgileri API üzerinden sorgulanarak kontrol edilir. Yanlış bilgiler girildiğinde uyarı mesajı gösterilir.

2.2

PROJELER

EKRANI

The screenshot displays the 'Projeler' (Projects) interface. On the left, a sidebar contains navigation icons for 'Projeler', 'Çalışanlar', and 'Ayarlar'. The main area features a table with the following data:

Proje Adı	Başlangıç Tarihi	Bitiş Tarihi	Gecikme Süresi (Gün)
ARAYÜZ	02.01.2025	05.01.2025	0
REKLAM	10.12.2024	02.01.2025	0
MOBİL	20.10.2024	28.12.2024	3
VTYS	01.11.2024	Original Bitiş Tarihi: 25.12.2024 Gecikme Süresi: 3	0

At the bottom of the table area, there is a button labeled 'Yeni proje ekle'. The Windows taskbar at the bottom shows the time as 21:09 on 28.12.2024.

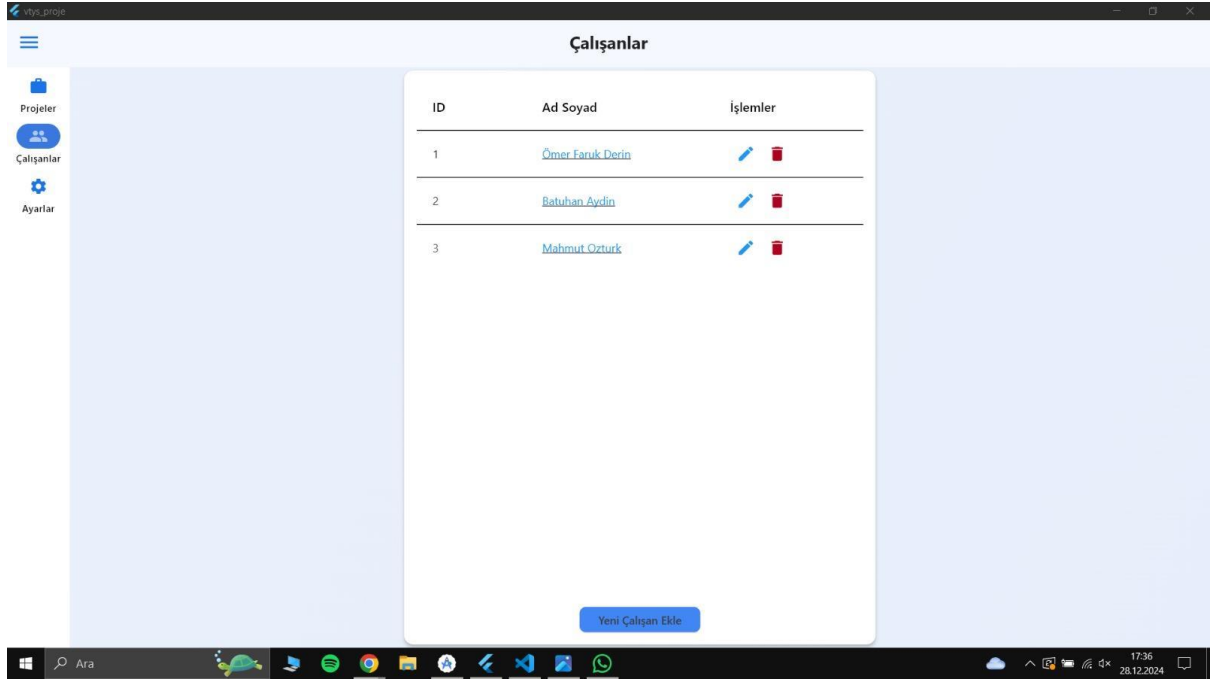
Bu arayüz giriş yapan kullanıcının projeleri görüntüleyip yönetebileceği sayfadır.

Projeler, API'den alınarak Datatable aracılığıyla dinamik olarak oluşturulur. Kullanıcılar, yeni bir proje eklemek için tasarlanmış özel bir dialog ekranı aracılığıyla proje bilgilerini girip kaydedebilir. Proje ekleme işlemi sırasında boş alan kontrolü yapılır, uygun hata mesajı ile yanıt verilir.

Projenin bitiş tarihi geçmişse gecikme süresi kırmızı renkle vurgulanır.

Proje adı üzerine tıklandığında, kullanıcı ilgili projenin görevlerini görüntüleyebileceği **Proje Görev Ekranı** sayfasına yönlendirilir. Bu sayfa, proje detaylarının yanı sıra çalışanlara ve ayarlara hızlı geçiş imkanı sunar.

2.3 ÇALIŞANLAR EKRANI

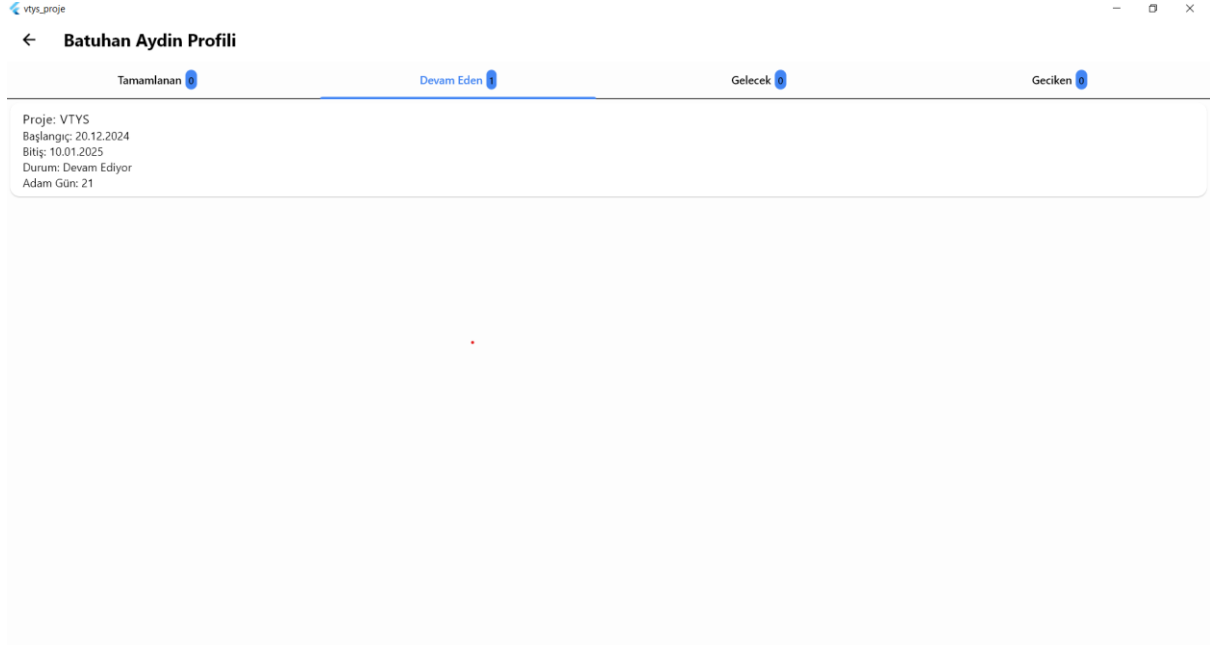


Kullanıcıların çalışan bilgilerini görüntüleyip yönetebileceği bir sayfadır. Bu sayfa, çalışanların detaylarına erişim, düzenleme, silme ve yeni çalışan ekleme gibi işlemleri sağlar.

Güncelleme, ekleme ve silme işlemleri başarılı olduğunda kullanıcıya bilgi mesajı gösterilir

Çalışan ad-soyadına tıklandığında, kullanıcı Çalışan Profil Ekranı 'na yönlendirilir.

2.4 ÇALIŞAN PROFİL EKRANI

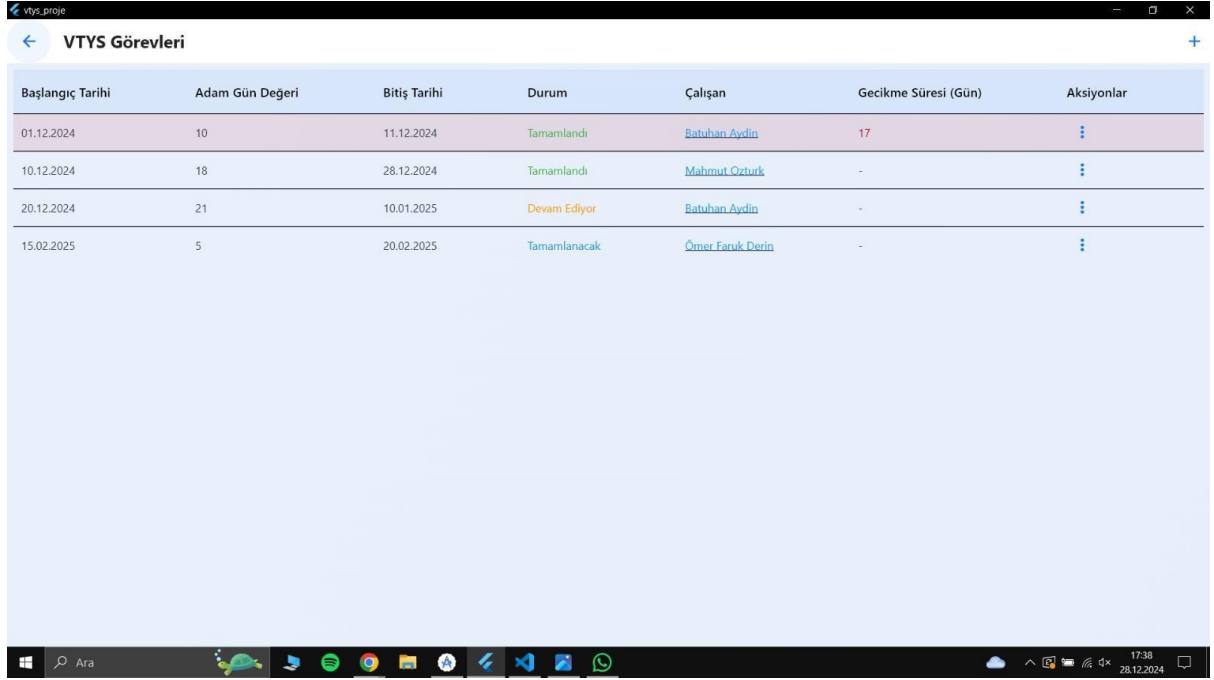


Bu sayfa, belirli bir çalışanın görevlerini görüntülemek için kullanılır. Görevler durumlarına göre farklı kategorilerde gösterilir

Her görev için proje adı, başlangıç ve bitiş tarihleri, durum bilgisi, adam gün ve gecikme süresi gibi detaylar gösterilir.

Kullanıcı, bir sekmeye tıklayarak yalnızca o kategorideki görevleri görüntüleyebilir.

2.5 PROJE DETAY EKRANI



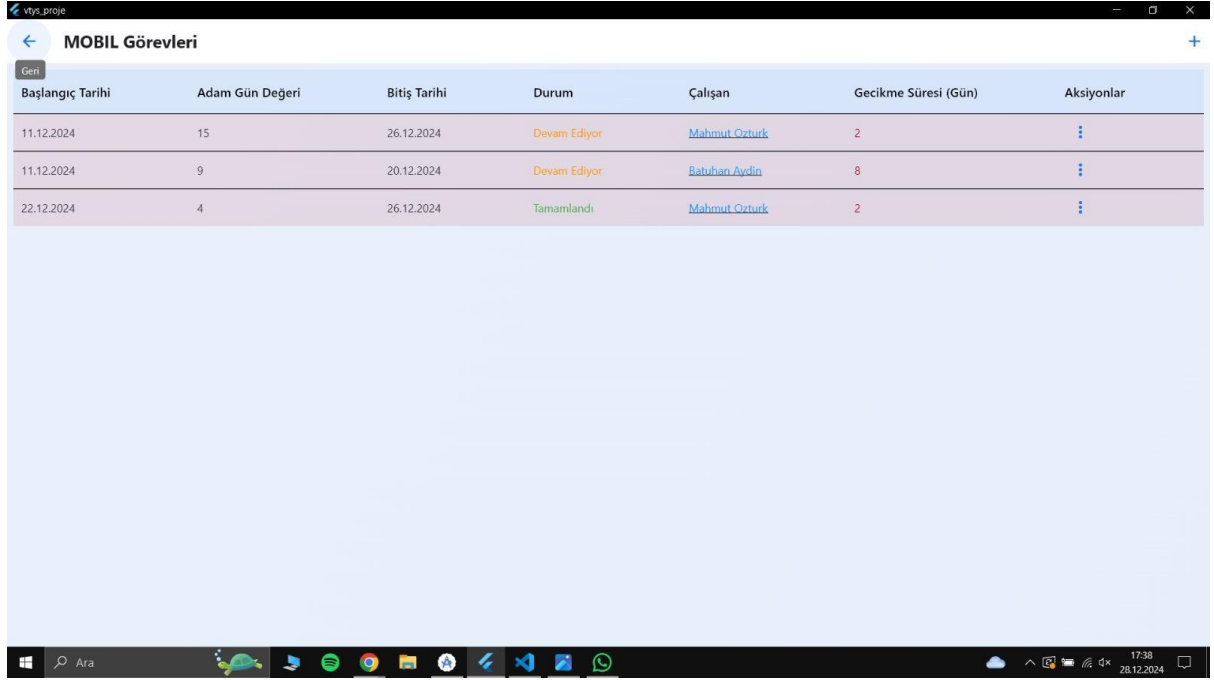
Başlangıç Tarihi	Adam Gün Değeri	Bitiş Tarihi	Durum	Çalışan	Gecikme Süresi (Gün)	Aksiyonlar
01.12.2024	10	11.12.2024	Tamamlandı	Batuhan Aydın	17	⋮
10.12.2024	18	28.12.2024	Tamamlandı	Mahmut Öztürk	-	⋮
20.12.2024	21	10.01.2025	Devam Ediyor	Batuhan Aydın	-	⋮
15.02.2025	5	20.02.2025	Tamamlanacak	Ömer Faruk Derin	-	⋮

Bu sayfa, seçili bir projeye ait görevleri listelemek, düzenlemek, yeni görev eklemek ve görevlerin durumlarını güncellemek için kullanılır.

Görev durumu "Tamamlanacak", "Devam Ediyor" veya "Tamamlandı" olarak değiştirilebilir.

Çalışanın ismine tıklayarak Çalışan Profil Ekranı'na gidilir.

2.6 PROJE DETAY EKRANI (GECİKMELİ)



Başlangıç Tarihi	Adam Gün Değeri	Bitiş Tarihi	Durum	Çalışan	Gecikme Süresi (Gün)	Aksiyonlar
11.12.2024	15	26.12.2024	Devam Ediyor	Mahmut Öztürk	2	⋮
11.12.2024	9	20.12.2024	Devam Ediyor	Batuhan Aydın	8	⋮
22.12.2024	4	26.12.2024	Tamamlandı	Mahmut Öztürk	2	⋮

En ileri tarihli görevin bitiş zamanı ile projenin bitiş zamanı karşılaştırılarak projenin gecikip gecikmediği kontrol edilir, en ileri tarihte biten gecikmeli görevin tarihinden projenin bitiş tarihi çıkarılarak projenin gecikme süresi hesaplanır ve anında güncellenir.

3. BACKEND

Kullanılan Çalışma Ortamı : **NODE.JS**

3.1 MYSQL BAĞLANTISI KURMA

```
//Veri tabanına bağlantı sağlama (Bağlantı limiti de eklendi)
const connection = mysql.createPool({
  host: 'localhost',
  user: 'root',
  password: '00_22111971',
  database: 'veritabanı',
  waitForConnections: true,
  connectionLimit: 4,
  queueLimit: 0
});
```

Bu kısım localde bulunan mysql veritabanı bağlantısını gerçekleştirmektedir. Şu anda localhost üzerinden çalışmaktadır fakat istenirse mysql in çalıştığı bir sunucu üzerinden de çalıştırılabilmektedir.

3.2 PROJE GET İŞLEMİ

```

// Projeleri gecikme durumlarıyla birlikte getirir
app.get('/projects', (req, res) => {
  const query = `
    WITH taskMetadata AS (
      SELECT
        p.project_id,
        p.start_date,
        CASE
          WHEN g.gid <= 0 THEN DATE_ADD(p.start_date, INTERVAL DATE_DIFF(CURRENT_DATE(), g.start_date) + 1 DAY)
          ELSE g.start_date
        END AS adjusted_task_date
      FROM Project p
    ),
    ProjectStats AS (
      SELECT
        p.project_id,
        COUNT(DISTINCT g.gid) AS totalTasks,
        SUM(g.gid <= 0) AS completedTasks,
        MAX(taskMetadata.adjusted_task_date) AS latest_task_date
      FROM Project p
    )
    SELECT
      p.project_id,
      p.start_date,
      CASE
        WHEN ps.latest_task_date > p.start_date THEN DATE_ADD(p.start_date, INTERVAL DATE_DIFF(ps.latest_task_date, p.start_date) + 1 DAY)
        ELSE p.start_date
      END AS totalDelay,
      ps.latest_task_date AS endGecikmeTarihi
    FROM ProjectStats ps
    ORDER BY p.project_id DESC
  `;

  connection.query(query, (error, results) => {
    if (error) {
      console.error('MySQL sorgu hatası:', error);
      return res.status(500).json({
        error: 'Veritabanı hatası',
        details: error.message
      });
    }

    const projectWithDelays = results.map(project => {
      const hasDelay = project.latest_task_date < new Date(project.start_date);
      return {
        ...project,
        originalEndDate: project.start_date,
        pAd: 'ARRAYÜZ',
        pBaslaTarih: "2025-01-01T21:00:00.000Z",
        pBitisTarih: "2025-01-04T21:00:00.000Z",
        pGecikmeSure: 0,
        "Kullanici_kID": 4,
        "totalTasks": 1,
        "completedTasks": "0",
        "latest_task_date": "2024-12-28T21:00:00.000Z",
        "totalDelay": 0,
        "enGecBitisTarih": "2024-12-28",
        "originalEndDate": "2025-01-04T21:00:00.000Z"
      };
    });

    res.json(projectWithDelays);
  });
});

```

Projelerin projeler ekranına getirilmesi işlemini yapmaktadır. Flutter gerekli metodlar yardımıyla bu endpointte istek atar ve nodejs bu istek karşısında tanımlanan sql sorgusu sayesinde projeleri veritabanından çeker ve flutter'a gönderir. Hata durumlarında nodejs konsola hata mesajını yazmaktadır.

```

[
  {
    "pID": 15,
    "pAd": "ARRAYÜZ",
    "pBaslaTarih": "2025-01-01T21:00:00.000Z",
    "pBitisTarih": "2025-01-04T21:00:00.000Z",
    "pGecikmeSure": 0,
    "Kullanici_kID": 4,
    "totalTasks": 1,
    "completedTasks": "0",
    "latest_task_date": "2024-12-28T21:00:00.000Z",
    "totalDelay": 0,
    "enGecBitisTarih": "2024-12-28",
    "originalEndDate": "2025-01-04T21:00:00.000Z"
  },
  {

```

Flutter'a gelen json tipindeki bir proje verisi (Projenin gecikme durum kontrolü için fazladan alanlar tanımlanmıştır.)

3.3 PROJE POST İŞLEMİ

```

// Yeni proje oluşturma işlemi
app.post('/projects', (req, res) => {
  console.log('Gelen veri:', req.body);

  const { pAd, pBaslicaTarih, pBitisTarih, kullanicid_KID } = req.body;

  if (!pAd || !pBaslicaTarih || !pBitisTarih || !kullanicid_KID) {
    return res.status(400).json({
      error: 'Gerekli alanlar eksik',
      details: {
        pAd: pAd ? 'Proje adı gerekli' : null,
        pBaslicaTarih: pBaslicaTarih ? 'Başlangıç tarihi gerekli' : null,
        pBitisTarih: pBitisTarih ? 'Bitiş tarihi gerekli' : null,
        kullanicid_KID: kullanicid_KID ? 'Kullanıcı ID gerekli' : null
      }
    });
  }

  if (pAd.length > 25) {
    return res.status(400).json({
      error: 'Gözetilmez',
      details: 'Proje adı 25 karakterden uzun olamaz'
    });
  }

  const now = new Date();
  const startDate = new Date(pBaslicaTarih);
  const endDate = new Date(pBitisTarih);

  if (!isNaN(startDate.getTime()) || !isNaN(endDate.getTime())) {
    return res.status(400).json({
      error: 'Gözetilmez tarih formatı',
      details: 'Tarihler yyyy-mm-dd formatında olmalı'
    });
  }

  if (endDate < startDate) {
    return res.status(400).json({
      error: 'Gözetilmez tarih aralığı',
      details: 'Bitiş tarihi başlangıç tarihinden önce olamaz'
    });
  }

  const checkUserQuery = 'SELECT KID FROM kullanicid WHERE KID = ?';
  if (error) {
    console.error('MySQL kullanicid kontrol hatası', error);
    return res.status(500).json({
      error: 'Veritabanı hatası',
      details: error.message
    });
  }

  if (results.length === 0) {
    return res.status(400).json({
      error: 'Gözetilmez kullanicid',
      details: 'Belirtilen kullanicid ID bulunamadı'
    });
  }

  const insertQuery = 'INSERT INTO proje (pAd, pBaslicaTarih, pBitisTarih, kullanicid_KID) VALUES (?, ?, ?, ?)';
  const values = [pAd, pBaslicaTarih, pBitisTarih, kullanicid_KID];

  connection.query(insertQuery, values, (error, results) => {
    if (error) {
      console.error('MySQL oluşturma hatası', error);
      return res.status(500).json({
        error: 'Veritabanı hatası',
        details: error.message
      });
    }

    res.status(201).json({
      message: 'Proje başarıyla eklendi',
      projectId: results.insertId
    });
  });
});

```

Bu endpoint ise veritabanına proje eklemek için kullanılmaktadır. Projeler ekranında proje ekleme işlemi gerçekleştirildikten sonra flutter üzerinden yollanan bilgiler burada tanımlanan “insert into” sql sorguları yardımıyla veritabanına kaydedilmektedir. Gönderilen verilerin tip kontrolleri de burada yapılmaktadır.

3.4 PROJE PUT İŞLEMİ

```
// Proje bitiş tarihini güncelleme
app.put('/projects/:id/update-end-date', (req, res) => {
  const projectId = req.params.id;
  const { newEndDate } = req.body;

  const query = `
    UPDATE Proje
    SET gBitisTarih = ?
    WHERE pID = ?
  `;

  connection.query(query, [newEndDate, projectId], (error, result) => {
    if (error) {
      console.error('MySQL güncelleme hatası:', error);
      return res.status(500).json({
        error: 'Veritabanı hatası',
        details: error.message
      });
    }

    if (result.affectedRows === 0) {
      return res.status(404).json({
        error: 'Proje bulunamadı'
      });
    }

    res.json({
      message: 'Proje bitiş tarihi başarıyla güncellendi',
      projectId: projectId,
      newEndDate: newEndDate
    });
  });
});
```

Projenin bitiş tarihinin güncelleme işleminden sorumlu endpointtir. Eğer 3.2 kısımdaki sorgudan gecikme gelirse bu endpoint çalışmaktadır.

3.5 GOREV GET İŞLEMİ

```
// Görevleri getirme işlemi (Görevler veritabanından da güncellenir)
app.get('/tasks/:projectId', (req, res) => {
  const projectId = req.params.projectId;
  const query = `
    SELECT g.*, c.cAdSoyad, c.cID
    FROM GOREV g
    LEFT JOIN Calisanlar c ON g.Calisanlar_CID = c.cID
    WHERE g.Proje_pID = ?
    ORDER BY g.gBitisTarih ASC
  `;

  connection.query(query, [projectId], (error, results) => {
    if (error) {
      console.error('MySQL sorgu hatası:', error);
      return res.status(500).json({ error: 'Veritabanı hatası' });
    }

    const now = new Date();
    const today = new Date(now.getFullYear(), now.getMonth(), now.getDate());
    const updateResults = results.map(task => {
      const startDate = new Date(task.gBitisTarih);
      const taskStartDate = new Date(startDate.getFullYear(), startDate.getMonth(), startDate.getDate());

      let newStatus = task.gDurum;

      if (task.gDurum !== 'Tamamlandı') {
        if (taskStartDate > today) {
          newStatus = 'Tamamlanacak';
        } else {
          newStatus = 'Devam Ediyor';
        }
      }

      if (newStatus !== task.gDurum) {
        const updateQuery = `UPDATE GOREV SET gDurum = ? WHERE gID = ?`;
        connection.query(updateQuery, [newStatus, task.gID], (updateError) => {
          if (updateError) {
            console.error('Durum güncelleme hatası:', updateError);
          }
        });
      }
    });

    return {
      ...task,
      gDurum: newStatus
    };
  });
});

res.json(updateResults);
});
```

Projede tanımlanan görevleri veritabanından alıp flutter'a iletmekle sorumludur. Ayrıca görevler tablosunda bulunan proje durum değiştirme işlemi yapıldığı zaman bu değişimi sql sorgusu ile veritabanına iletmektedir.

3.6 GOREV PUT İŞLEMİ

```
// Görev güncelleme işlemi
app.put('/tasks/:id', (req, res) => {
  const taskId = req.params.id;
  const { gDurum, gecikmeGun } = req.body;

  const query = 'UPDATE Gorev SET gDurum = ?, gecikmeGun = ? WHERE gID = ?';

  connection.query(query, [gDurum, gecikmeGun, taskId], (err, result) => {
    if (err) {
      res.status(500).json({ error: err.message });
      return;
    }
    res.json({ message: 'Görev başarıyla güncellendi.' });
  });
});
```

Bu endpoint ise görevin süresinin günler ilerledikçe kontrolü sonrasında durum değişikliği yaşamış olduğu durumlarda veritabanındaki durumunu güncellemek için kullanılmaktadır.

3.7 GOREV POST İŞLEMİ

```
// Yeni görev ekleme işlemi
app.post('/tasks', (req, res) => {
  console.log('Received task data:', req.body);

  const {
    gBaslaTarih,
    gBitisTarih,
    gAdanGun,
    gDurum,
    Calisanlar_cID,
    Proje_pID
  } = req.body;

  if (!gBaslaTarih || !gBitisTarih || !gAdanGun || !Calisanlar_cID || !Proje_pID) {
    return res.status(400).json({
      error: 'Gerekli alanlar eksik',
      receivedData: req.body
    });
  }

  const query = `
    INSERT INTO Gorev
    (gBaslaTarih, gBitisTarih, gAdanGun, gDurum, Calisanlar_cID, Proje_pID)
    VALUES (?, ?, ?, ?, ?, ?)
  `;

  connection.query(
    query,
    [gBaslaTarih, gBitisTarih, gAdanGun, gDurum || 'Tamamlanacak', Calisanlar_cID, Proje_pID],
    (err, result) => {
      if (err) {
        console.error('Veritabanı hatası:', err);
        res.status(500).json({
          error: err.message,
          sqlMessage: err.sqlMessage
        });
        return;
      }
      res.status(201).json({
        message: 'Görev başarıyla eklendi',
        id: result.insertId
      });
    }
  );
});
```

Proje detay ekranında bulunan görev ekleme butonundan alınan bilgileri sql sorguları yardımıyla veritabanına ekleme işlemini yapmaktadır.

3.8 GOREV DELETE İŞLEMİ

```
// Görev silme işlemi
app.delete('/tasks/:id', (req, res) => {
  const taskId = req.params.id;

  const query = 'DELETE FROM Gorev WHERE gID = ?';

  connection.query(query, [taskId], (err, result) => {
    if (err) {
      console.error('Görev silinirken bir hata oluştu:', err);
      return res.status(500).json({
        error: 'Veritabanı hatası',
        details: err.message
      });
    }

    if (result.affectedRows === 0) {
      return res.status(404).json({
        error: 'Görev bulunamadı.'
      });
    }

    res.json({
      message: 'Görev başarıyla silindi',
      taskId: taskId
    });
  });
});
```

Proje detay ekranında bulunan görev silme işlemini yapmaktadır. Kullanıcı görev sil butonuna bastığı zaman bu görevin veritabanından silinmesi için gereken sql sorgusunu veritabanına iletir.

3.9 GOREV GET İŞLEMİ

```
// Görevleri ve gecikme sürelerini getirme
app.get('/tasks/:projectId', (req, res) => {
  const projectId = req.params.projectId;
  const query = `
    SELECT
      g.*,
      c.cAdSoyad,
      p.pAd as projeAdi,
      CASE
        WHEN g.gDurum != 'Tamamlandı' AND CURDATE() > g.gBitisTarih
        THEN DATEDIFF(CURDATE(), g.gBitisTarih)
        ELSE 0
      END as gecikmeGun
    FROM Gorev g
    JOIN Calisanlar c ON g.calisanlar_cID = c.cID
    JOIN Proje p ON g.Proje_pID = p.pID
    WHERE g.Proje_pID = ?
  `;

  connection.query(query, [projectId], (err, results) => {
    if (err) {
      res.status(500).json({ error: err.message });
      return;
    }
    res.json(results);
  });
});
```

Bu endpoint ise görevlerin gecikme süreleriyle birlikte getirilmesinden sorumludur. Gün ilerlediği zaman gecikme süreleri, veri tabanından bu method yardımıyla alınmaktadır.

3.10 GÖREVLERİN GECİKME SÜRESİ KONTROLÜ

```
const scheduleTaskStatusUpdates = () => {
  const now = new Date();
  const night = new Date(
    now.getFullYear(),
    now.getMonth(),
    now.getDate() + 1,
    0, 0, 0
  );
  const timeToMidnight = night - now;

  setTimeout(() => {
    updateTaskStatuses();
    setInterval(updateTaskStatuses, 24 * 60 * 60 * 1000);
  }, timeToMidnight);
};
```

Görevlerin gecikme sürelerinin hesaplanmasında yardımcı bir metoddur. Her gün değişiminde görevlerin durumlarını ve gecikme sürelerini kontrol ederek görev durum kontrolüne yardımcı olur.

3.11 ÇALIŞANLAR GET İŞLEMİ

```
// Çalışanları VE görevlerinin ne kadarını tamamladığını getirme
app.get('/employees', (req, res) => {
  const query = `
    SELECT c.*,
           COUNT(DISTINCT g.Proje_pID) as totalProjects,
           SUM(CASE WHEN g.gDurum = 'Tamamlandı' THEN 1 ELSE 0 END) as completedTasks
    FROM Calisanlar c
    LEFT JOIN Gorev g ON c.cID = g.Calisanlar_cID
    GROUP BY c.cID
  `;

  connection.query(query, (err, results) => {
    if (err) {
      res.status(500).json({ error: err.message });
      return;
    }
    res.json(results);
  });
});
```

Bu endpoint ise çalışanların veritabanından getirilmesi işleminden sorumludur. Çalışanlar ekranında çalışanların id ve isimlerinin gösterimi için kullanılmaktadır. Ayrıca profilde tamamlanan ve tamamlanamamış projelerin gösterilmesinde kullanılmaktadır.

```
[
  {
    "cID": 1,
    "cAdSoyad": "Ömer Faruk Derin",
    "totalProjects": 1,
    "completedTasks": "0"
  },
  {
    "cID": 2,
    "cAdSoyad": "Batuhan Aydın",
    "totalProjects": 5,
    "completedTasks": "1"
  },
  {
    "cID": 3,
    "cAdSoyad": "Mahmut Ozturk",
    "totalProjects": 3,
    "completedTasks": "2"
  }
]
```

Flutter'a gelen json tipindeki veri

3.12 CALİSANLAR POST İŞLEMİ

```
// Yeni çalışan ekleme
app.post('/employees', (req, res) => {
  const { cAdSoyad } = req.body;

  if (!cAdSoyad) {
    return res.status(400).json({
      error: 'Eksik bilgi',
      details: 'Çalışan adı gereklidir'
    });
  }

  if (cAdSoyad.length > 50) {
    return res.status(400).json({
      error: 'Geçersiz veri',
      details: 'Çalışan adı 50 karakterden uzun olamaz'
    });
  }

  const insertQuery = 'INSERT INTO Calisanlar (cAdSoyad) VALUES (?)';
  connection.query(insertQuery, [cAdSoyad], (error, results) => {
    if (error) {
      console.error('MySQL ekleme hatası:', error);
      return res.status(500).json({
        error: 'Veritabanı hatası',
        details: error.message
      });
    }

    res.status(201).json({
      message: 'Çalışan başarıyla eklendi',
      employeeId: results.insertId
    });
  });
});
```

Bu endpoint sisteme yeni çalışan ekleme görevini üstlenmektedir. Çalışanlar sayfasında yeni çalışan ekle işlemi gerçekleştirildiği zaman tanımlanan sorgular yardımıyla yeni çalışanı veritabanına kaydeder.

3.13 CALİSANLAR DELETE İŞLEMİ


```

app.delete('/employees/:id', (req, res) => {
  const employeeId = req.params.id;
  const checkTasksQuery = 'SELECT COUNT(*) as taskCount FROM Gorev WHERE Calisanlar_cID = ?';

  connection.query(checkTasksQuery, [employeeId], (error, results) => {
    if (error) {
      return res.status(500).json({
        error: 'Veritabanı hatası',
        details: error.message
      });
    }

    if (results[0].taskCount > 0) {
      return res.status(400).json({
        error: 'Çalışan silinemez',
        details: 'Çalışanın aktif görevleri bulunmaktadır'
      });
    }

    // Çalışan herhangi bir görevde yer almıyorsa silinebilir ya da görevi bitmeli
    const deleteQuery = 'DELETE FROM Calisanlar WHERE cID = ?';

    connection.query(deleteQuery, [employeeId], (error, result) => {
      if (error) {
        return res.status(500).json({
          error: 'Veritabanı hatası',
          details: error.message
        });
      }

      if (result.affectedRows === 0) {
        return res.status(404).json({
          error: 'Çalışan bulunamadı'
        });
      }

      res.json({ message: 'Çalışan başarıyla silindi' });
    });
  });
});

```

Bu endpoint çalışanlar sayfasında bir çalışan silme işlemi çağrıldığı zaman çalışanın herhangi bir projede görev alıp almadığını kontrol eder eğer almıyorsa bu çalışanın veritabanından kaldırır ancak çalışan bir projede görev alıyorsa ve görevi tamamlanmış değil ise çalışanın sistemden kaldırılmasını engeller aksi takdirde ise çalışanın sistemden siler.

3.14 ÇALIŞANLAR PUT İŞLEMİ

```

// Çalışan ismi güncelleme
app.put('/employees/:id', (req, res) => {
  const employeeId = req.params.id;
  const { cAdSoyad } = req.body;

  if (!cAdSoyad) {
    return res.status(400).json({
      error: 'Eksik bilgi',
      details: 'Çalışan adı gereklidir'
    });
  }

  const updateQuery = 'UPDATE Calisanlar SET cAdSoyad = ? WHERE cID = ?';

  connection.query(updateQuery, [cAdSoyad, employeeId], (error, result) => {
    if (error) {
      return res.status(500).json({
        error: 'Veritabanı hatası',
        details: error.message
      });
    }

    if (result.affectedRows === 0) {
      return res.status(404).json({
        error: 'Çalışan bulunamadı'
      });
    }

    res.json({
      message: 'Çalışan başarıyla güncellendi',
      employeeId: employeeId
    });
  });
});

```

Çalışanlar sayfasında bulunan güncelleme işlemi sonrasında güncellenen çalışan ismi veritabanında güncelleme işlemini yapmaktadır

3.15 ÇALIŞANLAR GET İŞLEMİ

```
// Çalışan görevlerini getirme
app.get('/tasks/employee/:employeeId', (req, res) => {
  const employeeId = req.params.employeeId;
  const query = `
    SELECT g.*, p.pAd as projeAdi
    FROM Gorev g
    JOIN Proje p ON g.Proje_pID = p.pID
    WHERE g.Calisanlar_cID = ?
    ORDER BY g.gBaslaTarih
  `;

  connection.query(query, [employeeId], (err, results) => {
    if (err) {
      res.status(500).json({ error: err.message });
      return;
    }
    res.json(results);
  });
});
```

Bu kısım bir çalışanın bir projede yer aldığı bir görevin getirmek için kullanılmaktadır. (Sonrasında profil kısmında da kullanılacaktır.)

```
[
  {
    "gID": 49,
    "gBaslaTarih": "2025-02-14T21:00:00.000Z",
    "gBitisTarih": "2025-02-19T21:00:00.000Z",
    "gAdamGun": 5,
    "gDurum": "Tamamlanacak",
    "Calisanlar_cID": 1,
    "Proje_pID": 12,
    "projeAdi": "VTYS"
  }
]
```

Flutter'a ulaşan json formatındaki veri.

3.16 KULLANICI POST(KAYIT) İŞLEMİ

```
// Kayıt olma işlemi
app.post('/register', (req, res) => {
  const { kEmail, kSifre } = req.body;

  if (!kEmail || !kSifre) {
    return res.status(400).json({ error: 'Email ve şifre gerekli' });
  }

  const checkQuery = 'SELECT kID FROM kullanıcı WHERE kEmail = ?';
  connection.query(checkQuery, [kEmail], (error, results) => {
    if (error) {
      console.error('MySQL sorgu hatası:', error);
      return res.status(500).json({ error: 'Veritabanı hatası' });
    }

    if (results.length > 0) {
      return res.status(400).json({ error: 'Bu email zaten kayıtlı' });
    }

    const insertQuery = 'INSERT INTO kullanıcı (kEmail, kSifre) VALUES (?, ?)';
    connection.query(insertQuery, [kEmail, kSifre], (error, results) => {
      if (error) {
        console.error('MySQL ekleme hatası:', error);
        return res.status(500).json({ error: 'Veritabanı hatası' });
      }

      res.status(201).json({
        message: 'Kullanıcı başarıyla kaydedildi',
        userId: results.insertId,
      });
    });
  });
});
```

Kullanıcıların sisteme kaydını üstlenmektedir. Kullanıcıların girdiği e posta ve şifre bilgilerini sisteme yeni kullanıcı olarak kaydetmektedir.

3.17 KULLANICI POST(GİRİŞ) İŞLEMİ

```
// Giriş işlemi
app.post('/login', (req, res) => {
  const { kEmail, kSifre } = req.body;

  if (!kEmail || !kSifre) {
    return res.status(400).json({ error: 'Email ve şifre gerekli' });
  }

  const query = 'SELECT kID, kEmail FROM kullanıcı WHERE kEmail = ? AND kSifre = ?';

  connection.query(query, [kEmail, kSifre], (error, results) => {
    if (error) {
      console.error('MySQL sorgu hatası:', error);
      return res.status(500).json({ error: 'Veritabanı hatası' });
    }

    if (results.length === 0) {
      return res.status(401).json({ error: 'Geçersiz email veya şifre' });
    }

    res.json({
      kID: results[0].kID,
      kEmail: results[0].kEmail
    });
  });
});
```

Kullanıcıların girdiği e posta ve şifre bilgilerini veritabanına göndererek e posta ve şifre kontrolü yapmaktadır. Gerekli durumlarda kullanıcı hata mesajı almaktadır.

3.18 SUNUCU VE BAĞLANTI İŞLEMLERİ

```

// Nodejs sunucusu başlatma ve zaman kontrolcülerini başlatma
app.listen(port, () => {
  console.log('Node.js sunucusu http://localhost:${port} adresinde çalışıyor');
  updateTaskStatuses();
  scheduleTaskStatusUpdates();
});

// Global hata yakalayıcısı
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).json({
    error: 'Sunucu hatası',
    details: process.env.NODE_ENV === 'development' ? err.message : undefined
  });
});

// Mysql bağlantısı kapatma
process.on('SIGINT', () => {
  connection.end((err) => {
    if (err) {
      console.error('MySQL bağlantısı kapatılırken hata:');
    }
    process.exit();
  });
});

```

- app.listen metodu nodejs sunucusunu başlatır ve görevlerin durumlarını kontrol eden metodları çalıştırır.
- app.use metodu ise genel bir hata yakalayıcıdır nodejs sunucusunda bir hata olduğu zaman bu hatayı detaylıca konsolda göstermek amacıyla kullanılmaktadır.
- process.on metodu ise nodejs sunucusu kapatıldığı zaman nodejs mysql arasındaki bağlantının güvenli bir şekilde kapatılmasını sağlar.