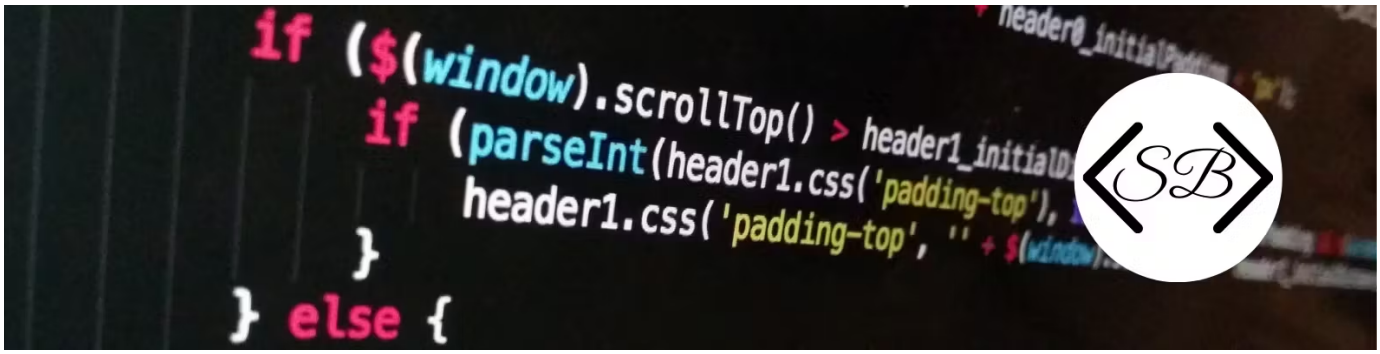

MERN-Blogify-Backend

Kullanılan Kütüphaneler:

mongoose : Veri tabanı
express : Route işlemleri için
nodemon : Proje otomatik izleme
bcrypt : Verileri şifreleme
jwt : Kullanıcı bilgileri koruma için
express-async-handler : Hataları yakalamak için
dotenv : Önemli bilgileri gizlemek için
nodemailer : Mail atabilmek için
multer cloudinary : Resimleri internete yüklemek için

1.Modelleri oluşturma:

User Model

```
const mongoose = require("mongoose"); //Mongo DB veritabanı ile baglantı kurmak için
kullanıldı
const crypto = require("crypto"); //Verilen veriyi şifrelemek için kullanılan özel bir
kütüphane

//Şema verileri:
const userSchema = new mongoose.Schema( //Mongo DB de bir şema olacagını ve içerik olarak
neler olacagını belirtigimiz kısım.
  {
    username: { //Alan adını temsil eder
      type: String, //Türü temsil eder
      required: true, //Doldurulması zorunlu oldugunu belirtir.
    },
    email: {
      type: String,
      required: true,
    },
    role: {
      type: String,
      required: true,
      enum: ["user", "admin"], //Alabilecegi 2 deger oldugunu belirtiriz
      default: "user", //Başlangıç olarak user olarak başlayacagını berlirtiriz.
```

```
    },
    password: {
      type: String,
      required: true,
    },
    lastLogin: {
      type: Date, //Veri türü Tarih oldugunu belirtilir.
      default: Date.now(), //Başlangıç olarak o anki tarihi alır
    },
    isVerified: {
      type: Boolean, //Bool veri türünü temsil eder.
      default: false, //Başlangıçta false olarak başlar
    },
    accountLevel: {
      type: String,
      enum: ["bronze", "silver", "gold"], //Alabilecegi degerler
      default: "bronze", //Başlangıç degeri
    },
    profilePicture: {
      type: String,
      default: "", //Başlangıç degeri
    },
    coverImage: {
      type: String,
      default: "",
    },
    bio: {
      type: String,
    },
    location: {
      type: String,
    },
    notificationPreferences: {
      email: { type: String, default: true }, //kullanıcıya bildirim göndermemizi
istedigi alanlar
      // diger bildirim türleri (sms)
    },
    genderUser: {
      type: String,
      enum: ["male", "female", "prefer not to say", "nun-binary"], //Alabilecegi
degerler.
    },
    profileViewrs: [{ type: mongoose.Schema.Types.ObjectId, ref: "User" }], //Bu alan
dizi olacak ve içine alacagı veri ref: "User" tablodan olacak. Referans tablonun ID
degerini alacak ve çagırırken işlemleri ID üzerinden gerçekleştirecek.
    followers: [{ type: mongoose.Schema.Types.ObjectId, ref: "User" }],
    following: [{ type: mongoose.Schema.Types.ObjectId, ref: "User" }],
    blockedUsers: [{ type: mongoose.Schema.Types.ObjectId, ref: "User" }],
    posts: [{ type: mongoose.Schema.Types.ObjectId, ref: "Post" }],
    likedPosts: [{ type: mongoose.Schema.Types.ObjectId, ref: "Post" }],
//Hepsi aynı şekilde Tablolardan ID degerlerini alır.
    passwordResetToken: { //Şifreyi degiştirmek için bir token oluşturucak o tokende
string olarak tutlacak
      type: String,
    },
    passwordResetExpires: { //Şifre nin değiştirildigi tarihi tutugumuz alan
```

```
      type: Date,
    },
    accountVerificationToken: { //Kullanıcıyı onaylamak için token gönderdigimiz kısım
      type: String,
    },
    accountVerificationExpires: { //Kullanıcının onaylandıgı tarih
      type: Date,
    },
  },
  {
    timestamps: true, //Hesap ne zaman oluşturuldugunu ve güncellendigi zamanları tutan
alanları temsil eder.
    toJSON: {
      virtuals: true, //Verilerin güncel hallerini sanal ortam olarak tutar.
    },
    toObject: {
      virtuals: true,//Verilerin güncel hallerini sanal ortam olarak tutar.
    },
  }
);

//Token oluşturma metotu
userSchema.methods.generatePasswordResetToken = function () { //User şeması içinde bir
metot olacak.
  //şifrelenmiş bir token oluşturma
  const resetToken = crypto.randomBytes(20).toString("hex");
  //Kullanıcın şifresini sıfırlaması için token üretiyoruz belirlenen yere kaydediyoruz
  this.passwordResetToken = crypto
    .createHash("sha256")
    .update(resetToken)
    .digest("hex");
  //Token geçerlilik süresini belirliyoruz
  this.passwordResetExpires = Date.now() + 10 * 60 * 1000;
  return resetToken; //Metot çagırıldıgında random 20 byte boyutundaki tokeni döndürür.
};

//! Maile onay için token gönderme
userSchema.methods.generateAccVerificationToken = function () {
  //Token oluşturma
  const resetToken = crypto.randomBytes(20).toString("hex");
  //Oluşturulan tokeni hashleme
  this.accountVerificationToken = crypto
    .createHash("sha256")
    .update(resetToken) // resetToken degerini alır ve şifrelenen degeri degiştirir.
    .digest("hex");

  //Token geçerlilik süresini ayarlama
  this.accountVerificationExpires = Date.now() + 10 * 60 * 1000; //! 10 dakika
  return resetToken; //Metot çagırıldıgında random 20 byte boyutundaki tokeni döndürür.
};

//User modellemesi bitti
const User = mongoose.model("User", userSchema);

module.exports = User;
```

Diğer alanlarda benzer şekilde işlemler içermektedir

2.Server.js Ana Dosya

```
const dotenv = require("dotenv"); //Gizlenencek verilerin tutlmasını sağlayan kütüphane.
dotenv.config(); //Bu sayfada kullanabilmek için ayarları geçerli kıldık.
const http = require("http"); //kullanılacak protokol
const express = require("express"); //Baglantıların çalışması için kullanıgımız
kütüphanenin ataması gerçekleştirildi.
const usersRouter = require("./routes/users/usersRouter"); //User router alanı tanımlandı
const {
  notFound,
  globalErrorHandler,
} = require("./middlewares/globalErrorHandler"); //Sorun çıktıgında hataların gelecegi
alanı toplamıştık o alanı çagırdık.
const categoryRouter = require("./routes/category/categoryRouter");//Kategory router
alanı tanımlandı
const postRouter = require("./routes/post/postRouter");//Post router alanı tanımlandı
const commentsRouter = require("./routes/comment/commentsRouter");//Yorum router alanı
tanımlandı
const sendEmail = require("./utils/sendEmail"); //kullanıcıya gönderecegimiz mail'in
ayarlamalarını yaptıgımız alan
require("./config/database")(); //Sayfa açıldıgında direkt çalışacagı için bir değişkene
atama geregi duymadık. Veritabanına baglandıgımız kısım
//!Server oluşturma kodları
const app = express(); //express func'nunu app'e üzerinden erişebilmeyi etkinleştirdik.

//Middlewarler

app.use(express.json()); //Gelen verileri JSON formatına dönüştürme işlemi yapılıyor
// Yönlendirme işlemleri..  http://localhost:1998/api/v1/_alan buraya geldiginde
çalışacak funclar
app.use("/users", usersRouter); //http://localhost:1998/api/v1/users
app.use("/categories", categoryRouter); //http://localhost:1998/api/v1/categories
app.use("/posts", postRouter); // http://localhost:1998/api/v1/posts
app.use("/comments", commentsRouter); // http://localhost:1998/api/v1/comments

app.use(notFound);// 404 sayfası
app.use(globalErrorHandler); //! Hata alınca gitmesi gereken alan

const server = http.createServer(app); //protokol sonrasına yeni bir server oluşturur ve
çagırılan alanı dinler.

//? Server Başlatma kodları
const PORT = process.env.PORT || 1998; //Portu belirigimiz kısım
server.listen(PORT, console.log(`${PORT} portu çalışıyor..`)); //Server'i çalıştırdıgımız
alan.
```

3.ENV Dosyası

```
MOGO_URL=  //veritabanı linki
JWT_KEY=   //JWT alanını kullanabilmek için siteden alınan key gerekir
GMAIL_PASS=  //Mail gidecek kişinin uygulamalar aracılıgıyla mail gönderebilmesini
sağlayan key
GMAIL_USER= // Mailin gidecegi mail adres
CLOUDINARY_NAME= //Resimleri yükledigimiz alanın adı siteden alınır
CLOUDINARY_KEY= //Aynı alanın Key bilgisi
CLOUDINARY_SECRET= //Aynı alanın gizli bilgisi
```

4.Veri Tabanı Bağlama

```
const mongoose = require("mongoose"); //Veritbanı kütüphanesi

const connectDB = async () => { //Func asenkron olacak
  try { //Hataları kontrol edebilmek için try-catch kullanıyoruz
    await mongoose.connect( //Veri tabanına baglanıyoruz
      process.env.MOGO_URL  //Veri tabanı linki gerek ama biz bilgileri gizlemiştik.
    );
    console.log("Veri tabanı baglantısı başarılı"); //Bilgilendirme.
  } catch (error) {
    console.log("Veri tabanı baglantısı başarısız", error.message);//Bilgilendirme.
  }
};

module.exports = connectDB; //Başka alanlardan çagırabilmek için export ediyoruz.
```

5.Kontroller'ler

User Controller:

```
const bcrypt = require("bcryptjs"); //Verileri şifrelemek için kütüphane
const crypto = require("crypto"); //otomatik random veri üretmek için kütüphane
const User = require("../../model/User/User");//Kullanıcı modelini entegre ettik
const generateToken = require("../../utils/generateToken"); //Token oluşturmayı entegere
ettik
const {asyncHandler,expressAsyncHandler} = require("express-async-handler"); //Genel hata
yöneticiyi entehre ettik
const sendEmail = require("../../utils/sendEmail"); //Mail atmak için func entegre ettik
const sendAccVerificationEmail = require("../../utils/sendAccVerificationEmail"); //Hesap
onaylama func entegre ettik.

//@desc Register a new user
//@route Post /api/v1/users/register
//@access public

exports.register = asyncHandler(async (req, res) => {

  const { username, email, password } = req.body; //Kullanıcıdan alınacak bilgiler

  const user = await User.findOne({ username }); //Kullanıcı varmı diye kontrollu
yapılıyor
```

```
  if (user) { //Kullanıcı varsa verecegi yanıt yoksa devam eder.
    throw new Error("Kullanıcı zaten kayıtlı");
  }
  //Kullanıcı yoksa yeni bir kullanıcı oluşturmak için bilgileri newUser'e atar.
  const newUser = new User({
    username,
    email,
    password,
    profilePicture: req?.file.path,
  });

  const salt = await bcrypt.genSalt(10);//Şifreyi güvenliye dönüştürme
  newUser.password = await bcrypt.hash(password, salt); //şifreyi belirlenen yönteme
dönüştürür.

  await newUser.save();//Kayıt işlemini gerçekleştirir.
  res
    .status(200)
    .json({ status: "OK", message: "Kullanıcı kayıt işlemi tamam", newUser });
});

//@desc User Login
//@route Post /api/v1/users/login
//@access public

exports.login = asyncHandler(async (req, res) => {
  const { username, password } = req.body; //Bilgileri kullanıcıdan alır.

  const user = await User.findOne({ username });// Kullanıcı kontrolu yapılır.
  if (!user) {
    throw new Error("Kullanıcı bulunamadı"); //Kullanıcı yoksa bu hatayı verir.
  }
  const isMatched = await bcrypt.compare(password, user?.password); //şifrelerin eşleşip
eşleşmedigini kontrol edilir.
  if (!isMatched) { //Eşleşmiyorsa aşagıdaki hatayı verir.
    throw new Error("Giriş bilgileri doğru değil");
  }

  user.lastLogin = new Date();// Son giriş zamanını güncelleme için veritabanında bir
güncelleme işlemi gerçekleştirilir.
  res.json({
    status: "success",
    email: user?.email,
    _id: user?._id,
    username: user?.username,
    role: user?.role,
    token: generateToken(user), //Kullanıcı giriş yaparken bu kısımda ona özgü bir token
üretilir ve kullanıcıya geri gönderilir bu bilgiler ise taraycıda tutulur.
    profilePicture: user?.profilePicture,
    isVerified: user?.isVerified,
  });
});

//@desc Get profile
//@route Post /api/v1/users/profile/:id
//@access public
```

```
exports.getProfile = asyncHandler(async (req, res, next) => { //Tek bir kullanıcının
bilgilerini getirme kodları
  const id = req.userAuth._id; //Giriş yapan kullanıcın ID'sini alır
  const user = await User.findById(id) //Kullanıcılar arasından o kulanıcyı arar.
    .populate({ //Baglantılı alanlar için getirilecek alanları belirlenir.
      path: "posts",
      model: "Post",
    })
    .populate({
      path: "following",
      model: "User",
    })
    .populate({
      path: "blockedUsers",
      model: "User",
    })
    .populate({
      path: "profileViewrs",
      model: "User",
    });
  res.json({
    status: "succes",
    message: "Profile girildi",
    user,
  });
});

//@desc Block user
//@route Post /api/v1/users/block/:userIdToBlock
//@access privte

exports.blockUser = asyncHandler(async (req, res) => {
  //Giriş yapan kişinin  istedigi kullanıcıyı bloklamasını saglıyoruz bu kodta
  const userIdToBlock = req.params.userIdToBlock; //Bloklanacak kullanıcının hesabında
oldugunu varsayarak o ID'yi alarız.
  const userToBlock = await User.findById(userIdToBlock); //O ID'yi tabloda ararız.
  if (!userToBlock) { //Yoksa aşagıdaki hatayı veririz.
    throw new Error("Kullanıcı yok");
  }
  // Kendi kendini bloklamasın diye karşılaştırma yapıyoruz.
  const userBlocking = req.userAuth._id; //Giriş yapan kişinin ID'si alıyoruz
  // Kendi olmadıgını kontrol ediyoruz eger kendi ise aşagıdaki hatayı veriyoruz.
  if (userIdToBlock.toString() === userBlocking.toString()) {
    throw new Error("Kendi kendini bloklayamazsın");
  }
  //Bloklamak istedigi kişinin zaten bloklu olup olmadıgını kontrolunu sğlıyoruz
  const currentUser = await User.findById(userBlocking);
  // Zaten bloklumu kontrolu
  if (currentUser?.blockedUsers?.includes(userIdToBlock)) {
    throw new Error("Kullanıcı zaten bloklu");
  }
  //Bloklu degil ise bloklananlar içine gönderiyoruz.
  currentUser.blockedUsers.push(userIdToBlock);
  await currentUser.save();
  res.json({
```

```
    message: "Kullanıcı başarı ile bloklandı",
    status: "Başarılı",
  });
});

//@desc unBlock user
//@route Post /api/v1/users/unblock/:userIdToBlock
//@access privte

exports.unblockuser = asyncHandler(async (req, res) => {
  //Kişiyi blokdan çıkarmak.
  const userIdToUnBlock = req.params.userIdToUnBlock; //Üzerinde durdugu kişinin ID'ni
alır.
  const userToUnBlock = await User.findById(userIdToUnBlock); //Kişiyi tablodan arar.
  if (!userToUnBlock) { //Yoksa hata verir
    throw new Error("Böyle bir kullanıcı yok");
  }
  //Kendimizi buluyoruz
  const userUnBlocking = req.userAuth._id;
  const currentUser = await User.findById(userUnBlocking);

  //Kullanıcının verilerinin içindeki bloklular dizisindekilerden verdigimiz ID'yi arar.
Yoksa hata verir.
  if (!currentUser.blockedUsers.includes(userIdToUnBlock)) {
    throw new Error("Bu kullanıcı bloklu değil");
  }
  //Kullanıcıyı bloklular listesinden arar uymuyorsa bir şey yapmaz ama oyuyorsa listeye
dahil etmez.
  currentUser.blockedUsers = currentUser.blockedUsers.filter(
    (id) => id.toString() !== userIdToUnBlock.toString()
  );
  //Listenin son halini kaydeder.
  await currentUser.save();
  res.json({
    status: "Başarılı",
    message: "Kullanıcı blokdan çıkarıldı",
  });
});

//@desc   Kimler profilime baktı
//@route  GET /api/v1/users/profile-viewer/:userProfileId
//@access Private

exports.profileViewers = asyncHandler(async (req, res) => {
  // Kullanıcıların sayfalarına kimlerin baktıgını bulma
  const userProfileId = req.params.userProfileId; //Aranan kişi ID'sini alma

  const userProfile = await User.findById(userProfileId); //Tabloda varmı diye kontrol
etme
  if (!userProfile) { //Kullanıcı yoksa bu hatayı ver.
    throw new Error("Böyle bir kullanıcı yok");
  }
  //Kendi bilgilerini bulmak
  const currentUserId = req.userAuth._id;
  // Bakılan kişinin ziyesretçilerinin içinden arama yapar ve giriş yapan kişinin ID'si
olup olmadıgını kontrol eder.
```

```
  if (userProfile?.profileViewrs?.includes(currentUserId)) { //Varsa aşagdadki hatayı
verir.
    throw new Error("Zaten bakmışsın");
  }
  //Eger yoksa giriş yapan kişinin ID'sini, ziyaret edilen kişinin ziyaretçilerine ekleme
yapar.
  userProfile.profileViewrs.push(currentUserId);
  await userProfile.save(); //Tabloyu kayıt eder.
  res.json({
    message: "Profile bakılanlar listesine eklendi",
    status: "Başarılı",
  });
});

//@desc   Follwing user
//@route  PUT /api/v1/users/following/:userIdToFollow
//@access Private

exports.followingUser = asyncHandler(async (req, res) => {
  //Giriş yapan kişiyi buluyoruz.
  const currentUserId = req.userAuth._id;
  //! Takip edecegimiz kişiyi arıyoruz
  const userToFollowId = req.params.userToFollowId;
  //Takip edecegimiz kişinin kendimiz olmadıgını belli ediyoruz
  if (currentUserId.toString() === userToFollowId.toString()) {
    throw new Error("Kendini takip edemezsiz");
  }
  //Giriş yapan kişinin takip ettiklerine ekler
  await User.findByIdAndUpdate(
    currentUserId,
    {
      $addToSet: { following: userToFollowId }, //addToSet: Mongo DB kodudur vevar olan
degere, yeni bir deger eklemek için kullınılır.
    },
    {
      new: true,
    }
  );
  //Aranan kişinin takipçilerine eklemek için kullanılır
  await User.findByIdAndUpdate(
    userToFollowId,
    {
      $addToSet: { followers: currentUserId }, //addToSet: Mongo DB kodudur ve yeni bir
deger eklemek için kullınılır.
    },
    {
      new: true,
    }
  );
  //işlem sonucunu gönder
  res.json({
    status: "Başarılı",
    message: "Takip etme kodu başarılı",
  });
});
```

```
//@desc   UnFollwing user
//@route  PUT /api/v1/users/unfollowing/:userIdToUnFollow
//@access Private

exports.unFollowingUser = asyncHandler(async (req, res) => {
  //Kendini bulma
  const currentUserId = req.userAuth._id;
  //! Takipden çıkacagın kişiyi bulma
  const userToUnFollowId = req.params.userToUnFollowId;

  //Kendi kendini takipden çıkamazsın
  if (currentUserId.toString() === userToUnFollowId.toString()) {
    throw new Error("Kendi kendini takipden çıkamazsın");
  }
  //Takip edilenlerden çıkarma kodları
  await User.findByIdAndUpdate(
    currentUserId,
    {
      $pull: { following: userToUnFollowId }, //Pull kodu Mongo DB ye özeldir ve bir
degeri çıkarmak için kullanılır
    },
    {
      new: true,
    }
  );
  //Takip edenlerden çıkarma
  await User.findByIdAndUpdate(
    userToUnFollowId,
    {
      $pull: { followers: currentUserId }, //Pull kodu Mongo DB ye özeldir ve bir degeri
çıkrmak için kullanılır
    },
    {
      new: true,
    }
  );
  //send the response
  res.json({
    status: "başarılı",
    message: "Takipden çıkma işlemi gerçekleştirildi",
  });
});

// @route   POST /api/v1/users/forgot-password
// @desc   Forgot password
// @access  Public

exports.forgotpassword = expressAsyncHandler(async (req, res) => {
  const { email } = req.body; //Mail'i kullanıcıdan alırız.
  //DB de kullanıcının mailini arıyoruz
  const userFound = await User.findOne({ email });
  if (!userFound) {
    throw new Error("Email iniz sistemde kayıtlı degil");
  }
  //Yeni token oluşturma
  const resetToken = await userFound.generatePasswordResetToken();
```

```
  //Degişimi kaydediyoruz
  await userFound.save();

  //Şifreyi yenileye bilmesi için şifre yenileme url'sini token ile mail atıyoruz
  sendEmail(email, resetToken);
  res
    .status(200)
    .json({ message: "Şifre yenileme mesajı gönderildi", resetToken });
});

// @route   POST /api/v1/users/reset-password/:resetToken
// @desc   Reset password
// @access  Public

exports.resetPassword = expressAsyncHandler(async (req, res) => {
  //email de gelen parametreyi alma
  const { resetToken } = req.params;
  const { password } = req.body;
  //Veritabındaki token ile dışardan gelen token karşılaştırılır.
  const cryptoToken = crypto
    .createHash("sha256")
    .update(resetToken)
    .digest("hex");
  //Bizim verecegimiz özelliklere uyan kişileri buluyoruz
  const userFound = await User.findOne({
    passwordResetToken: cryptoToken, //Veritabanındaki token ile aynı mı diye kontrol
eder.
    passwordResetExpires: { $gt: Date.now() }, // $gt : token belirtilen süreden geçip
geçmedigini kontrol etmek için karşılaştırma yapma kodudur ve MONGODB'ye özelir.
Tablodaki passwordResetExpires alanındaki deger şimdiki zamandan büyük olup olmadıgını
kontrol eder.
  });
  if (!userFound) { //Dönen deger false ise aşagıdaki hatayı verir.
    throw new Error("Şifre yenilemek için verilen süre bitmiş olabilir");
  }
  //Şİfreyi değiştiriyoruz
  const salt = await bcrypt.genSalt(10); //Girilen şifreyi hashleme yöntemi belirlenir.
  userFound.password = await bcrypt.hash(password, salt); //Şifreyi hashlenir.
  userFound.passwordResetExpires = undefined; //Bu alanı boşaltırız
  userFound.passwordResetToken = undefined;   //Bu alanı boşaltırız
  //İşlem sonucunu döndürüyoruz
  await userFound.save(); //Veriyi DB'ye kayıt ederiz.
  res.status(200).json({ message: "Şifre yenileme işlemi başarılı" });
});

// @route   POST /api/v1/users/account-verification-email/
// @desc    Send Account verification email
// @access  Private

exports.accountVerificationEmail = expressAsyncHandler(async (req, res) => {
  //Kullanıcıyı aktifleştirmek.
  const user = await User.findById(req?.userAuth?._id); //Giriş yapan kullanıcının
ID'sini alma
  if (!user) { //Kullanıcıyı yoksa verecegi hata
    throw new Error("Böyle bir kullanıcı yok");
  }
```

```
  //Token i oluşturma alanı
  const token = await user.generateAccVerificationToken();
  //tokeni kullanıcının belirlenen alanına kayıt eder
  await user.save();
  //Tokeni ile birlikte URL'yi mail atma
  sendAccVerificationEmail(user?.email, token);
  res.status(200).json({
    message: `Hesabı onaylamak için bilirim bu maile gönderildi: ${user?.email}`,
  });
});

// @route   POST /api/v1/users/verify-account/:verifyToken
// @desc    Verify token
// @access  Private

exports.verifyAccount = expressAsyncHandler(async (req, res) => {
  //Kullanıcıyı etkinleştirme
  const { verifyToken } = req.params; //Token i URL'nin sonuna ekleyip gönderdigimiz için
ordan alırız.
  //Veri tabındaki token ile kullanıcıdan aldıgımız tokeni karşılaştırıyoruz
  const cryptoToken = crypto
    .createHash("sha256")
    .update(verifyToken)
    .digest("hex");
  //Token süre kontrolünü sağlama
  const userFound = await User.findOne({
    accountVerificationToken: cryptoToken,
    accountVerificationExpires: { $gt: Date.now() },
  });
  if (!userFound) { //Sürede sorun varsa aşagıdaki hatayı verir.
    throw new Error(
      "Hesabı onaylamak için gönderilen token tarihi geçmiş olabilir"
    );
  }
  //Kullanıcı bilgileri güncellendi
  userFound.isVerified = true;
  userFound.accountVerificationExpires = undefined;
  userFound.accountVerificationToken = undefined;
  //Sonuç
  await userFound.save();
  res.status(200).json({ message: "Hesap onaylandı" });
});

//@desc  Upload profile image
//@route  PUT /api/v1/users/upload-profile-image
//@access Private

exports.uploadeProfilePicture = asyncHandler(async (req, res) => {
  // Giriş yapan kullanıcıyı arama
  const userFound = await User.findById(req?.userAuth?._id);
  if (!userFound) { //Kullanıcyı DB'de kontrolünü sağladık
    throw new Error("Kullanıcı Yok");
  }
  const user = await User.findByIdAndUpdate( //Kullanıcın ID'sine göre güncelleme
yapıyoruz.
    req?.userAuth?._id,
```

```
      {
        $set: { profilePicture: req?.file?.path }, //resim i güncelleme
      },
      {
        new: true, //yeni degeri döndürür.
      }
    );

    // Sonuçu yazdırma
    res.json({
      status: "Başarılı",
      message: "Kullanıcı resim güncellendi",
      user,
    });
  });

  //@desc   Upload cover image
  //@route  PUT /api/v1/users/upload-cover-image
  //@access Private

  exports.uploadeCoverImage = asyncHandler(async (req, res) => {
    // Kullanıcıyı arama
    const userFound = await User.findById(req?.userAuth?._id);
    if (!userFound) {
      throw new Error("Böyle bir kullanıcı yok");
    }
    const user = await User.findByIdAndUpdate(
      req?.userAuth?._id,
      {
        $set: { coverImage: req?.file?.path }, // set:  güncelleme operatörüdür ve bir
  belgedeki alanların değerlerini güncellemek için kullanılır
      },
      {
        new: true,
      }
    );

    //Sonucu gönderme
    res.json({
      status: "Başarılı",
      message: "Kullancı küçük resim güncellendi",
      user,
    });
  });

  //@desc   Update username/email
  //@route  PUT /api/v1/users/update-profile
  //@access Private

  exports.updateUserProfile = asyncHandler(async (req, res) => {
    //!Kullanıcı kontorlu
    const userId = req.userAuth?._id;
    const userFound = await User.findById(userId); //Kullanıcıyı DB'de bulma
    if (!userFound) { //Kullanıcı yoksa verecegi hata
      throw new Error("Kullanıcı bulunamadı");
    }
```

```
  console.log(userFound);
  //! Kullanıcı email ve adı degiştirme işlemi
  const { username, email } = req.body; //Bilgileri kullanıcdan alıyoruz.
  const post = await User.findByIdAndUpdate( //Kişiyi arayıp güncelliyoruz.
    userId,
    { //Güncellenecek alanalar
      email: email ? email : userFound?.email,
      username: username ? username : userFound?.username,
    },
    {
      new: true,
      runValidators: true, //DB'deki kuralları kontrol et yine
    }
  );//Sonucu yazdır.
  res.status(201).json({
    status: "Başarılı",
    message: "Kullanıcı başarı ile güncellendi",
    post,
  });
});
```

Post Controller:

```
const asyncHandler = require("express-async-handler");
const Category = require("../../model/Category/Category");
const Post = require("../../model/Post/Post");
const User = require("../../model/User/User");
const expressAsyncHandler = require("express-async-handler");
//@desc  Create a post
//@route POST /api/v1/posts
//@access Private

exports.createPost = asyncHandler(async (req, res) => {
  const { title, content, categoryId } = req.body; //Kullanıcıdan post verilerini alma

  const postFound = await Post.findOne({ title });//Post kontollu
  if (postFound) {
    throw new Error("Post zaten var");
  }
  //Create post
  const post = await Post.create({ //Alınan degerleri tek bir  degere atıp DB'ye yükleme
    title,
    content,
    category: categoryId,
    author: req?.userAuth?._id,
    image: req?.file?.path,
  });
  //Kullanıcının postlarını güncelleme
  await User.findByIdAndUpdate(
    req?.userAuth?._id,
    {
      $push: { posts: post._id }, //$push: var olan degerlerin üstüne ekler.
    },
    {
      new: true, //Eklenen degerleri getirir.
```

```
    }
  );

  //Categorinin postlarını güncelleme
  await Category.findByIdAndUpdate(
    req?.userAuth?._id,
    {
      $push: { posts: post._id },//$push: var olan degerlerin üstüne ekler.
    },
    {
      new: true,
    }
  );

  //Sonuçları bildirme
  res.json({
    status: "Başarılı",
    message: "Post yüklendi",
    post,
  });
});

//@desc  Get all posts
//@route GET /api/v1/posts
//@access Private

exports.getPosts = asyncHandler(async (req, res) => {
  //Hangi kullanıcının baktıgını belirliyoruz
  const loggedInUserId = req.userAuth?._id;
  const currentTime = new Date();
  //Bu kullanıcı bloclananlar arasında varmı kontrolu saglanıyor
  const usersBlockingLoggedInuser = await User.find({
    blockedUsers: loggedInUserId,
  });
  // Engellenleri tek tek getirme Id lerine göre
  const blockingUsersIds = usersBlockingLoggedInuser?.map((user) => user?._id);
  //! sorgu ile category ve özel yazı arama
  const category = req.query.category; //Kullanıcının seçtigi kategoriyi tutar.
  const searchTerm = req.query.searchTerm; //Kullanıcının aradıgı kelimeyi tutar.
  //Blok içinde olmayanları getirir
  let query = {
    author: { $nin: blockingUsersIds }, // $nin demek not in anlamına gelir ve belli alan
haricindekileri getirir.
    $or: [
      //or veya anlamında kullanılır.
      {
        shedduledPublished: { $lte: currentTime }, //lte   "less than or equal" (küçük
veya eşit) anlamına gelir ve yüklendigi tarih şuanki zamandan büyük olamaması için
yapılan bir işlemdir.
        shedduledPublished: null,
      },
    ],
  };

  if (category) {
```

```
    query.category = category; //Uygun postlardan categorisi kullanıcının seçtigini
getirir.
  }
  if (searchTerm) {
    query.title = { $regex: searchTerm, $options: "i" }; //Burada ara regex ile arama
yapılıyor ve options ile ise büyük küçük harf farkı kaldırılıyor.
  }
  const page = parseInt(req.query.page, 10) || 1; //Sayfalama için kullanıcıdan sayfa
numarasını alır, eger deger numara gelmezse deger olarak 1 atar.
  const limit = parseInt(req.query.limit, 10) || 5; //Kaç tane post görebilecek
  const startIndex = (page - 1) * limit; //Sayfada kaçıncı post ile başlancak
  const endIndex = page * limit; //Son olarak kaçıncı posta biter onu belirlendi.
  const total = await Post.countDocuments(query); //kaçtane post var

  const posts = await Post.find(query) //Aranan kiretere göre post getirme
    .populate({
      path: "author",
      model: "User",
      select: "email role username",
    })
    .populate("category") //referans tablo
    .skip(startIndex) //sayfadan kaçıncı postdan itibaren gösterilsin ayarı
    .limit(limit) //her sayfada kaç tane post götersilsin
    .sort({ createdAt: -1 }); //En yeni en başa gelmesi için
  // Sayflama sonuçları
  const pagination = {};
  if (endIndex < total) {
    //toplam post belirlenen sayfalama sayısından büyükse
    pagination.next = { //pagination'a yeni bir anahtar ekliyoruz.
      page: page + 1, //sayfa artır
      limit,
    };
  }

  if (startIndex > 0) {
    pagination.prev = { //pagination'a yeni bir anahtar ekliyoruz.
      page: page - 1, //sayfayı eksiltiyoruz.
      limit,
    };
  }

  res.status(201).json({ //sonuçları yazdırıyoruz.
    status: "Başarılı",
    message: "Postlar getirildi",
    posts,
  });
});

//@desc  Get single post
//@route GET /api/v1/posts/:id
//@access PUBLIC
exports.getPost = asyncHandler(async (req, res) => { //Tekli post getirme
  const post = await Post.findById(req.params.id); //Post'u arama
  res.status(201).json({
    status: "Başarılı",
    message: "Post getirildi",
```

```
    post,
  });
});

//@desc  Delete Post
//@route DELETE /api/v1/posts/:id
//@access Private

exports.deletePosts = asyncHandler(async (req, res) => { //Post'u silme
  await Post.findByIdAndDelete(req.params.id); //post'u bulup silme
  res.status(201).json({
    status: "Başarılı",
    message: "Post silindi",
  });
});

//@desc  update Posts
//@route PUT /api/v1/posts/:id
//@access Private

exports.updatePosts = asyncHandler(async (req, res) => { //Post'u güncelleme
  const post = await Post.findByIdAndUpdate(req.params.id, req.body, { //Post'u bulup
güncelleme
    new: true,//yeni degeri getirme
    runValidators: true,
  });
  res.status(201).json({
    status: "Başarılı",
    message: "Categori güncellendi",
    post,
  });
});

//@desc   liking a Post
//@route  PUT /api/v1/posts/likes/:id
//@access Private

exports.likePost = expressAsyncHandler(async (req, res) => {
  //Post sahibinin id sini alma
  const { id } = req.params;
  //Kullanıcının giriş yaptıgı id yi bulma
  const userId = req.userAuth._id;
  //Postu bulma
  const post = await Post.findById(id);
  if (!post) {
    throw new Error("Post yok");
  }
  //Begenenlere ekleme

  await Post.findByIdAndUpdate(
    id,
    {
      $addToSet: { likes: userId },
    },
    { new: true }
  );
```

```
  // Dislike deki degeri değiştirme kodu
  post.dislikes = post.dislikes.filter(
    (dislike) => dislike.toString() !== userId.toString()
  );
  //Sonuçu kaydet
  await post.save();
  res.status(200).json({ message: "Post begenilenlere eklendi.", post });
});

//@desc   disliking a Post
//@route  PUT /api/v1/posts/dislikes/:id
//@access Private

exports.disLikePost = expressAsyncHandler(async (req, res) => {
  //Post sahibini bulma
  const { id } = req.params;
  //Kullanıcı ıd sini alma
  const userId = req.userAuth._id;
  //Post bulma
  const post = await Post.findById(id);
  if (!post) {
    throw new Error("Post not found");
  }
  //Dislike kısımını güncelleme

  await Post.findByIdAndUpdate(
    id,
    {
      $addToSet: { dislikes: userId },
    },
    { new: true }
  );
  // Likelenen alandan çıkarma
  post.likes = post.likes.filter(
    (like) => like.toString() !== userId.toString()
  );
  //Sonuçu kaydet
  await post.save();
  res.status(200).json({ message: "Post disliked successfully.", post });
});

//@desc   clapong a Post
//@route  PUT /api/v1/posts/claps/:id
//@access Private

exports.claps = expressAsyncHandler(async (req, res) => {
  //postu bulma
  const { id } = req.params;
  //Postu db de bulma
  const post = await Post.findById(id);
  if (!post) {
    throw new Error("Post bulunmuyor");
  }
  //Alkışları değiştirme
  const updatedPost = await Post.findByIdAndUpdate(
    id,
```

```
      {
        $inc: { claps: 1 },
      },
      {
        new: true,
      }
    );
    res.status(200).json({ message: "Post clapped successfully.", updatedPost });
  });

  //@desc   Shedule a post
  //@route  PUT /api/v1/posts/schedule/:postId
  //@access Private

  exports.schedule = expressAsyncHandler(async (req, res) => {
    //Bilgileri alma
    const { scheduledPublish } = req.body;
    const { postId } = req.params;
    //ID veya paylaşım yokmu kontrol et
    if (!postId || !scheduledPublish) {
      throw new Error("ID veya paylaşım yok");
    }
    //Post u arama
    const post = await Post.findById(postId);
    if (!post) {
      throw new Error("Post yok ....");
    }
    //kullanıcının post'unyazarı olup olmadığını kontrol edin
    if (post.author.toString() !== req.userAuth._id.toString()) {
      throw new Error("Kendi gönderinizi paylaşabilirsiniz");
    }
    // Ne zaman paylaşıldıgını güncelleme
    const scheduleDate = new Date(scheduledPublish);
    const currentDate = new Date();
    if (scheduleDate < currentDate) {
      throw new Error("Tarih eskisinden daha eski olamaz");
    }
    //Post güncellendi
    post.shedduledPublished = scheduledPublish;
    await post.save();
    res.json({
      status: "Başarılı",
      message: "Post programı güncellendi",
      post,
    });
  });

  //@desc   post  view counta
  //@route  PUT /api/v1/posts/:id/post-views-count
  //@access Private

  exports.postViewCount = expressAsyncHandler(async (req, res) => {
    //Postu ID yi bulma
    const { id } = req.params;
    //Kullanıcı yı bulma
    const userId = req.userAuth._id;
```

```
  //Postu bulma
  const post = await Post.findById(id);
  if (!post) {
    throw new Error("Post yok");
  }
  //Post izlenmeyi degiştirme

  await Post.findByIdAndUpdate(
    id,
    {
      $addToSet: { postViews: userId },
    },
    { new: true }
  ).populate("author");
  await post.save();
  res.status(200).json({ message: "Post bakan sayısı değiştirildi", post });
});
```

Diğer Controllerlerde'de Benzer işlemler Yapılıyor.

Ara Yazılımlar

isLoggin

```
const jwt = require("jsonwebtoken"); //Token oluşturmak için kütüphane
const User = require("../model/User/User"); //User model
const isLoggin = (req, res, next) => { //Giriş metot'umuz
  const token = req.headers.authorization?.split(" ")[1]; //Giriş yapan kullanıcının
token'ini alma
  jwt.verify(token, process.env.JWT_KEY, async (err, decoded) => { //alınan token deşifre
etme ve kontrol etme metotu
    //Kullanıcı Id sini bulma ve id'ye kayıtlı token'i çözme işlemi
    const userId = decoded?.user?.id;
    //Kullanıcı bilgilerinin tümünü getirdik
    const user = await User.findById(userId).select("username email role _id");
    //Kullanıcı bilgilerini req objesine geçirme
    req.userAuth = user;
    if (err) { //yapılan işlemler sırasında hata varsa burası çalışır.
      const err = new Error("Geçersiz token");
      next(err);
    } else { //yoksa kod devam eder.
      next();
    }
  });
};

module.exports = isLoggin;
```

AccountVerification

```
const User = require("../model/User/User"); //User Modeli çagırılır

const checkAccountVerification = async (req, res, next) => {
```

```
  try {
    //Kullanıcı ara
    const user = await User.findById(req.userAuth._id);
    //Hesap onaylanmış mı kontol et
    if (user?.isVerified) {
      next(); //Eger onaylanmışsa devam et deriz. Yazmasak devam etmez bekler veya hata
verir.
    } else {
      res.status(401).json({ message: "Hesap onaylanmamış" }); //Degilse hata verir ve
durur.
    }
  } catch (error) {
    res.status(500).json({ message: "Server Hatası", error });
  }
};

module.exports = checkAccountVerification;
```

globalErrorHandler

```
//Hatalı alanlarda vermesi gereken kısım
const globalErrorHandler = (err, req, res, next) => { //err oldugu için tüm alanlar hata
oldugunda bu alana gelir ve hatalarını burada degerlendirir.
  //status mesajı
  const status = err?.status ? err?.status : "Hatalı";
  //Hata mesajı
  const message = err?.message;
  //Hatanın nerede oldugunu veren alan
  const stack = err?.stack;
  res.status(500).json({ //geriye altdaki degerleri döndür.
    status,
    message,
    stack,
  });
};

//404 hatalarında vermesi gereken alan
const notFound = (req, res, next) => { //eger sayfa yok ise genel olarak hepsinde bu
hatayı verir.
  const err = new Error(`Aradığınız bu ${req.originalUrl} sayfa bulunamadı`);
  next(err); //burada err oldugu için aynı şekilde tüm alanlar buraya gelip hatalarını
degerlendirirler.
};

module.exports = { notFound, globalErrorHandler };
```

7. Router'ler

User Router

```
const multer = require("multer"); //resim yükleme için gereken kütüphane
const express = require("express"); //router işlemleri için
```

```
const {
  register,
  login,
  getProfile,
  blockUser,
  unblockuser,
  profileViewers,
  followingUser,
  unFollowingUser,
  forgotpassword,
  resetPassword,
  accountVerificationEmail,
  verifyAccount,
  uploadeProfilePicture,
  uploadeCoverImage,
  updateUserProfile,
} = require("../../controllers/users/usersCtrl"); //kullanılan metotlar
const isLoggin = require("../../middlewares/isLoggin"); //Kullanıcının giriş yapması
gereken yönlendirmeler için gereken metot.
const storage = require("../../utils/fileUpload"); //Resim yükleme için gerekli metot
//!Dosya yükleme ara yazılımı
const upload = multer({ storage }); //Kullanabilmek için değişkene atadık.

//*Kütüphaneyi kullana bilmek için değişkene atadım.
const usersRouter = express.Router();

//! Register sayfasına gönderir.
usersRouter.post("/register", upload.single("profilePicture"), register);
//! Login sayfasına gönderir.
usersRouter.post("/login", login);
//! Kullanıcı resim yükleme
usersRouter.put(
  "/upload-profile-image",
  isLoggin,
  upload.single("file"),
  uploadeProfilePicture
);
//! Kullanıcı küçük resim yükleme
usersRouter.put(
  "/upload-cover-image",
  isLoggin,
  upload.single("file"),
  uploadeCoverImage
);
//! ID ye göre girme sayfasına gönderir.
usersRouter.get("/profile/", isLoggin, getProfile);
//! kullanıcı adı ve mail güncelleme
usersRouter.put("/update-profile/", isLoggin, updateUserProfile);
//! kullanıcı engelleme
usersRouter.put("/block/:userIdToBlock/", isLoggin, blockUser);
//! kullanıcı engelini kaldırma
usersRouter.put("/unblock/:userIdToUnBlock/", isLoggin, unblockuser);
//! Profile bakanları görme
usersRouter.get("/profile-viewer/:userProfileId/", isLoggin, profileViewers);
//! Takip etme
usersRouter.put("/following/:userToFollowId/", isLoggin, followingUser);
```

```
//! Takipden çıkma
usersRouter.put("/unfollowing/:userToUnFollowId/", isLoggin, unFollowingUser);
//! Şifremi unuttum
usersRouter.post("/forgot-password/", forgotpassword);
//! Şifreyi resetleme
usersRouter.post("/reset-password/:resetToken/", resetPassword);
//! Hesap Onaylama Mail
usersRouter.put(
  "/account-verification-email/",
  isLoggin,
  accountVerificationEmail
);
//! Hesap Onaylama Mail
usersRouter.put("/account-verification/:verifyToken/", isLoggin, verifyAccount);

//*Kullana bilmek için eksport ediyorum
module.exports = usersRouter;
```

Post Router

```
const express = require("express"); //router işlemleri için
const isLoggin = require("../../middlewares/isLoggin"); //Kullanıcının giriş yapması
gereken yönlendirmeler için gereken metot.
const {
  createPost,
  getPost,
  getPosts,
  deletePosts,
  updatePosts,
  likePost,
  disLikePost,
  claps,
  schedule,
} = require("../../controllers/posts/posts");//kullanılan metotlar
const checkAccountVerification = require("../../middlewares/isAccountVerified"); //Hesap
onay metotu
const storage = require("../../utils/fileUpload"); //resim yükleyebilmek için metot
const multer = require("multer"); //cloudinary kütüphanesi
const postRouter = express.Router(); //yönlendirme baglantısı

//!Dosya yükleme ara yazılımı
const upload = multer({ storage });

//Post yönlendirme oluşturma
postRouter.post(
  "/",
  isLoggin, //yeni bir post oluşturmak için giriş yapması gerek anlamına gelir
  checkAccountVerification, //yeni bir post oluşturmak için hesap onaylanmış olması gerek
anlamına gelir
  upload.single("file"), //resim yükleye bilmesi için gerekir
  createPost //yeni post oluşturma metotu
);

postRouter.get("/", getPosts);
```

```
postRouter.get("/:id", getPost);
postRouter.delete("/:id", isLoggin, deletePosts); //giriş yapması gerek
postRouter.put("/:id", isLoggin, updatePosts);//giriş yapması gerek
postRouter.put("/likes/:id", isLoggin, likePost);//giriş yapması gerek
postRouter.put("/dislikes/:id", isLoggin, disLikePost);//giriş yapması gerek
postRouter.put("/claps/:id", isLoggin, claps);//giriş yapması gerek
postRouter.put("/schedule/:postId", isLoggin, schedule);//giriş yapması gerek

module.exports = postRouter;
```

Diger Router işlmeleride benzer şekilde yapılmıştır.

8.Diğer İşlemler

Generate Token

```
const jwt = require("jsonwebtoken"); //Token oluşturabilmek için kullanılan kütüphane

const generateToken = (user) => { //Token oluşturma metot
  const payload = { //obje oluşturuyoruz
    user: {
      id: user.id, //bu metot'a bir deger gelecek o gelen degerden ID'yi alıyoruz.
  };
  const token = jwt.sign(payload, process.env.JWT_KEY, {
 //1. aldıgı ilke deger: payload obje'sini jwt.sing ile şifreler ve geri bir deger
döndürür.
//2. aldıgı ikinci ise şifreleme anahtarı bunu gizli oldugu için .env dosyasında tutuk.
    expiresIn: 36000,
  });  // expiresIn : süreyi belirtmek için kullanılır. "anykey" bir anahtardır bu jwt yi
çözmek için çözülecegi yerde bunu girmek gerekir.
  return token; //metot çagırıldıgında token'i döndürür.
};

module.exports = generateToken;
```

File yükleme

```
const cloudinary = require("cloudinary").v2; //resim yükelemek için kullacagımız
kütüphane
require("dotenv").config(); //gizli dosyaları almak için
const { CloudinaryStorage } = require("multer-storage-cloudinary"); //resim yükelemek
için kullacagımız kütüphane

//Ayarlar cloudinary
cloudinary.config({
  cloud_name: process.env.CLOUDINARY_NAME,
  api_key: process.env.CLOUDINARY_KEY,
  api_secret: process.env.CLOUDINARY_SECRET,
});

//Kullanılacak alan
const storage = new CloudinaryStorage({
```

```
  cloudinary, //resimin yüklenecegi hesap bilgileri gelir
  allowedFormats: ["jpg", "png", "jpeg"], //yüklencek formatlar
  params: { //parametreler
    folder: "blogify-api", //yüklenecek klasör isimi
    transformation: [{ width: 500, height: 500, crop: "limit" }], //özel ayarlar
  },
});

module.exports = storage;
```

Hesap onay mail

```
const nodemailer = require("nodemailer"); //mail göndermek için kullanılan kütüphane
require("dotenv").config(); //gizli bilgileri almak için

//Mail gönderme fonksiyonu

const sendAccVerificationEmail = async (to, resetToken) => {
  try {
    //Gönderici oluşturma
    const tarnsporter = nodemailer.createTransport({ //Mail atıcı oluşturuyoruz.
      host: "smtp.gmail.com", //kullanılacak host
      port: 587, //genel olarak kullanılan port mail atmak için
      secure: false, //güvenli olup olmadığını ayarlarız
      auth: { //hesap bilgileri mail atılacak mail bilgileri gibi
        user: process.env.GMAIL_USER, //burada belirtilen mailden mailler atılacak,
        pass: process.env.GMAIL_PASS  //uygulamalara özel kullanılmak için alınan kod,
      },
    });
    //Gidecek mesaj
    const message = {
      to,
      subject: "Account is Verify", //Başlık
      html: `
        <p>You are receiving this email because you (or someone else) have requested
verify you account</p>
        <p>Please click on the following link, or paste this into your browser to
complete the process:</p>
        <p>https://blogify-inovotek.netlify.app/reset-password/${resetToken}</p>
        <p>If you did not request this, please ignore this email and your password will
remain unchanged.</p>
        `,//İçerik
    };
    //Mail'i gönderme işlemi gerçekleşir.
    const info = await tarnsporter.sendMail(message);
  } catch (error) {
    console.log(error);
    throw new Error("Email sending failed");
  }
};

module.exports = sendAccVerificationEmail;
```

Şifre yenileme mail'i(Yukardaki işlemler ile çok benzer)

```
const nodemailer = require("nodemailer");
require("dotenv").config();

//Mail gönderme fonksiyonu

const sendEmail = async (to, resetToken) => {
  try {
    //Gönderici oluşturma
    const tarnsporter = nodemailer.createTransport({
      host: "smtp.gmail.com",
      port: 587,
      secure: false,
      auth: {
        user: process.env.GMAIL_USER, //user email address,
        pass: process.env.GMAIL_PASS,
      },
    });
    //Gidecek mesaj
    const message = {
      to,
      subject: "Password reset",
      html: `
        <p>You are receiving this email because you (or someone else) have requested the
reset of a password.</p>
        <p>Please click on the following link, or paste this into your browser to
complete the process:</p>
        <p>https://blogify-inovotek.netlify.app/reset-password/${resetToken}</p>
        <p>If you did not request this, please ignore this email and your password will
remain unchanged.</p>
        `,
    };
    //send the email
    const info = await tarnsporter.sendMail(message);
    console.log("Email mesaj", info.messageId);
  } catch (error) {
    console.log(error);
    throw new Error("Email sending failed");
  }
};

module.exports = sendEmail;
```