Santiye Maliyet Yonetim

Bu proje Anıl Tarar tarafından FullStack bir web uygulaması olması amacıyla hazırlanmıştır. Bu dökümantasyon bu projeyi geliştirme aşamasındayken yapılan işlemleri barındıracaktır. Sadece proje dökümantasyonu olmaktan ziyade öğretici / serüven bazlı bir dökümantasyon olacaktır. Kodların kendisiyle beraber mümkün olduğunca açıklamaları da olacaktır.

Bana ulaşmak isterseniz

- Linkedin ⇒ https://www.linkedin.com/in/anil-tarar/
- Mail ⇒ aniltararr@gmail.com

Proje Amacı

Firmaların şantiye bazında yaptıkları tüm giderleri kategorize ederek kayit altina alabileceği, kullanıcı yetkilerine göre yönetebileceği ve raporlama yapabileceği bir SaaS platform.

Kullanıcı Rolleri ve Yetkileri

- Admin : Tüm şantiyleri ve kullanıcıları yönetir.

- Editör (Şantiye Şefi) : Sadece yetkili olduğu şantiyeleri yönetir. Maliyet ve kullanıcı ataması yapabilir
- User (Kullanıcı) : Sadece atanmış olduğu şantiyedeki verileri görüntüleyebilir.

Ana Modüller ve Açıklamaları

1. Authentication Modülü : Kayıt Olma, Giriş Yapma , Şifre Yenileme, JWT Auth , Role Auth
2. Kullanıcı Yönetimi : Kullanıcı CRUD
3. Şantiye Yönetimi : Şantiye CRUD , Kullanıcıları Şantiyeye Atama
4. Maliyet Kategori : Maliyet Kategori CRUD'ı
5. Maliyet Kayit : Maliyet Kategoriler Veri CRUD'ı, İlgili Şantiyeye Maliyet Kategori Eklemek
6. Raporlama ve Grafikler : Şantiye ve Maliyet Kategorisi bazlı grafikler
7. PDF Rapor Alma : Şantiyeye ait bilgilerin export edilmesi

Veritabanı Modelleri

1. **User**

- `name`
- `email`
- `password`
- `role`
- `assignedSites`

---

2. **Site**

- `name`
- `location`
- `budget`

- startDate
- endDate
- createdBy

---

3. CostCategory

- name
- description
- isGlobal
- siteId

---

4. Cost

- title
- description
- unit
- quantity
- unitPrice
- netAmount
- taxRate
- taxAmount
- grossAmount
- moneyType
- costCategory
- siteId
- createdBy
- fileUrl
- createdAt

- `updatedAt`

Backend Süreçleri

Projenin açılması ve mongoDB eklenmesi

Terminal üzerinden projenin açılması ve paketlerin kurulması

```
//terminal üzerinde eklenecek paketler.
npm init -y
npm i express cors nodemon bcryptjs dotenv jsonwebtoken mongoose
```

src/server.js açılması

```
const express = require('express');
const cors = require('cors');
const dotenv = require('dotenv');

const app = express();
dotenv.config();

app.use(cors());
app.use(express.json());

app.get('/', (req, res) => {
   res.send('Welcome to the API!');
});



app.listen(process.env.PORT, ()=>{
   console.log(`Server is running on port http://localhost:${process.env.PORT}
`);
})
```

.env dosyasının açılması

```
PORT = 3005
MONGO_URI= mongodb+srv://root:<dbpassword>@sitecostmanagementapp.wgaxm6p.mongodb.net/?retryWrites=true&w=majority&appName=SiteCostManagementApp
```

config/databaseConfig.js dosyasının acılması

```
//database config dosyası
const mongoose = require("mongoose");

const connectDatabase = async () => {
 try {
  await mongoose.connect(process.env.MONGO_URI).then(() => {
    console.log("*****MongoDB Connection Successful*****");
   });
 } catch (err) {
   console.log(err);
 }
};

module.exports = connectDatabase;
```

mongoDB'ye bağlanmak için yazdığımız fonksiyon. Server calisirken önce db'ye bağlanacak, sonrasında server listen yapacak şekilde ayarlama yapacağız.

server.js 'e mongoDB connect'in eklenmesi

```
const express = require('express');
const cors = require('cors');
const dotenv = require('dotenv');
const connectDatabase = require('./config/databaseConfig');

const app = express();
```

```
dotenv.config();

app.use(cors());
app.use(express.json());

app.get('/', (req, res) => {
   res.send('Welcome to the API!');
});

connectDatabase().then(()=>{
   app.listen(process.env.PORT,()=>{
      console.log(`Server is running on port ${process.env.PORT}`);
   })
})
```

User İşlemleri

Burada User için gereken model tanımlamaları, controller'i ve token işlemlerini yapacağım.

User Model'inin oluşturulması.

```
// model/user.js
const mongoose = require("mongoose");

const userSchema = new mongoose.Schema(
 {
   firstName: {
     type: String,
     required: true,
     trim: true,
   },
   lastName: {
     type: String,
     required: true,
```

```
      trim: true,
    },
    email: {
      type: String,
      required: true,
      unique: true,
      trim: true,
    },
    password: {
      type: String,
      required: true,
    },
    role: {
      type: String,
      enum: ["user", "editor", "admin"],
      default: "user",
    },
  },
  { timestamps: true }
);

module.exports = mongoose.model("User", userSchema);
```

Kullanıcılarımızın ne şekilde depolanacağını burada belirliyoruz. Bu model kullanıcı verimizin datalarını tutacak.

Token Modelinin oluşturulması

```
const mongoose = require("mongoose");

const tokenSchema = new mongoose.Schema(
  {
    userId: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "User",
      required: true,
```

```
    },
    refreshToken: {
      type: String,
      required: true,
    },
  },
  { timestamps: true }
);

module.exports = mongoose.model("Token", tokenSchema);
```

Database üzerinde refreshToken tutacağız. Kullanıcıların isteklerini bu tokenlar sayesinde sağlayacağız.

controller/authController.js dosyasının yazılması.

```
const User = require("../model/user.js");
const Token = require("../model/token.js");
const bcrypt = require("bcryptjs");
const jwt = require("jsonwebtoken");

// Token Oluşturma Fonksiyonu
const generateTokens = async (user) => {
  // Kısa süreli erişim token'ı
  const accessToken = jwt.sign(
    {
      userId: user._id,
      email: user.email,
    },
    process.env.ACCESS_TOKEN_SECRET,
    { expiresIn: "15m" }
  );

  // Uzun süreli refresh token'ı
```

```
  const refreshToken = jwt.sign(
    {
      userId: user._id,
    },
    process.env.REFRESH_TOKEN_SECRET,
    { expiresIn: "7d" }
  );

  //   DB'de kayıtlı olan token'ı sil.
  await Token.deleteMany({ userId: user._id });

  // Yeni token'ı oluştur ve kaydet.
  const newRefreshToken = await Token.create({
    userId: user._id,
    refreshToken: refreshToken,
    date: new Date(),
  });

  return { accessToken, refreshToken };
};

// Kullanıcı Kaydı
const register = async (req, res) => {
  try {
    const { firstName, lastName, email, password } = req.body;
    if (!firstName || !lastName || !email || !password) {
      return res.status(400).json({
        message:
          "Ad, soyad, email ve şifre gereklidir. Lütfen tüm alanları doldurun.",
      });
    }

    // Kullanıcı veritabanında var mı kontrol et
    const existingUser = await User.findOne({
      email,
    });
```

```
if (existingUser) {
  return res.status(400).json({
    message: "Bu email adresi zaten kayıtlı.",
  });
}

// Şifreyi hashle
const salt = await bcrypt.genSalt(10);
const hashedPassword = await bcrypt.hash(password, salt);

// Yeni kullanıcıyı oluştur
const user = await User.create({
  firstName,
  lastName,
  email,
  password: hashedPassword,
  role: "user",
});

// Token'ları oluştur
const tokens = await generateTokens(user);

const userWithoutPassword = {
  firstName: user.firstName,
  lastName: user.lastName,
  email: user.email,
  role: user.role,
  tokens: tokens,
};

res.cookie("refreshToken", tokens.refreshToken, {
  httpOnly: true,
  secure: false,
  sameSite: "none",
});
```

```
    res.cookie("accessToken", tokens.accessToken, {
      httpOnly: true,
      secure: false,
      sameSite: "none",
    });

    res.status(201).json(userWithoutPassword);
  } catch (error) {
    res.status(500).json({
      message: "Kullanıcı kaydı sırasında bir hata oluştu.",
      error: error.message,
    });
  }
};

// Tokenları Yenileme
const tokenRefresh = async (req, res) => {
  try {
    // RefreshToken'ın eski halini al
    const oldRefreshToken = req.cookies.refreshToken || req.body.refreshToke
n;
    if (!oldRefreshToken) {
      return res.status(401).json({
        message: "Refresh token bulunamadı. Lütfen tekrar giriş yapın.",
      });
    }

    // DB'de kayıtlı olan token'ı bul
    const storedToken = await Token.findOne({
      refreshToken: oldRefreshToken,
    });
    if (!storedToken) {
      return res.status(401).json({
        message:
          "Refresh token veritabanında bulunamadı. Lütfen tekrar giriş yapın.",
```

```
      });
    }
    //  Kullanıcıyı bul
    const user = await User.findById(storedToken.userId);

    if (!user) {
      return res.status(401).json({
        message: "Kullanıcı bulunamadı. Lütfen tekrar giriş yapın.",
      });
    }

    // Yeni token'ları oluştur
    const tokens = await generateTokens(user);

    res.cookie("refreshToken", tokens.refreshToken, {
      httpOnly: true,
      secure: false,
      sameSite: "none",
    });
    res.cookie("accessToken", tokens.accessToken, {
      httpOnly: true,
      secure: false,
      sameSite: "none",
    });

    return res.status(200).json(tokens);
  } catch (error) {
    res.status(500).json({
      message: "Token yenileme sırasında bir hata oluştu.",
      error: error.message,
    });
  }
};

// Kullanıcı Girişi
const login = async (req, res) => {
```

```
  try {
    const { email, password } = req.body;
    if (!email || !password) {
      return res.status(400).json({
        message: "Email ve şifre gereklidir. Lütfen tüm alanları doldurun.",
      });
    }

    // Kullanıcıyı veritabanında bul
    const user = await User.findOne({ email });
    if (!user) {
      return res.status(404).json({
        message: "Bu email adresi ile kayıtlı kullanıcı bulunamadı.",
      });
    }

    const isPasswordMatch = await bcrypt.compare(password, user.passwor
d);
    if (!isPasswordMatch) {
      return res.status(401).json({
        message: "Şifre yanlış. Lütfen tekrar deneyin.",
      });
    }

    // Token'ları oluştur
    const tokens = await generateTokens(user);
    const userWithoutPassword = {
      firstName: user.firstName,
      lastName: user.lastName,
      email: user.email,
      role: user.role,
      tokens: tokens,
    };

    res.cookie("refreshToken", tokens.refreshToken, {
      httpOnly: true,
```

```
      secure: false,
      sameSite: "none",
    });
    res.cookie("accessToken", tokens.accessToken, {
      httpOnly: true,
      secure: false,
      sameSite: "none",
    });
    res.status(200).json(userWithoutPassword);
  } catch (error) {
    res.status(500).json({
      message: "Giriş sırasında bir hata oluştu.",
      error: error.message,
    });
  }
};

// Kullanıcı Çıkışı
const logout = async (req,res) => {
    try {
        const refreshToken = req.cookies.refreshToken || req.body.refreshToken;
        if (!refreshToken) {
            return res.status(400).json({
                message: "Çıkış yapmak için refresh token gereklidir.",
            });
        }
        const deletedToken = await Token.deleteOne({ refreshToken });
        if(!deletedToken){
            return res.status(400).json({
                message: "Refresh token veritabanında bulunamadı. Çıkış yaparken
bir sorun oluştu.",
            });
        }

        res.clearCookie("refreshToken", {
            httpOnly: true,
```

```
            secure: false,
            sameSite: "none",
        });
        res.clearCookie("accessToken", {
            httpOnly: true,
            secure: false,
            sameSite: "none",
        });

        res.status(200).json({
            message: "Başarıyla çıkış yapıldı.",
        });

    } catch (error) {
        res.status(500).json({
            message: "Çıkış sırasında bir hata oluştu.",
            error: error.message,
        });
    }
}

module.exports = {
    register,
    login,
    logout,
    tokenRefresh
}
```

✅ register(req, res)

Yeni kullanıcı kaydını yapar.

- Email adresinin daha önce kullanılıp kullanılmadığını kontrol eder.
- Şifreyi hash'leyip veritabanına kaydeder.
- Erişim ve yenileme token'ları üretip kullanıcıya gönderir.

✅ `login(req, res)`

Kullanıcı girişi sağlar.

- Email ve şifre kontrolü yapar.
- Şifre doğruysa access ve refresh token üretir ve cookie olarak kaydeder.
- Giriş yapan kullanıcının bilgilerini döner.

✅ `logout(req, res)`

Kullanıcının çıkış yapmasını sağlar.

- Refresh token'ı veritabanından siler.
- Tarayıcıdaki token cookie'lerini temizler.

✅ `tokenRefresh(req, res)`

Oturum süresi dolan kullanıcılar için yeni token üretir.

- Gönderilen refresh token geçerliyse, yeni access ve refresh token oluşturur.
- Yeni token'ları cookie olarak döner.

🔐 `generateTokens(user)`

Access ve refresh token üretir.

- Refresh token'ı veritabanında saklar.
- Önceki token'ları siler.

authRoute.js dosyasının oluşturulması

```
const express = require('express');
const router = express.Router();

const { register, login, logout, tokenRefresh } = require('../controller/authContr
oller.js');

router.post('/register', register);
router.post('/login', login);
```

```
router.post('/logout', logout);
router.post('/token-refresh', tokenRefresh);

module.exports = router;
```

authRoute bilgilerinin server.js'e eklenmesi

```
const express = require('express');
const cors = require('cors');
const dotenv = require('dotenv');
const cookieParser = require('cookie-parser');
const connectDatabase = require('./config/databaseConfig');

const authRoute = require("./routes/authRoute.js")


const app = express();
dotenv.config();

app.use(cors());
app.use(express.json());
app.use(cookieParser());

app.get('/', (req, res) => {
   res.send('Welcome to the API!');
});


app.use("/auth", authRoute);

connectDatabase().then(()=>{
   app.listen(process.env.PORT,()=>{
      console.log(`Server is running on port ${process.env.PORT}`);
```

```
    })
})
```

Swagger’in kurulması

Swagger ile birlikte projemizin API’larını dökümante edeceğiz. Öncelikle gerekli paketlerin kurulumunu yapalım.

```
npm i swagger-jsdoc
npm i swagger-ui-express
```

Paket kurulumları sonrasında src/config/swagger.config.js dosyasını açıyoruz ve içerisini şu şekilde dolduruyoruz.

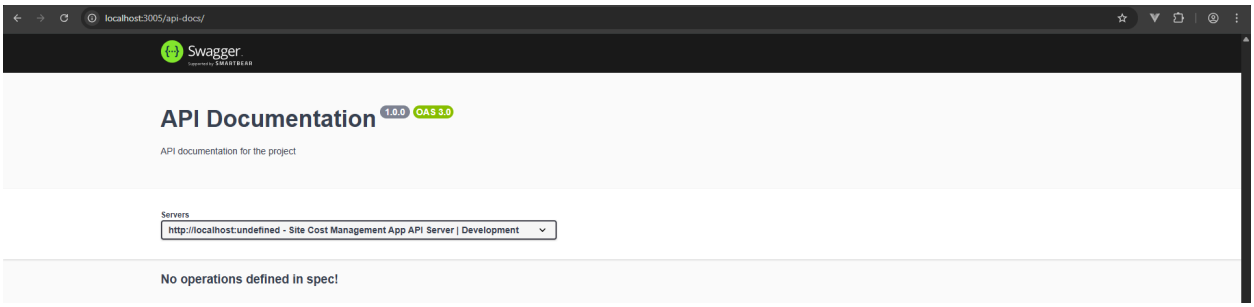
```
//swagger.config.js
const swaggerJSDoc = require('swagger-jsdoc');

const swaggerOptions = {
  definition: {
    openapi: "3.0.0",
    info: {
      title: "API Documentation",
      version: "1.0.0",
      description: "API documentation for the project",
    },
    servers: [
      {
        url: `http://localhost:${process.env.API_PORT}`,
        description: "Site Cost Management App API Server | Development",
      },
    ],
  },
  apis: ["./src/routes/*.js"],
};
```

```
const swaggerSpec = swaggerJSDoc(swaggerOptions);

module.exports = swaggerSpec;
```

Swagger'i UI olarak görmek için server.js'te kullanıma açmamız gerekmekte.

```
// packages
const express = require('express');
const cors = require('cors');
const dotenv = require('dotenv');
const cookieParser = require('cookie-parser');
const swaggerUi = require('swagger-ui-express');

// routes import
const authRoute = require("./routes/authRoute.js")

const app = express();
dotenv.config();
app.use(cors());
app.use(express.json());
app.use(cookieParser());

// config import
const swaggerSpec = require('./config/swagger.config.js');
const connectDatabase = require('./config/databaseConfig');

app.get('/', (req, res) => {
   res.send('Welcome to the API!');
});

// Routes
app.use("/auth", authRoute);


// Swagger
app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(swaggerSpec));
```

```
connectDatabase().then(()⇒{
   app.listen(process.env.PORT,()⇒{
      console.log(`Server is running on port ${process.env.PORT}`);
   })
})
```



Artık URL'e /api-docs yazıp gittiğimizde swagger ui karsımıza cıkıyor

Auth Route'a özel swagger yapılandırması

Evet swagger'imizi tanımladık ancak ui'de ne şekilde gözükeceğini her route özelinde tanımlamamız lazım. Ben burada ai'dan yardım almayı tercih ediyorum. VS Code üzerinde copilot'u açarak controller ve route dosyamı verip benim için swagger metinlerini oluştur diyorum.

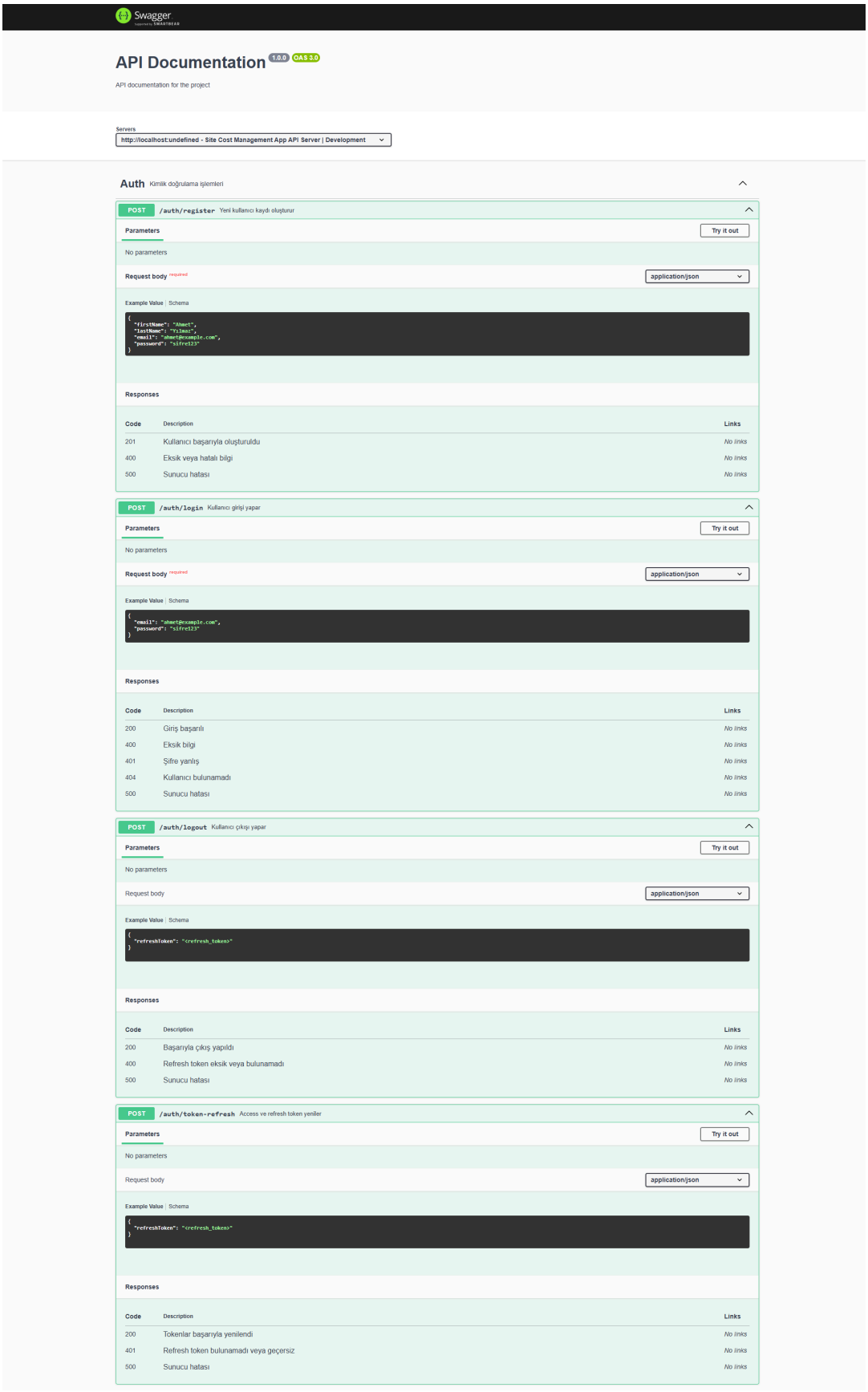
```
// authRoute.js
const express = require('express');
const router = express.Router();

const { register, login, logout, tokenRefresh } = require('../controller/authContr
oller.js');
/**
 * @swagger
 * tags:
 *   name: Auth
 *   description: Kimlik doğrulama işlemleri
 */
```

```
/**
 * @swagger
 * /auth/register:
 *   post:
 *     summary: Yeni kullanıcı kaydı oluşturur
 *     tags: [Auth]
 *     requestBody:
 *       required: true
 *       content:
 *         application/json:
 *           schema:
 *             type: object
 *             required:
 *               - firstName
 *               - lastName
 *               - email
 *               - password
 *             properties:
 *               firstName:
 *                 type: string
 *                 example: Ahmet
 *               lastName:
 *                 type: string
 *                 example: Yılmaz
 *               email:
 *                 type: string
 *                 example: ahmet@example.com
 *               password:
 *                 type: string
 *                 example: sifre123
 *     responses:
 *       201:
 *         description: Kullanıcı başarıyla oluşturuldu
 *       400:
 *         description: Eksik veya hatalı bilgi
```

```
 *       500:
 *         description: Sunucu hatası
 */

/**
 * @swagger
 * /auth/login:
 *   post:
 *     summary: Kullanıcı girişi yapar
 *     tags: [Auth]
 *     requestBody:
 *       required: true
 *       content:
 *         application/json:
 *           schema:
 *             type: object
 *             required:
 *               - email
 *               - password
 *             properties:
 *               email:
 *                 type: string
 *                 example: ahmet@example.com
 *               password:
 *                 type: string
 *                 example: sifre123
 *     responses:
 *       200:
 *         description: Giriş başarılı
 *       400:
 *         description: Eksik bilgi
 *       401:
 *         description: Şifre yanlış
 *       404:
 *         description: Kullanıcı bulunamadı
 *       500:
```

```
 *       description: Sunucu hatası
 */

/**
 * @swagger
 * /auth/logout:
 *   post:
 *     summary: Kullanıcı çıkışı yapar
 *     tags: [Auth]
 *     requestBody:
 *       required: false
 *       content:
 *         application/json:
 *           schema:
 *             type: object
 *             properties:
 *               refreshToken:
 *                 type: string
 *                 example: <refresh_token>
 *     responses:
 *       200:
 *         description: Başarıyla çıkış yapıldı
 *       400:
 *         description: Refresh token eksik veya bulunamadı
 *       500:
 *         description: Sunucu hatası
 */

/**
 * @swagger
 * /auth/token-refresh:
 *   post:
 *     summary: Access ve refresh token yeniler
 *     tags: [Auth]
 *     requestBody:
 *       required: false
```

```
 *      content:
 *        application/json:
 *          schema:
 *            type: object
 *            properties:
 *              refreshToken:
 *                type: string
 *                example: <refresh_token>
 *    responses:
 *      200:
 *        description: Tokenlar başarıyla yenilendi
 *      401:
 *        description: Refresh token bulunamadı veya geçersiz
 *      500:
 *        description: Sunucu hatası
 */

router.post('/register', register);
router.post('/login', login);
router.post('/logout', logout);
router.post('/token-refresh', tokenRefresh);

module.exports = router;
```


Swagger
API Documentation 1.0.0 OAS 3.0
API documentation for the project
Servers
http://localhost:undefined - Site Cost Management App API Server | Development
Auth Kimlik doğrulama işlemleri
POST /auth/register Yeni kullanıcı kaydı oluşturur
Parameters
Try it out
No parameters
Request body required
application/json
Example Value | Schema
{
"firstName": "Ahmet",
"lastName": "Yılmaz",
"email": "ahmet@example.com",
"password": "sifre123"
}
Responses
Code Description Links
201 Kullanıcı başarıyla oluşturuldu No links
400 Eksik veya hatalı bilgi No links
500 Sunucu hatası No links
POST /auth/login Kullanıcı girişi yapar
Parameters
Try it out
No parameters
Request body required
application/json
Example Value | Schema
{
"email": "ahmet@example.com",
"password": "sifre123"
}
Responses
Code Description Links
200 Giriş başarılı No links
400 Eksik bilgi No links
401 Şifre yanlış No links
404 Kullanıcı bulunamadı No links
500 Sunucu hatası No links
POST /auth/logout Kullanıcı çıkışı yapar
Parameters
Try it out
No parameters
Request body
application/json
Example Value | Schema
{
"refreshToken": "crefresh_token3"
}
Responses
Code Description Links
200 Başarıyla çıkış yapıldı No links
400 Refresh token eksik veya bulunamadı No links
500 Sunucu hatası No links
POST /auth/token-refresh Access ve refresh token yeniler
Parameters
Try it out
No parameters
Request body
application/json
Example Value | Schema
{
"refreshToken": "crefresh_token3"
}
Responses
Code Description Links
200 Tokenlar başarıyla yenilendi No links
401 Refresh token bulunamadı veya geçersiz No links
500 Sunucu hatası No links

Görüldüğü gibi hepsi geldi.

Auth Middleware ile beraber token kontrolünün sağlanması.

Evet biz accessToken ve refreshToken olusturuyoruz login/register esnasında. Bunun amacı nedir ? Yapılacak işlemlerde kullanıcı giriş yapmış mı yapmamış mı onu görmek. Giriş yapmamış olan kullanıcı işlem yapamaz.

```
//midlewares/authMidleware.js
const jwt = require("jsonwebtoken");

const verifyAccessToken = (req, res, next) => {
 const token =
  req.cookies.accessToken || req.headers["authorization"]?.split(" ")[1];
 if (!token) {
  return res.status(401).json({ message: "Access token bulunamadı. " });
 }

 try {
  const decoded = jwt.verify(token, process.env.ACCESS_TOKEN_SECRET);
  req.user = decoded;
  next();
 } catch (error) {
  return res.status(403).json({ message: "Geçersiz access token." });
 }
};

const verifyRefreshToken = (req, res, next) => {
 const token =
  req.cookies.refreshToken || req.headers["authorization"]?.split(" ")[1];
 if (!token) {
  return res.status(401).json({ message: "Refresh token bulunamadı." });
 }
 try {
  const decoded = jwt.verify(token, process.env.REFRESH_TOKEN_SECRET);
```

```
    req.user = decoded;
    next();
  } catch (error) {
    return res.status(403).json({ message: "Geçersiz refresh token." });
  }
};

module.exports = {
  verifyAccessToken,
  verifyRefreshToken,
};
```

Bunları şuanda değil, auth dısındaki herhangi bir işlemin olduğu esnada middleware olarak kullanacağız.

Check Role ile beraber rol middleware'inin yazılması

Evet kullanıcı giriş yapmış mı yapmamış mı şuanda onu authMiddleware ile kontrol ediyoruz. Peki ya sadece giriş yapmak her api'ye istek atma yetkisi vermeli mi ? Hayır! Bunun için rollendirme kontrolü de eklemeliyiz. O yüzden kısa bir middleware yazıyoruz.

```
//middlewares/checkRole.js
const checkRole = (...roles) => {
   return (req,res,next)=>{
      if(!req.user){
         return res.status(401).json({
            message: "Kullanıcı doğrulanamadı. Lütfen giriş yapın.",
         });
      }
      const userRole = req.user.role;
      if(!roles.includes(userRole)){
         return res.status(403).json({
            message: "Bu işlem için yetkiniz yok.",
         });
```

```
        }
        next();
    }
}

module.exports = checkRole;
```

Şantiye İşlemleri

User, authMiddleware, checkRole gibi işlemleri yaparak kullanıcı bazlı işlemlerin bir kısmını hallettik. Şimdi sırada projenin temellerinde olan şantiye işlemleri var.

Site Model'inin yazılması

```
//model/site.js
const mongoose = require('mongoose');

const siteSchema = new mongoose.Schema({
    name:{
        type: String,
        required: true,
        trim: true,
    },
    location: {
        type: String,
        required: true,
        trim: true,
    },
    budget:{
        type: Number,
        required: true,
    },
    startDate: {
        type: Date,
        required: true,
```

```
    },
    endDate: {
        type: Date,
        required: true,
    },
    createdBy:{
        type: mongoose.Schema.Types.ObjectId,
        ref: 'User',
        required: true,
    },
}, { timestamps: true });


module.exports = mongoose.model('Site', siteSchema);
```

Site Controller'in yazılması

```
const mongoose = require("mongoose");
const Site = require("../model/site.js");
const User = require("../model/user.js");

const createSite = async (req, res) => {
  try {

    const { name, location, budget, startDate, endDate } = req.body;
    if (!name || !location || !budget || !startDate || !endDate) {
      return res.status(400).json({
        message: "Tüm alanlar gereklidir. Lütfen tüm alanları doldurun.",
      });
    }
    // kullanıcının kimliğini doğrula
    const userId = req.user.userId; // req.user, kimlik doğrulama middleware tar
afından ayarlanmalıdır
    const user = await User.findById(userId);
    if (!user) {
```

```
      return res.status(404).json({
        message: "Kullanıcı bulunamadı. Lütfen tekrar giriş yapın.",
      });
    }
    const createdBy = new mongoose.Types.ObjectId(userId);
    // Yeni site oluştur
    const newSite = await Site.create({
      name,
      location,
      budget,
      startDate,
      endDate,
      createdBy,
    });
    return res.status(201).json({
      message: "Site başarıyla oluşturuldu.",
      site: newSite,
    });
  } catch (error) {
    return res.status(500).json({
      message: "Site oluşturulurken bir hata oluştu.",
      error: error.message,
    });
  }
};

module.exports={ createSite };
```

Site Route'in yazılması

```
const express = require("express");
const router = express.Router();
const { createSite } = require("../controller/siteController.js");
const {verifyAccessToken} = require("../middlewares/authMiddleware.js")
const {checkRole} = require("../middlewares/checkRole.js");
```

```
/**
 * @swagger
 * tags:
 *   name: Site
 *   description: Şantiye işlemleri
 */

/**
 * @swagger
 * /site/create:
 *   post:
 *     summary: Yeni site (şantiye) oluşturur
 *     tags: [Site]
 *     security:
 *       - bearerAuth: []
 *     requestBody:
 *       required: true
 *       content:
 *         application/json:
 *           schema:
 *             type: object
 *             required:
 *               - name
 *               - location
 *               - budget
 *               - startDate
 *               - endDate
 *             properties:
 *               name:
 *                 type: string
 *                 example: Merkez Şantiye
 *               location:
 *                 type: string
 *                 example: İstanbul
 *               budget:
 *                 type: number
```

```
 *               example: 1000000
 *             startDate:
 *               type: string
 *               format: date
 *               example: 2025-08-01
 *             endDate:
 *               type: string
 *               format: date
 *               example: 2026-08-01
 *     responses:
 *       201:
 *         description: Site başarıyla oluşturuldu
 *       400:
 *         description: Tüm alanlar gereklidir
 *       404:
 *         description: Kullanıcı bulunamadı
 *       500:
 *         description: Sunucu hatası
 */

router.post("/create", verifyAccessToken, checkRole("admin"), createSite);

module.exports = router;
```

Site Route'un server.js'e eklenmesi

```
// packages
const express = require('express');
const cors = require('cors');
const dotenv = require('dotenv');
const cookieParser = require('cookie-parser');
const swaggerUi = require('swagger-ui-express');

// routes import
const authRoute = require("./routes/authRoute.js")
```

```
const siteRoute = require("./routes/siteRoute.js");

const app = express();
dotenv.config();
app.use(cors({
   origin:"http://localhost:3000",
   credentials: true
}));
app.use(express.json());
app.use(cookieParser());

// config import
const swaggerSpec = require('./config/swagger.config.js');
const connectDatabase = require('./config/databaseConfig');

app.get('/', (req, res) => {
   res.send('Welcome to the API!');
});

// Routes
app.use("/auth", authRoute);
app.use("/site", siteRoute);


// Swagger
app.use('/api-docs', swaggerUi.serve, swaggerUi.setup(swaggerSpec));

connectDatabase().then(()=>{
   app.listen(process.env.PORT,()=>{
      console.log(`Server is running on port ${process.env.PORT}`);
   })
})
```

CostCategory Modelinin oluşturulması

```
//model/costCategory.js
const mongoose = require('mongoose');

const costCategorySchema = new mongoose.Schema({
name:{
    type: String,
    required: true,
    trim: true
},
description:{
    type: String,
    required: false,
    trim: true
},
isGlobal:{
    type: Boolean,
    required: true,
    default: false
},
siteId:{
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Site',
    required:true,
}
}, { timestamps: true });

module.exports = mongoose.model('CostCategory', costCategorySchema);
```

Cost Category ile beraber maliyetler için oluşturulacak kategorilerimiz model tipini belirledik. Ad, açıklama, genellik (her şantiyede maliyet kategorisi olarak gözükecek mi), ve şantiye id'si olacak şekilde ayarladım.

CostCategory Controller'in yazılması

```
const mongoose = require("mongoose");
const CostCategory = require("../model/costCategory.js");
const Site = require("../model/site.js");
const User = require("../model/user.js");

const getCostCategories = async (req, res) ⇒ {
  try {
    const costCategories = await CostCategory.find();
    return res.status(200).json({
      costCategoriesCount: costCategories.length,
      message: "Cost categories başarıyla listelendi.",
      costCategories,
    });
  } catch (error) {
    return res
      .status(500)
      .json({ message: "Cost categories listelenirken bir hata oluştu." });
  }
};

const getCostCategoriesBySiteId = async (req, res) ⇒ {
  try {
    const { siteId } = req.params;
    const costCategories = await CostCategory.find({ siteId });

    if (costCategories.length === 0) {
      return res
        .status(404)
        .json({ message: "Bu şantiyeye ait hiç cost category bulunamadı." });
    }

    return res.status(200).json({
      costCategoriesCount: costCategories.length,
      message: "Cost categories başarıyla listelendi.",
      costCategories,
```

```
    });
  } catch (error) {
    return res
      .status(500)
      .json({ message: "Cost categories listelenirken bir hata oluştu." });
  }
};

const createCostCategory = async (req, res) => {
  try {
    const { name, description, isGlobal, siteId } = req.body;
    if (!name || !siteId) {
      return res.status(400).json({
        message: "Tüm alanlar gereklidir. Lütfen tüm alanları doldurun.",
      });
    }

    const userId = req.user.userId;
    const user = await User.findById(userId);
    if (!user) {
      return res
        .status(404)
        .json({ message: "Kullanıcı bulunamadı. Lütfen tekrar giriş yapın." });
    }
    const createdBy = new mongoose.Types.ObjectId(userId);

    const site = await Site.findById(siteId);
    if (!site) {
      return res.status(404).json({ message: "Site bulunamadı." });
    }

    const newCostCategory = await CostCategory.create({
      name,
      description,
      isGlobal: isGlobal || false,
      siteId,
```

```
      createdBy,
    });
    return res.status(201).json({
      message: "Cost category başarıyla oluşturuldu.",
      costCategory: newCostCategory,
    });
  } catch (error) {
    return res
      .status(500)
      .json({ message: "Cost category oluşturulurken bir hata oluştu." });
  }
};

const deleteCostCategory = async (req, res) => {
  try {
    const { id } = req.params;
    const deletedCategory = await CostCategory.findByIdAndDelete(id);
    if (!deletedCategory) {
      return res.status(404).json({ message: "Cost category bulunamadı." });
    }
    return res.status(200).json({
      message: "Cost category başarıyla silindi.",
      costCategory: deletedCategory,
    });
  } catch (error) {
    return res
      .status(500)
      .json({ message: "Cost category silinirken bir hata oluştu." });
  }
};

const updateCostCategory = async (req, res) => {
  try {
    const { id } = req.params;
    const { name, description, isGlobal, siteId } = req.body;
```

```
    const updatedCategory = await CostCategory.findByIdAndUpdate(
      id,
      { name, description, isGlobal, siteId },
      { new: true }
    );

    if (!updatedCategory) {
      return res.status(404).json({ message: "Cost category bulunamadı." });
    }

    return res.status(200).json({
      message: "Cost category başarıyla güncellendi.",
      costCategory: updatedCategory,
    });
  } catch (error) {
    return res
      .status(500)
      .json({ message: "Cost category güncellenirken bir hata oluştu." });
  }
};

module.exports = {
  createCostCategory,
  deleteCostCategory,
  updateCostCategory,
  getCostCategories,
  getCostCategoriesBySiteId,
};
```

CostCategoryController Route'in yazılması

```
const express = require("express");
const router = express.Router();
const {
  createCostCategory,
```

```
  deleteCostCategory,
  updateCostCategory,
  getCostCategories,
  getCostCategoriesBySiteId,
} = require("../controller/costCategoryController.js");
const { verifyAccessToken } = require("../middlewares/authMiddleware.js");
const { checkRole } = require("../middlewares/checkRole.js");

/**
 * @swagger
 * tags:
 *   name: CostCategory
 *   description: Maliyet kategorisi yönetimi
 */

/**
 * @swagger
 * /cost-category:
 *   post:
 *     summary: Yeni bir maliyet kategorisi oluşturur
 *     tags: [CostCategory]
 *     security:
 *       - bearerAuth: []
 *     requestBody:
 *       required: true
 *       content:
 *         application/json:
 *           schema:
 *             type: object
 *             properties:
 *               name:
 *                 type: string
 *                 example: Beton
 *               description:
 *                 type: string
 *                 example: Duvar ve kalıplara döküm işlemi için
```

```
 *             isGlobal:
 *               type: boolean
 *               example: false
 *             siteId:
 *               type: string
 *               example: 68c1df3c93e7321cce38cb81
 *     responses:
 *       201:
 *         description: Cost category başarıyla oluşturuldu
 *         content:
 *           application/json:
 *             schema:
 *               type: object
 *               properties:
 *                 message:
 *                   type: string
 *                   example: Cost category başarıyla oluşturuldu.
 *                 costCategory:
 *                   $ref: '#/components/schemas/CostCategory'
 *       400:
 *         description: Eksik veya hatalı alanlar
 *       401:
 *         description: Kullanıcı doğrulanamadı
 *       403:
 *         description: Bu işlem için yetkiniz yok
 *       404:
 *         description: Site bulunamadı
 *       500:
 *         description: Sunucu hatası
 */

/**
 * @swagger
 * /cost-category:
 *   get:
 *     summary: Tüm maliyet kategorilerini listeler
```

```
 *    tags: [CostCategory]
 *    security:
 *      - bearerAuth: []
 *    responses:
 *      200:
 *        description: Cost categories başarıyla listelendi
 *        content:
 *          application/json:
 *            schema:
 *              type: object
 *              properties:
 *                costCategoriesCount:
 *                  type: integer
 *                  example: 3
 *                message:
 *                  type: string
 *                  example: Cost categories başarıyla listelendi.
 *                costCategories:
 *                  type: array
 *                  items:
 *                    $ref: '#/components/schemas/CostCategory'
 *      401:
 *        description: Kullanıcı doğrulanamadı
 *      500:
 *        description: Sunucu hatası
 */

/**
 * @swagger
 * /cost-category/{siteId}:
 *   get:
 *     summary: Belirli bir şantiyeye ait maliyet kategorilerini listeler
 *     tags: [CostCategory]
 *     security:
 *       - bearerAuth: []
 *     parameters:
```

```
 *      - in: path
 *        name: siteId
 *        required: true
 *        schema:
 *          type: string
 *        description: Şantiyenin ID'si
 *        example: 68c1df3c93e7321cce38cb81
 *    responses:
 *      200:
 *        description: Cost categories başarıyla listelendi
 *        content:
 *          application/json:
 *            schema:
 *              type: object
 *              properties:
 *                costCategoriesCount:
 *                  type: integer
 *                  example: 2
 *                message:
 *                  type: string
 *                  example: Cost categories başarıyla listelendi.
 *                costCategories:
 *                  type: array
 *                  items:
 *                    $ref: '#/components/schemas/CostCategory'
 *      401:
 *        description: Kullanıcı doğrulanamadı
 *      404:
 *        description: Bu şantiyeye ait hiç cost category bulunamadı.
 *      500:
 *        description: Sunucu hatası
 */

/**
 * @swagger
 * /cost-category/{id}:
```

```
*   put:
*     summary: Bir maliyet kategorisini günceller
*     tags: [CostCategory]
*     security:
*       - bearerAuth: []
*     parameters:
*       - in: path
*         name: id
*         required: true
*         schema:
*           type: string
*         description: Güncellenecek cost category ID'si
*     requestBody:
*       required: true
*       content:
*         application/json:
*           schema:
*             type: object
*             properties:
*               name:
*                 type: string
*                 example: Beton
*               description:
*                 type: string
*                 example: Güncellenmiş açıklama
*               isGlobal:
*                 type: boolean
*                 example: false
*               siteId:
*                 type: string
*                 example: 68c1df3c93e7321cce38cb81
*     responses:
*       200:
*         description: Cost category başarıyla güncellendi
*         content:
*           application/json:
```

```
 *           schema:
 *             type: object
 *             properties:
 *               message:
 *                 type: string
 *                 example: Cost category başarıyla güncellendi.
 *               costCategory:
 *                 $ref: '#/components/schemas/CostCategory'
 *       401:
 *         description: Kullanıcı doğrulanamadı
 *       404:
 *         description: Cost category bulunamadı.
 *       500:
 *         description: Sunucu hatası
 */

/**
 * @swagger
 * /cost-category/{id}:
 *   delete:
 *     summary: Bir maliyet kategorisini siler
 *     tags: [CostCategory]
 *     security:
 *       - bearerAuth: []
 *     parameters:
 *       - in: path
 *         name: id
 *         required: true
 *         schema:
 *           type: string
 *         description: Silinecek cost category ID'si
 *     responses:
 *       200:
 *         description: Cost category başarıyla silindi
 *         content:
 *           application/json:
```

```
 *           schema:
 *             type: object
 *             properties:
 *               message:
 *                 type: string
 *                 example: Cost category başarıyla silindi.
 *               costCategory:
 *                 $ref: '#/components/schemas/CostCategory'
 *       401:
 *         description: Kullanıcı doğrulanamadı
 *       404:
 *         description: Cost category bulunamadı.
 *       500:
 *         description: Sunucu hatası
 */

/**
 * @swagger
 * components:
 *   schemas:
 *     CostCategory:
 *       type: object
 *       properties:
 *         _id:
 *           type: string
 *           example: 68c1df3c93e7321cce38cb81
 *         name:
 *           type: string
 *           example: Beton
 *         description:
 *           type: string
 *           example: Duvar ve kalıplara döküm işlemi için
 *         isGlobal:
 *           type: boolean
 *           example: false
 *         siteId:
```

```
 *         type: string
 *         example: 68c1df3c93e7321cce38cb81
 *       createdBy:
 *         type: string
 *         example: 64d4e7f2b2c1a2b3c4d5e6f8
 *       createdAt:
 *         type: string
 *         format: date-time
 *       updatedAt:
 *         type: string
 *         format: date-time
 */

router.get("/", verifyAccessToken, getCostCategories);
router.get("/:siteId", verifyAccessToken, getCostCategoriesBySiteId);
router.post("/", verifyAccessToken, checkRole("admin"), createCostCategory);
router.delete(
  "/:id",
  verifyAccessToken,
  checkRole("admin"),
  deleteCostCategory
);
router.put("/:id", verifyAccessToken, checkRole("admin"), updateCostCategor
y);

module.exports = router;
```

CostCategory ControllerRoute'in Server.js'e eklenmesi

```
// packages
const express = require("express");
const cors = require("cors");
const dotenv = require("dotenv");
const cookieParser = require("cookie-parser");
const swaggerUi = require("swagger-ui-express");
```

```
// routes import
const authRoute = require("./routes/authRoute.js");
const siteRoute = require("./routes/siteRoute.js");
const costCategoryRoute = require("./routes/costCategoryRoute.js");

const app = express();
dotenv.config();
app.use(
  cors({
    origin: "http://localhost:3000",
    credentials: true,
  })
);
app.use(express.json());
app.use(cookieParser());

// config import
const swaggerSpec = require("./config/swagger.config.js");
const connectDatabase = require("./config/databaseConfig");

app.get("/", (req, res) => {
  res.send("Welcome to the API!");
});

// Routes
app.use("/auth", authRoute);
app.use("/site", siteRoute);
app.use("/cost-category", costCategoryRoute);

// Swagger
app.use("/api-docs", swaggerUi.serve, swaggerUi.setup(swaggerSpec));

connectDatabase().then(() => {
  app.listen(process.env.PORT, () => {
    console.log(`Server is running on port ${process.env.PORT}`);
```

```
  });
});
```

Cost Model’in Oluşturulması

```
const mongoose = require("mongoose");

const CostSchema = new mongoose.Schema(
  {
    title: {
      type: String,
      required: true,
      trim: true,
    },
    description: {
      type: String,
      trim: true,
    },
    unit: {
      type: String,
      enum: [
        "adet",
        "kg",
        "litre",
        "metre",
        "santim",
        "koli",
        "paket",
        "diger",
      ],
      required: true,
      default: "adet",
    },
    quantity: { type: Number, required: true, default: 1 },
    unitPrice: { type: Number, required: true },
```

```
// Ayrı ayrı tutulan tutarlar
netAmount: { type: Number }, // Vergisiz toplam
taxRate: { type: Number, default: 0 }, // % KDV
taxAmount: { type: Number }, // Vergi tutarı
grossAmount: { type: Number }, // Vergili toplam

moneyType: {
  type: String,
  enum: [
    "TRY",
    "USD",
    "EUR",
    "GBP",
    "JPY",
    "CNY",
    "AUD",
    "CAD",
    "CHF",
    "SEK",
    "NOK",
    "DKK",
  ],
  required: true,
  default: "TRY",
},
costCategory: {
  type: mongoose.Schema.Types.ObjectId,
  ref: "CostCategory",
  required: true,
},
siteId: {
  type: mongoose.Schema.Types.ObjectId,
  ref: "Site",
  required: true,
},
```

```
    createdBy: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "User",
      required: true,
    },
    fileUrl: {
      type: String,
      trim: true,
    },
  },
  { timestamps: true }
);

// Pre-save hook ile otomatik hesaplama
CostSchema.pre("save", function (next) {
  this.netAmount = this.quantity * this.unitPrice; // Vergisiz toplam
  this.taxAmount = this.netAmount * (this.taxRate / 100); // Vergi tutarı
  this.grossAmount = this.netAmount + this.taxAmount; // Vergili toplam
  next();
});

module.exports = mongoose.model("Cost", CostSchema);
```

Bir maliyet gireceğimiz zaman öncelikle, Maliyet kategorisinin oluşmuş olması gerekmektedir. Bu detayları göz önünde bulundurarak maliyet için uygun bir model yapısı oluşturduk. Şimdi Temel olarak bu maliyet'e uygun CRUD işlemlerini yapalım.

Cost Controller'in Yazılması

Cost (Maliyet / Gider) için bütün servislerimizi yazalım

```
const mongoose = require("mongoose");
const Cost = require("../model/costModel.js");
const CostCategory = require("../model/costCategory.js");
const Site = require("../model/site.js");
```

```
const User = require("../model/user.js");

const createCost = async (req, res) => {
 try {
  const {
   title,
   description,
   unit,
   quantity,
   unitPrice,
   taxRate,
   moneyType,
   costCategory,
   siteId,

   fileUrl,
  } = req.body;

  // validatör ile gerekli alanları kontrol et || sonra yapılacak
  if (
   !title ||
   !unit ||
   !quantity ||
   !unitPrice ||
   !moneyType ||
   !costCategory ||
   !siteId
  ) {
   return res.status(400).json({
    message: "Lütfen tüm gerekli alanları doldurun.",
   });
  }

  // Şantiye var mı kontrol et
  const site = await Site.findById(siteId);
  if (!site) {
```

```
    return res.status(404).json({
      message: "Belirtilen şantiye bulunamadı.",
    });
  }
  // Kullanıcı var mı kontrol et

  const userId = req.user.userId;
  const user = await User.findById(userId);
  if (!user) {
    return res
      .status(404)
      .json({ message: "Kullanıcı bulunamadı. Lütfen tekrar giriş yapın." });
  }
  const createdBy = new mongoose.Types.ObjectId(userId);

  // Maliyet kategorisi var mı kontrol et
  const category = await CostCategory.findById(costCategory);
  if (!category) {
    return res.status(404).json({
      message: "Belirtilen maliyet kategorisi bulunamadı.",
    });
  }
  // Yeni maliyet oluştur

  const newCost = await Cost.create({
    title,
    description,
    unit,
    quantity,
    unitPrice,
    taxRate,
    moneyType,
    costCategory,
    siteId,
    createdBy,
    fileUrl,
```

```
    });

    return res.status(201).json({
      message: "Maliyet başarıyla oluşturuldu.",
      cost: newCost,
    });
  } catch (error) {
    return res.status(500).json({
      message: "Gider oluşturulurken bir hata oluştu.",
      error: error.message,
    });
  }
};

const getCostsBySiteId = async (req, res) => {
  try {
    const { siteId } = req.params;
    // Şantiye var mı kontrol et
    const site = await Site.findById(siteId);
    if (!site) {
      return res.status(404).json({
        message: "Belirtilen şantiye bulunamadı.",
      });
    }
    // Cost'un hepsini al, içerisindeki costCategory alanını populate et
    const costs = await Cost.find({ siteId: siteId }).populate(
      "costCategory",
      "name description"
    );
    return res.status(200).json({
      message: "Giderler başarıyla alındı.",
      costs: costs,
    });
  } catch (error) {
    return res.status(500).json({
      message: "Giderler alınırken bir hata oluştu.",
```

```
      error: error.message,
    });
  }
};

const updateCostById = async (req, res) => {
  try {
    const { costId } = req.params;
    const updateData = req.body;

    const updatedCost = await Cost.findByIdAndUpdate(costId, updateData, {
      new: true,
      runValidators: true, // ← enum, min/max vs. çalışır
      context: "query", // ← bazı validator'lar için gerekli
    });

    if (!updatedCost) {
      return res.status(404).json({ message: "Gider bulunamadı." });
    }

    return res.status(200).json({
      message: "Gider başarıyla güncellendi.",
      cost: updatedCost,
    });
  } catch (error) {
    return res.status(500).json({
      message: "Gider güncellenirken bir hata oluştu.",
      error: error.message,
    });
  }
};

const deleteCostById = async (req,res) => {
  try {
    const { costId } = req.params;
    const deletedCost = await Cost.findByIdAndDelete(costId);
```

```
    if (!deletedCost) {
      return res.status(404).json({
        message: "Gider bulunamadı.",
      });
    }
    return res.status(200).json({
      message: "Gider başarıyla silindi.",
      cost: deletedCost,
    });
  } catch (error) {
    return res.status(500).json({
      message: "Gider silinirken bir hata oluştu.",
      error: error.message,
    });
  }
}

module.exports = {
  createCost,
  getCostsBySiteId,
  updateCostById,
  deleteCostById
};
```

Cost Route'nin yazılması

Yapmış oldugumuz fonksiyonları ilgili url'lere atayalım, swagger için uygun alanları olusturalım.

```
const express = require('express');
const router = express.Router();

const { createCost,getCostsBySiteId,updateCostById,deleteCostById } = requi
re('../controller/costController.js');
const { verifyAccessToken } = require('../middlewares/authMiddleware.js');
const { checkRole } = require('../middlewares/checkRole.js');
```

```
/**
 * @swagger
 * tags:
 *   name: Cost
 *   description: Gider yönetimi
 */

/**
 * @swagger
 * /cost:
 *   post:
 *     summary: Yeni bir gider oluşturur
 *     tags: [Cost]
 *     security:
 *       - bearerAuth: []
 *     requestBody:
 *       required: true
 *       content:
 *         application/json:
 *           schema:
 *             type: object
 *             required:
 *               - title
 *               - unit
 *               - quantity
 *               - unitPrice
 *               - moneyType
 *               - costCategory
 *               - siteId
 *             properties:
 *               title:
 *                 type: string
 *                 example: C25 Beton Dökümü
 *               description:
 *                 type: string
```

```
*               example: Temel ve perde duvarlar için beton dökümü
*             unit:
*               type: string
*               example: m3
*             quantity:
*               type: number
*               example: 12.5
*             unitPrice:
*               type: number
*               example: 1500
*             taxRate:
*               type: number
*               example: 20
*             moneyType:
*               type: string
*               example: TRY
*             costCategory:
*               type: string
*               example: 68c1f417cc6b6b534764747d
*             siteId:
*               type: string
*               example: 6898ec00ca777605c79a7106
*             fileUrl:
*               type: string
*               example: https://example.com/uploads/beton-faturasi.pdf
*     responses:
*       201:
*         description: Gider başarıyla oluşturuldu
*         content:
*           application/json:
*             schema:
*               type: object
*               properties:
*                 message:
*                   type: string
*                   example: Maliyet başarıyla oluşturuldu.
```

```
 *               cost:
 *                 $ref: '#/components/schemas/Cost'
 *       400:
 *         description: Eksik veya hatalı alanlar
 *       401:
 *         description: Kullanıcı doğrulanamadı
 *       403:
 *         description: Bu işlem için yetkiniz yok
 *       404:
 *         description: Şantiye veya kategori bulunamadı
 *       500:
 *         description: Sunucu hatası
 */

/**
 * @swagger
 * /cost/{siteId}:
 *   get:
 *     summary: Belirli bir şantiyeye ait giderleri listeler
 *     tags: [Cost]
 *     security:
 *       - bearerAuth: []
 *     parameters:
 *       - in: path
 *         name: siteId
 *         required: true
 *         schema:
 *           type: string
 *         description: Şantiyenin ID'si
 *         example: 6898ec00ca777605c79a7106
 *     responses:
 *       200:
 *         description: Giderler başarıyla alındı
 *         content:
 *           application/json:
 *             schema:
```

```
 *             type: object
 *             properties:
 *               message:
 *                 type: string
 *                 example: Giderler başarıyla alındı.
 *               costs:
 *                 type: array
 *                 items:
 *                   $ref: '#/components/schemas/Cost'
 *       401:
 *         description: Kullanıcı doğrulanamadı
 *       403:
 *         description: Bu işlem için yetkiniz yok
 *       404:
 *         description: Şantiye bulunamadı
 *       500:
 *         description: Sunucu hatası
 */

/**
 * @swagger
 * /cost/{costId}:
 *   put:
 *     summary: Bir gideri günceller
 *     tags: [Cost]
 *     security:
 *       - bearerAuth: []
 *     parameters:
 *       - in: path
 *         name: costId
 *         required: true
 *         schema:
 *           type: string
 *         description: Güncellenecek gider ID'si
 *     requestBody:
 *       required: true
```

```
*     content:
*       application/json:
*         schema:
*           type: object
*           properties:
*             title:
*               type: string
*               example: C25 Beton Dökümü (Güncellendi)
*             description:
*               type: string
*               example: Güncellenmiş açıklama
*             unit:
*               type: string
*               example: m3
*             quantity:
*               type: number
*               example: 15
*             unitPrice:
*               type: number
*               example: 1600
*             taxRate:
*               type: number
*               example: 20
*             moneyType:
*               type: string
*               example: TRY
*             costCategory:
*               type: string
*               example: 68c1f417cc6b6b534764747d
*             siteId:
*               type: string
*               example: 6898ec00ca777605c79a7106
*             fileUrl:
*               type: string
*               example: https://example.com/uploads/beton-faturasi-v2.pdf
*     responses:
```

```
 *      200:
 *        description: Gider başarıyla güncellendi
 *        content:
 *          application/json:
 *            schema:
 *              type: object
 *              properties:
 *                message:
 *                  type: string
 *                  example: Gider başarıyla güncellendi.
 *                cost:
 *                  $ref: '#/components/schemas/Cost'
 *      401:
 *        description: Kullanıcı doğrulanamadı
 *      403:
 *        description: Bu işlem için yetkiniz yok
 *      404:
 *        description: Gider bulunamadı
 *      500:
 *        description: Sunucu hatası
 */

/**
 * @swagger
 * /cost/{costId}:
 *   delete:
 *     summary: Bir gideri siler
 *     tags: [Cost]
 *     security:
 *       - bearerAuth: []
 *     parameters:
 *       - in: path
 *         name: costId
 *         required: true
 *         schema:
 *           type: string
```

```
 *       description: Silinecek gider ID'si
 *     responses:
 *       200:
 *         description: Gider başarıyla silindi
 *         content:
 *           application/json:
 *             schema:
 *               type: object
 *               properties:
 *                 message:
 *                   type: string
 *                   example: Gider başarıyla silindi.
 *                 cost:
 *                   $ref: '#/components/schemas/Cost'
 *       401:
 *         description: Kullanıcı doğrulanamadı
 *       403:
 *         description: Bu işlem için yetkiniz yok
 *       404:
 *         description: Gider bulunamadı
 *       500:
 *         description: Sunucu hatası
 */

/**
 * @swagger
 * components:
 *   schemas:
 *     Cost:
 *       type: object
 *       properties:
 *         _id:
 *           type: string
 *           example: 68d0f9c12a15e3364966d86c
 *         title:
 *           type: string
```

```
*           example: C25 Beton Dökümü
*         description:
*           type: string
*           example: Temel ve perde duvarlar için beton dökümü
*         unit:
*           type: string
*           example: m3
*         quantity:
*           type: number
*           example: 12.5
*         unitPrice:
*           type: number
*           example: 1500
*         taxRate:
*           type: number
*           example: 20
*         moneyType:
*           type: string
*           example: TRY
*         costCategory:
*           type: string
*           example: 68c1f417cc6b6b534764747d
*         siteId:
*           type: string
*           example: 6898ec00ca777605c79a7106
*         createdBy:
*           type: string
*           example: 64d4e7f2b2c1a2b3c4d5e6f8
*         fileUrl:
*           type: string
*           example: https://example.com/uploads/beton-faturasi.pdf
*         createdAt:
*           type: string
*           format: date-time
*         updatedAt:
*           type: string
```

```
 *         format: date-time
 */

router.post('/', verifyAccessToken, checkRole('admin'), createCost);
router.get('/:siteId', verifyAccessToken, checkRole('admin'), getCostsBySiteI
d);
router.put('/:costId', verifyAccessToken, checkRole('admin'), updateCostById);
router.delete('/:costId', verifyAccessToken, checkRole('admin'), deleteCostBy
d);

module.exports = router;
```

Morgan Kütüphanesinin Kurulması

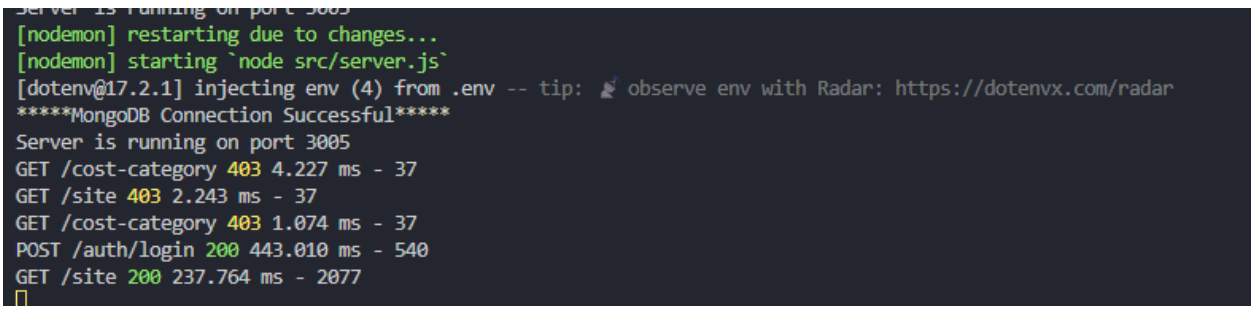
Morgan ile birlikte server'imize atılan istekleri, dönüş süreleri gibi bilgileri terminale yazdırarak, görüntülemeyi sağlar.

```
npm i morgan
```

Terminal üzerinden indirdikten sonra, server.js üzerine app.use ile morgan'ı kullanıma açacağız

```
//server.js
const morgan = require('morgan')

app.use(morgan("dev"));
```

Artık atılan isteklerin hepsini terminal üzerinde canlı olarak görebileceğiz.



Express Validatör Kurulması

Express Validatör ile birlikte projemizdeki atılan isteklerdeki parametrelerin doğruluğunu kontrol altına alacağız. Özellikle body'den gelen verileri kontrol edeceğiz. Controller dosyalarının içerisi daha da sadeleşecek.

```
npm i express-validator
```

ValidateMiddleware 'in yazılması

src/middlewares/validateMiddleware.js dosyasını oluşturalım. Bu bir middleware olacak. Bu sayede route'lar üzerinde role kontrolü yapar gibi validasyon kontrolü yapacağız.

```
const { validationResult } = require("express-validator");

const validate = (req, res, next) => {
  const errors = validationResult(req);
  // eğer errors boş değilse, bloğa gir
  if (!errors.isEmpty()) {
    return res.status(400).json({
      status: "Error",
      message: "Validation Error",
      errors: errors.array().map((error) => ({
        field: error.path,
        message: error.msg,
      })),
    });
  }
  next();
};

module.exports = validate;
```

validate diye bir fonksiyon acıp req,res,next özellikleriyle aşağı geçiyoruz. validationResult özelliği express validatör'e özel bir method. yakalanan hataları

tutuyor. tüm hataları errors'a atıyoruz ve eğer errors boş değilse diyerek bir return res dönüyoruz. Döndüğümüz response içerini hataların gözükeceği biçimde özelleştiriyoruz.

userValidator.js dosyasının yazılması

user'in login/register işlemleri için doğrulama ayarlarını vereceğimiz kısım. Biz bu bilgileri body'den alıyoruz o yüzden bütün ilerleyiş ona göre olacak.

```
//userValidator.js
const { body } = require("express-validator");

const userValidationRules = {
  email: body("email")
    .trim()
    .isEmail()
    .withMessage("Geçerli bir email adresi giriniz.")
    .normalizeEmail()
    .notEmpty()
    .withMessage("Email alanı boş bırakılamaz.")
    .toLowerCase(),

  password: body("password")
    .trim()
    .isLength({ min: 6, max: 20 })
    .withMessage("Şifre en az 6 karakter - en fazla 20 karakterolmalıdır.")
    .notEmpty()
    .withMessage("Şifre alanı boş bırakılamaz.")
    .matches(
      /^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-z\d@$!%*?&]{6,}$/
    )
    .withMessage(
      "Parola en az bir büyük harf, bir küçük harf, bir rakam ve bir özel karakter i
çermelidir."
    ),
```

```
    firstName: body("firstName")
    .trim()
    .notEmpty()
    .withMessage("İsim alanı boş bırakılamaz.")
    .isLength({ min: 2, max: 30 })
    .withMessage("İsim en az 2 karakter - en fazla 30 karakter olmalıdır.")
    .matches(/^[A-Za-zğüşöçıİĞÜŞÖÇ\s]+$/)
    .withMessage("İsim sadece harf ve boşluk içerebilir."),

    lastName: body("lastName")
    .trim()
    .notEmpty()
    .withMessage("Soyisim alanı boş bırakılamaz.")
    .isLength({ min: 2, max: 30 })
    .withMessage("Soyisim en az 2 karakter - en fazla 30 karakter olmalıdır.")
    .matches(/^[A-Za-zğüşöçıİĞÜŞÖÇ\s]+$/)
    .withMessage("Soyisim sadece harf ve boşluk içerebilir."),

};

module.exports = userValidationRules;
```

body'den gelen alanlara body("bodyUzerindeKarsilanacakAlanAdi") şeklinde belirterek ilgili kuralları yazıyoruz.

Validator için index.js dosyasının yazılması

Validatör'ün ana dizini index.js üzerinde hangi validasyon nerede çalışacak diye belirliyoruz.

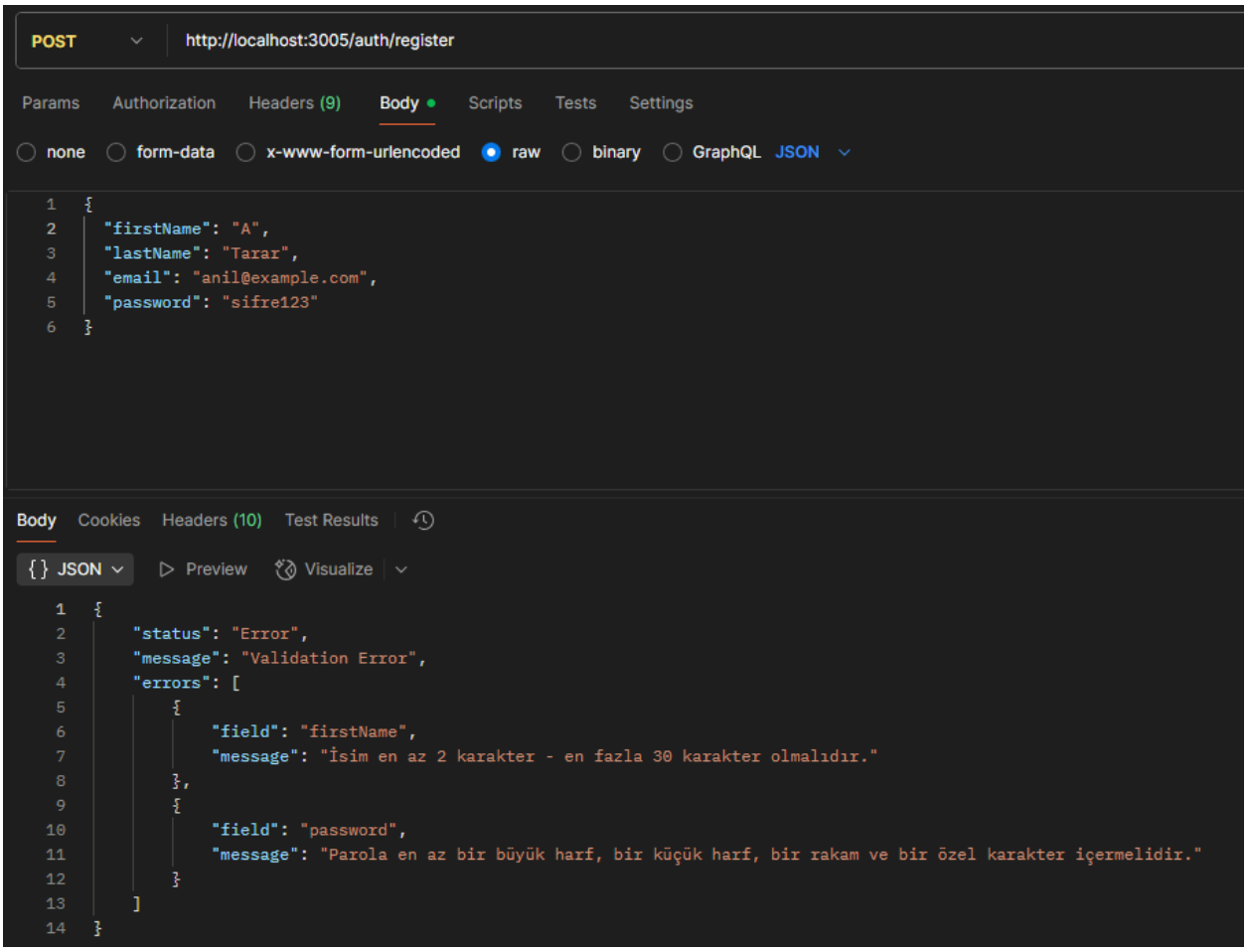
```
const validate = require("../middlewares/validateMiddleware.js");
const userValidator = require("./userValidator.js");

// Register Validasyon Kuralları
const validateRegistration = [
  userValidator.firstName,
```

```
  userValidator.lastName,
  userValidator.email,
  userValidator.password,
  validate,
];

// Login Validasyon Kuralları
const validateLogin = [userValidator.email, userValidator.password, validate];

module.exports = {
  validateRegistration,
  validateLogin,
};
```

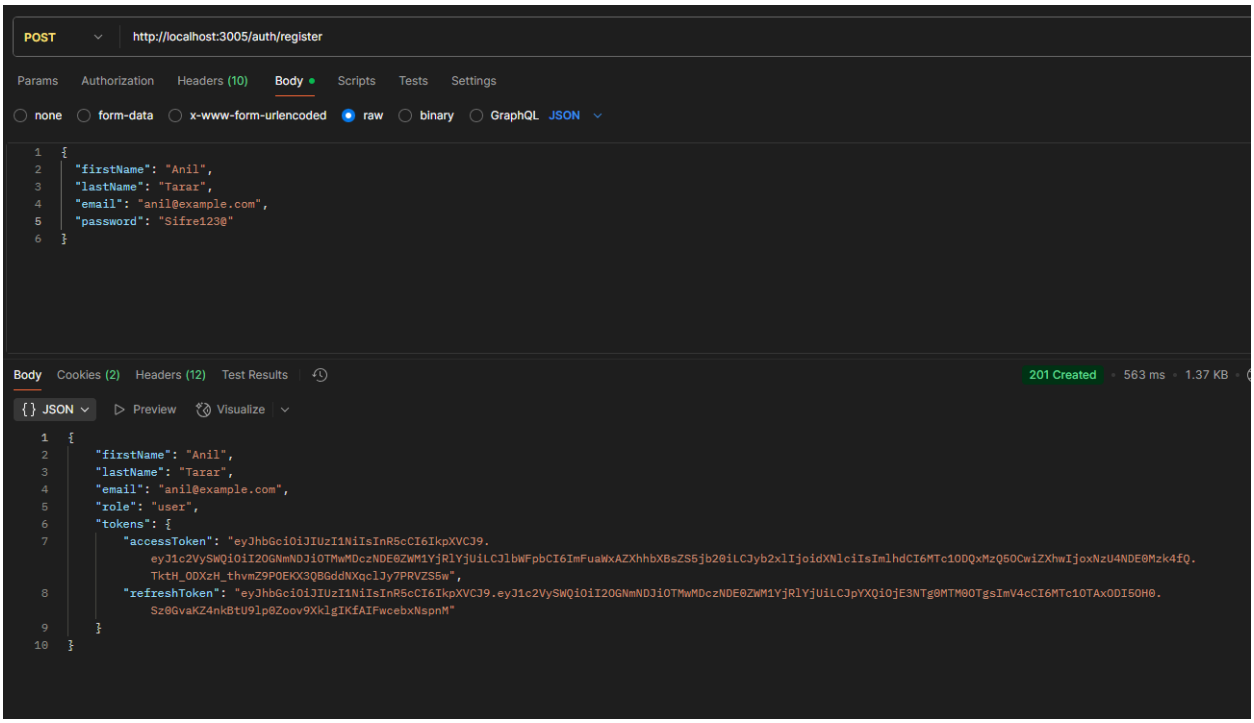
Register ve Login'e özel olacak şekilde hangi kuralları içereceğini belirledik.

Validator işlemlerini auth route içerisinde eklenmesi

```
router.post("/register", validateRegistration, register);
router.post("/login", validateLogin, login);
```



Örneğin register işleminde eksik olan bölümler bu şekilde response veriyor.



siteValidator.js dosyasının açılması

```
const { body,param } = require("express-validator");

const siteValidationRules = {
 name: body("name")
   .trim()
   .notEmpty()
   .withMessage("Şantiye adı boş bırakılamaz.")
   .isLength({ min: 2, max: 50 })
   .withMessage(
     "Şantiye adı en az 2 karakter - en fazla 50 karakter olmalıdır."
   ),

 location: body("location")
   .trim()
   .notEmpty()
   .withMessage("Şantiye konumu boş bırakılamaz.")
   .isLength({ min: 2, max: 100 })
   .withMessage(
```

```
    "Şantiye konumu en az 2 karakter - en fazla 100 karakter olmalıdır."
  ),

budget: body("budget")
  .notEmpty()
  .withMessage("Bütçe alanı boş bırakılamaz.")
  .isFloat({ gt: 0 })
  .withMessage("Bütçe pozitif bir sayı olmalıdır."),

startDate: body("startDate")
  .notEmpty()
  .withMessage("Başlangıç tarihi boş bırakılamaz.")
  .isISO8601()
  .withMessage(
    "Geçerli bir başlangıç tarihi giriniz (YYYY-MM-DD formatında)."
  ),

endDate: body("endDate")
  .notEmpty()
  .withMessage("Bitiş tarihi boş bırakılamaz.")
  .isISO8601()
  .withMessage("Geçerli bir bitiş tarihi giriniz (YYYY-MM-DD formatında).")
  .custom((value, { req }) => {
    if (new Date(value) <= new Date(req.body.startDate)) {
      throw new Error("Bitiş tarihi, başlangıç tarihinden sonra olmalıdır.");
    }
    return true;
  }),

assignedUsers: body("assignedUsers")
  .optional()
  .isArray()
  .withMessage("Atanan kullanıcılar alanı bir dizi olmalıdır."),
  // assignedUsers.* yaparak dizinin her bir elemanını kontrol ediyoruz
assignedUsersIds: body("assignedUsers.*")
  .optional()
```

```
    .isMongoId()
    .withMessage("Atanan kullanıcı ID'leri geçerli bir ObjectId olmalıdır."),

  siteId: param("siteId")
    .isMongoId()
    .withMessage("Geçerli bir şantiye ID'si giriniz.")
};

module.exports = siteValidationRules;
```

Body'de veya param'da (url'den alınacak data) kullanılacak alanlar için kurallar yazıyoruz. Bu kuralları tek tek validator index.js'de gruplandıracağız.

```
const validate = require("../middlewares/validateMiddleware.js");
const userValidator = require("./userValidator.js");
const siteValidator = require("./siteValidator.js");

// **AUTH VALIDATION RULES**
// Register Validasyon Kuralları
const validateRegistration = [
  userValidator.firstName,
  userValidator.lastName,
  userValidator.email,
  userValidator.password,
  validate,
];

// Login Validasyon Kuralları
const validateLogin = [userValidator.email, userValidator.password, validate];

// **SITE VALIDATION RULES**

const validateSiteCreation = [
  siteValidator.name,
  siteValidator.location,
  siteValidator.budget,
```

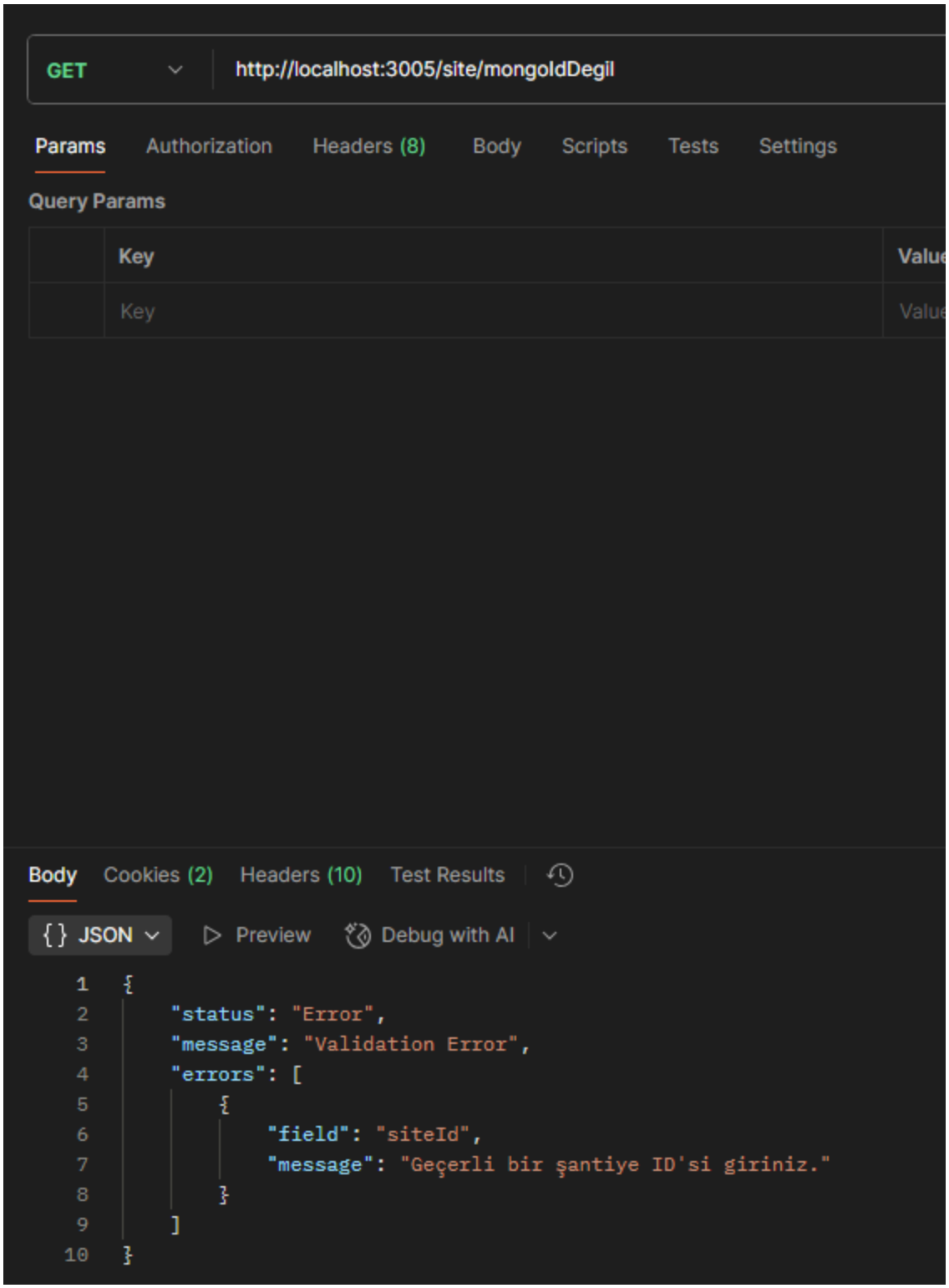
```
  siteValidator.startDate,
  siteValidator.endDate,
  siteValidator.assignedUsers,
  validate,
];
const validateGetSiteById = [siteValidator.siteId, validate];

module.exports = {
  validateRegistration,
  validateLogin,
  validateSiteCreation,
  validateGetSiteById,
};
```

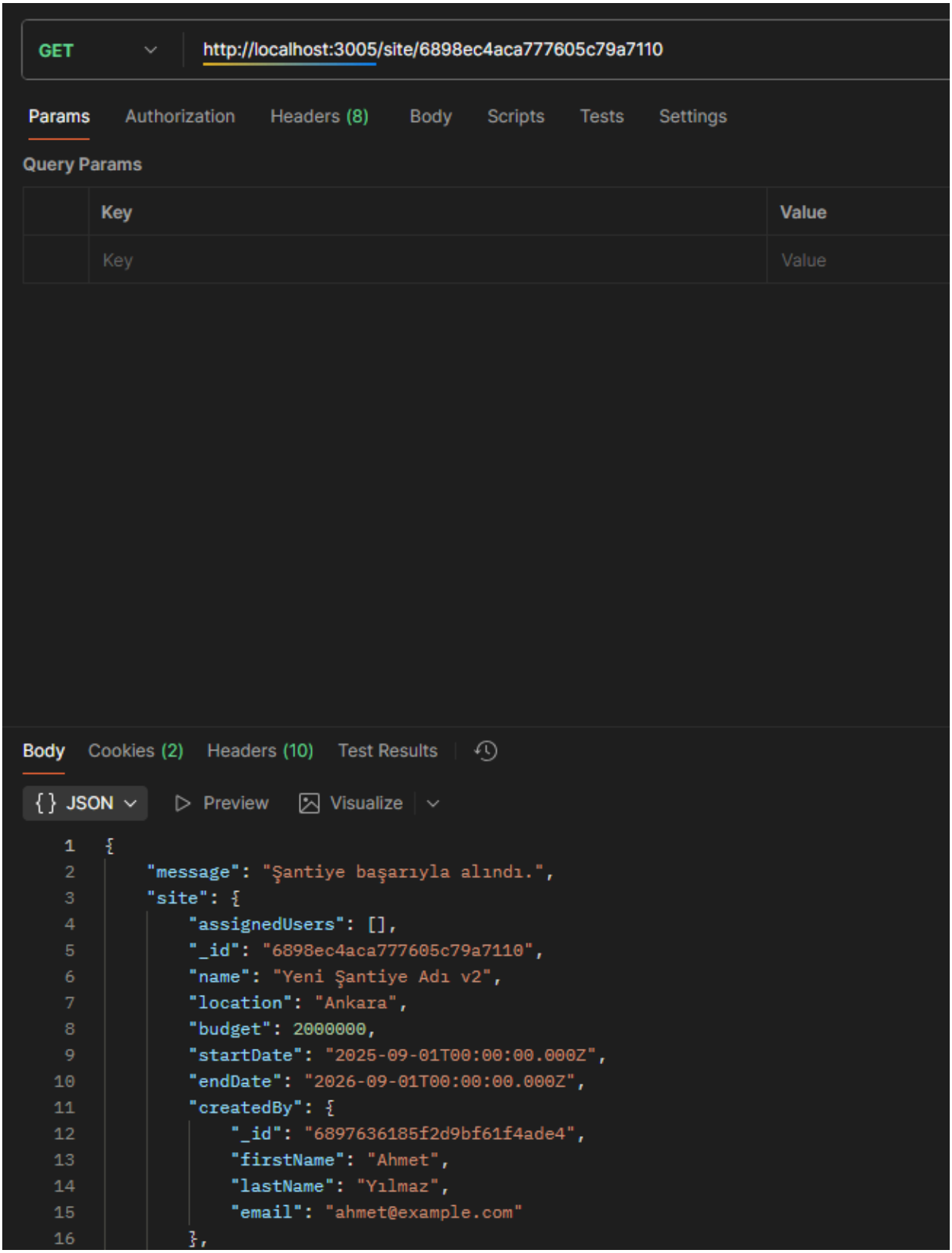
Site Creation için ve getSiteById için olusturdugum alanı export ediyorum. İlgili route'da controller'dan önce middleware olarak calıstıracağım.

```
router.post(
  "/",
  verifyAccessToken,
  checkRole("admin"),
  validateSiteCreation,
  createSite
);
router.get("/", verifyAccessToken, checkRole("admin"), getAllSites);
router.get(
  "/:siteId",
  verifyAccessToken,
  checkRole("admin"),
  validateGetSiteById,
  getSiteById
);
router.delete(
  "/:siteId",
  verifyAccessToken,
  checkRole("admin"),
```

```
  deleteSiteById
);
router.put("/:siteId", verifyAccessToken, checkRole("admin"), updateSiteById);
module.exports = router;
```

kırmızı ile işaretlediğimiz alanlar ilgili kuralları kontrol ediyor. örnek ekran görüntüleri.


GET
http://localhost:3005/site/mongoldDegil
Params
Authorization
Headers (8)
Body
Scripts
Tests
Settings
Query Params
Key
Value
Key
Value
Body
Cookies (2)
Headers (10)
Test Results
JSON
Preview
Debug with AI
{
"status": "Error",
"message": "Validation Error",
"errors": [
{
"field": "siteId",
"message": "Geçerli bir şantiye ID'si giriniz."
}
]
}


GET
http://localhost:3005/site/6898ec4aca777605c79a7110
Params
Authorization
Headers (8)
Body
Scripts
Tests
Settings
Query Params
Key
Value
Key
Value
Body
Cookies (2)
Headers (10)
Test Results
{ } JSON
Preview
Visualize
{
"message": "Şantiye başarıyla alındı.",
"site": {
"assignedUsers": [],
"_id": "6898ec4aca777605c79a7110",
"name": "Yeni Şantiye Adı v2",
"location": "Ankara",
"budget": 2000000,
"startDate": "2025-09-01T00:00:00.000Z",
"endDate": "2026-09-01T00:00:00.000Z",
"createdBy": {
"_id": "6897636185f2d9bf61f4ade4",
"firstName": "Ahmet",
"lastName": "Yılmaz",
"email": "ahmet@example.com"
},

Cost Validator'ün kurulması

Belirli bir maliyet kategorise ait, maliyet'i eklerken de belli kurallarımız olmalı. Buradaki en büyük sorun şu ; oluşturma işlemleri için zorunlu, güncelleme işlemleri için opsiyonel alanlar var. Aynı alanı 2 sefer tanımlamak gerekiyor. Bunu en temiz şekilde yazabilmenin bir yolunu bulmamız gerek. Bu yol da createRule adında bir fonksiyondan geçiyor.

```
//costValidator.js
const { body, param } = require("express-validator");

const createRule = (fieldName, isOptional = false) => {
  const rules = {
    title: (optional) => {
      let chain = body("title").trim();
      if (!optional) chain = chain.notEmpty().withMessage("Maliyet başlığı boş bı
rakılamaz.");
      else chain = chain.optional();
      return chain.isLength({ min: 2, max: 100 })
        .withMessage("Maliyet başlığı en az 2 karakter - en fazla 100 karakter ol
malıdır.");
    },

    description: (optional) => {
      let chain = body("description").trim();
      if (optional) chain = chain.optional();
      return chain.isLength({ max: 500 })
        .withMessage("Maliyet açıklaması en fazla 500 karakter olmalıdır.");
    },

    unit: (optional) => {
      let chain = body("unit").trim();
      if (!optional) chain = chain.notEmpty().withMessage("Birim boş bırakılama
z.");
      else chain = chain.optional();
      return chain.isIn(["adet", "kg", "litre", "metre", "santim", "koli", "paket", "dig
```

```
er"])
        .withMessage("Birim sadece şu değerlerden biri olabilir: adet, kg, litre, m
etre, santim, koli, paket, diger");
    },

    quantity: (optional) => {
      let chain = body("quantity");
      if (!optional) chain = chain.notEmpty().withMessage("Miktar boş bırakılam
az.");
      else chain = chain.optional();
      return chain.isFloat({ gt: 0 }).withMessage("Miktar 0'dan büyük bir sayı ol
malıdır.");
    },

    unitPrice: (optional) => {
      let chain = body("unitPrice");
      if (!optional) chain = chain.notEmpty().withMessage("Birim fiyat boş bırakıl
amaz.");
      else chain = chain.optional();
      return chain.isFloat({ gt: 0 }).withMessage("Birim fiyat 0'dan büyük bir say
ı olmalıdır.");
    },

    taxRate: (optional) => {
      let chain = body("taxRate");
      if (!optional) chain = chain.notEmpty().withMessage("Vergi oranı boş bırakı
lamaz.");
      else chain = chain.optional();
      return chain.isFloat({ min: 0 }).withMessage("Vergi oranı 0 veya daha büyü
k bir sayı olmalıdır.");
    },

    moneyType: (optional) => {
      let chain = body("moneyType").trim();
      if (!optional) chain = chain.notEmpty().withMessage("Para birimi boş bırakı
lamaz.");
```

```
      else chain = chain.optional();
      return chain.isIn(["TRY", "USD", "EUR", "GBP", "JPY", "CNY", "AUD", "CA
D", "CHF", "SEK", "NOK", "DKK"])
        .withMessage("Para birimi geçerli bir değer olmalıdır.");
    },

    costCategory: (optional) => {
      let chain = body("costCategory");
      if (!optional) chain = chain.notEmpty().withMessage("Maliyet kategorisi bo
ş bırakılamaz.");
      else chain = chain.optional();
      return chain.isMongoId().withMessage("Geçerli bir maliyet kategorisi ID'si
giriniz.");
    },

    siteId: (optional) => {
      let chain = body("siteId");
      if (!optional) chain = chain.notEmpty().withMessage("Şantiye ID'si boş bıra
kılamaz.");
      else chain = chain.optional();
      return chain.isMongoId().withMessage("Geçerli bir şantiye ID'si giriniz.");
    },
  };

  return rules[fieldName] ? rules[fieldName](isOptional) : null;
};

const costValidationRules = {
  // Static rules
  costId: param("costId").isMongoId().withMessage("Geçerli bir maliyet/gider I
D'si giriniz."),
  siteIdParam: param("siteId").isMongoId().withMessage("Geçerli bir şantiye I
D'si giriniz."),
  fileUrl: body("fileUrl").optional().isURL().withMessage("Geçerli dosya URL'si
giriniz."),
```

```
  // Required rules
  title: createRule("title", false),
  description: createRule("description", false),
  unit: createRule("unit", false),
  quantity: createRule("quantity", false),
  unitPrice: createRule("unitPrice", false),
  taxRate: createRule("taxRate", false),
  moneyType: createRule("moneyType", false),
  costCategory: createRule("costCategory", false),
  siteId: createRule("siteId", false),

  // Optional rules
  titleOptional: createRule("title", true),
  descriptionOptional: createRule("description", true),
  unitOptional: createRule("unit", true),
  quantityOptional: createRule("quantity", true),
  unitPriceOptional: createRule("unitPrice", true),
  taxRateOptional: createRule("taxRate", true),
  moneyTypeOptional: createRule("moneyType", true),
  costCategoryOptional: createRule("costCategory", true),
  siteIdOptional: createRule("siteId", true),
};

module.exports = costValidationRules;
```

createRule ile birlikte önce fieldName'i alıyoruz (hangi alan oldugunu belirttiğimiz kısım.) sonrasında isOptional kontrolü yapıyoruz, default olarak false aliyoruz.

cost'a ait validasyonların index.js de gruplandırılması.

```
const validate = require("../middlewares/validateMiddleware.js");
const userValidator = require("./userValidator.js");
const siteValidator = require("./siteValidator.js");
const costValidationRules = require("./costValidator.js");

// **AUTH VALIDATION RULES**
```

```
// Register Validasyon Kuralları
const validateRegistration = [
  userValidator.firstName,
  userValidator.lastName,
  userValidator.email,
  userValidator.password,
  validate,
];
const validateLogin = [userValidator.email, userValidator.password, validate];

// **SITE VALIDATION RULES**

const validateSiteCreation = [
  siteValidator.name,
  siteValidator.location,
  siteValidator.budget,
  siteValidator.startDate,
  siteValidator.endDate,
  siteValidator.assignedUsers,
  siteValidator.assignedUsersIds,
  validate,
];
const validateGetSiteById = [siteValidator.siteId, validate];
const validateDeleteSiteById = [siteValidator.siteId, validate];

const validateUpdateSiteById = [
  siteValidator.siteId,
  siteValidator.nameOptional,
  siteValidator.locationOptional,
  siteValidator.budgetOptional,
  siteValidator.startDateOptional,
  siteValidator.endDateOptional,
  siteValidator.assignedUsersOptional,
  siteValidator.assignedUsersIdsOptional,
  validate,
];
```

```
// **Cost VALIDATION RULES**
const validateCostCreation = [
  costValidationRules.title,
  costValidationRules.description,
  costValidationRules.unit,
  costValidationRules.quantity,
  costValidationRules.unitPrice,
  costValidationRules.taxRate,
  costValidationRules.moneyType,
  costValidationRules.costCategory,
  costValidationRules.siteId,
  costValidationRules.fileUrl,
  validate,
];

const validateGetCostBySiteId = [costValidationRules.siteIdParam, validate];
const validateDeleteCostById = [costValidationRules.costId, validate];

const validateUpdateCostById = [
  costValidationRules.costId,
  costValidationRules.titleOptional,
  costValidationRules.descriptionOptional,
  costValidationRules.unitOptional,
  costValidationRules.quantityOptional,
  costValidationRules.unitPriceOptional,
  costValidationRules.taxRateOptional,
  costValidationRules.moneyTypeOptional,
  costValidationRules.costCategoryOptional,
  costValidationRules.siteIdOptional,
  costValidationRules.fileUrl,
  validate,
];

module.exports = {
  // Auth
```

```
  validateRegistration,
  validateLogin,
  // Site
  validateSiteCreation,
  validateGetSiteById,
  validateDeleteSiteById,
  validateUpdateSiteById,
  // Cost
  validateCostCreation,
  validateGetCostBySiteId,
  validateDeleteCostById,
  validateUpdateCostById,
};
```

Gruplanan Cost Validator'lerin costRoute'a eklenmesi

```
//costRoute.js
router.post(
  "/",
  verifyAccessToken,
  checkRole("admin"),
  validateCostCreation,
  createCost
);
router.get(
  "/:siteId",
  verifyAccessToken,
  checkRole("admin"),
  validateGetCostBySiteId,
  getCostsBySiteId
);
router.put(
  "/:costId",
  verifyAccessToken,
  checkRole("admin"),
```

```
  validateUpdateCostById,
  updateCostById
);
router.delete(
  "/:costId",
  verifyAccessToken,
  checkRole("admin"),
  validateDeleteCostById,
  deleteCostById
);
```

Multer ile beraber dosyaların eklenmesi

Multer ile beraber projemize dosya yükleme işlemlerini yapiyoruz. Öncelikle bir config dosyası ile middleware haline getiriyoruz. Sonrasında içeride gerekli ayarlamalari yapiyoruz.

```
// config / multerConfig.js
const multer = require("multer");
const path = require("path");
const fs = require("fs");

const storage = multer.diskStorage({
  destination: (req, file, cb) => {
    const siteId = req.body.siteId;
    if (!siteId) {
      return cb(new Error("Multer Error | Site ID gereklidir!"), null);
    }
    // Şantiyeye özel klasör oluşturma
    const uploadPath = path.join(__dirname, "..", "uploads", "costs", siteId);
    // klasör yapısı => uploads/costs/{siteId}
    //eğer klasör yoksa oluştur
    fs.mkdirSync(uploadPath, { recursive: true });
    cb(null, uploadPath);
  },
```

```
  filename: (req, file, cb) => {
    const ext = path.extname(file.originalname);
    // uniqueSuffix oluşturma
    const uniqueSuffix = Date.now() + "-" + Math.round(Math.random() * 1e9);
    const fileName = `cost-${uniqueSuffix}${ext}`;
    cb(null, fileName);
  },
});

const fileFilter = (req, file, cb) => {
  // Sadece belirli dosya türlerine izin ver
  const allowedMimeTypes = [
    "image/jpeg",
    "image/png",
    "image/jpg",
    "application/pdf",
  ];
  if (allowedMimeTypes.includes(file.mimetype)) {
    cb(null, true);
  } else {
    cb(
      new Error(
        "Multer Error | Geçersiz dosya türü! Sadece JPEG, PNG, JPG ve PDF dos
yalarına izin verilir."
      ),
      false
    );
  }
};

const upload = multer({
  storage: storage,
  limits: { fileSize: 15 * 1024 * 1024 }, // 15MB dosya boyutu sınırı
  fileFilter: fileFilter,
});
```

```
module.exports = upload;
```

upload'ı dısarıya export ediyoruz.

```
//route'a middleware olarak upload'ı eklemek. Swagger'dan file almak

/**
 * @swagger
 * /cost:
 *   post:
 *     summary: Yeni bir gider oluşturur
 *     tags: [Cost]
 *     security:
 *       - bearerAuth: []
 *     requestBody:
 *       required: true
 *       content:
 *         multipart/form-data:
 *           schema:
 *             type: object
 *             required:
 *               - title
 *               - unit
 *               - quantity
 *               - unitPrice
 *               - moneyType
 *               - costCategory
 *               - siteId
 *             properties:
 *               title:
 *                 type: string
 *                 example: C25 Beton Dökümü
 *               description:
 *                 type: string
 *                 example: Temel ve perde duvarlar için beton dökümü
```

```
 *             unit:
 *               type: string
 *               example: m3
 *             quantity:
 *               type: number
 *               example: 12.5
 *             unitPrice:
 *               type: number
 *               example: 1500
 *             taxRate:
 *               type: number
 *               example: 20
 *             moneyType:
 *               type: string
 *               example: TRY
 *             costCategory:
 *               type: string
 *               example: 68c1f417cc6b6b534764747d
 *             siteId:
 *               type: string
 *               example: 6898ec00ca777605c79a7106
 *             file:
 *               type: string
 *               format: binary
 *               description: Fatura veya belge dosyası (JPEG, PNG, JPG, PDF - M
ax 15MB)
 *     responses:
 *       201:
 *         description: Gider başarıyla oluşturuldu
 *       400:
 *         description: Dosya hatası veya eksik alanlar
 *       401:
 *         description: Kullanıcı doğrulanamadı
 *       403:
 *         description: Bu işlem için yetkiniz yok
 *       404:
```

```
 *       description: Şantiye veya kategori bulunamadı
 *     500:
 *       description: Sunucu hatası
 */

router.post(
  "/",
  verifyAccessToken,
  checkRole("admin"),
  upload.single("file"),
  validateCostCreation,
  createCost
);
```

Admin İşlemleri İçin userController'in yazılması

Şuana kadar sistemimizde checkRole ile beraber rol kontrolü yaptığımız bir middleware var ancak senaryosal olarak admin'in ihtiyacı olan servisler yazmamıştık. Projede altyapısı hazır olan bir özelliği kullanılabilir hale getirmek için temel user crud işlemlerini admin üzerinde kullanılabilir hale getirmek istiyorum. Bu yüzden user için bir controller yazalım.

```
const mongoose = require("mongoose");
const User = require("../model/user.js");

const getAllUsers = async (req, res) => {
  try {
    const users = await User.find().select("-password"); // Parolayı hariç tut
    return res.status(200).json({
      message: "Kullanıcılar başarıyla alındı.",
      users: users,
    });
  } catch (error) {
    return res.status(500).json({
      message: "Kullanıcılar alınırken bir hata oluştu.",
```

```
      error: error.message,
    });
  }
};
const getUserById = async (req, res) => {
    try {
        const userId = req.params.userId;
        if (!mongoose.Types.ObjectId.isValid(userId)) {
            return res.status(400).json({ message: "Geçersiz kullanıcı ID'si." });
        }
        const user = await User.findById(userId).select("-password");
        if (!user) {
            return res.status(404).json({ message: "Kullanıcı bulunamadı." });
        }
        return res.status(200).json({
            message: "Kullanıcı başarıyla alındı.",
            user: user,
        });
    } catch (error) {
        return res.status(500).json({
            message: "Kullanıcı alınırken bir hata oluştu.",
            error: error.message,
            });
    }
}

const updateUserRole = async (req, res) => {
  try {
    const { userId } = req.params;
    const { role } = req.body;
    if (!mongoose.Types.ObjectId.isValid(userId)) {
      return res.status(400).json({ message: "Geçersiz kullanıcı ID'si." });
    }
    const user = await User.findById(userId);
    if (!user) {
      return res.status(404).json({ message: "Kullanıcı bulunamadı." });
```

```
    }
    user.role = role;
    await user.save();
    return res.status(200).json({
      message: "Kullanıcı rolü başarıyla güncellendi.",
      user: user,
    });
  } catch (error) {
    return res.status(500).json({
      message: "Kullanıcı rolü güncellenirken bir hata oluştu.",
      error: error.message,
    });
  }
};

const deleteUserById = async (req, res) => {
try {
    const userId = req.params.userId;
    if (!mongoose.Types.ObjectId.isValid(userId)) {
        return res.status(400).json({ message: "Geçersiz kullanıcı ID'si." });
    }
    const user = await User.findByIdAndDelete(userId);
    if (!user) {
        return res.status(404).json({ message: "Kullanıcı bulunamadı." });
    }
    return res.status(200).json({
        message: "Kullanıcı başarıyla silindi.",
        user: user,
    });

} catch (error) {
    return res.status(500).json({
        message: "Kullanıcı silinirken bir hata oluştu.",
        error: error.message,
    });
}
```

```
}

module.exports = {
  getAllUsers,
  updateUserRole,
  getUserById,
  deleteUserById
};
```

userRoute'in yazılması

Oluşturduğumuz yeni userController'i route a açıp, gerekli middleware'leri hazırlayalım.

```
const express = require("express");
const router = express.Router();
const {
  getAllUsers,
  updateUserRole,
  getUserById,
  deleteUserById,
} = require("../controller/userController.js");
const { checkRole } = require("../middlewares/checkRole.js");
const { verifyAccessToken } = require("../middlewares/authMiddleware.js");

/**
 * @swagger
 * tags:
 *   name: User
 *   description: Kullanıcı yönetimi
 */

/**
 * @swagger
 * /user:
 *   get:
```

```
 *    summary: Tüm kullanıcıları listeler
 *    tags: [User]
 *    security:
 *      - bearerAuth: []
 *    responses:
 *      200:
 *        description: Kullanıcılar başarıyla alındı
 *        content:
 *          application/json:
 *            schema:
 *              type: object
 *              properties:
 *                message:
 *                  type: string
 *                  example: Kullanıcılar başarıyla alındı.
 *                users:
 *                  type: array
 *                  items:
 *                    $ref: '#/components/schemas/User'
 *      401:
 *        description: Kullanıcı doğrulanamadı
 *      403:
 *        description: Bu işlem için yetkiniz yok
 *      500:
 *        description: Sunucu hatası
 */

/**
 * @swagger
 * /user/{userId}:
 *   get:
 *     summary: Belirli bir kullanıcıyı getirir
 *     tags: [User]
 *     security:
 *       - bearerAuth: []
 *     parameters:
```

```
 *      - in: path
 *        name: userId
 *        required: true
 *        schema:
 *          type: string
 *        description: Kullanıcının ID'si
 *        example: 64d4e7f2b2c1a2b3c4d5e6f8
 *    responses:
 *      200:
 *        description: Kullanıcı başarıyla alındı
 *        content:
 *          application/json:
 *            schema:
 *              type: object
 *              properties:
 *                message:
 *                  type: string
 *                  example: Kullanıcı başarıyla alındı.
 *                user:
 *                  $ref: '#/components/schemas/User'
 *      400:
 *        description: Geçersiz kullanıcı ID'si
 *      401:
 *        description: Kullanıcı doğrulanamadı
 *      403:
 *        description: Bu işlem için yetkiniz yok
 *      404:
 *        description: Kullanıcı bulunamadı
 *      500:
 *        description: Sunucu hatası
 */

/**
 * @swagger
 * /user/{userId}:
 *   put:
```

```
*    summary: Kullanıcının rolünü günceller
*    tags: [User]
*    security:
*      - bearerAuth: []
*    parameters:
*      - in: path
*        name: userId
*        required: true
*        schema:
*          type: string
*        description: Güncellenecek kullanıcının ID'si
*        example: 64d4e7f2b2c1a2b3c4d5e6f8
*    requestBody:
*      required: true
*      content:
*        application/json:
*          schema:
*            type: object
*            required:
*              - role
*            properties:
*              role:
*                type: string
*                enum: [admin, user]
*                example: admin
*                description: Yeni kullanıcı rolü
*    responses:
*      200:
*        description: Kullanıcı rolü başarıyla güncellendi
*        content:
*          application/json:
*            schema:
*              type: object
*              properties:
*                message:
*                  type: string
```

```
 *                 example: Kullanıcı rolü başarıyla güncellendi.
 *               user:
 *                 $ref: '#/components/schemas/User'
 *       400:
 *         description: Geçersiz kullanıcı ID'si veya eksik alanlar
 *       401:
 *         description: Kullanıcı doğrulanamadı
 *       403:
 *         description: Bu işlem için yetkiniz yok
 *       404:
 *         description: Kullanıcı bulunamadı
 *       500:
 *         description: Sunucu hatası
 */

/**
 * @swagger
 * /user/{userId}:
 *   delete:
 *     summary: Belirli bir kullanıcıyı siler
 *     tags: [User]
 *     security:
 *       - bearerAuth: []
 *     parameters:
 *       - in: path
 *         name: userId
 *         required: true
 *         schema:
 *           type: string
 *         description: Silinecek kullanıcının ID'si
 *         example: 64d4e7f2b2c1a2b3c4d5e6f8
 *     responses:
 *       200:
 *         description: Kullanıcı başarıyla silindi
 *         content:
 *           application/json:
```

```
 *           schema:
 *             type: object
 *             properties:
 *               message:
 *                 type: string
 *                 example: Kullanıcı başarıyla silindi.
 *               user:
 *                 $ref: '#/components/schemas/User'
 *       400:
 *         description: Geçersiz kullanıcı ID'si
 *       401:
 *         description: Kullanıcı doğrulanamadı
 *       403:
 *         description: Bu işlem için yetkiniz yok
 *       404:
 *         description: Kullanıcı bulunamadı
 *       500:
 *         description: Sunucu hatası
 */

/**
 * @swagger
 * components:
 *   schemas:
 *     User:
 *       type: object
 *       properties:
 *         _id:
 *           type: string
 *           example: 64d4e7f2b2c1a2b3c4d5e6f8
 *         firstName:
 *           type: string
 *           example: Ali
 *         lastName:
 *           type: string
 *           example: Veli
```

```
 *       email:
 *         type: string
 *         format: email
 *         example: ali@veli.com
 *       role:
 *         type: string
 *         enum: [admin, user]
 *         example: admin
 *       createdAt:
 *         type: string
 *         format: date-time
 *         example: 2025-01-01T12:00:00.000Z
 *       updatedAt:
 *         type: string
 *         format: date-time
 *         example: 2025-01-01T12:00:00.000Z
 */

router.get("/", verifyAccessToken, checkRole("admin"), getAllUsers);
router.get("/:userId", verifyAccessToken, checkRole("admin"), getUserById);
router.put("/:userId", verifyAccessToken, checkRole("admin"), updateUserRol
e);
router.delete(
  "/:userId",
  verifyAccessToken,
  checkRole("admin"),
  deleteUserById
);

module.exports = router;
```

Böylelikle artık admin işlemleri için de kullanılabilir bir servisimiz hazırlanmış oldu.

Kapanış

Bu proje, şantiye bazlı maliyet yönetimi

Bu proje, şantiye bazlı maliyet yönetimi için geliştirilmiş tam kapsamlı bir backend uygulamasıdır. JWT tabanlı kimlik doğrulama, rol bazlı yetkilendirme (Admin/Editor/User), şantiye ve maliyet CRUD işlemleri, dosya yükleme, input validasyon ve Swagger API dökümantasyonu içermektedir.
Node.js ve Express.js kullanılarak geliştirilmiş, MongoDB ile veri yönetimi sağlanmıştır. Proje boyunca clean code prensipleri, middleware pattern'leri ve RESTful API standartlarına dikkat edilmiştir. Multer ile dosya yükleme, express-validator ile güvenli veri girişi, bcrypt ile şifre güvenliği ve JWT ile oturum yönetimi implement edilmiştir.
Proje, firmaların şantiyelerindeki tüm giderleri kategorize ederek kayıt altına alabilmesini, kullanıcı yetkilerine göre yönetebilmesini ve temel raporlama yapabilmesini sağlar. Geliştirme süreci boyunca yaşanan zorluklar ve alınan kararlar detaylı olarak dökümante edilmiştir.

Projenin senaryosal olarak tüm hatları oluşturulmamıştır. Ancak altyapı olarak istenilen çoğu yeni özelliği kaldırabilecek bir temele sahiptir. Fırsat bulduğum zaman, projeyi geliştirmeye devam edebilirim.
Herkese iyi kodlamalar, hoşçakalın 😃.

-Anıl Tarar