* github.com/Ankit9835/jay-bhagyata

#Adding Functionalities to Coupon

```
export const createCoupon = asyncHandler(async (req, res) => {
    const {code, discount} = req.body

    if (!code || !discount) {
        throw new CustomError("Code and discount are required", 400)
    }

    // check id code already exists

    const coupon = await Coupon.create({
        code,
        discount
    })

    res.status(200).json({
        success: true,
        message: "Coupon created successfully",
        coupon
    })
})
```

* To find All Coupons

```
export const getAllCoupons = asyncHandler( async (req, res) => {
    const allCoupons = await Coupon.find(); or await Coupon.find({});

    if (!allCoupons) {
        throw new CustomError("No Coupons found", 400)

    }

    res.status(200).json({
        success: true,
        allCoupons
    })
})
```

nodemailer & mailtrap //for sending mails

```
// create reusable transporter object using the default SMTP transport
let transporter = nodemailer.createTransport({
  host: "smtp.ethereal.email",
  port: 587,
  secure: false, // true for 465, false for other ports
  auth: {
    user: testAccount.user, // generated ethereal user
    pass: testAccount.pass, // generated ethereal password
  },
});
```

* Always need this Object to send mails

⇓ Store this in .env

```
PORT=5000
MONGODB_URL=mongodb://localhost:27017/ecomm
JWT_SECRET=yoursecret
JWT_EXPIRY=7d

S3_ACCESS_KEY=youraccesskey
S3_SECRET_ACCESS_KEY=yoursecretaccesskey
S3_BUCKET_NAME=yourbucketname
S3_REGION=yourregion

SMTP_MAIL_HOST=smtp.mailtrap.io
SMTP_MAIL_PORT=2525
SMTP_MAIL_USERNAME=yourusername
SMTP_MAIL_PASSWORD=yourpassword
SMTP_SENDER_EMAIL=some
```

need ⇒

index.js

```
otenv.config()

onst config = {
    PORT: process.env.PORT || 5000,
    MONGODB_URL: process.env.MONGODB_URL || "mongodb://localhost:27017/ecomm",
    JWT_SECRET: process.env.JWT_SECRET || "yoursecret",
    JWT_EXPIRY: process.env.JWT_EXPIRY || "30d",
    S3_ACCESS_KEY: process.env.S3_ACCESS_KEY,
    S3_SECRET_ACCESS_KEY: process.env.S3_SECRET_ACCESS_KEY,
    S3_BUCKET_NAME: process.env.S3_BUCKET_NAME,
    S3_REGION: process.env.S3_REGION,
    SMTP_MAIL_HOST: process.env.SMTP_MAIL_HOST,
    SMTP_MAIL_PORT: process.env.SMTP_MAIL_PORT,
    SMTP_MAIL_USERNAME: process.env.SMTP_MAIL_USERNAME,
    SMTP_MAIL_PASSWORD: process.env.SMTP_MAIL_PASSWORD,
    SMTP_SENDER_EMAIL: process.env.SMTP_SENDER_EMAIL,
}

export default config
```

creating transporter file

```
import config from "./index.js"

const transporter = nodemailer.createTransport({
    host: config.SMTP_MAIL_HOST,
    port: config.SMTP_MAIL_PORT,
    // secure: false, // true for 465, false for other ports
    auth: {
      user: config.SMTP_MAIL_USERNAME, // generated ethereal user
      pass: config.SMTP_MAIL_PASSWORD, // generated ethereal password
    },
})


export default transporter
```

for sending the mail

```
// send mail with defined transport object
let info = await transporter.sendMail({
  from: '"Fred Foo 👻" <foo@example.com>', // sender address
  to: "bar@example.com, baz@example.com", // list of receivers
  subject: "Hello ✔", // Subject line
  text: "Hello world?", // plain text body
  html: "<b>Hello world?</b>", // html body
});
```

mailhelper.js

```
import config from "../config/index.js"

import transporter from "../config/transporter.config.js"

const mailHelper = async (option) => {

    const message = {
        from: config.SMTP_SENDER_EMAIL,
        to: option.email,
        subject: option.subject,
        text: option.message
    }

    await transporter.sendMail(message)
}


export default mailHelper
```

in authcontroller.js

```
export const forgotPassword = asyncHandler(async (req, res) => {
    const {email} = req.body
    //no email
    const user = await User.findOne({email})

    if (!user) {
        throw new CustomError("User not found", 404)
    }

    const resetToken = user.generateForgotPasswordToken()

    await user.save({validateBeforeSave: false})
}) //Update user with the resetted token
```

stored forgot token

in user schema

```
generateForgotPasswordToken: function (){
    const forgotToken = crypto.randomBytes(20).toString("hex")

    // just to encrypt the token generated by crypto
    this.forgotPasswordToken = crypto
    .createHash("sha256")
    .update(forgotToken)
    .digest("hex")

    //time for token to expire
    this.forgotPasswordExpiry = Date.now() + 20 * 60 * 1000

    return forgotToken //we are returning it
}
```

so that we can store it

just save it and dont do any validation in user shema

(we only do validation check when user login / time)

* req.protocol automatically detects 'https' or 'http'

* const resetUrl = `${req.protocol}://${req.get("host")}`

will give the domain name of localhost

```
const resetUrl = `${req.protocol}://${req.get("host")}/api/v1/auth/password/reset/${resetToken}`

const message = `Your password reset token is as follows \n\n ${resetUrl} \n\n if this was not requested by you, please ignore.`

try {
    // const options = {} //could define Options first then mailHelper(options) or define in the mailHelper itself
    await mailHelper({
        email: user.email,
        subject: "Password reset mail",
        message
    })
} catch (error) {
    // here we did try first & then catch because we do not want user to get affected as if error occurs will occur because of our fault.

}
```

```
} catch (error) {
    user.forgotPasswordToken = undefined
    user.forgotPasswordExpiry = undefined

    await user.save({validateBeforeSave: false})

    throw new CustomError(error.message || "Email could not be sent", 500)
}
```

```
export const resetPassword = asyncHandler(async (req, res) => {
    const {token: resetToken} = req.params
    const {password, confirmPassword} = req.body

    const resetPasswordToken = crypto
    .createHash("sha256")
    .update(resetToken)
    .digest("hex")

    const user = await User.findOne({
        forgotPasswordToken: resetPasswordToken,
        forgotPasswordExpiry: { $gt : Date.now() }
    })
})
```

* Here we will find User based on token because we do not have email as well as set password by the user

will forget Password the reset will be only valid for 20 minutes as previously defined. so if those 20 minutes are not yet exceeded and are greater than now password will be reset

```
    if (!user) {
        throw new CustomError( "password reset token in invalid or expired", 400)
    }

    if (password !== confirmPassword) {
        throw new CustomError("password does not match", 400)
    }

    user.password = password;
    user.forgotPasswordToken = undefined
    user.forgotPasswordExpiry = undefined

    await user.save()
})
```

```
    user.forgotPasswordExpiry = undefined

    await user.save()

    // optional

    const token = user.getJWTtoken()
    res.cookie("token", token, cookieOptions)

    res.status(200).json({
        success: true,
        user, //your choice
    })
})
```

```
export const updateCoupon = asyncHandler(async (req, res) => {
    const {id: couponId} = req.params
    const {action} = req.body

    // action is boolean or not

    const coupon = await Coupon.findByIdAndUpdate(
        couponId, //from where
        {
            active: action //what
        },
        {
            new: true,
            runValidators: true // updates
        }
    )
})

export const getAllCoupons = asyncHandler( async (req, res) => {…
})
```

```
if (!coupon) {
    throw new CustomError("Coupon not found", 404)
}

res.status(200).json({
    success: true,
    message: "Coupon updated"
})
```

```
import { Router } from "express";
import { getProfile, login, logout, signUp } from "../controllers/auth.controller";
import { isLoggedIn } from "../middlewares/auth.middleware";

const router = Router()

router.post("/signup", signUp)
router.post("/login", login)
router.get("/logout", logout)
router.get("/profile", isLoggedIn, getProfile)

export default router;
```

Controllers

Middleware

```
controllers/coupon.controller.js";
import { isLoggedIn, authorize } from "../middlewares/auth.middleware";
import AuthRoles from "../utils/authRoles.js";



const router = Router()

router.post("/", isLoggedIn, authorize(AuthRoles.ADMIN), createCoupon)
router.delete("/:id", isLoggedIn, authorize(AuthRoles.ADMIN, AuthRoles.MODERATOR),
deleteCoupon)
export default router;
```

name "id" should match what we set in Schema

Middlewares

- PUT : To completely replace a resource
  PATCH : To partially replace a resource

In index.js

```
import { Router } from "express";
import authRoutes from "./auth.route.js"
import couponRoutes from "./coupon.route.js"
import collectionRoutes from "./collection.route.js"

const router = Router()
router.use("/auth", authRoutes)
router.use("/coupon", couponRoutes)
router.use("/collection", collectionRoutes)

export default router
```

collection.route.js

```
import AuthRoles from "../utils/authRoles.js";

const router = Router()
will hit /api/v1/collection
router.post("/", isLoggedIn, authorize(AuthRoles.ADMIN), createCollection)
router.put("/:id", isLoggedIn, authorize(AuthRoles.ADMIN), updateCollection)

// delete a single collection
router.delete("/:id", isLoggedIn, authorize(AuthRoles.ADMIN), deleteCollection)

//get all collection
router.get("/", getAllCollections)

export default router;
```

```
import { Router } from "express";
import { createCoupon, deleteCoupon, getAllCoupons, updateCoupon } from "../
controllers/coupon.controller.js";
import { isLoggedIn, authorize } from "../middlewares/auth.middleware";
import AuthRoles from "../utils/authRoles.js";

const router = Router()

router.post("/", isLoggedIn, authorize(AuthRoles.ADMIN), createCoupon)
router.delete("/:id", isLoggedIn, authorize(AuthRoles.ADMIN, AuthRoles.MODERATOR),
deleteCoupon)
router.put("/action/:id", isLoggedIn, authorize(AuthRoles.ADMIN, AuthRoles.MODERATOR),
updateCoupon)
router.get("/", isLoggedIn, authorize(AuthRoles.ADMIN, AuthRoles.MODERATOR),
getAllCoupons)

export default router;
```

* POST request : to send data
GET request : just hit the route and get data
eg: Login will have post request as we send data
logout will be get request

* in app.js we used :
app.use("/api/v1/", routes) which means every single route will be appended by /api/v1...

```
import { Router } from "express";
import { getProfile, login, logout, signUp, forgotPassword, resetPassword } from "../
controllers/auth.controller";
import { isLoggedIn } from "../middlewares/auth.middleware";

const router = Router()

router.post("/signup", signUp)
router.post("/login", login)
router.get("/logout", logout)

router.post("/password/forgot/", forgotPassword)
router.post("/password/reset/:token", resetPassword)

router.get("/profile", isLoggedIn, getProfile)

export default router;
```