Unit Testing G1 Pet Adoption System – HappyTails

Testing framework used – Mocha @10.8.2

Assertion library used – Chai @4.3.4

Other – Sinon @19.0.2

1. OTPControllers
   a. sendGmailOTP

```
const sendGmailOTP = async ({_id,email},res) => {
    try {
        const otp = `${Math.floor(Math.random() * 9000 + 1000)}` //create 4 digit OTP
        const mailOptions = { //email details
            from: process.env.USER_EMAIL,
            to: email,
            subject: "Verify Your HappyTails account",
            html: `<div style="font-family: Arial, sans-serif; color: #333; line-height: 1.6;
            </div>`,
        }
        const hashedOTP = await bcrypt.hash(otp, 10);
        const newOTPVerification = new OTPverification({ //create OTP details in db
            userID: _id,
            otp: hashedOTP,
            createdAt: Date.now(),
            expireAt: Date.now() + 3600000,
            email:email,
        });
        await newOTPVerification.save();
        let emailTransporter = await setGmailForOTP();
        await emailTransporter.sendMail(mailOptions); //send OTP email
    } catch (error) {
        res.json({
            success:false,
            message:error.message,
        })
    }
}
```

Test cases and results:

```
describe("sendGmailOTP", () => {
    it("should successfully send OTP email", async function () {
        this.timeout(10000);
        const userData = {
            _id: "user123",
            email: "test@example.com",
        };
        const fakeTransporter = {
            sendMail: sinon.stub().resolves(),
        };
        sinon.stub(bcrypt, "hash").resolves("hashedOTP");
        const saveStub = sinon.stub().resolves();
        sinon.stub(OTPverification.prototype, "save").callsFake(saveStub);
        const oauth2ClientStub = {
            setCredentials: sinon.stub(),
            getAccessToken: sinon
                .stub()
                .callsFake((callback) => callback(null, "fake-token")),
        };
        sinon.stub(google.auth, "OAuth2").returns(oauth2ClientStub);
        sinon.stub(nodemailer, "createTransport").returns(fakeTransporter);
        await otpController.sendGmailOTP(userData, res);
        expect(saveStub.calledOnce).to.be.true;
        expect(fakeTransporter.sendMail.calledOnce).to.be.true;
    });
```

This case checks the condition where the Gmail is sent properly when all necessary conditions are met.

```
it("should handle errors during OTP email sending", async () => {
    const userData = {
        _id: "user123",
        email: "test@example.com",
    };
    sinon.stub(bcrypt, "hash").rejects(new Error("Bcrypt error"));
    await otpController.sendGmailOTP(userData, res);
    expect(
        jsonStub.calledWith({
            success: false,
            message: "Bcrypt error",
        })
    ).to.be.true;
});
```

This test case will handle any unexpected error in the sending of emails.

Results:

```
sendGmailOTP
  ✓ should successfully send OTP email (206ms)
  ✓ should handle errors during OTP email sending
```

b. setGmailForOTP

```
const setGmailForOTP = async () => {
    try {
        const oauth2Client = new OAuth2( //use google OAuth2
            process.env.CLIENT_ID_SMTP,
            process.env.CLIENT_SECRET_SMTP,
            "https://developers.google.com/oauthplayground"
        );
        oauth2Client.setCredentials({
            refresh_token: process.env.REFRESH_TOKEN,
        });
        const accessToken = await new Promise((resolve, reject) => {
            oauth2Client.getAccessToken((err, token) => {
                if (err) {
                    console.log("*ERR: ", err)
                    reject();
                }
                resolve(token);
            });
        });
        const transporter = nodemailer.createTransport({ //specify transport details
            service: "gmail",
            auth: {
              type: "OAuth2",
              user: process.env.USER_EMAIL,
              accessToken,
              clientId: process.env.CLIENT_ID_SMTP,
              clientSecret: process.env.CLIENT_SECRET_SMTP,
              refreshToken: process.env.REFRESH_TOKEN,
            },
        });
```

Test Cases and results:

```
describe("setGmailForOTP", () => {
    it("should successfully create a transporter", async () => {
        const oauth2ClientStub = {
            setCredentials: sinon.stub(),
            getAccessToken: sinon
                .stub()
                .callsFake((callback) => callback(null, "fake-token")),
        };
        sinon.stub(google.auth, "OAuth2").returns(oauth2ClientStub);
        const fakeTransporter = { fake: "transporter" };
        sinon.stub(nodemailer, "createTransport").returns(fakeTransporter);
        const result = await otpController.setGmailForOTP();
        expect(result).to.deep.equal(fakeTransporter);
    });
});
```

This test case checks the condition where the email transporter is created successfully.

```
it("should handle access token error correctly", async () => {
    oauth2ClientStub = sinon
        .stub(OAuth2.prototype, "getAccessToken")
        .callsFake((callback) => {
            callback(new Error("Access token error"), null);
        });
    const res = {
        json: sinon.spy(),
    };
    await setGmailForOTP(res);
    expect(res.json.calledOnce).to.be.true;
    expect(res.json.firstCall.args[0]).to.deep.equal({
        success: false,
        message: "Access token error",
    });
    expect(oauth2ClientStub.calledOnce).to.be.true;
});
```

This condition checks the condition where the access token is not generated properly and return an error

```
it("should handle general errors in try-catch block", async () => {
    mockOAuth2Client.getAccessToken.callsFake((callback) => {
        callback(null, "mock-token");
    });
    const transportError = new Error("invalid_client");
    nodeMailerStub = sinon
        .stub(nodemailer, "createTransport")
        .throws(transportError);

    const res = {
        json: sinon.spy(),
    };

    await setGmailForOTP(res);
    expect(res.json.calledOnce, "res.json should be called once").to.be.true;
    expect(res.json.firstCall.args[0]).to.deep.equal({
        success: false,
        message: "invalid_client",
    });
});
```

This condition checks the case for any unexpected error in the function which can occur due to database error or other reasons.

Results:

```
OTP Controllers
  setGmailForOTP
    ✓ should successfully create a transporter (417ms)
```

```
✓ should handle access token error correctly
✓ should handle general errors in try-catch block (242ms)
```

c. checkOTP

```
async function checkOTP(req,res){
    try {
        const {email,OTP}=req.body;
        if(!email || !OTP) {
            return res.status(400).json({ success: false, message: "Empty OTP details" });
        } else { //check for OTP correctness
            const OTPverificationRecords=await OTPverification.find({email:email});
            if(OTPverificationRecords.length<=0) {
                return res.status(400).json({ success: false, message: "Account is either invalid or already been verified" });
            } else {
                const {expireAt}=OTPverificationRecords[0];
                const hashedOTP=OTPverificationRecords[0].otp;
                if(expireAt < Date.now()) {
                    await OTPverification.deleteMany({email:email});
                    return res.status(400).json({ success: false, message: "OTP has been expired, please request again" });
                } else {
                    const validOTP=bcrypt.compare(OTP,hashedOTP);
                    if(!validOTP) {
                        return res.status(404).json({ success: false, message: "Invalid OTP, try again" });
                    } else { //verify user
                        await loginschema.updateOne({email:email},{verified:true});
                        await OTPverification.deleteMany({email:email});
                        res.json({success : true,message:"verified",isVerified:true,user:{email:email,isVerified:true}})
                    }
                }
            }
        }
    } catch (error) {···
```

Test Cases and results:

```
describe("checkOTP", () => {
    it("should return error for empty OTP details", async () => {
        req = {
            body: {},
        };
        await otpController.checkOTP(req, res);
        expect(statusStub.calledWith(400)).to.be.true;
        expect(
            jsonStub.calledWith({
                success: false,
                message: "Empty OTP details",
            })
        ).to.be.true;
    });
```

This case checks the condition where the OTP details are not found.

```
it("should return error for invalid account", async () => {
    req = {
        body: {
            email: "test@example.com",
            OTP: "1234",
        },
    };
    sinon.stub(OTPverification, "find").resolves([]);
    await otpController.checkOTP(req, res);
    expect(statusStub.calledWith(400)).to.be.true;
    expect(
        jsonStub.calledWith({
            success: false,
            message: "Account is either invalid or already been verified",
        })
    ).to.be.true;
});
```

This condition where the account is invalid and such error are handled.

```
it("should return error for expired OTP", async () => {
    req = {
        body: {
            email: "test@example.com",
            OTP: "1234",
        },
    };
    sinon.stub(OTPverification, "find").resolves([
        {
            expireAt: Date.now() - 1000,
            otp: "hashedOTP",
        },
    ]);
    sinon.stub(OTPverification, "deleteMany").resolves();
    await otpController.checkOTP(req, res);
    expect(statusStub.calledWith(400)).to.be.true;
    expect(
        jsonStub.calledWith({
            success: false,
            message: "OTP has been expired, please request again",
        })
    ).to.be.true;
});
```

This test case checks the condition where the OTP entered has expired.

```
it("should return error for invalid OTP", async () => {
    req = {
        body: {
            email: "test@example.com",
            OTP: "1234",
        },
    };
    sinon.stub(OTPverification, "find").resolves([
        {
            expireAt: Date.now() + 3600000,
            otp: "hashedOTP",
        },
    ]);
    sinon.stub(bcrypt, "compare").returns(false);
    await otpController.checkOTP(req, res);
    expect(statusStub.calledWith(404)).to.be.true;
    expect(
        jsonStub.calledWith({
            success: false,
            message: "Invalid OTP, try again",
        })
    ).to.be.true;
});
```

This test condition will check the case where OTP entered does not match the databases entry.

```
it("should successfully verify OTP", async () => {
    const email = "test@example.com";
    req = { body: { email, OTP: "1234",}};
    sinon.stub(OTPverification, "find").resolves([
        {expireAt: Date.now() + 3600000,otp: "hashedOTP",}
    ]);
    sinon.stub(bcrypt, "compare").returns(true);
    sinon.stub(loginschema, "updateOne").resolves();
    sinon.stub(OTPverification, "deleteMany").resolves();
    await otpController.checkOTP(req, res);
    expect(
        jsonStub.calledWith({
            success: true,
            message: "verified",
            isVerified: true,
            user: {
                email,
                isVerified: true,
            },
        })
    ).to.be.true;
});
```

This condition checks the condition where all details entered are correct and OTP is correct.

```
it("should handle unexpected errors", async () => {
    req = {
        body: {
            email: "test@example.com",
            OTP: "1234",
        },
    };
    sinon.stub(OTPverification, "find").rejects(new Error("Database error"));
    await otpController.checkOTP(req, res);
    expect(
        jsonStub.calledWith({
            success: false,
            message: "Database error",
        })
    ).to.be.true;
});
```

This condition check the case where any unexpected error is handled.

Results:

```
checkOTP
  ✓ should return error for empty OTP details
  ✓ should return error for invalid account
  ✓ should return error for expired OTP
  ✓ should return error for invalid OTP
  ✓ should successfully verify OTP
  ✓ should handle unexpected errors
```

d. postResendOTP

```
async function postResendOTP(req,res) {
    try {
        const email=req.body.email;
        const user = await loginschema.findOne({email:email}); //get user through email
        if(!user) {
            return res.status(404).json({ success:false, message: "No such user exists" });
        } else {
            await OTPverification.deleteMany({email:email});
            const data={
                _id:user._id,
                email:email,
            }
            await sendGmailOTP(data,res); //send OTP email again
            res.json({success:true, message: "Email sent 12332,4"})
        }
    } catch (error) {
        res.json({
            success:false,
            message:error.message,
        })
    }
}
```

```
describe("postResendOTP", () => {
    it("should return error for non-existent user", async () => {
        req = {
            body: {
                email: "test@example.com",
            },
        };
        sinon.stub(loginschema, "findOne").resolves(null);
        await otpController.postResendOTP(req, res);
        expect(statusStub.calledWith(404)).to.be.true;
        expect(
            jsonStub.calledWith({
                success: false,
                message: "No such user exists",
            })
        ).to.be.true;
    });
```

This test case checks the condition where the email entered by the user doesn't exists in the users database.

```
it("should successfully resend OTP", async function () {
    this.timeout(10000); // Increase timeout for this specific test
    const email = "test@example.com";
    req = {body: { email },};
    const userData = {_id: "user123",email,};
    sinon.stub(loginschema, "findOne").resolves(userData);
    sinon.stub(OTPverification, "deleteMany").resolves();
    const fakeTransporter = {sendMail: sinon.stub().resolves(),};
    sinon.stub(bcrypt, "hash").resolves("hashedOTP");
    sinon.stub(OTPverification.prototype, "save").resolves();
    const oauth2ClientStub = {
        setCredentials: sinon.stub(),
        getAccessToken: sinon
            .stub()
            .callsFake((callback) => callback(null, "fake-token")),
    };
    sinon.stub(google.auth, "OAuth2").returns(oauth2ClientStub);
    sinon.stub(nodemailer, "createTransport").returns(fakeTransporter);
    await otpController.postResendOTP(req, res);
    expect(jsonStub.calledWith({success: true,message: "Email sent 12332,4",})).to.be.true;
});
```

This condition checks for the case where OTP is sent successfully when correct email is entered.

```
it("should handle unexpected errors", async () => {
    req = {
        body: {
            email: "test@example.com",
        },
    };
    sinon.stub(loginschema, "findOne").rejects(new Error("Database error"));
    await otpController.postResendOTP(req, res);
    expect(
        jsonStub.calledWith({
            success: false,
            message: "Database error",
        })
    ).to.be.true;
});
```

This test case checks the condition when some unexpected error occurs happens.

Results:

```
postResendOTP
  ✓ should return error for non-existent user
  ✓ should successfully resend OTP (116ms)
  ✓ should handle unexpected errors
```

Test coverage:

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	98.87	98	100	98.86	
controllers	98.71	97.82	100	98.71	
OTPControllers.js	96.61	92.85	100	96.61	25,44
loginControllers.js	100	100	100	100	
models	100	100	100	100	
OTPverification.js	100	100	100	100	
loginschema.js	100	100	100	100	
service	100	100	100	100	
auth.js	100	100	100	100	

This is the test coverage for the OTP controller, as we can see that the 97% of code is covered where 2 statement remain uncovered which is due to errors faced in making these test cases. But other than that each and every function in the controller was covered.