**Explanation of the Code in Simple Terms**

This code **queues email-sending tasks** instead of sending emails immediately. It uses **Bull**, a Node.js library for job queues, backed by **Redis**.

**1️⃣ What is Happening Here?**

1. **Connect to Redis**
   * A job queue named **"emailQueue"** is created and connected to Redis (127.0.0.1:6379).
   * Redis acts as a temporary storage for jobs.
2. **Queue a Job When a User Requests Email Sending**
   * When a user sends a POST request to /send-email, the server **does NOT send the email immediately**.
   * Instead, the job is **added to the queue**.
3. **A Worker Picks Up the Job and Processes It Asynchronously**
   * The worker fetches jobs from the queue.
   * It simulates email sending by waiting (setTimeout for 3 seconds).
   * Finally, it logs that the email has been sent.

**2️⃣ Code Breakdown**

**📌 Step 1: Connect to Redis**

const Queue = require('bull');

const jobQueue = new Queue('emailQueue', {

redis: {

host: '127.0.0.1',

port: 6379

}

});

✔ **Creates a new queue named "emailQueue"**.  
✔ **Connects to Redis** running on 127.0.0.1:6379.  
✔ Redis stores the jobs temporarily and processes them later.

**📌 Step 2: Add a Job to the Queue**

app.post('/send-email', async (req, res) => {

const { email, subject, message } = req.body;

if (!email || !subject || !message) {

return res.status(400).json({ error: "Missing required fields" });

}

// Add email task to the queue

await jobQueue.add({ email, subject, message });

res.json({ message: `Email job queued for ${email}` });

});

✔ **When a POST /send-email request is made** with an email, subject, and message:

* The request **does not immediately send the email**.
* Instead, it **adds the task to the queue** (jobQueue.add(...)).
* **The user gets an immediate response**:  
  { message: "Email job queued for user@example.com" }.

**📌 Step 3: Process the Job in the Background**

jobQueue.process(async (job) => {

console.log(`Sending email to ${job.data.email}...`);

// Simulate sending email (Replace this with an actual email service)

await new Promise(resolve => setTimeout(resolve, 3000));

console.log(`Email sent to ${job.data.email}!`);

});

✔ **A worker constantly checks for new jobs in the queue**.  
✔ When a new job is found, it:

* Extracts **email, subject, message** from the job.
* **Simulates sending an email** (setTimeout for 3 seconds).
* Logs when the email is "sent".

**Final Code:**

const express = require('express');

const Queue = require('bull'); // Node.js library for job queues, backed by Redis.

const app = express();

app.use(express.json()); // For parsing the JSON request

/\*\*

 \* Connect to Redis

 \*      A job queue named "emailQueue" is created and connected to Redis (127.0.0.1:6379).

 \*      Redis acts as a temporary storage for jobs.

 \*\*/

const jobQueue = new Queue('emailQueue', {

    redis: {

        host: '127.0.0.1',

        port: 6379

    }

});

/\*\*

 \* Queue a Job When a User Requests Email Sending

 \*      When a user sends a POST request to /send-email, the server does NOT send the email immediately.

 \*      Instead, the job is added to the queue.

 \*/

app.post('/send-email', async (req, res) => {

    const { email, subject, message } = req.body;

    if(!email || !subject || !message){

        return res.status(400).json({ error: "Missing required fields!" })

    }

    // Add email task to the Queue

    await jobQueue.add({ email, subject, message });

    res.json({ message: `Email job queued for ${email}` });

});

/\*\*

 \* Process Job (Worker): A Worker Picks Up the Job and Processes It Asynchronously

 \*      The worker fetches jobs from the queue.

 \*      It simulates email sending by waiting (setTimeout for 3 seconds).

 \*      Finally, it logs that the email has been sent.

 \*\*/

jobQueue.process( async (job) => {

    console.log(`Sending email to ${job.data.email}...`);

    // Simulate sending email (Replace this with an actual email service)

    await new Promise(resolve => setTimeout(resolve, 3000));

    console.log(`Email sent to ${job.data.email}!`);

});

// Start Server

app.listen(8000, () => {

    console.log("Server is running on port 8000");

});

**3️⃣ Testing the Queue**

**📌 Step 1: Start Redis Server**

Before running the code, ensure **Redis** is running.

* If Redis is **not installed**, install it via:

sudo apt update && sudo apt install redis -y # Ubuntu

brew install redis # macOS

* Start Redis:

redis-server

**📌 Step 2: Start the Node.js Server**

Run the application:

node server.js

or if using nodemon:

nodemon server.js

**📌 Step 3: Send a Request to Queue an Email**

Use **cURL** or **Postman**:

**cURL Command**

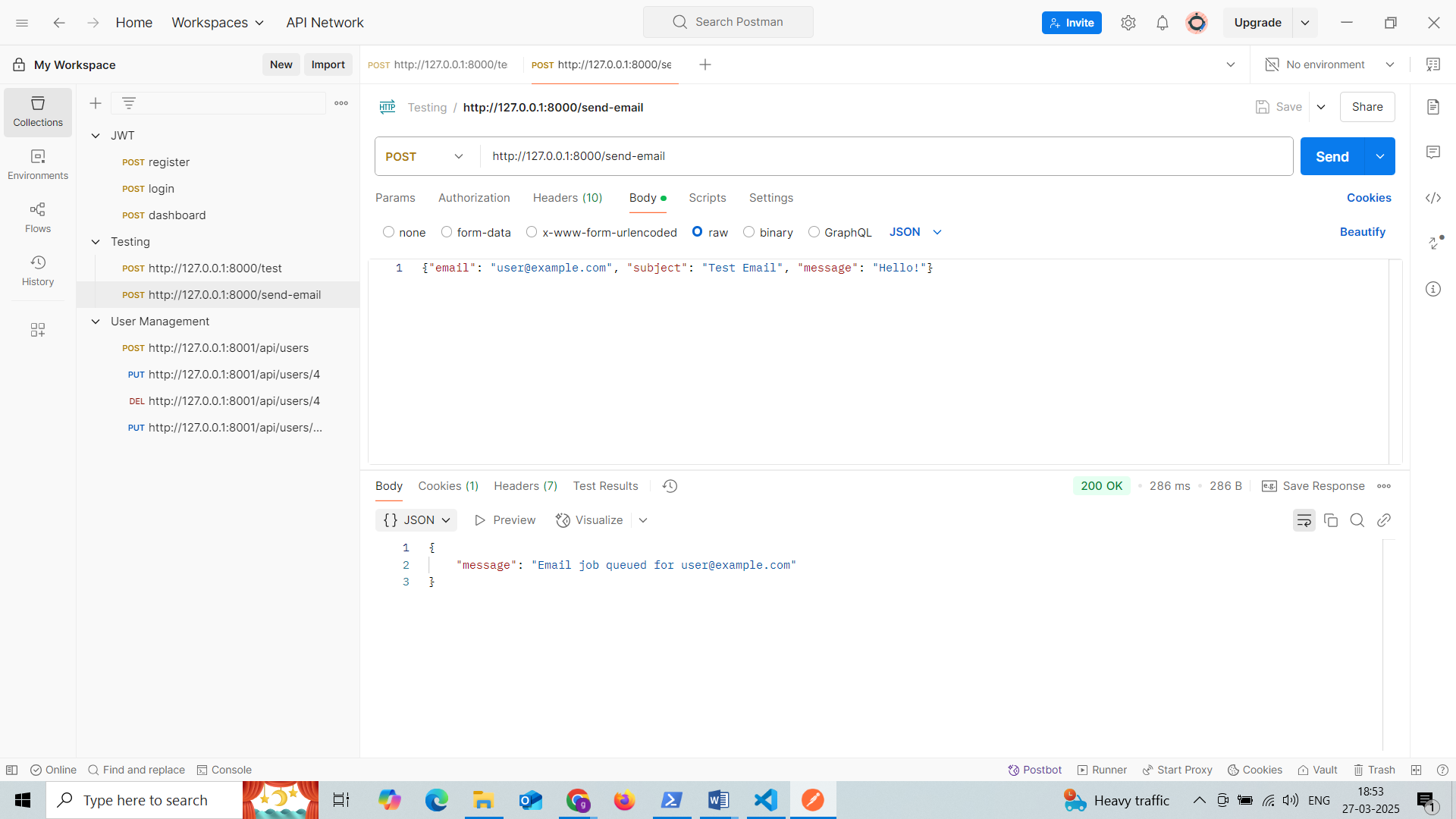
curl -X POST http://127.0.0.1:8000/send-email \

-H "Content-Type: application/json" \

-d '{"email": "user@example.com", "subject": "Test Email", "message": "Hello!"}'

✔ **Expected Response**

{ "message": "Email job queued for user@example.com" }



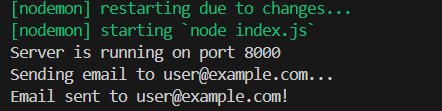
**📌 Step 4: Verify the Logs**

In the terminal running the server, you should see:

Sending email to user@example.com...

(3 seconds delay)

Email sent to [user@example.com](mailto:user@example.com)!



🎉 **This confirms the job was processed successfully!**

### ****✅ Replacing Simulated Email Sending with an Actual Email Service (Nodemailer)****

Now, let's **replace the simulated email process** with an actual email service using **Nodemailer**.

## **🔹 Step 1: Install Nodemailer**

Run the following command in your terminal to install Nodemailer:

npm install nodemailer

## **🔹 Step 2: Configure a Real Email Service in** jobQueue.js

Modify **jobQueue.js** to send real emails using **Gmail, Outlook, or SMTP**.

📌 **Updated jobQueue.js**

const Queue = require('bull');

const nodemailer = require('nodemailer');

require('dotenv').config();

// Initialize the email queue (using Redis)

const emailQueue = new Queue('emailQueue', {

redis: { host: '127.0.0.1', port: 6379 }

});

// Configure Email Transporter

const transporter = nodemailer.createTransport({

service: 'gmail', // You can use 'Outlook', 'Yahoo', or SMTP settings

auth: {

user: process.env.EMAIL\_USER, // Your email

pass: process.env.EMAIL\_PASS // Your email password or app password

}

});

// Process Email Jobs in the Background

emailQueue.process(async (job) => {

const { email, subject, message } = job.data;

console.log(`📧 Sending email to: ${email}...`);

try {

// Send email using Nodemailer

await transporter.sendMail({

from: `"Your App Name" <${process.env.EMAIL\_USER}>`,

to: email,

subject: subject,

text: message

});

console.log(`✅ Email sent to ${email}!`);

} catch (error) {

console.error(`❌ Error sending email to ${email}:`, error);

}

});

module.exports = { emailQueue };

## **🔹 Step 3: Update** .env **File with Email Credentials**

Add the following lines to your .env file:

EMAIL\_USER=your-email@gmail.com

EMAIL\_PASS=your-app-password

📌 **Important Note:**  
If using **Gmail**, you need to enable **"Less Secure Apps"** or generate an **App Password**.  
🔗 **Generate App Password**

## **🔹 Step 4: Queue an Email After Google OAuth Login**

📌 **create a new file - email\_service.js [**to send an email after login]

const express = require('express');

const passport = require('passport');

const session = require('express-session');

const { emailQueue } = require('./jobQueue'); // Import Redis queue

require('dotenv').config();

const app = express();

app.use(session({ secret: "SESSION\_SECRET", resave: false, saveUninitialized: true }));

app.use(passport.initialize());

app.use(passport.session());

// Google OAuth Authentication Routes (Login)

app.get('/auth/google',

passport.authenticate("google", { scope: ["profile", "email"] })

);

// Google OAuth Callback

app.get('/auth/google/callback',

passport.authenticate('google', { failureRedirect: '/' }),

(req, res) => {

// 🎯 Queue an email after successful login

emailQueue.add({

email: req.user.emails[0].value,

subject: "Welcome to Our App!",

message: `Hi ${req.user.displayName},\n\nYou have successfully logged in!\n\nBest Regards,\nYour App Team`

});

res.redirect('/profile');

}

);

// Profile Page

app.get('/profile', (req, res) => {

res.send(`Welcome ${req.user.displayName}`);

});

app.listen(8000, () => console.log(`Server running on port 8000`));

## **🔹 Step 5: Start Everything & Test**

### ****1️⃣ Start Redis****

If using WSL:

sudo service redis-server start

If using Docker:

docker run --name redis-server -d -p 6379:6379 redis

### ****2️⃣ Start Your Node.js Server****

node index.js

### ****3️⃣ Test Google OAuth Login****

1. Open **http://localhost:8000/auth/google** in your browser.
2. Log in using **Google OAuth**.
3. After login, you should receive an **actual email** in your inbox! 🎉

## **✅ Summary**

* **What Changed?**
  + **Used Nodemailer** for real email sending.
  + **Integrated it into Bull Queue** for background email processing.
* **Why is this Useful?**
  + Emails are **sent asynchronously** without slowing down the server.
  + **Scalable** solution for sending multiple emails.