Name :- Utkarsh Lambade **Roll no :- 62**

Sem: VI **Subject :- Honors(Devops)**

Aim :

To design, develop, and deploy a secure backend application using modern web technologies. The system must support user registration with email verification, login functionality, and a JWT-based token authentication mechanism for protected API access. The project should be deployed using Nginx as a reverse proxy and secured with HTTPS via Let’s Encrypt.

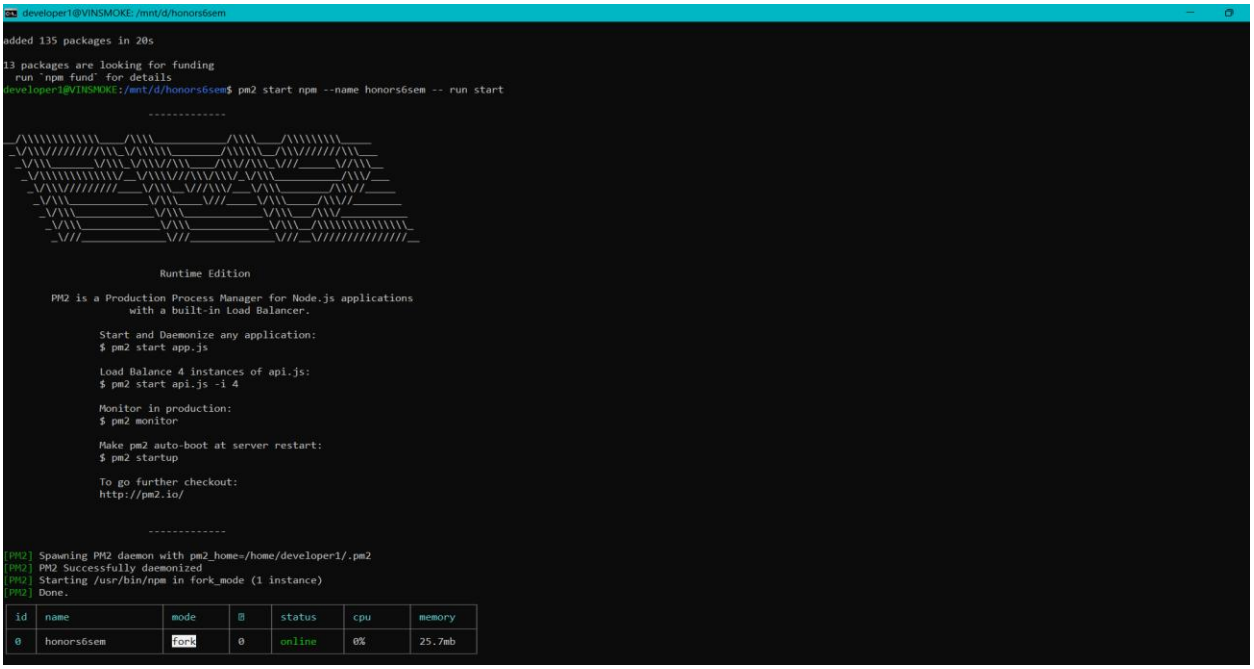
Abstract

This project implements a full-stack secure user registration and login system on a WSL Ubuntu environment. The application utilizes Express.js for backend logic, MongoDB Atlas for database operations, JWT for authentication, and Nodemailer with Gmail for email verification and password reset. The backend is deployed using Nginx as a reverse proxy and secured with HTTPS via Let's Encrypt.

Table of Contents

---

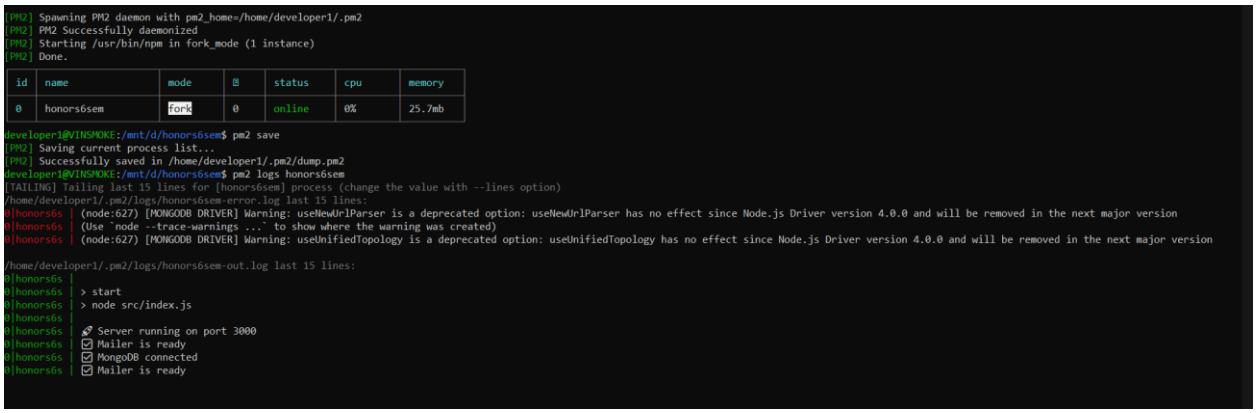
1. Environment Setup on WSL Ubuntu

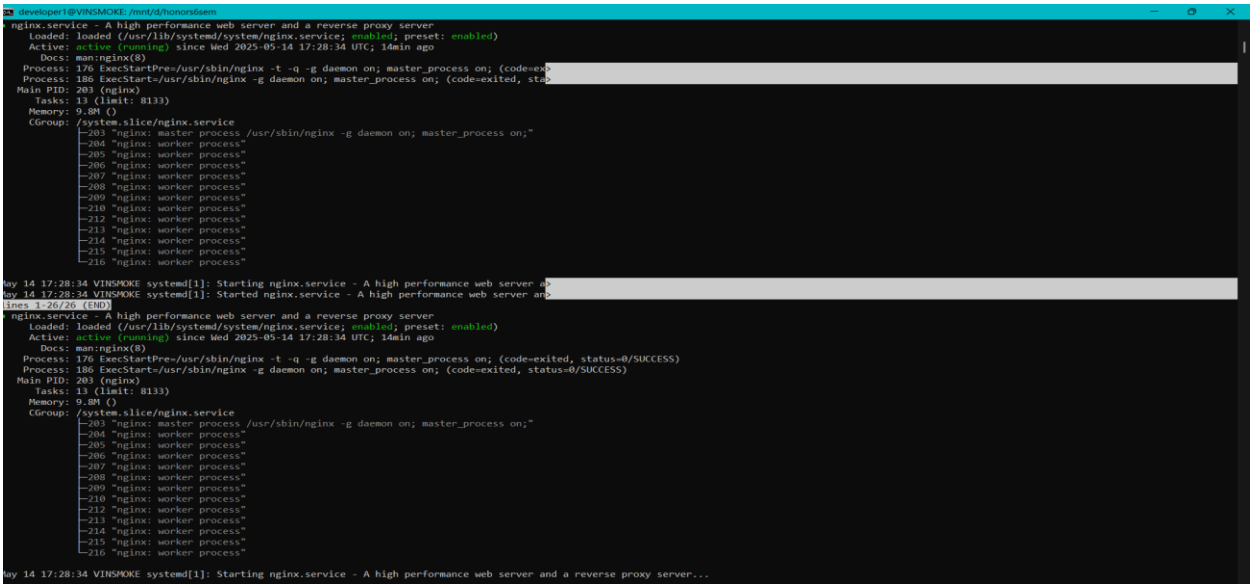


Configured a Node.js development environment on WSL Ubuntu. This ensures Linux-based development on Windows, making deployment more production-like.

- Installed Node.js, npm
- Configured and used PM2 to manage the Node.js app

- Installed Nginx for reverse proxy



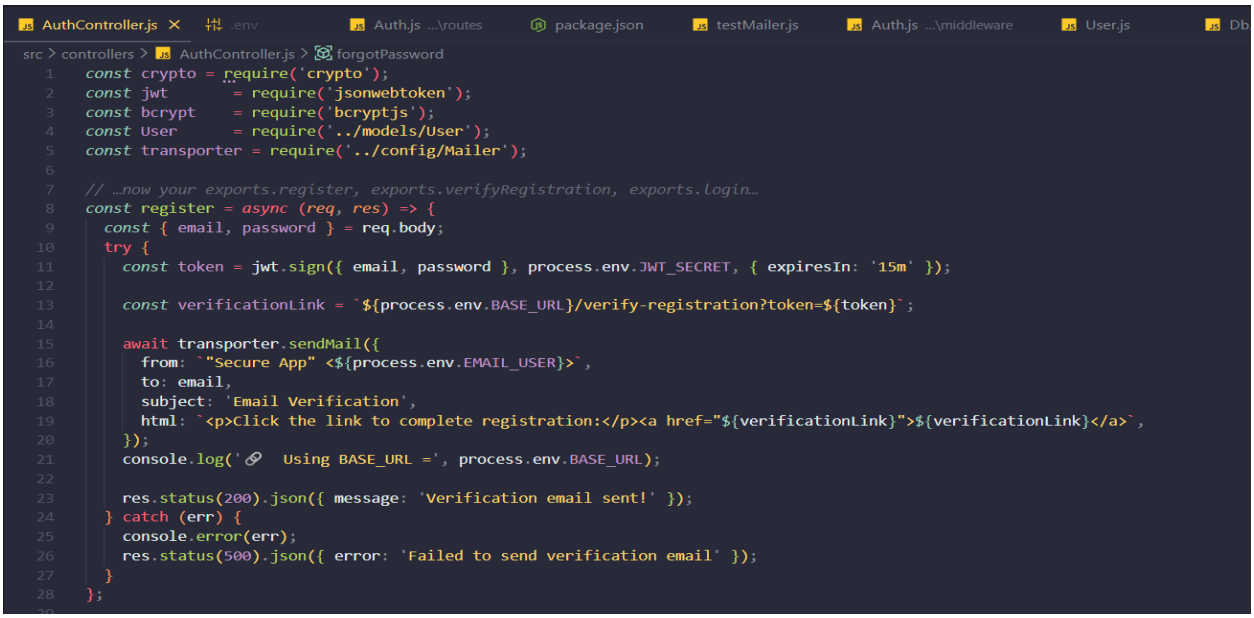

developer1@VINSMOKE: /mnt/d/honors6sem
● nginx.service - A high performance web server and a reverse proxy server
Loaded: loaded (/usr/lib/systemd/system/nginx.service; enabled; preset: enabled)
Active: active (running) since Wed 2025-05-14 17:28:34 UTC; 14min ago
Docs: man:nginx(8)
Process: 176 ExecStartPre=/usr/sbin/nginx -t -q -g daemon on; master_process on; (code=ex
Process: 186 ExecStart=/usr/sbin/nginx -g daemon on; master_process on; (code=exited, ste
Main PID: 203 (nginx)
Tasks: 13 (limit: 8133)
Memory: 9.8M ()
CGroup: /system.slice/nginx.service
├─203 "nginx: master process /usr/sbin/nginx -g daemon on; master_process on;"
├─204 "nginx: worker process"
├─205 "nginx: worker process"
├─206 "nginx: worker process"
├─207 "nginx: worker process"
├─208 "nginx: worker process"
├─209 "nginx: worker process"
├─210 "nginx: worker process"
├─212 "nginx: worker process"
├─213 "nginx: worker process"
├─214 "nginx: worker process"
├─215 "nginx: worker process"
└─216 "nginx: worker process"
May 14 17:28:34 VINSMOKE systemd[1]: Starting nginx.service - A high performance web server a
May 14 17:28:34 VINSMOKE systemd[1]: Started nginx.service - A high performance web server an
lines 1-26/26 (END)
● nginx.service - A high performance web server and a reverse proxy server
Loaded: loaded (/usr/lib/systemd/system/nginx.service; enabled; preset: enabled)
Active: active (running) since Wed 2025-05-14 17:28:34 UTC; 14min ago
Docs: man:nginx(8)
Process: 176 ExecStartPre=/usr/sbin/nginx -t -q -g daemon on; master_process on; (code=exited, status=0/SUCCESS)
Process: 186 ExecStart=/usr/sbin/nginx -g daemon on; master_process on; (code=exited, status=0/SUCCESS)
Main PID: 203 (nginx)
Tasks: 13 (limit: 8133)
Memory: 9.8M ()
CGroup: /system.slice/nginx.service
├─203 "nginx: master process /usr/sbin/nginx -g daemon on; master_process on;"
├─204 "nginx: worker process"
├─205 "nginx: worker process"
├─206 "nginx: worker process"
├─207 "nginx: worker process"
├─208 "nginx: worker process"
├─209 "nginx: worker process"
├─210 "nginx: worker process"
├─212 "nginx: worker process"
├─213 "nginx: worker process"
├─214 "nginx: worker process"
├─215 "nginx: worker process"
└─216 "nginx: worker process"
May 14 17:28:34 VINSMOKE systemd[1]: Starting nginx.service - A high performance web server and a reverse proxy server...

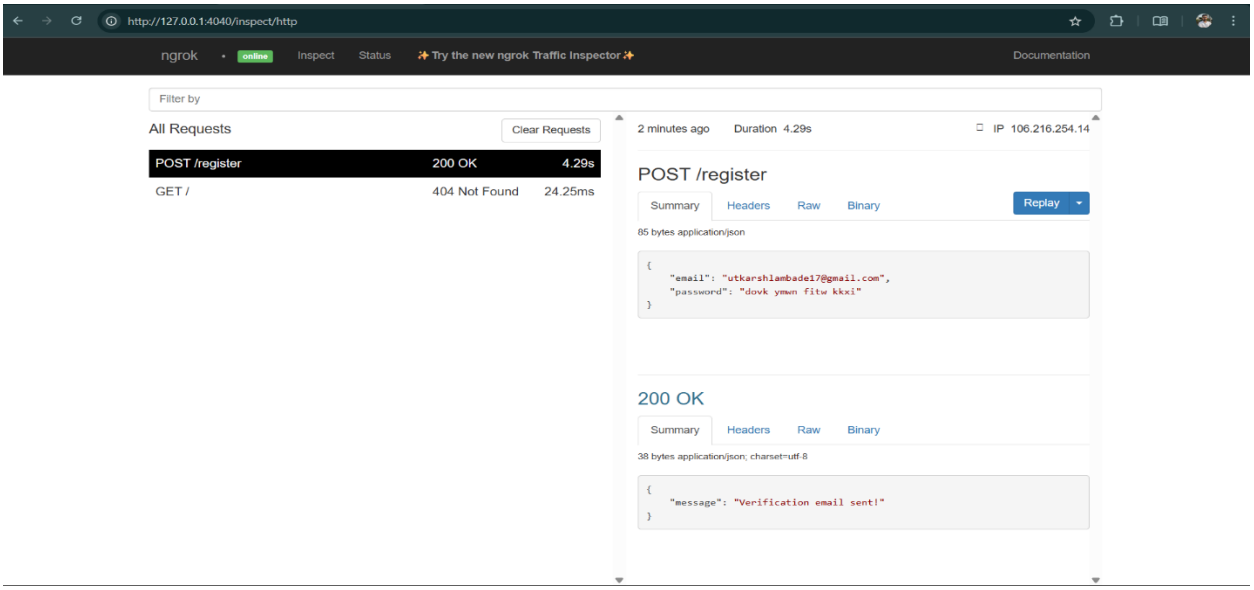
2. Backend Development using Express.js (4 Marks)

We created a robust backend server using Express.js with the following endpoints:

- `POST /register` - Initiates email registration
- `POST /verify-registration` - Verifies token, creates user
- `POST /login` - Logs in and returns JWT
- `GET /protected` - Accessible only via valid JWT

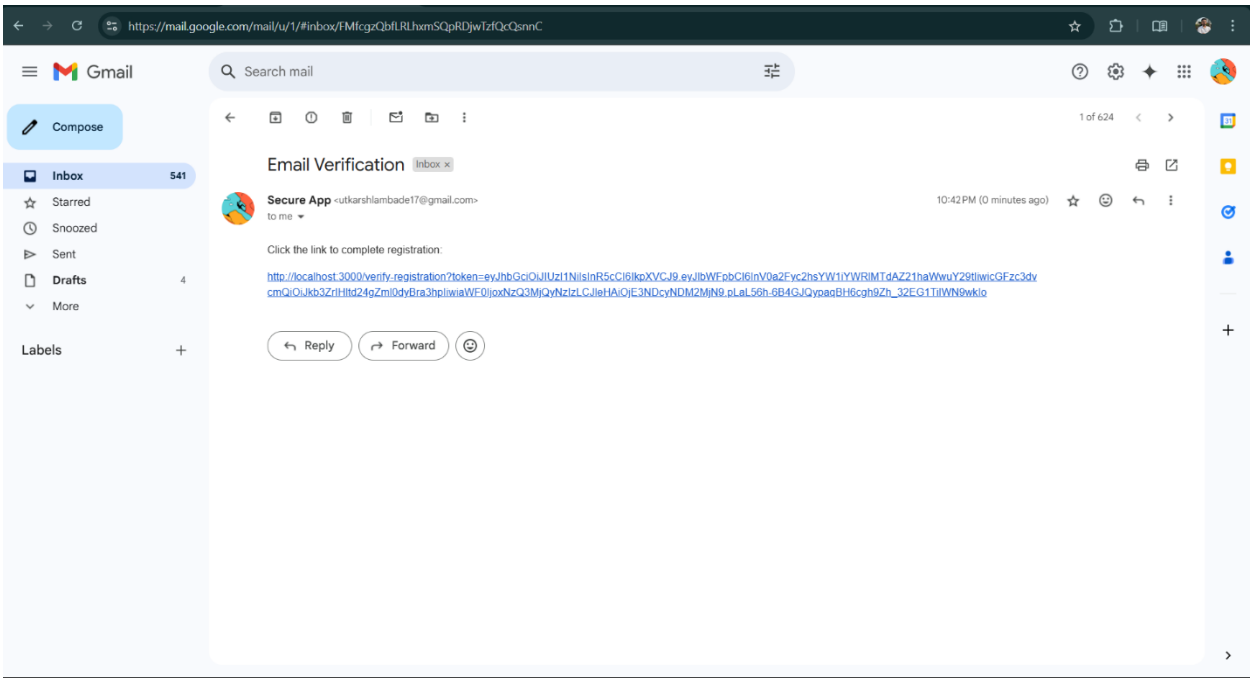
1.Email registration





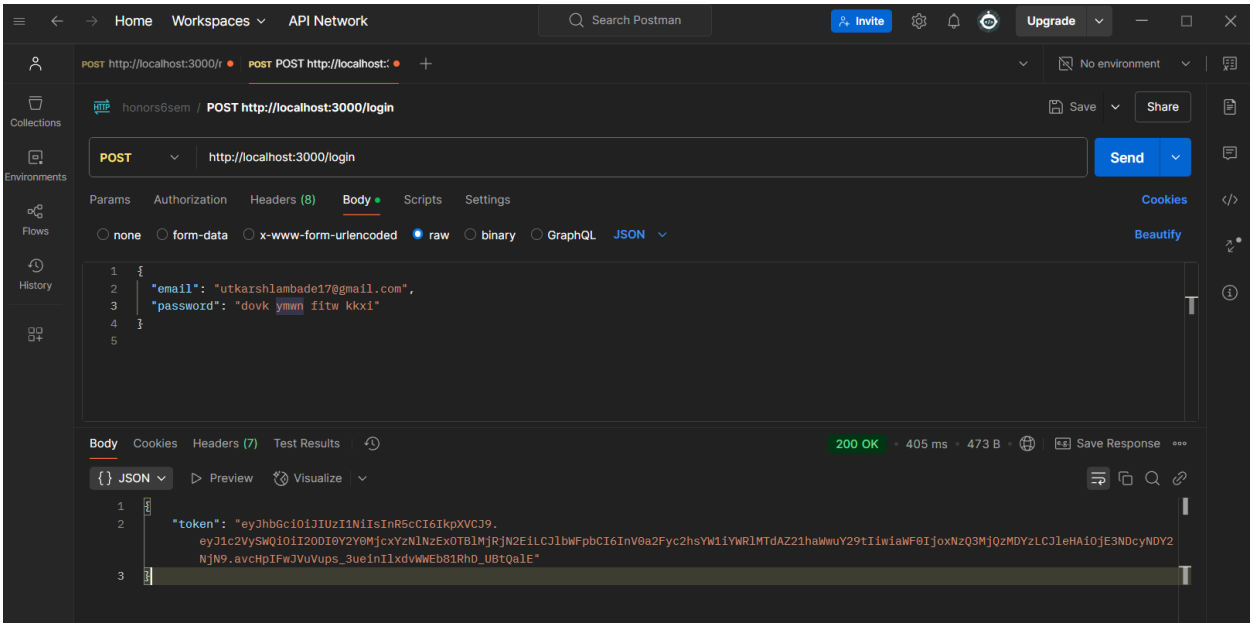
2. Verifying token and Creating user

```
const verifyRegistration = async (req, res) => {
  const { token } = req.query;
  try {
    const payload = jwt.verify(token, process.env.JWT_SECRET);

    let user = await User.findOne({ email: payload.email });
    if (user) return res.status(400).json({ message: 'User already exists' });

    user = new User({
      email: payload.email,
      password: payload.password,
      isVerified: true,
    });

    await user.save();

    res.status(201).json({ message: 'User registered successfully!' });
  } catch (err) {
    console.error(err);
    res.status(400).json({ error: 'Invalid or expired token' });
  }
};
```



3. Logs in and returns JWT

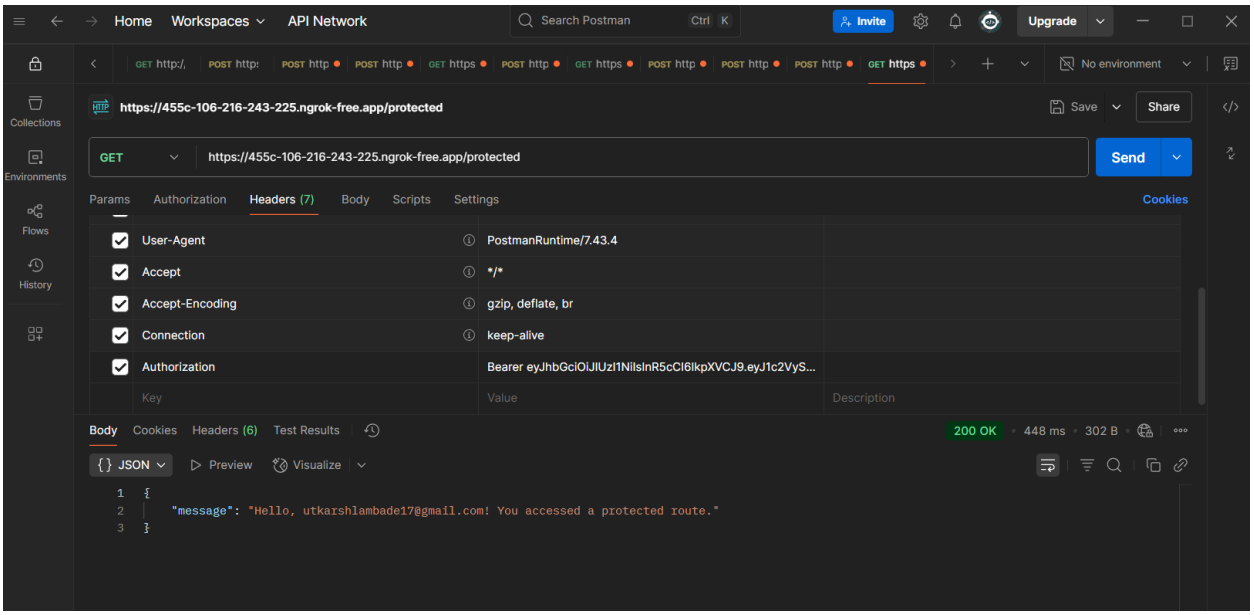
```
//Login Controller
const login = async (req, res) => {
  const { email, password } = req.body;
  try {
    // 1. Find user
    const user = await User.findOne({ email });
    if (!user) return res.status(400).json({ message: 'Invalid credentials' });

    // 2. Check if verified
    if (!user.isVerified) {
      return res.status(403).json({ message: 'Please verify your email first' });
    }

    // 3. Compare passwords
    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) return res.status(400).json({ message: 'Invalid credentials' });

    // 4. Sign JWT
    const token = jwt.sign(
      { userId: user._id, email: user.email },
      process.env.JWT_SECRET,
      { expiresIn: '1h' }
    );

    res.status(200).json({ token });
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: 'Server error' });
  }
};

```



4. Accessible only via valid JWT

src > middleware > Auth.js > ...

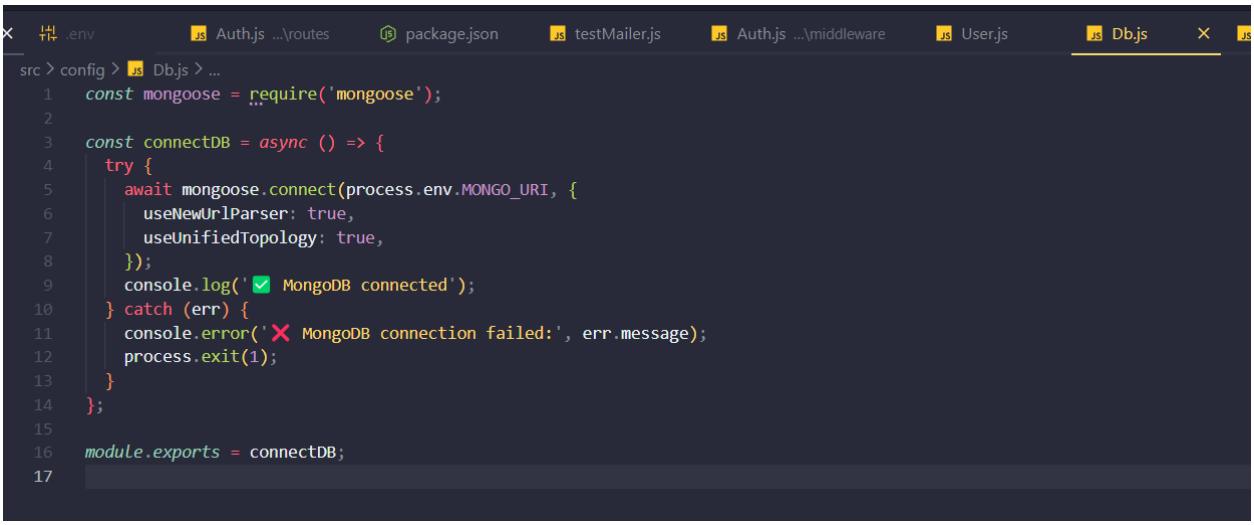
```
1   // Implement Login & Protected Routes
2   const jwt = require('jsonwebtoken');
3
4   module.exports = (req, res, next) => {
5     const authHeader = req.headers.authorization;
6     if (!authHeader) return res.status(401).json({ message: 'No token provided' });
7
8     const token = authHeader.split(' ')[1];
9     try {
10      const payload = jwt.verify(token, process.env.JWT_SECRET);
11      req.user = payload;           // you can access payload.email later
12      next();
13    } catch (err) {
14      return res.status(403).json({ message: 'Invalid or expired token' });
15    }
16  };
17
```

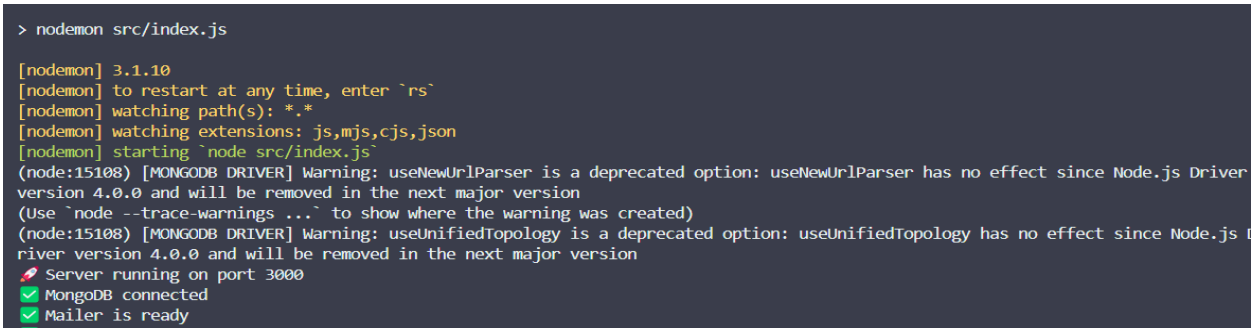


3. MongoDB Atlas Integration

MongoDB Atlas was used for storing user credentials and tokens. The database is securely connected via Mongoose.

1.Connection setup

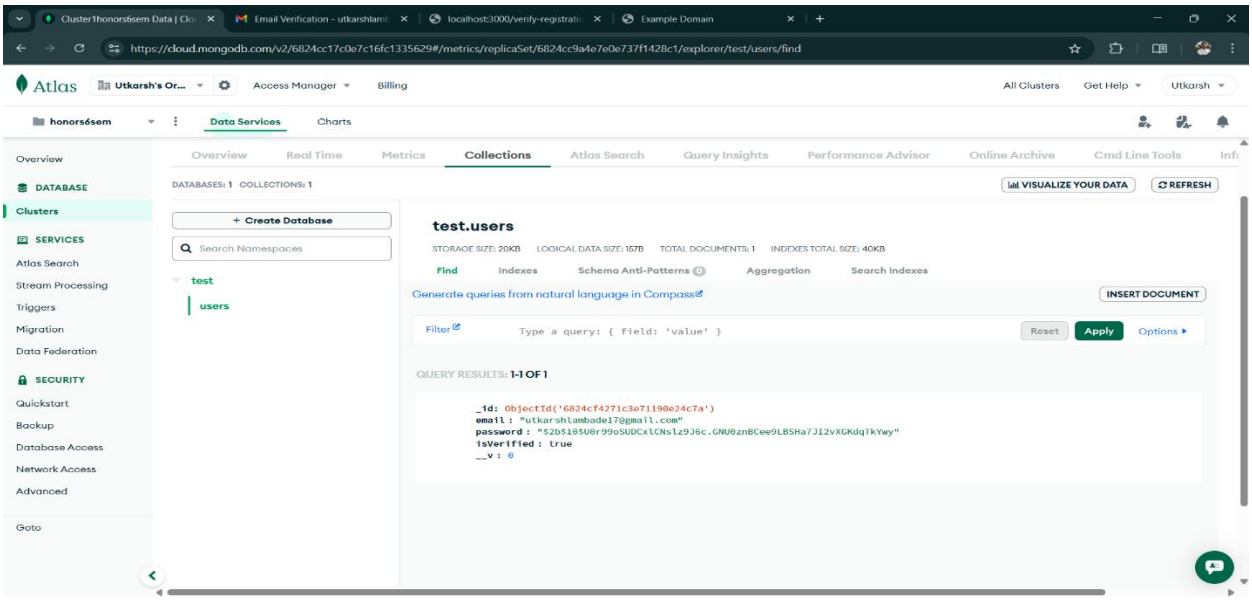




2.Storing User in Db

src > models > User.js > ...

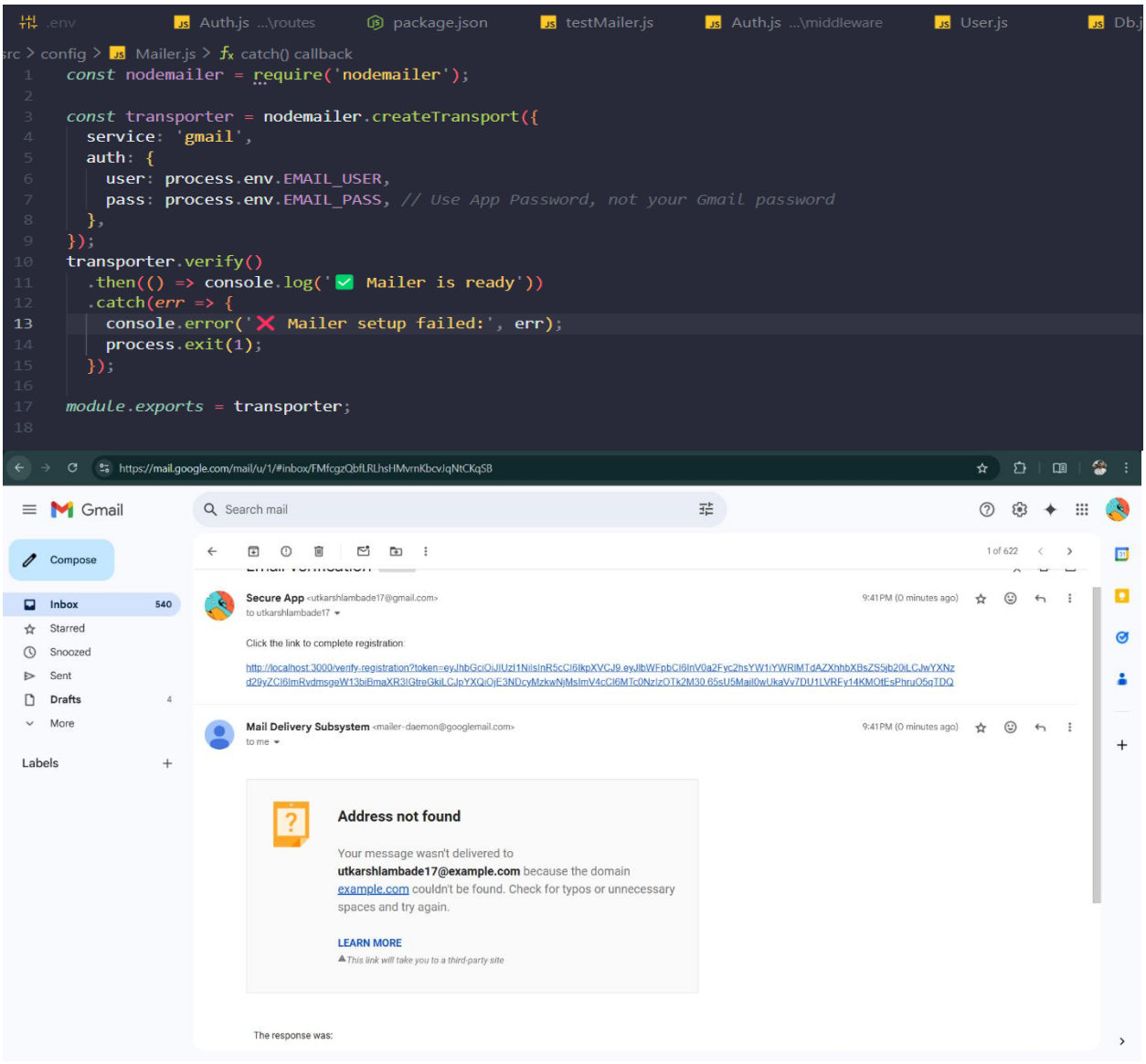
```
const mongoose = require('mongoose');
const bcrypt = require('bcryptjs');

const UserSchema = new mongoose.Schema({
  email:       { type: String, required: true, unique: true },
  password:    { type: String, required: true },
  isVerified: { type: Boolean, default: false },
  resetToken:       String,
  resetTokenExpiry: Date
});

// Hash password before save (for both new and reset)
UserSchema.pre('save', async function(next) {
  if (!this.isModified('password')) return next();
  const salt = await bcrypt.genSalt(10);
  this.password = await bcrypt.hash(this.password, salt);
  next();
});

module.exports = mongoose.model('User', UserSchema);
```



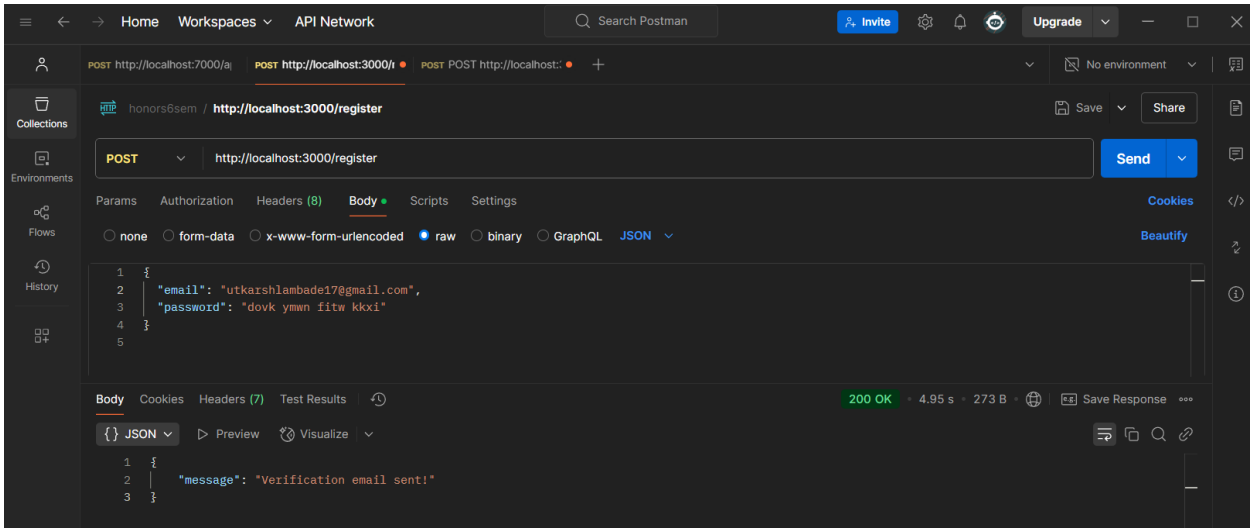
4. Email Registration Flow with Nodemailer (10 Marks)

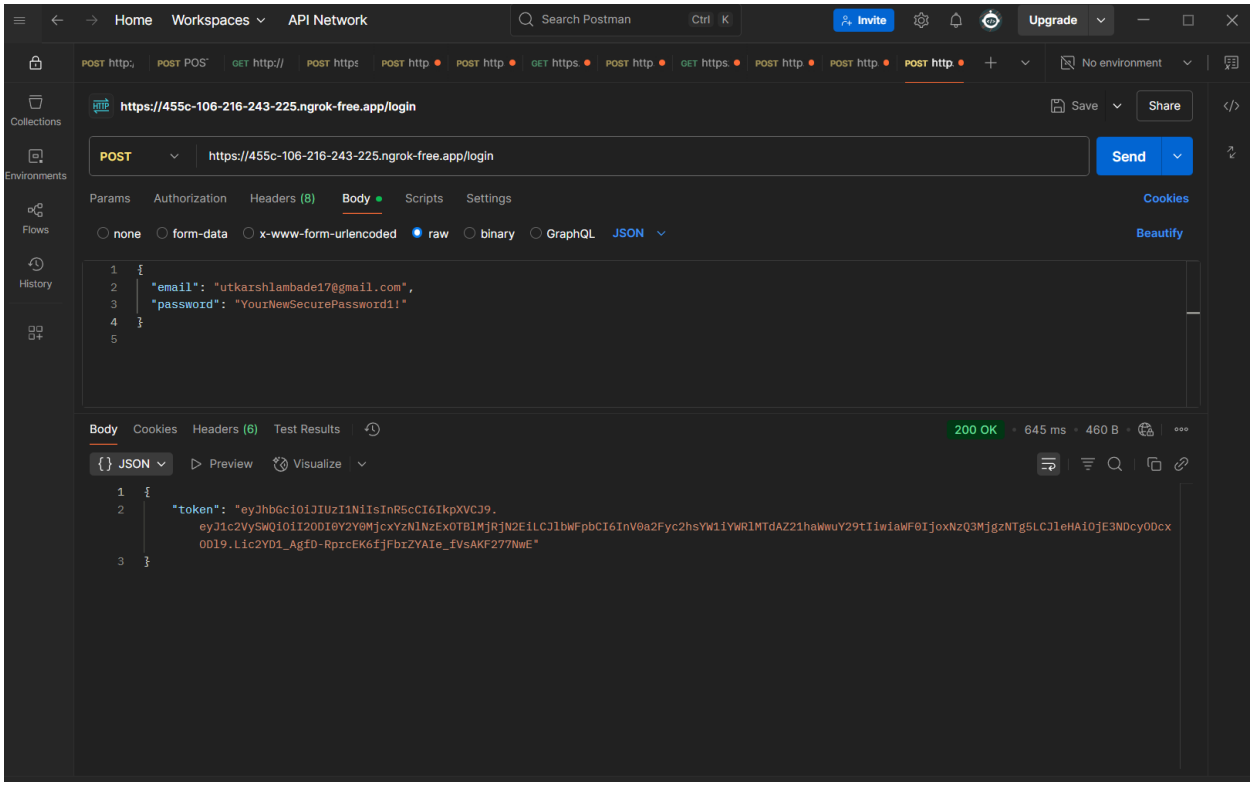
Implemented user registration flow:

- User registers with email(NodeMailer setup)
- Receives verification email with token link



- Link verifies token and creates


Home
Workspaces
API Network
Search Postman
Invite
Upgrade
No environment
Collections
Environments
Flows
History
honors6sem / http://localhost:3000/register
Save
Share
POST
http://localhost:3000/register
Send
Params
Authorization
Headers (8)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"email": "utkarshlambade17@gmail.com",
"password": "dovk ymwn fitw kkxi"
}
Body
Cookies
Headers (7)
Test Results
200 OK
4.95 s
273 B
Save Response
JSON
Preview
Visualize
{
"message": "Verification email sent!"
}

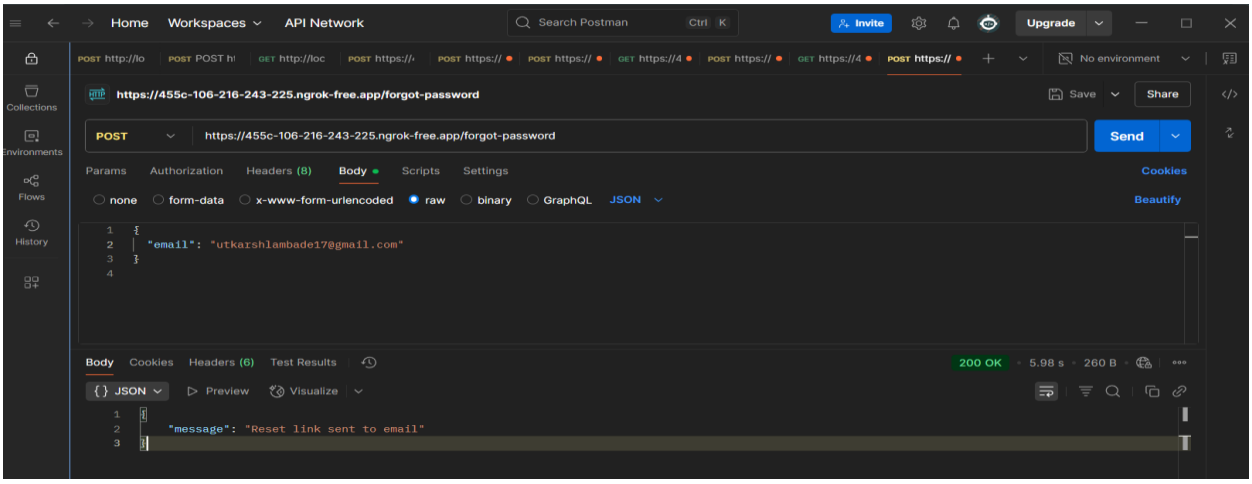

Home
Workspaces
API Network
Search Postman
Ctrl K
Invite
Upgrade
No environment
Collections
Environments
Flows
History
https://455c-106-216-243-225.ngrok-free.app/login
Save
Share
POST
https://455c-106-216-243-225.ngrok-free.app/login
Send
Params
Authorization
Headers (8)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"email": "utkarshlambade17@gmail.com",
"password": "YourNewSecurePassword1!"
}
Body
Cookies
Headers (6)
Test Results
200 OK
645 ms
460 B
JSON
Preview
Visualize
{
"token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJ1c2VySWQiOiI2ODI0Y2Y0MjcxYzNlNzExOTBlMjRjN2EiLCJlbWFpbCI6InV0a2Fyc2hsYW1iYWRlMTdAZ21haWwuY29tIiwiaWF0IjoxNzQ3MjgzNTg5LCJleHAiOjE3NDcyODcx
ODl9.Lic2YD1_AgfD-RprcEK6fjFbrZYAIe_fVsAKF277NwE"
}

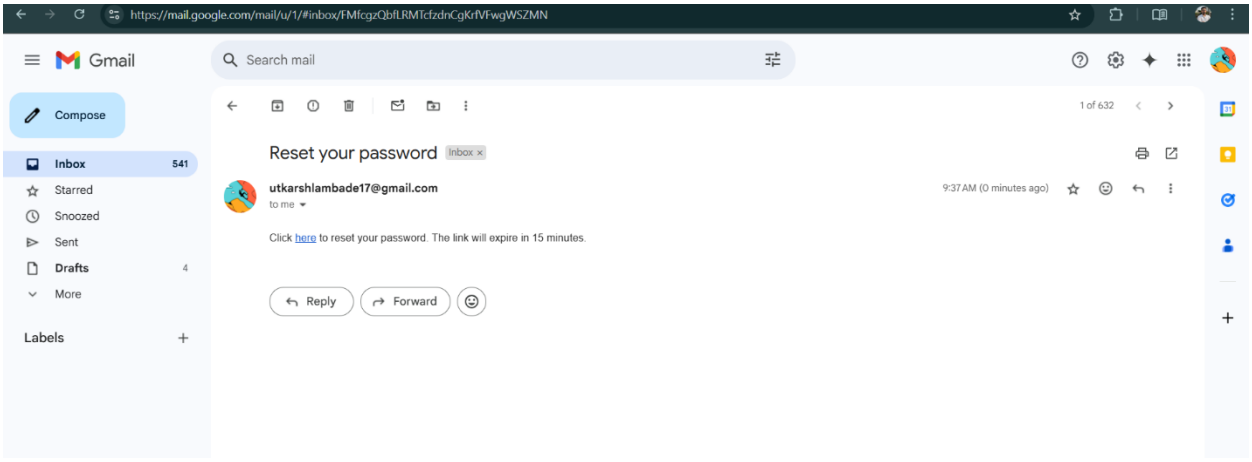
Password reset flow:

- User requests reset

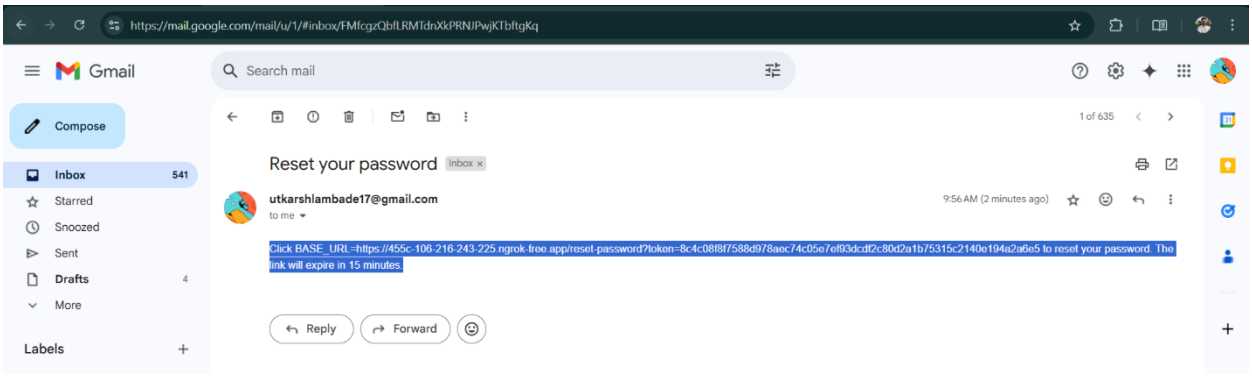
```
// Reset Password
const resetPassword = async (req, res) => {
  const { token, newPassword } = req.body;
  try {
    const user = await User.findOne({
      resetToken: token,
      resetTokenExpiry: { $gt: Date.now() }
    });

    if (!user) return res.status(400).json({ message: 'Invalid or expired token' });

    user.password = newPassword;
    user.resetToken = undefined;
    user.resetTokenExpiry = undefined;
    await user.save();

    res.status(200).json({ message: 'Password updated successfully' });
  } catch (err) {
    res.status(500).json({ message: 'Error resetting password', error: err.message });
  }
};
```

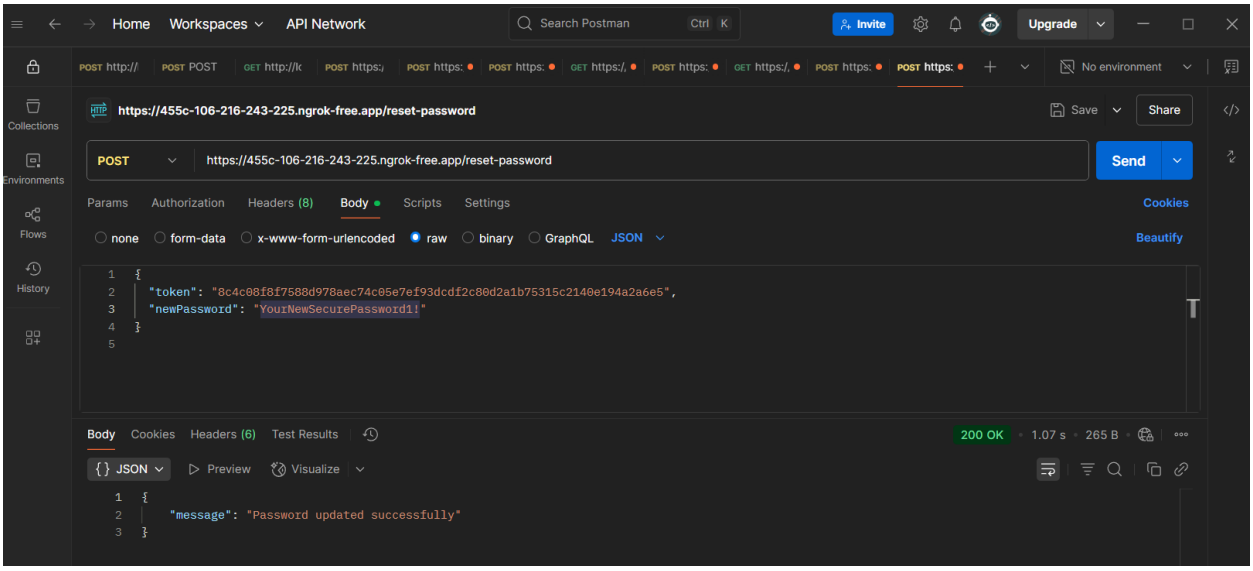
- Mail with reset token sent





- Token link resets password

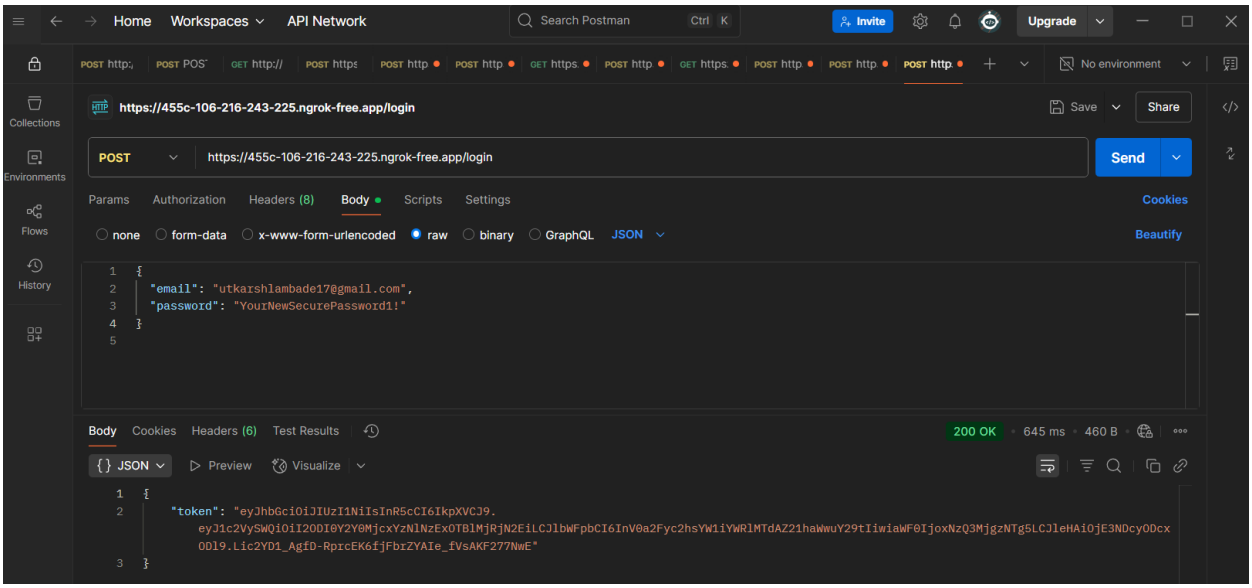


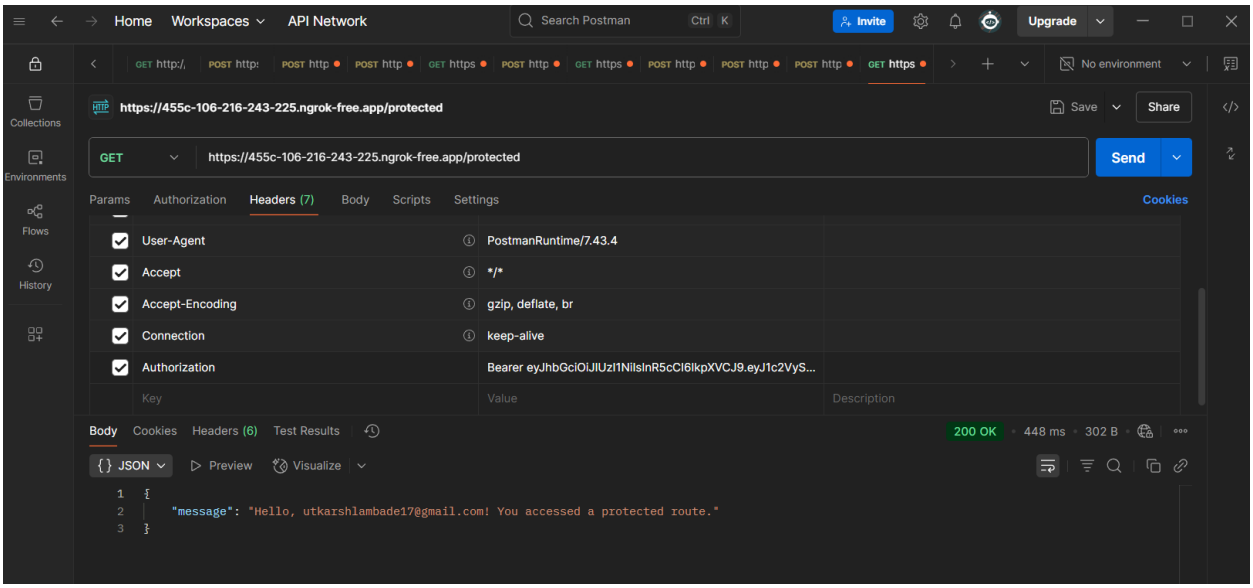


5. JWT Authentication Mechanism

JSON Web Tokens (JWT) are used to authenticate and authorize users.

- On successful login, JWT is issued
- Token is required to access protected route





6. Login Functionality

Implemented via `POST /login`. Compares hashed password and returns JWT on success.

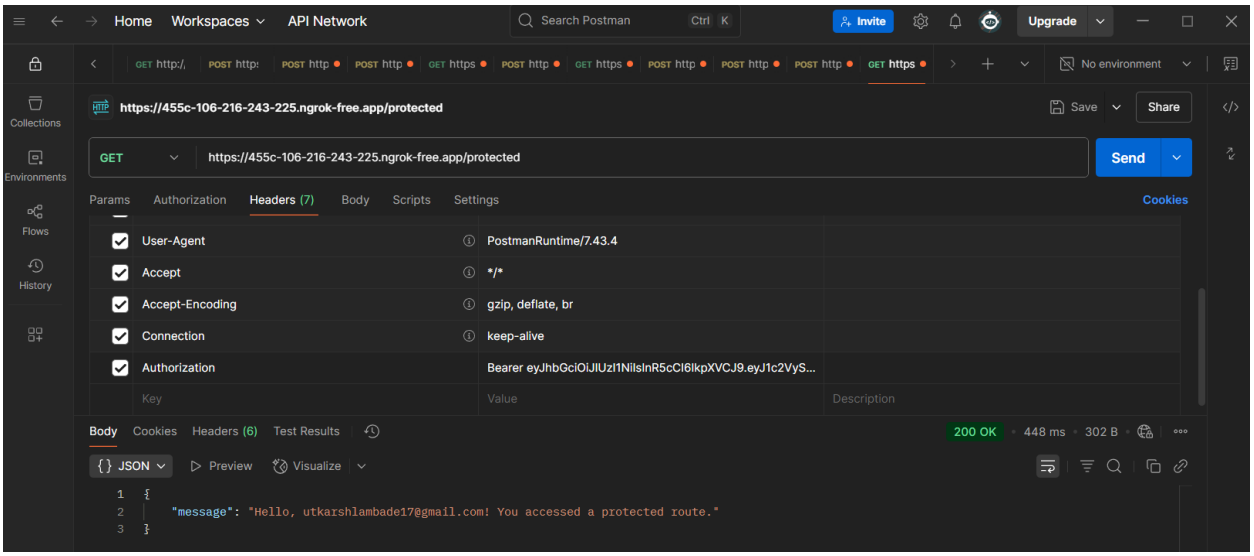
src > middleware > Auth.js > ...

```
// Implement Login & Protected Routes
const jwt = require('jsonwebtoken');

module.exports = (req, res, next) => {
  const authHeader = req.headers.authorization;
  if (!authHeader) return res.status(401).json({ message: 'No token provided' });

  const token = authHeader.split(' ')[1];
  try {
    const payload = jwt.verify(token, process.env.JWT_SECRET);
    req.user = payload;           // you can access payload.email later
    next();
  } catch (err) {
    return res.status(403).json({ message: 'Invalid or expired token' });
  }
};

```

```
//Login Controller
const login = async (req, res) => {
  const { email, password } = req.body;
  try {
    // 1. Find user
    const user = await User.findOne({ email });
    if (!user) return res.status(400).json({ message: 'Invalid credentials' });

    // 2. Check if verified
    if (!user.isVerified) {
      return res.status(403).json({ message: 'Please verify your email first' });
    }

    // 3. Compare passwords
    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) return res.status(400).json({ message: 'Invalid credentials' });

    // 4. Sign JWT
    const token = jwt.sign(
      { userId: user._id, email: user.email },
      process.env.JWT_SECRET,
      { expiresIn: '1h' }
    );

    res.status(200).json({ token });
  } catch (err) {
    console.error(err);
    res.status(500).json({ message: 'Server error' });
  }
};
```

7. Protected Route Access using JWT

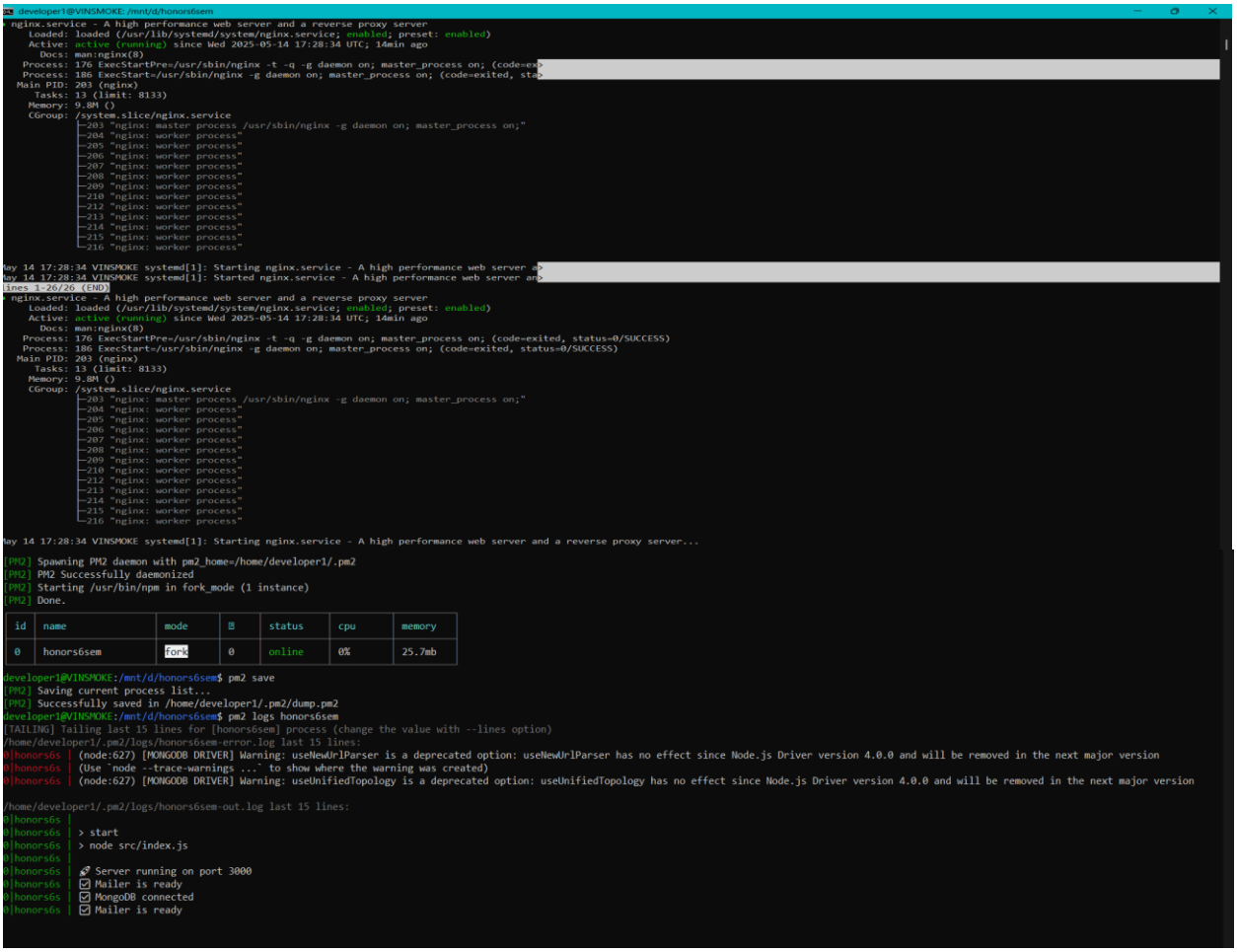
`GET /protected` endpoint validates JWT before serving response.

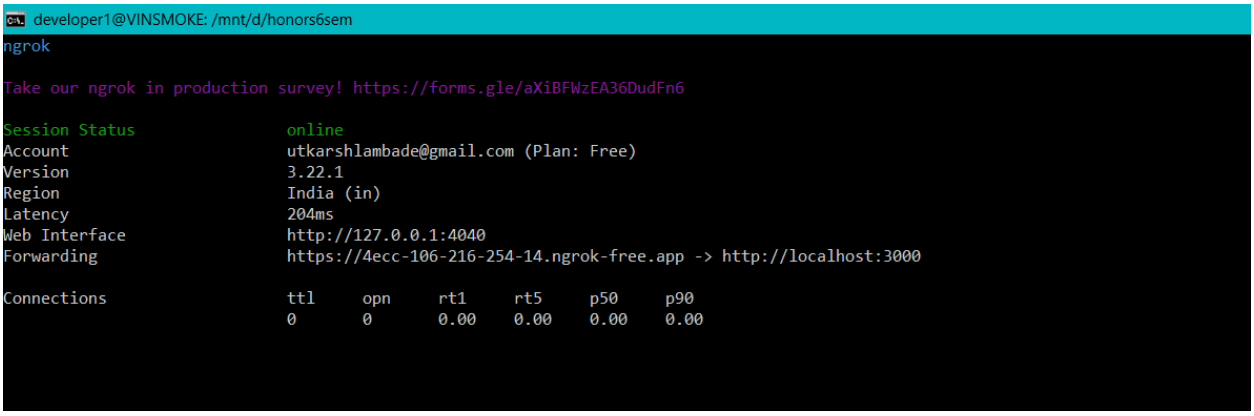



```
const express = require('express');
const router = express.Router();

const {
  register,
  verifyRegistration,
  login,
  forgotPassword,
  resetPassword
} = require('../controllers/authController');
const authMiddleware = require('../middleware/auth');

router.post('/register', register);
router.get('/verify-registration', verifyRegistration);
router.post('/login', login);                    // ← ensure this line is pres
router.post('/forgot-password', forgotPassword);
router.post('/reset-password', resetPassword);
router.get('/protected', authMiddleware, (req, res) => {
  res.json({
    message: `Hello, ${req.user.email}! You accessed a protected route.`
  });
});

module.exports = router;
```

8. Nginx Reverse Proxy & HTTPS with Let's Encrypt

Nginx forwards requests to Node.js app and is secured using Certbot-generated HTTPS certificates.





9. One-Way Password Hashing

Passwords are hashed using bcrypt before being saved to the database. This ensures security against database breaches.

src > models > User.js > ...

```
const mongoose = require('mongoose');
const bcrypt = require('bcryptjs');

const UserSchema = new mongoose.Schema({
  email:       { type: String, required: true, unique: true },
  password:    { type: String, required: true },
  isVerified: { type: Boolean, default: false },
  resetToken:        String,
  resetTokenExpiry: Date
});

// Hash password before save (for both new and reset)
UserSchema.pre('save', async function(next) {
  if (!this.isModified('password')) return next();
  const salt = await bcrypt.genSalt(10);
  this.password = await bcrypt.hash(this.password, salt);
  next();
});

module.exports = mongoose.model('User', UserSchema);

```

10. Conclusion

All components were successfully implemented and tested using Postman. The system supports secure user registration, login, email verification, password reset, and protected route access, making it a full-fledged secure authentication system suitable for production environments.

GitHub Repository

- Repository Name: `secure-auth-mailer-service`
- LINK - https://github.com/utkarshjain2004/express-secure-auth-service
- Description: A secure user authentication system built with Node.js, Express, MongoDB Atlas, JWT, and Nodemailer, deployed via Nginx on WSL Ubuntu with HTTPS support.