

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 (4 Marks)
 2. Backend Development using Express.js (4 Marks)
 3. MongoDB Atlas Integration (4 Marks)
 4. Email Registration Flow with Nodemailer (10 Marks)
 5. JWT Authentication Mechanism (10 Marks)
 6. Login Functionality (3 Marks)
 7. Protected Route Access using JWT (3 Marks)
 8. Nginx Reverse Proxy & HTTPS with Let's Encrypt (2 Marks)
 9. One-Way Password Hashing (2 Marks)
 10. Conclusion
-

1. Environment Setup on WSL Ubuntu

[illegible]

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

```
[PM2] Spawning PM2 daemon on pm2_home=/home/developer1/.pm2
[PM2] PM2 Successfully daemonized
[PM2] Starting /usr/bin/npm in fork_mode (1 instance)
[PM2] Done.
```

id	name	mode	B	status	cpu	memory
0	honors6sem	Fork	0	online	0%	25.7mb

```
developer1@VINSHUKKI:~/mnt/d/honors6sem$ pm2 save
[PM2] Saving current process list...
[PM2] Successfully saved in /home/developer1/.pm2/dump.pm2
developer1@VINSHUKKI:~/mnt/d/honors6sem$ pm2 logs honors6sem
[TAILING] Tailing last 15 lines for [honors6sem] process (change the value with --lines option)
/home/developer1/.pm2/logs/honors6sem-error.log last 15 lines:
0|honors6sem [node:627] [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTopology has no effect since Node.js Driver version 4.0.0 and will be removed in the next major version
0|honors6sem [Use mode --trace-warnings ...] to show where the warning was created)
0|honors6sem [node:627] [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTopology has no effect since Node.js Driver version 4.0.0 and will be removed in the next major version
/home/developer1/.pm2/logs/honors6sem-out.log last 15 lines:
0|honors6sem
0|honors6sem > start
0|honors6sem > node src/index.js
0|honors6sem
0|honors6sem Server running on port 3000
0|honors6sem [x] Mailer is ready
0|honors6sem [x] MongoDB connected
0|honors6sem [x] Mailer is ready
```

```
developer1@VINSMOKE:/mnt/d/honorsitem
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...
May 14 17:28:34 VINSMOKE systemd[1]: Started nginx.service - A high performance web server and a reverse proxy server...
Linux 5.10.200 (EUD)

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...
```

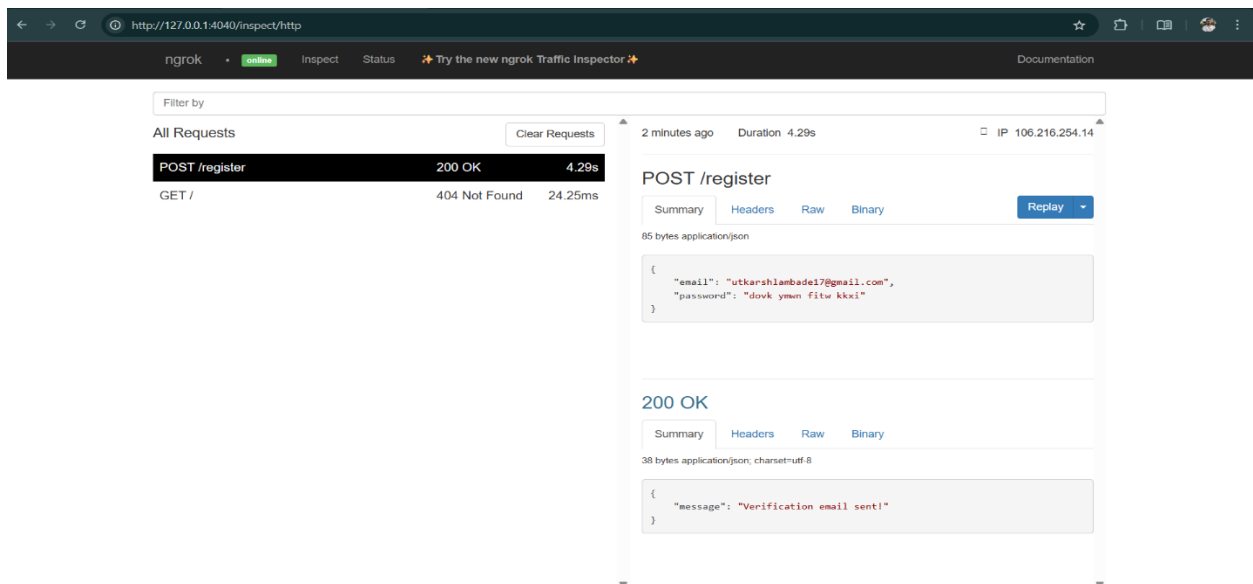
2. Backend Development using Express.js (4 Marks)

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

```
AuthController.js x .env Auth.js ...routes package.json testMailer.js Auth.js ...middleware User.js Db
src > controllers > AuthController.js > forgotPassword
1 const crypto = require('crypto');
2 const jwt = require('jsonwebtoken');
3 const bcrypt = require('bcryptjs');
4 const User = require('../models/User');
5 const transporter = require('../config/Mailer');
6
7 // ...now your exports.register, exports.verifyRegistration, exports.Login...
8 const register = async (req, res) => {
9   const { email, password } = req.body;
10  try {
11    const token = jwt.sign({ email, password }, process.env.JWT_SECRET, { expiresIn: '15m' });
12
13    const verificationLink = `${process.env.BASE_URL}/verify-registration?token=${token}`;
14
15    await transporter.sendMail({
16      from: "Secure App" <${process.env.EMAIL_USER}>,
17      to: email,
18      subject: 'Email Verification',
19      html: `<p>Click the link to complete registration:</p><a href="${verificationLink}">${verificationLink}</a>`,
20    });
21    console.log(' Using BASE_URL =', process.env.BASE_URL);
22
23    res.status(200).json({ message: 'Verification email sent!' });
24  } catch (err) {
25    console.error(err);
26    res.status(500).json({ error: 'Failed to send verification email' });
27  }
28  };
29
```

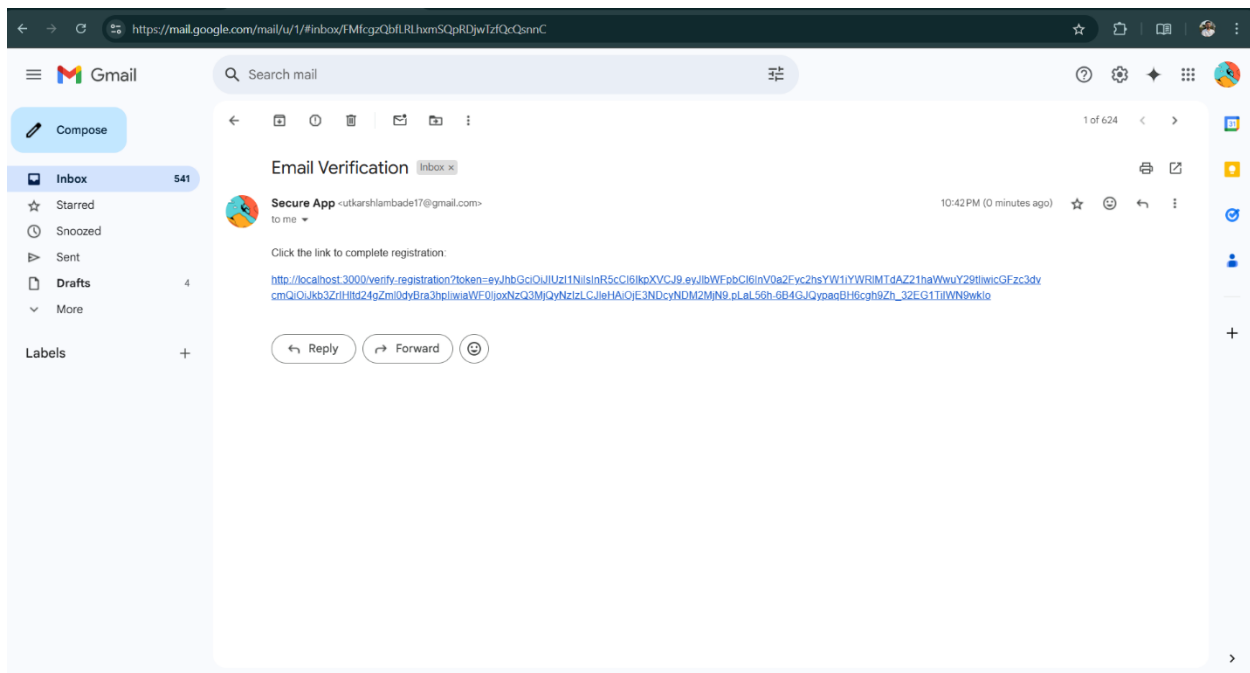


2. Verifying token and Creating user

```

30  const verifyRegistration = async (req, res) => {
31    const { token } = req.query;
32    try {
33      const payload = jwt.verify(token, process.env.JWT_SECRET);
34
35      let user = await User.findOne({ email: payload.email });
36      if (user) return res.status(400).json({ message: 'User already exists' });
37
38      user = new User({
39        email: payload.email,
40        password: payload.password,
41        isVerified: true,
42      });
43
44      await user.save();
45
46      res.status(201).json({ message: 'User registered successfully!' });
47    } catch (err) {
48      console.error(err);
49      res.status(400).json({ error: 'Invalid or expired token' });
50    }
51  };

```



3. Logs in and returns JWT

```

53 //Login Controller
54 const login = async (req, res) => {
55   const { email, password } = req.body;
56   try {
57     // 1. Find user
58     const user = await User.findOne({ email });
59     if (!user) return res.status(400).json({ message: 'Invalid credentials' });
60
61     // 2. Check if verified
62     if (!user.isVerified) {
63       return res.status(403).json({ message: 'Please verify your email first' });
64     }
65
66     // 3. Compare passwords
67     const isMatch = await bcrypt.compare(password, user.password);
68     if (!isMatch) return res.status(400).json({ message: 'Invalid credentials' });
69
70     // 4. Sign JWT
71     const token = jwt.sign(
72       { userId: user._id, email: user.email },
73       process.env.JWT_SECRET,
74       { expiresIn: '1h' }
75     );
76
77     res.status(200).json({ token });
78   } catch (err) {
79     console.error(err);
80     res.status(500).json({ message: 'Server error' });
81   }
82 };
83

```

The screenshot shows the Postman interface for a POST request to `http://localhost:3000/login`. The request body is a JSON object with the following fields:

```

{
  "email": "utkarshlambade17@gmail.com",
  "password": "dovk ymmn fitw kkxi"
}

```

The response is a 200 OK status with a response time of 405 ms and a body size of 473 B. The response body is a JSON object containing a token:

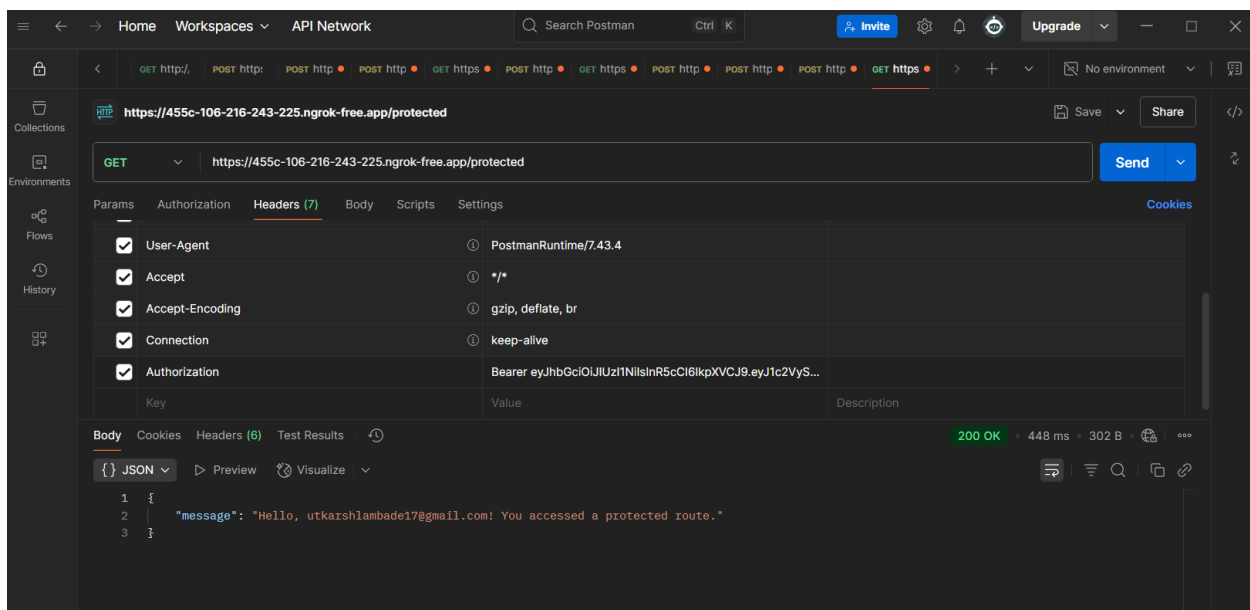
```

{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2V5SWQ1OjI0ODIyY2Y0MjcxcXZlbnN2eX0tYm9jaWw1IiwiaWF0IjoxNzQ3MjQzNDYzLCJleHAiOjE3NDcyNDYzNjQ3LCJ1b250a1EiOiJ1b250a1Ei"
}

```

4. Accessible only via valid JWT

```
src > middleware > JS 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

```
src > config > Db.js > ...
1  const mongoose = require('mongoose');
2
3  const connectDB = async () => {
4    try {
5      await mongoose.connect(process.env.MONGO_URI, {
6        useNewUrlParser: true,
7        useUnifiedTopology: true,
8      });
9      console.log('✅ MongoDB connected');
10     } catch (err) {
11       console.error('❌ MongoDB connection failed:', err.message);
12       process.exit(1);
13     }
14   };
15
16   module.exports = connectDB;
17
```

```
> nodemon src/index.js

[nodemon] 3.1.10
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node src/index.js`
(node:15108) [MONGODB DRIVER] Warning: useNewUrlParser is a deprecated option: useNewUrlParser has no effect since Node.js Driver version 4.0.0 and will be removed in the next major version
(node:15108) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTopology has no effect since Node.js Driver version 4.0.0 and will be removed in the next major version
🔥 Server running on port 3000
✅ MongoDB connected
✅ Mailer is ready
```

2.Storing User in Db


```

src > models > JS User.js > ...
1  const mongoose = require('mongoose');
2  const bcrypt = require('bcryptjs');
3
4  const UserSchema = new mongoose.Schema({
5    email: { type: String, required: true, unique: true },
6    password: { type: String, required: true },
7    isVerified: { type: Boolean, default: false },
8    resetToken: String,
9    resetTokenExpiry: Date
10 });
11
12 // Hash password before save (for both new and reset)
13 UserSchema.pre('save', async function(next) {
14   if (!this.isModified('password')) return next();
15   const salt = await bcrypt.genSalt(10);
16   this.password = await bcrypt.hash(this.password, salt);
17   next();
18 });
19
20 module.exports = mongoose.model('User', UserSchema);
21

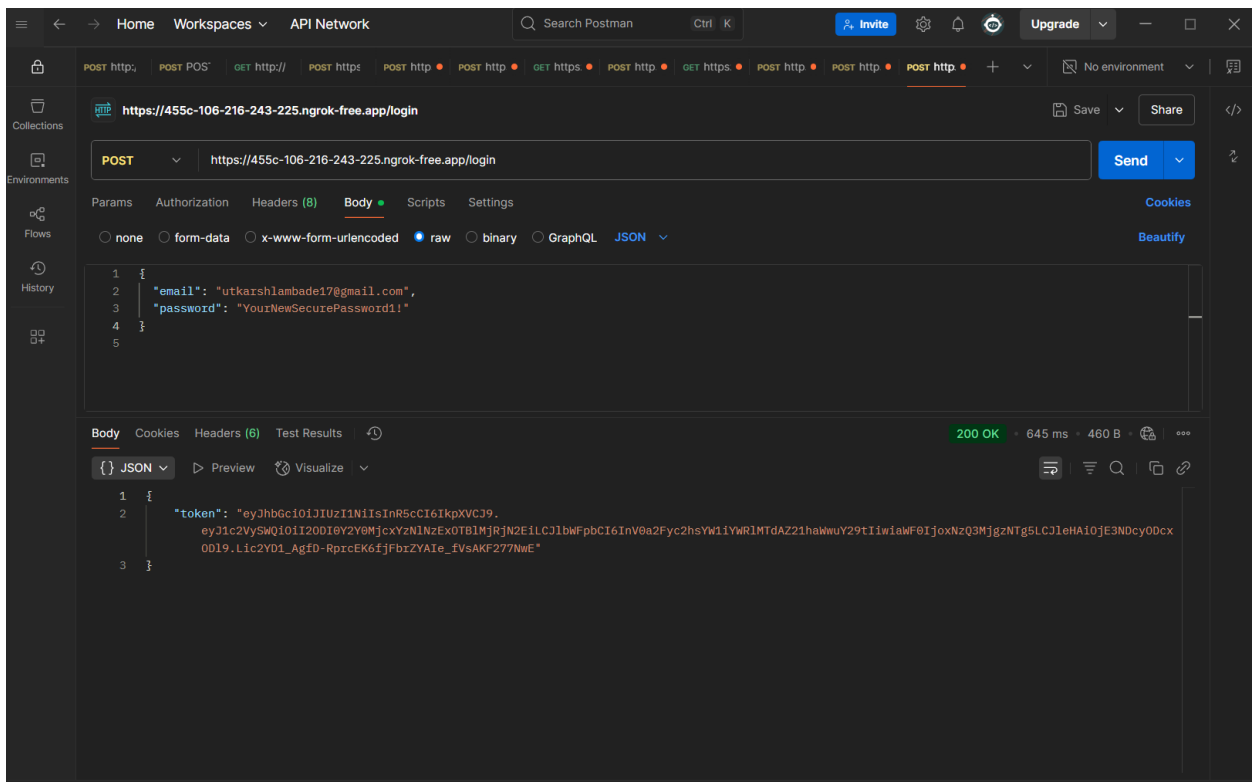
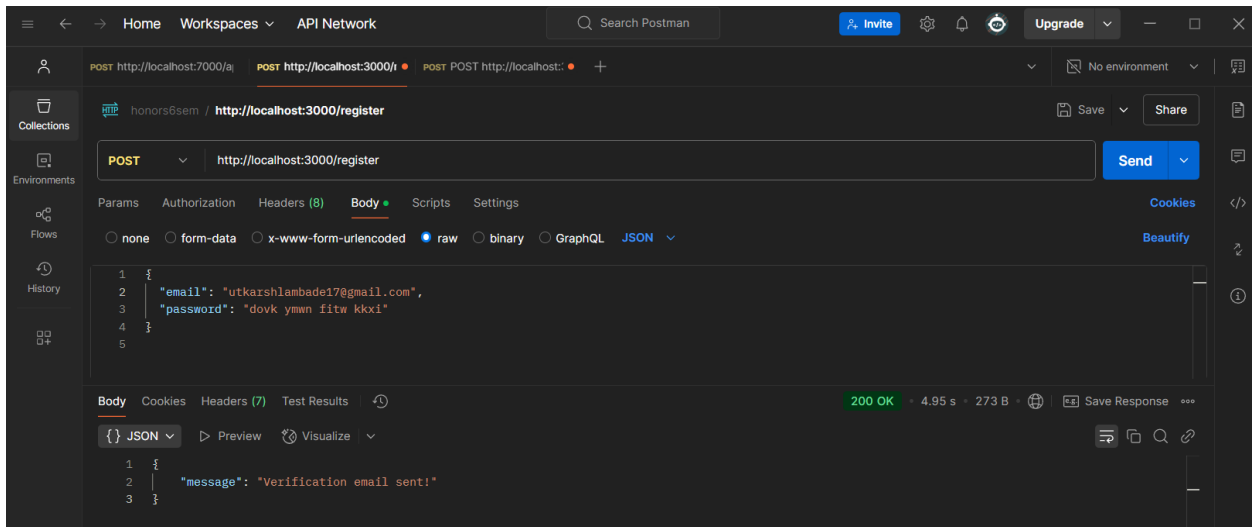
```

The screenshot shows the MongoDB Atlas web interface. The browser address bar displays the URL: <https://cloud.mongodb.com/v2/6824cc17c0e7c16fc1335629#/metrics/replicaSet/6824cc9a4e7e0e737f1428c1/explorer/test/users/find>. The interface includes a sidebar with navigation options like Overview, DATABASE, Clusters, SERVICES, and SECURITY. The main content area is titled 'test.users' and shows a single document in the 'users' collection. The document details are as follows:

Field	Value
_id	ObjectId('6824cf4271c3e71190e24c7a')
email	"utkarshlambade17@gmail.com"
password	"52b518508r99oSUDCXLcNsLz9J6c.GNU8zn8Cee9LBSHa7J12vXGKdq1kYwy"
isVerified	true
__v	0

The interface also includes a search bar, a filter input, and buttons for 'Visualize Your Data' and 'Refresh'.

- Link verifies token and creates

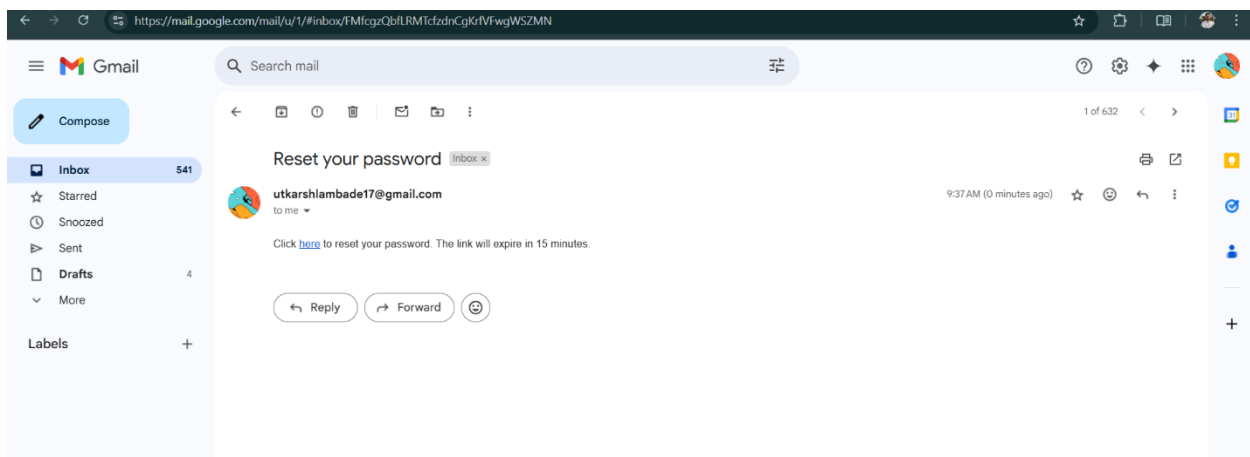
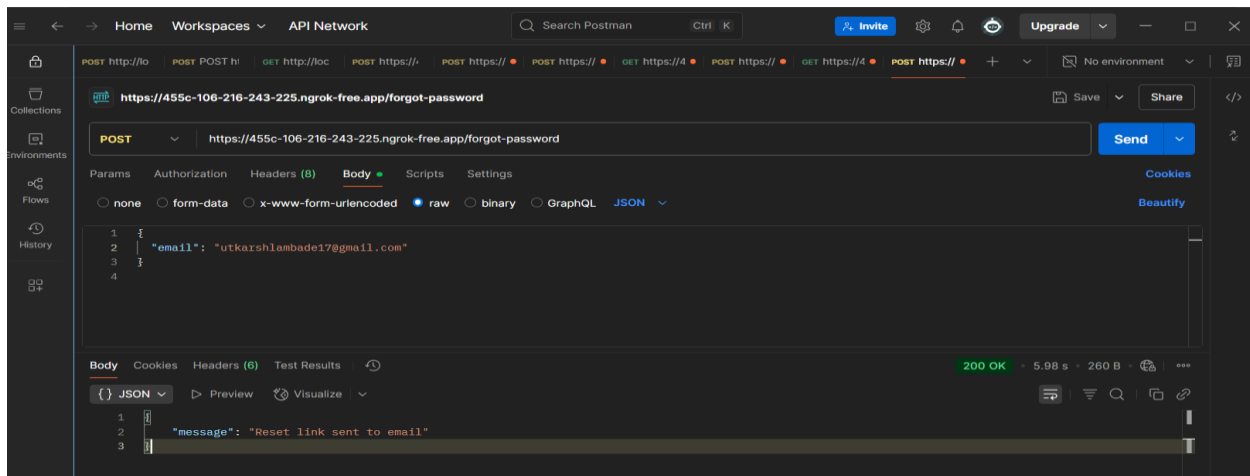


Password reset flow:

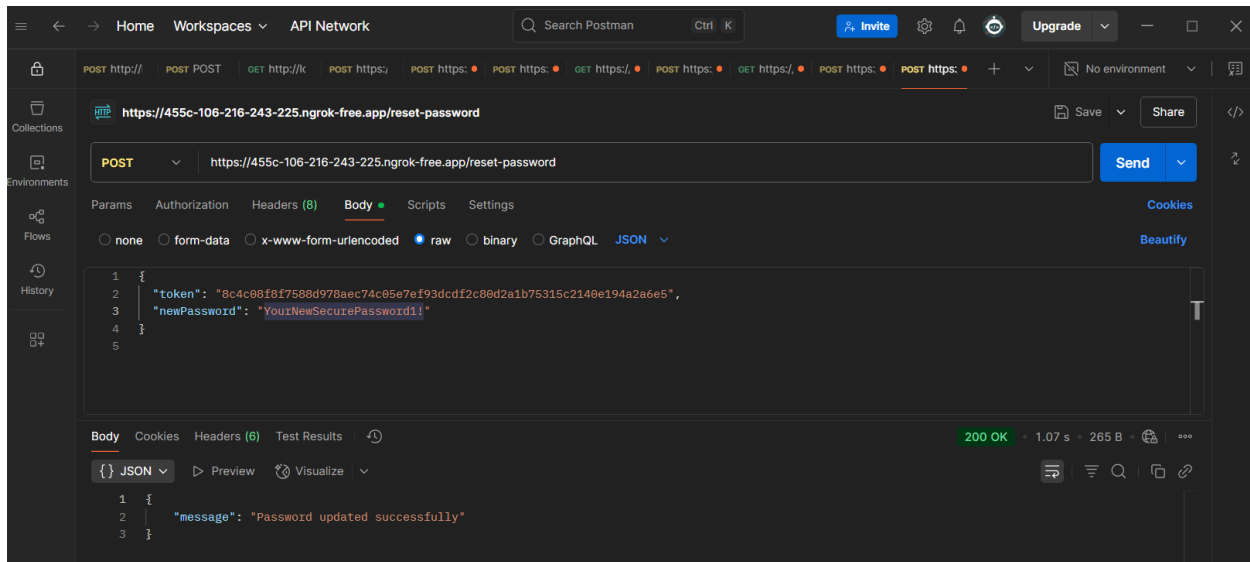
- User requests reset

```
113 // Reset Password
114 const resetPassword = async (req, res) => {
115   const { token, newPassword } = req.body;
116   try {
117     const user = await User.findOne({
118       resetToken: token,
119       resetTokenExpiry: { $gt: Date.now() }
120     });
121
122     if (!user) return res.status(400).json({ message: 'Invalid or expired token' });
123
124     user.password = newPassword;
125     user.resetToken = undefined;
126     user.resetTokenExpiry = undefined;
127     await user.save();
128
129     res.status(200).json({ message: 'Password updated successfully' });
130   } catch (err) {
131     res.status(500).json({ message: 'Error resetting password', error: err.message });
132   }
133 };
```

- Mail with reset token sent



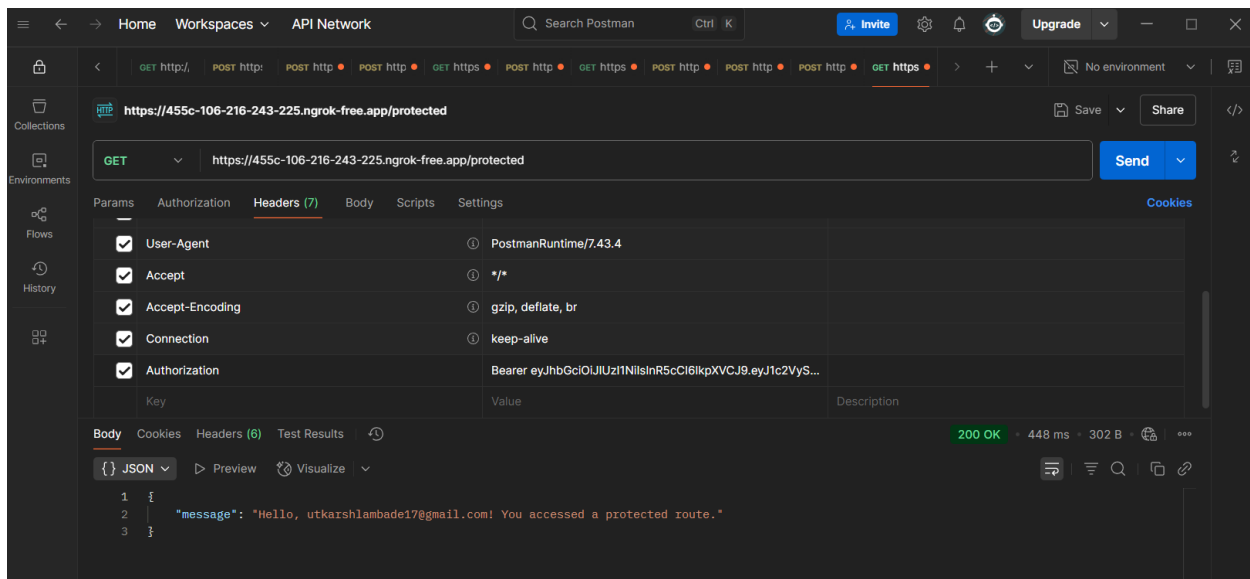
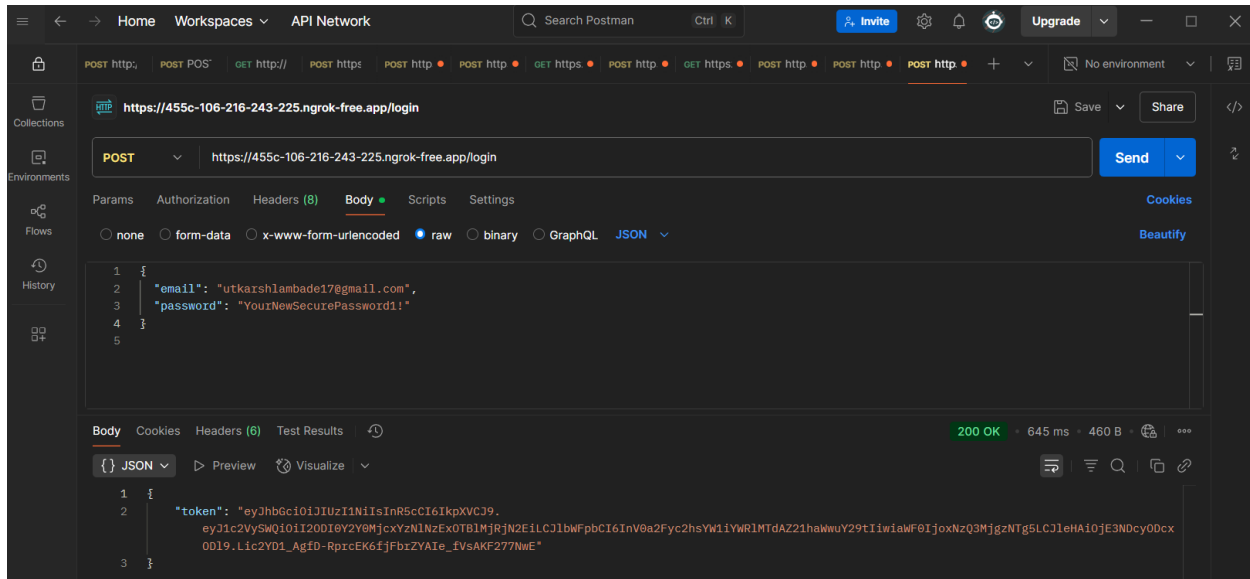
- Token link resets password



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 > .js 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
```

```
// 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' });
    }

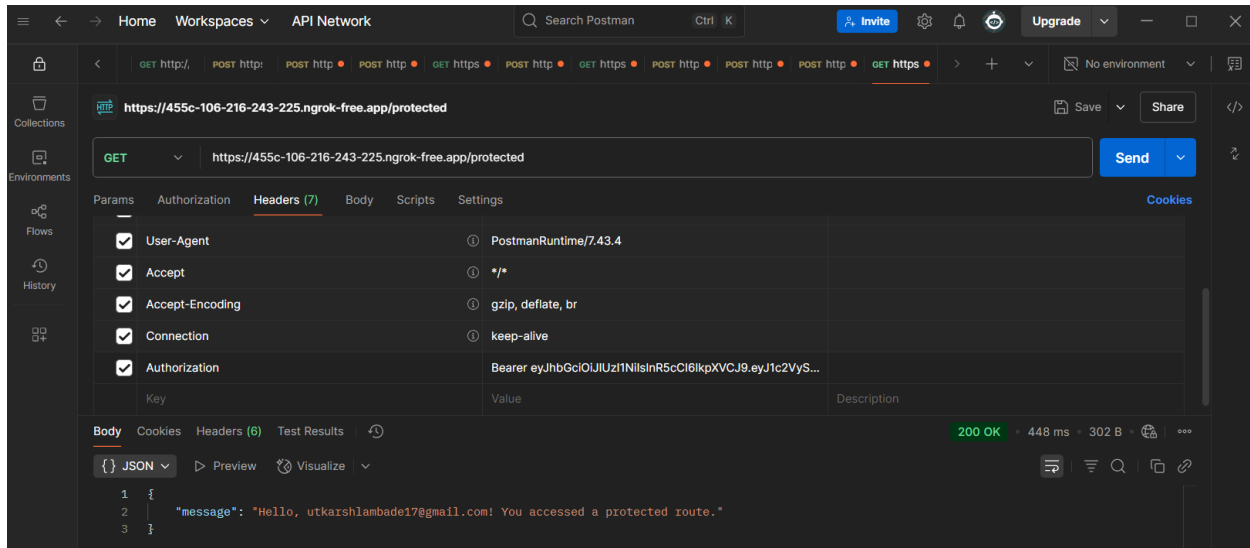
    // 3. Compare passwords
    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) return res.status(400).json({ message: 'Invalid credentials' });

    // 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.



```
1  const express = require('express');
2  const router = express.Router();
3
4  const {
5    register,
6    verifyRegistration,
7    login,
8    forgotPassword,
9    resetPassword
10 } = require('../controllers/authController');
11 const authMiddleware = require('../middleware/auth');
12
13 router.post('/register', register);
14 router.get('/verify-registration', verifyRegistration);
15 router.post('/login', login); // ← ensure this line is present
16 router.post('/forgot-password', forgotPassword);
17 router.post('/reset-password', resetPassword);
18 router.get('/protected', authMiddleware, (req, res) => {
19   res.json({
20     message: `Hello, ${req.user.email}! You accessed a protected route.`
21   });
22 });
23
24 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.

```
developer1@VINSMOKE:/mnt/d/honors6sem$ systemctl status nginx.service
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 (0)
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...
May 14 17:28:34 VINSMOKE systemd[1]: Started nginx.service - A high performance web server and a reverse proxy server.
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 (0)
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...
[PM2] Spawning PM2 daemon with pm2_home=/home/developer1/.pm2
[PM2] PM2 Successfully daemonized
[PM2] Starting /usr/bin/npm in fork_mode (1 instance)
[PM2] Done.



| id | name       | mode | B | status | cpu | memory |
|----|------------|------|---|--------|-----|--------|
| 0  | honors6sem | Fork | 0 | online | 0%  | 25.7mb |



developer1@VINSMOKE:/mnt/d/honors6sem$ pm2 save
[PM2] Saving current process list...
[PM2] Successfully saved in /home/developer1/.pm2/dump.pm2
developer1@VINSMOKE:/mnt/d/honors6sem$ pm2 logs honors6sem
[TAILING] Tailing last 15 lines for [honors6sem] process (change the value with --lines option)
/home/developer1/.pm2/logs/honors6sem-error.log last 15 lines:
0|honors6sem (node:627) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTopology has no effect since Node.js Driver version 4.0.0 and will be removed in the next major version
0|honors6sem (node:627) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTopology has no effect since Node.js Driver version 4.0.0 and will be removed in the next major version
0|honors6sem (node:627) [MONGODB DRIVER] Warning: useUnifiedTopology is a deprecated option: useUnifiedTopology has no effect since Node.js Driver version 4.0.0 and will be removed in the next major version
/home/developer1/.pm2/logs/honors6sem-out.log last 15 lines:
0|honors6sem > start
0|honors6sem > node src/index.js
0|honors6sem Server running on port 3000
0|honors6sem [x] Mailer is ready
0|honors6sem [x] MongoDB connected
0|honors6sem [x] Mailer is ready
```

```
GA developer1@VINSMOKE:/mnt/d/honors6sem$ ngrok

Take our ngrok in production survey! https://forms.gle/aXiBFwzEA36DudFn6

Session Status      online
Account             utkarshlambade@gmail.com (Plan: Free)
Version             3.22.1
Region              India (in)
Latency              204ms
Web Interface        http://127.0.0.1:4040
Forwarding           https://4ecc-106-216-254-14.ngrok-free.app -> http://localhost:3000

Connections          ttl    opn    rt1    rt5    p50    p90
                    0      0      0.00   0.00   0.00   0.00
```

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 > ...
1  const mongoose = require('mongoose');
2  const bcrypt = require('bcryptjs');
3
4  const UserSchema = new mongoose.Schema({
5    email:      { type: String, required: true, unique: true },
6    password:   { type: String, required: true },
7    isVerified: { type: Boolean, default: false },
8    resetToken: String,
9    resetTokenExpiry: Date
10  });
11
12  // Hash password before save (for both new and reset)
13  UserSchema.pre('save', async function(next) {
14    if (!this.isModified('password')) return next();
15    const salt = await bcrypt.genSalt(10);
16    this.password = await bcrypt.hash(this.password, salt);
17    next();
18  });
19
20  module.exports = mongoose.model('User', UserSchema);
21
```

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.