USER AUTHENTICATION WITH JWT AND EMAIL VERIFICATION

Author: dev_ibratimoth

Tech Stack: Node.js, Express, PostgreSQL, Sequelize, JWT, Nodemailer, bcrypt and crypto

Overview

This document outlines the process of implementing user authentication using **JWT (JSON Web Token)**, with a focus on **email verification** before account activation and password reset.

Tech Stack

- Node.js + Express – Server-side logic
- PostgreSQL – Relational database
- Sequelize – ORM for DB modeling and queries
- bcrypt – Password hashing
- jsonwebtoken (JWT) – Authentication tokens
- nodemailer – Email services
- crypto – Generating secure random tokens
- dotenv – Secure config

Workflow summary

i. User Registration
   - Validates user input using custom validators
   - Checks if the user already exists
   - Password is hashed using bcrypt
   - Generates a 6-digit numeric email verification token
   - Stores token with expiration timestamp in DB
   - Sends email via Nodemailer for account verification

ii. Email Verification
   - Validates token from user input
   - Checks token existence and expiry using Sequelize's Op.gt
   - Marks user as verified and clears token fields
   - Sends welcome email upon success

iii. User Login
   - Verifies if the user is registered and verified

- Compares password with the hashed one using bcrypt
- On success, issues **JWT access and refresh tokens**
- Saves last login timestamp

iv. Forgot Password

- Accepts user's email and checks verification status
- Generates secure reset token using crypto. randomBytes
- Saves token and expiry (1 hour)
- Sends reset link via email with token embedded

v. Password Reset

- Accepts new password and token from email
- Validates password and token expiry
- Hashes and updates password
- Clears token fields and notifies user of successful reset

vi. User Deletion

- Finds user by primary key and removes the record

vii. Logout

- Clears the JWT token from cookies

Example of my code snipet for Authentication

```
const { Op } = require('sequelize'); // Import Op from Sequelize
const User = require('./../models/authModel');
const bcrypt = require('bcryptjs');
const crypto = require('crypto');
const { validateRegisterInput, validatePassword } = require('../utils/validators');
const { validateVerificationCode } = require('../utils/validators');
const { sendResponse } = require('../utils/responses');
const { sendVerificationEmail, sendWelcomeEmail, sendPasswordResetEmail, sendResetSuccessEmail } = require('../nodemail
const generateTokenAndSetCookie = require('../utils/generateToken');

const registerController = async (req, res) => {
    try {
        const { name, email, password } = req.body;

        // Validate input
        const validation = validateRegisterInput({ name, email, password });
        if (!validation.valid) {
            return sendResponse(res, 400, false, validation.message);
        }

        // Check if user already exists
        const existingUser = await User.findOne({ where: { email } }); // Using Sequelize findOne method
        if (existingUser) {
            return sendResponse(res, 400, false, 'Already Registered, please login');
        }

        // Hash password
        const hashedPassword = await bcrypt.hash(password, 12);

        // Create verification token and expiry time
        const verificationToken = Math.floor(100000 + Math.random() * 900000).toString();
        const verificationTokenExpiresAt = Date.now() + 24 * 60 * 60 * 1000; // 24 hours from now
```

```
        // Create new user
        const user = await User.create({ // Using Sequelize create method
            name,
            email,
            password: hashedPassword,
            verificationToken,
            verificationTokenExpiresAt,
        });

        // Generate token and set cookie
        // generateTokenAndSetCookie(res, user.id); // Ensure you're using the correct user ID field

        // Send verification email
        await sendVerificationEmail(user.email, verificationToken);

        // Return success response
        sendResponse(res, 201, true, "User registered successfully", {
            ...user.get(), // Use get() method to retrieve the user instance data
            password: undefined,  // Don't send the password back in the response
        });

    } catch (error) {
        console.error(error);
        sendResponse(res, 500, false, 'Error in registration', error);
    }
};

const verifyEmail = async (req, res) => {
    const { code } = req.body;
```

```
    try {
        // Validate verification code input
        const validation = validateVerificationCode(code); // Optional: Implement this function if needed
        if (!validation.valid) {
            return sendResponse(res, 400, false, validation.message);
        }

        // Find user by verification token and check if the token is still valid
        const user = await User.findOne({
            where: {
                verificationToken: code,
                verificationTokenExpiresAt: {
                    [Op.gt]: new Date() // Using Sequelize's operator for greater than
                }
            }
        });

        if (!user) {
            return sendResponse(res, 400, false, "Invalid or expired verification code");
        }

        // Update user status and remove verification fields
        user.isVerified = true;
        user.verificationToken = null; // Better to set to null
        user.verificationTokenExpiresAt = null;
        await user.save();

        // Send welcome email
        await sendWelcomeEmail(user.email, user.name);
```

```
        // Send success response
        sendResponse(res, 200, true, "Email verified successfully", {
            ...user.get(), // Use get() to retrieve user data
            password: undefined // Don't send the password back
        });
    } catch (error) {
        console.error(error);
        sendResponse(res, 500, false, 'Error while verifying email', error);
    }
};

const loginController = async (req, res) => {
    const { email, password } = req.body;

    try {
        // Check if all required fields are provided
        if (!email || !password) {
            return sendResponse(res, 400, false, 'All fields must be provided.');
        }

        // Find user by email using Sequelize
        const user = await User.findOne({ where: { email } });

        // If user not found
        if (!user || user.isVerified !== true) {
            return sendResponse(res, 400, false, 'Invalid credentials or account not verified.');
        }

        // Check if password is valid
        const isPasswordValid = await bcrypt.compare(password, user.password);
        if (!isPasswordValid) {
            return sendResponse(res, 400, false, 'Invalid credentials.');
```

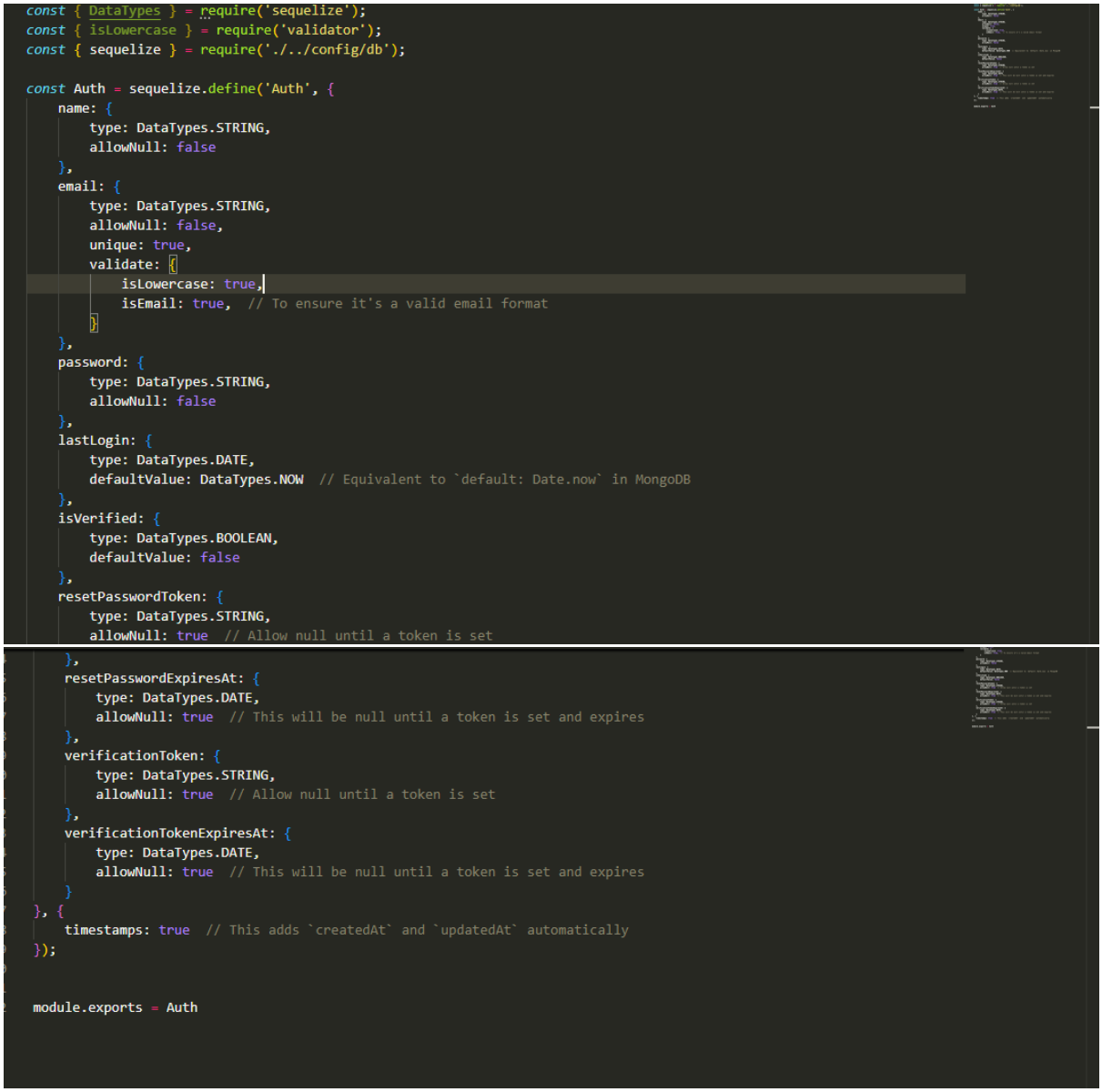
```
        }

        // Generate token and set cookie
        const { accessToken, refreshToken } = generateTokenAndSetCookie(res, user.id);

        // Update last login timestamp
        await user.update({ lastLogin: new Date() });

        // Return successful response, omitting the password from the user object
        return sendResponse(res, 200, true, 'Logged in successfully', {
            user: {
                id: user.id,
                name: user.name,
                email: user.email,
                lastLogin: user.lastLogin,
                isVerified: user.isVerified,
            }, tokens: {
                accessToken,
                refreshToken
            }
        });

    } catch (error) {
        console.error('Error in login:', error);
        return sendResponse(res, 500, false, 'An error occurred during login.');
    }
};

const forgotPassword = async (req, res) => {
    const { email } = req.body;

    try {
        // Check if the email is provided
        if (!email) {
            return sendResponse(res, 400, false, 'Email is required');
        }

        // Find the user by email using Sequelize
        const user = await User.findOne({ where: { email } });

        // If user does not exist, return error
        if (!user || user.isVerified !== true) {
            return sendResponse(res, 404, false, 'User not found or account not verified');
        }

        // Generate a secure reset token
        const resetToken = crypto.randomBytes(20).toString('hex');

        // Set token expiration time (1 hour)
        const resetTokenExpiresAt = Date.now() + 1 * 60 * 60 * 1000; // 1 hour

        // Update user with the reset token and expiration time
        user.resetPasswordToken = resetToken;
        user.resetPasswordExpiresAt = resetTokenExpiresAt;

        // Save the user with updated information
        await user.save();

        // Construct the password reset URL
        const resetUrl = `http://localhost:4003/api/auth/reset-password/${resetToken}`;

        // Send password reset email
        await sendPasswordResetEmail(user.email, resetUrl);
```

```
const forgotPassword = async (req, res) => {

        // Respond with success message
        return sendResponse(res, 200, true, 'Password reset link sent to your email', { resetToken });

    } catch (error) {
        console.error('Error in forgotPassword:', error);

        // Respond with error message
        return sendResponse(res, 500, false, 'An error occurred while processing your request');
    }
};

const resetPassword = async (req, res) => {
    const { token } = req.params;
    const { password } = req.body;

    try {
        // Validate request parameters and body
        if (!token || !password) {
            return sendResponse(res, 400, false, 'Reset token and new password must be provided.');
        }

        // Validate input
        const validation = validatePassword({ password });
        if (!validation.valid) {
            return sendResponse(res, 400, false, validation.message);
        }
        // Find user by reset token and check if it has expired
        const user = await User.findOne({
            where: {
                resetPasswordToken: token,
                resetPasswordExpiresAt: { [Op.gt]: Date.now() }, // Ensure using Sequelize operators
            where: {
                resetPasswordExpiresAt: { [Op.gt]: Date.now() }, // Ensure using Sequelize operators
            },
        });

        // If user not found or token is invalid/expired
        if (!user) {
            return sendResponse(res, 400, false, 'Invalid or expired reset token.');
        }

        // Hash the new password
        const hashedPassword = await bcrypt.hash(password, 10);
        user.password = hashedPassword;
        user.resetPasswordToken = null; // Clear reset token
        user.resetPasswordExpiresAt = null; // Clear expiration time

        // Save updated user information
        await user.save();

        // Send success email
        await sendResetSuccessEmail(user.email);

        // Send response indicating success
        return sendResponse(res, 200, true, 'Password reset successful.');

    } catch (error) {
        console.error('Error in resetPassword:', error);
        return sendResponse(res, 500, false, 'An error occurred while resetting the password.');
    }
};
```

```
const deleteUserById = async (req, res) => {
    const { id } = req.params; // Get the user ID from the request parameters

    try {
        // Find the user by ID
        const user = await User.findByPk(id);

        // Check if the user exists
        if (!user) {
            return sendResponse(res, 404, false, 'User not found');
        }

        // Delete the user
        await User.destroy({
            where: { id },
        });

        // Send a success response
        sendResponse(res, 200, true, 'User deleted successfully');
    } catch (error) {
        console.error(error);
        sendResponse(res, 500, false, 'Error deleting user', error);
    }
};

const logout = async (req, res) => {
    res.clearCookie('token')
    res.status(200).json({ success: true, message: "Logged out successfully" })
}
```

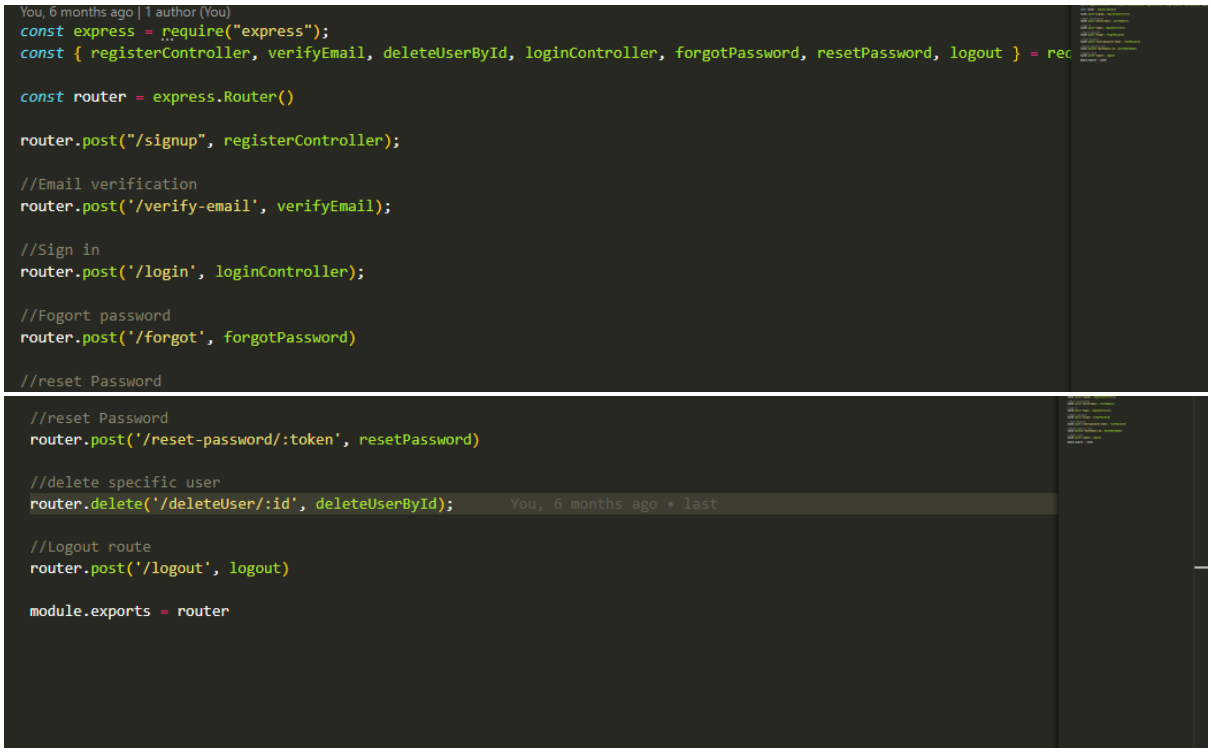
```
module.exports = { registerController, verifyEmail, deleteUserById, loginController, forgotPassword, resetPassword , lo
```

Example of my model schema structure



```
const { DataTypes } = require('sequelize');
const { isLowercase } = require('validator');
const { sequelize } = require('./../config/db');

const Auth = sequelize.define('Auth', {
    name: {
        type: DataTypes.STRING,
        allowNull: false
    },
    email: {
        type: DataTypes.STRING,
        allowNull: false,
        unique: true,
        validate: {
            isLowercase: true,
            isEmail: true,  // To ensure it's a valid email format
        }
    },
    password: {
        type: DataTypes.STRING,
        allowNull: false
    },
    lastLogin: {
        type: DataTypes.DATE,
        defaultValue: DataTypes.NOW  // Equivalent to `default: Date.now` in MongoDB
    },
    isVerified: {
        type: DataTypes.BOOLEAN,
        defaultValue: false
    },
    resetPasswordToken: {
        type: DataTypes.STRING,
        allowNull: true  // Allow null until a token is set
    },
    resetPasswordExpiresAt: {
        type: DataTypes.DATE,
        allowNull: true  // This will be null until a token is set and expires
    },
    verificationToken: {
        type: DataTypes.STRING,
        allowNull: true  // Allow null until a token is set
    },
    verificationTokenExpiresAt: {
        type: DataTypes.DATE,
        allowNull: true  // This will be null until a token is set and expires
    }
}, {
    timestamps: true  // This adds `createdAt` and `updatedAt` automatically
});


module.exports = Auth
```

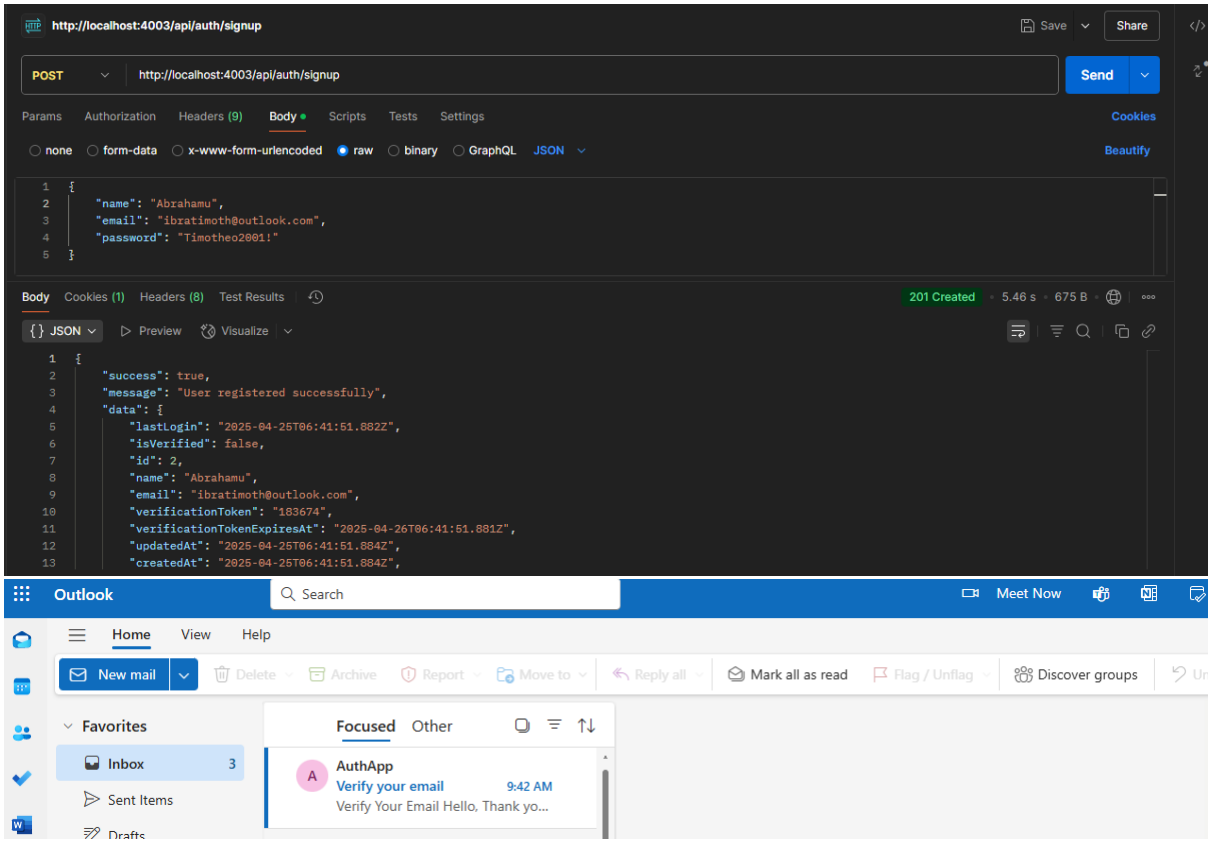
My routes



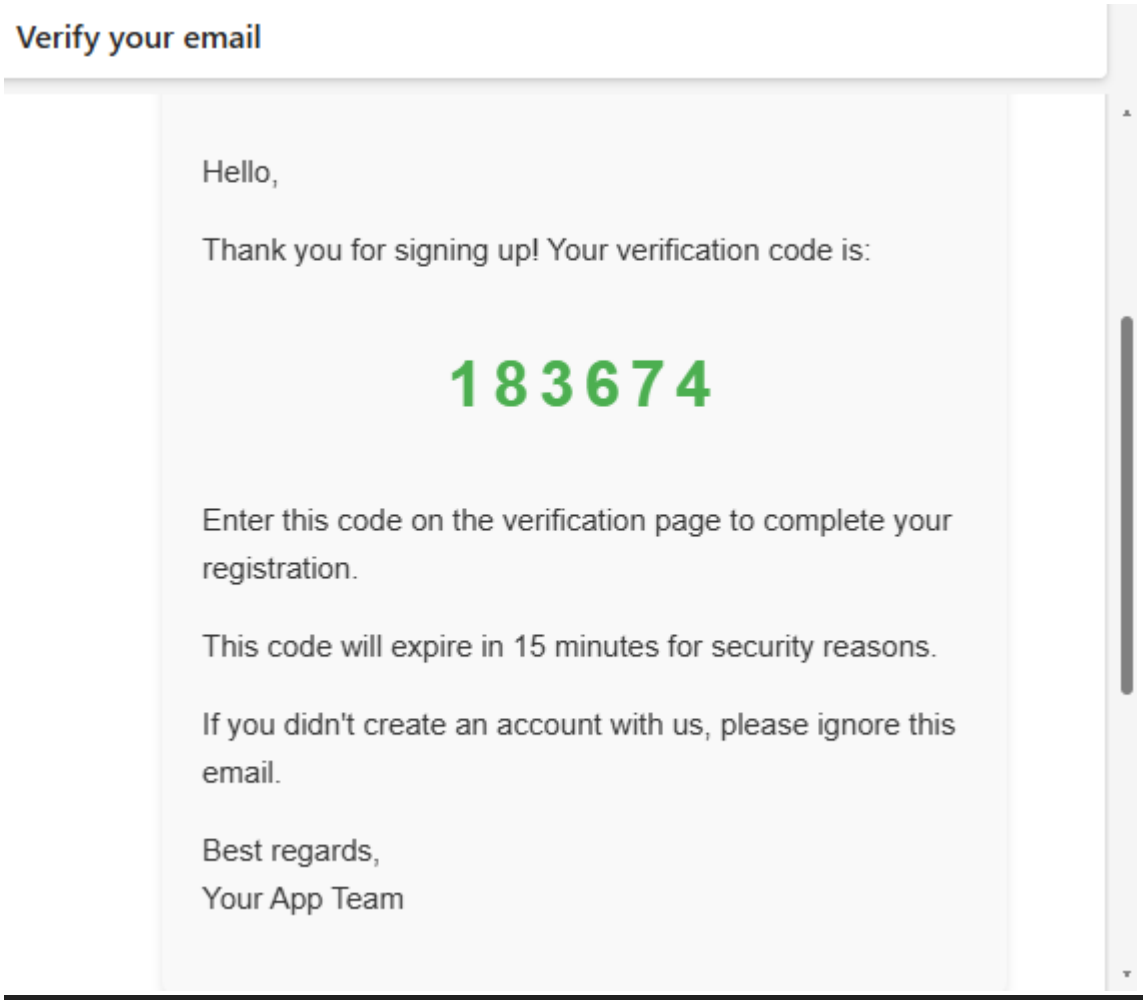
```
You, 6 months ago | 1 author (You)
const express = require("express");
const { registerController, verifyEmail, deleteUserById, loginController, forgotPassword, resetPassword, logout } = rec

const router = express.Router()

router.post("/signup", registerController);

//Email verification
router.post('/verify-email', verifyEmail);

//Sign in
router.post('/login', loginController);

//Fogort password
router.post('/forgot', forgotPassword)

//reset Password
```

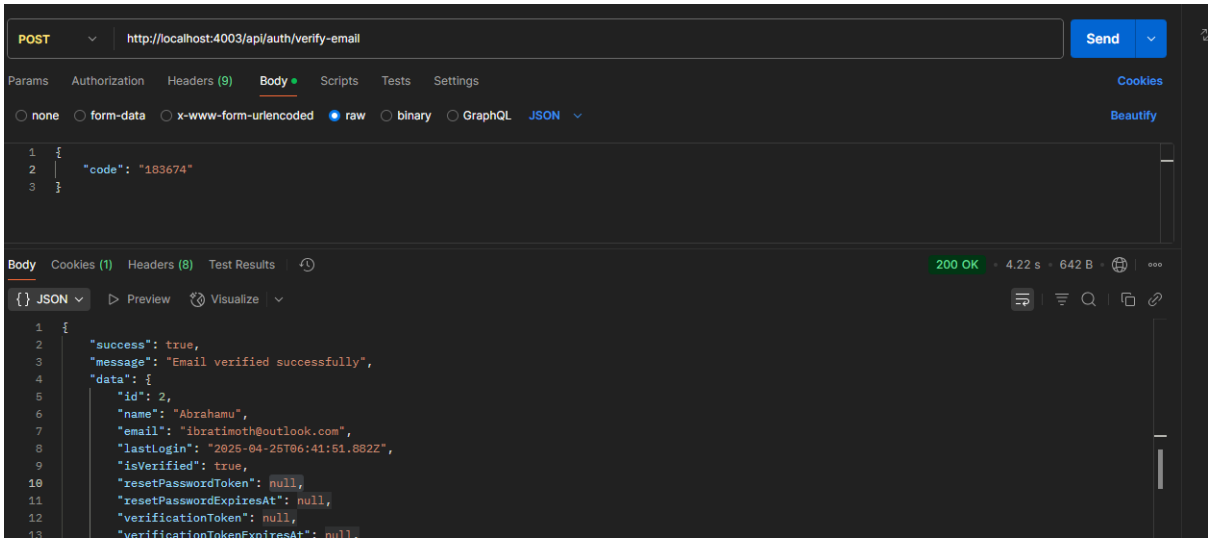
```
//reset Password
router.post('/reset-password/:token', resetPassword)

//delete specific user
router.delete('/deleteUser/:id', deleteUserById);       You, 6 months ago • last

//Logout route
router.post('/logout', logout)

module.exports = router
```

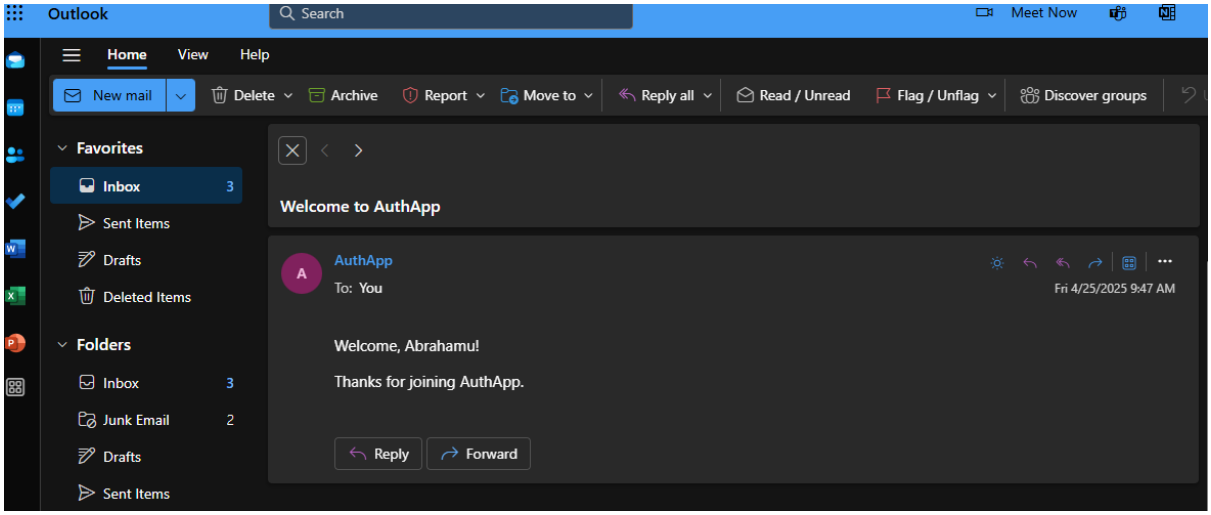
How it works

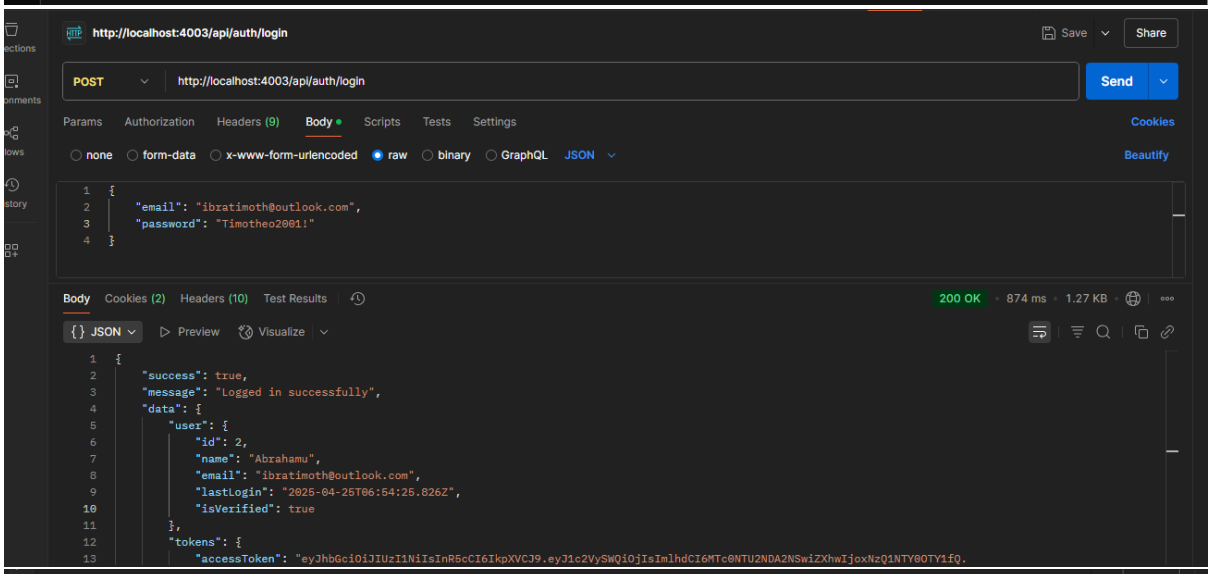

http://localhost:4003/api/auth/signup
Save
Share
POST
http://localhost:4003/api/auth/signup
Send
Params
Authorization
Headers (9)
Body
Scripts
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"name": "Abrahamu",
"email": "ibratimoth@outlook.com",
"password": "Timotheo2001!"
}
Body
Cookies (1)
Headers (8)
Test Results
201 Created
5.46 s
675 B
{} JSON
Preview
Visualize
{
"success": true,
"message": "User registered successfully",
"data": {
"lastLogin": "2025-04-25T06:41:51.882Z",
"isVerified": false,
"id": 2,
"name": "Abrahamu",
"email": "ibratimoth@outlook.com",
"verificationToken": "183674",
"verificationTokenExpiresAt": "2025-04-26T06:41:51.881Z",
"updatedAt": "2025-04-25T06:41:51.884Z",
"createdAt": "2025-04-25T06:41:51.884Z",

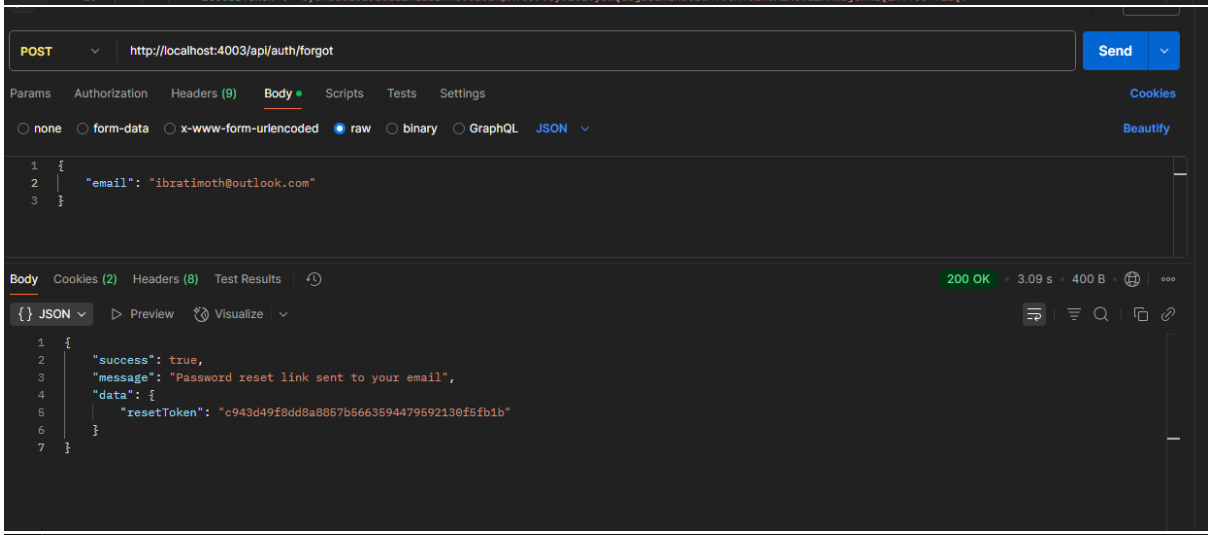
Outlook
Search
Meet Now
Home
View
Help
New mail
Delete
Archive
Report
Move to
Reply all
Mark all as read
Flag / Unflag
Discover groups
Favorites
Inbox
3
Sent Items
Drafts
Focused
Other
AuthApp
Verify your email
9:42 AM
Verify Your Email Hello, Thank yo...

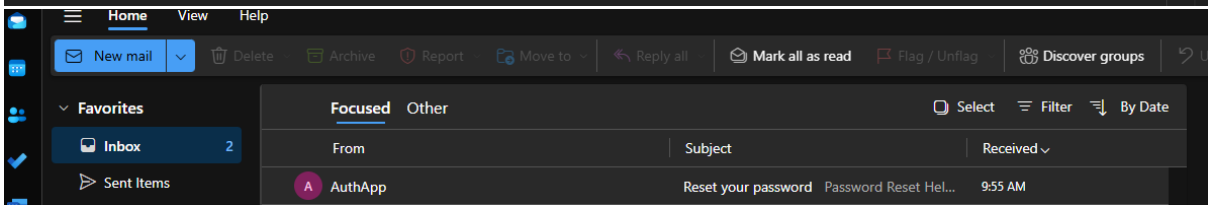

Verify your email
Hello,
Thank you for signing up! Your verification code is:
183674
Enter this code on the verification page to complete your registration.
This code will expire in 15 minutes for security reasons.
If you didn't create an account with us, please ignore this email.
Best regards,
Your App Team

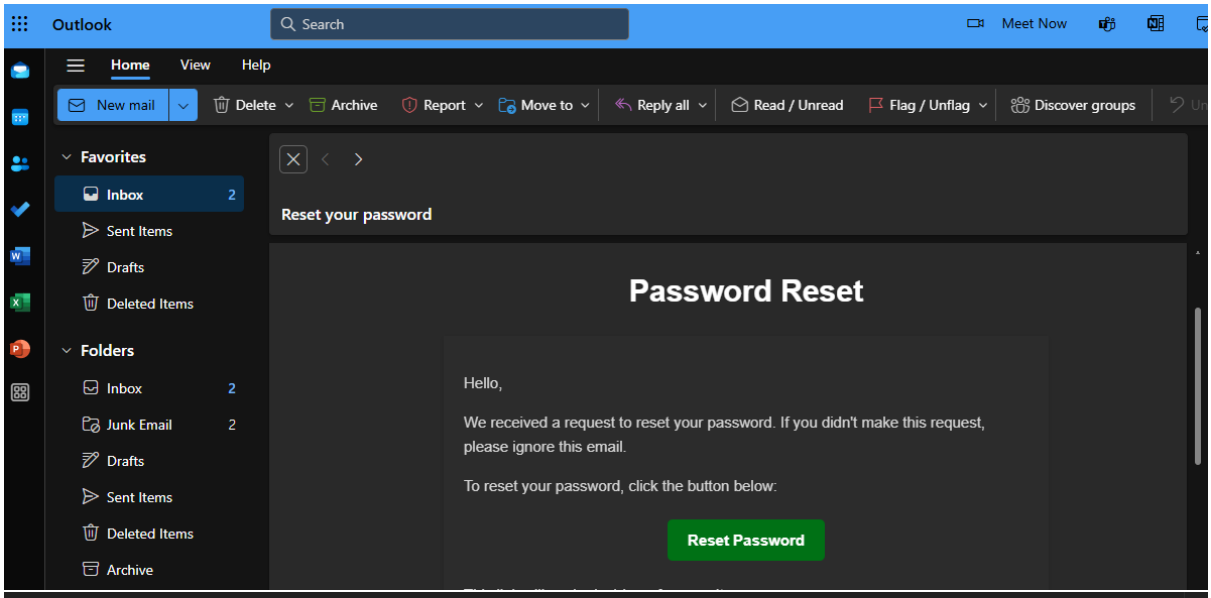

POST
http://localhost:4003/api/auth/verify-email
Send
Params
Authorization
Headers (9)
Body
Scripts
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"code": "183674"
}
Body
Cookies (1)
Headers (8)
Test Results
200 OK
4.22 s
642 B
JSON
Preview
Visualize
{
"success": true,
"message": "Email verified successfully",
"data": {
"id": 2,
"name": "Abrahamu",
"email": "ibratimoth@outlook.com",
"lastLogin": "2025-04-25T06:41:51.882Z",
"isVerified": true,
"resetPasswordToken": null,
"resetPasswordExpiresAt": null,
"verificationToken": null,

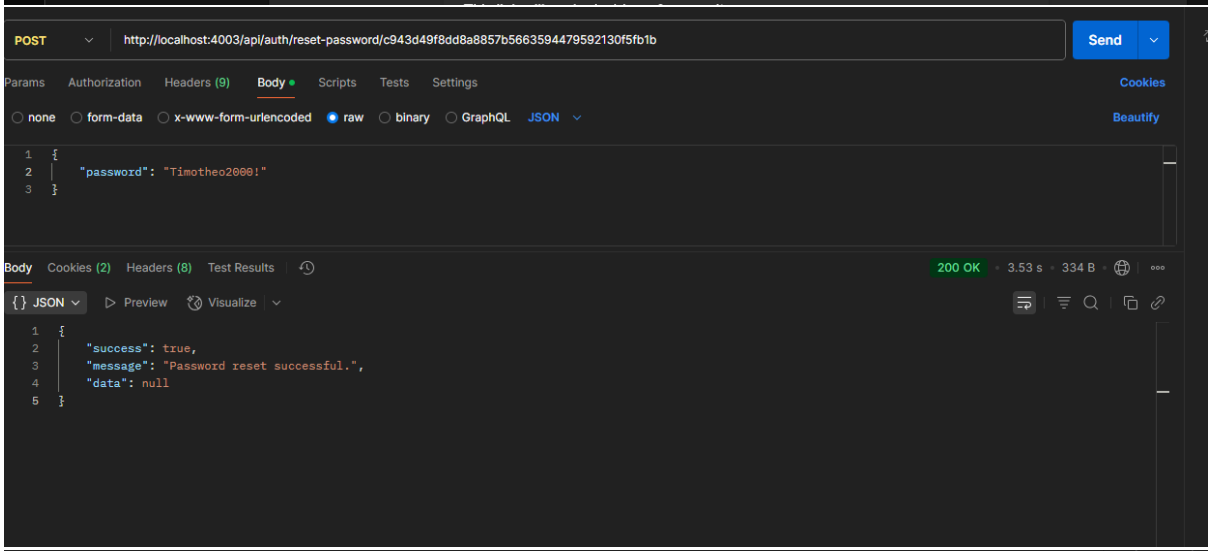

Outlook
Search
Meet Now
Home
View
Help
New mail
Delete
Archive
Report
Move to
Reply all
Read / Unread
Flag / Unflag
Discover groups
Favorites
Inbox 3
Sent Items
Drafts
Deleted Items
Folders
Inbox 3
Junk Email 2
Drafts
Sent Items
Welcome to AuthApp
AuthApp
To: You
Fri 4/25/2025 9:47 AM
Welcome, Abrahamu!
Thanks for joining AuthApp.
Reply
Forward

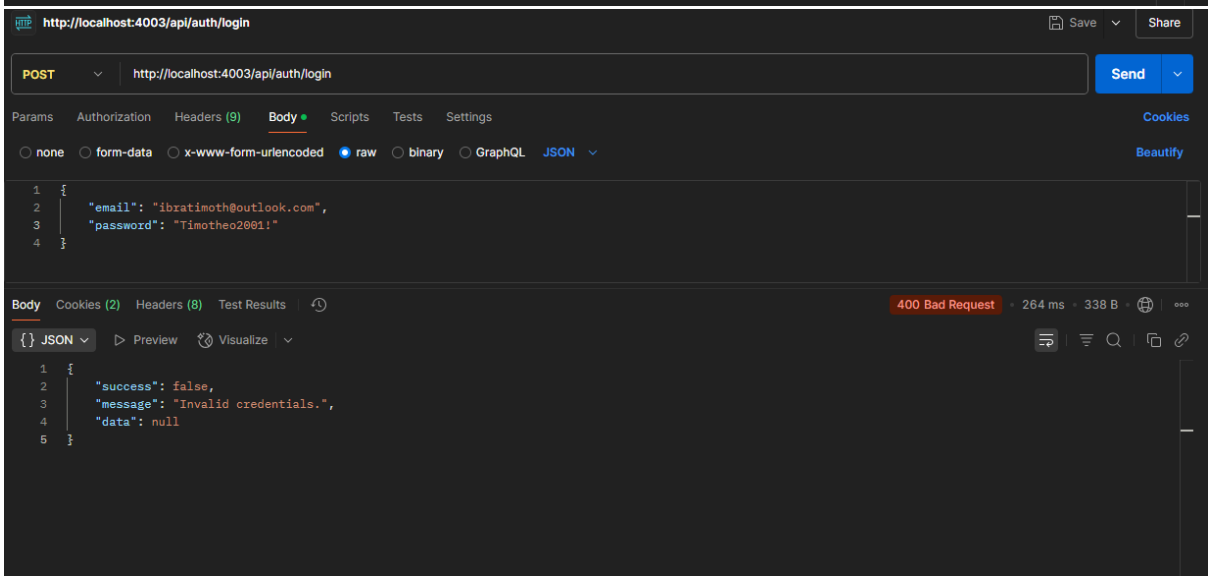

http://localhost:4003/api/auth/login
Save
Share
POST
http://localhost:4003/api/auth/login
Send
Params
Authorization
Headers (9)
Body
Scripts
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"email": "ibratimoth@outlook.com",
"password": "Timotheo2001!"
}
Body
Cookies (2)
Headers (10)
Test Results
200 OK
874 ms
1.27 KB
JSON
Preview
Visualize
{
"success": true,
"message": "Logged in successfully",
"data": {
"user": {
"id": 2,
"name": "Abrahamu",
"email": "ibratimoth@outlook.com",
"lastLogin": "2025-04-25T06:54:25.826Z",
"isVerified": true
},
"tokens": {
"accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOjIsImlhdCI6MTc0NTU2NDA2NSwiZXhwIjoxNzQ1NTY0OTY1fQ.

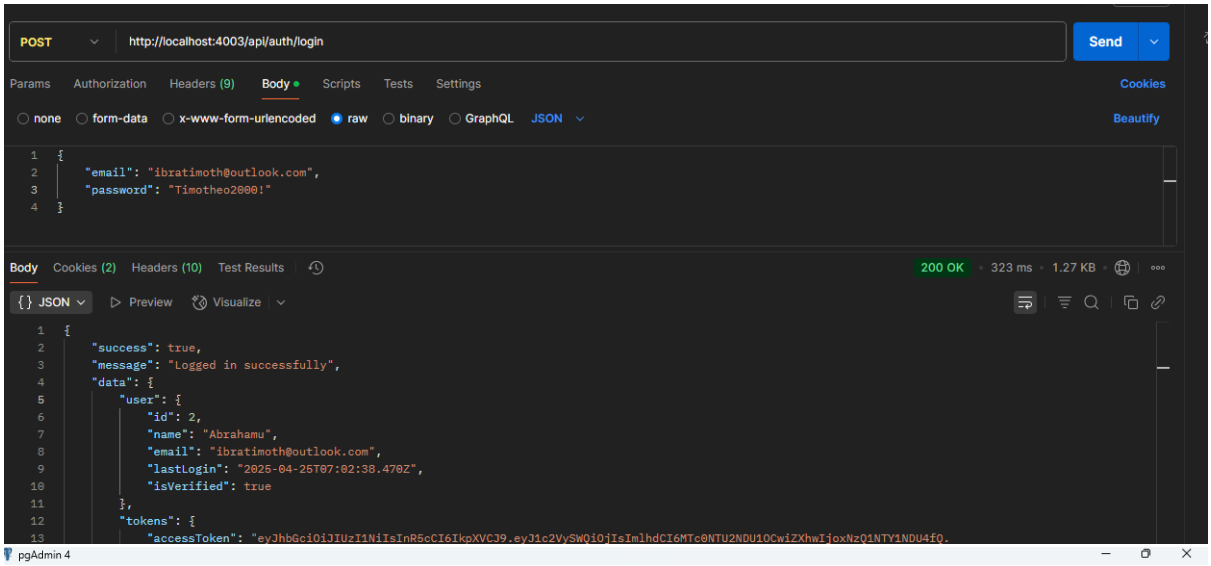

POST
http://localhost:4003/api/auth/forgot
Send
Params
Authorization
Headers (9)
Body
Scripts
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"email": "ibratimoth@outlook.com"
}
Body
Cookies (2)
Headers (8)
Test Results
200 OK
3.09 s
400 B
JSON
Preview
Visualize
{
"success": true,
"message": "Password reset link sent to your email",
"data": {
"resetToken": "c943d49f8dd8a8857b5663594479592130f5fb1b"
}
}

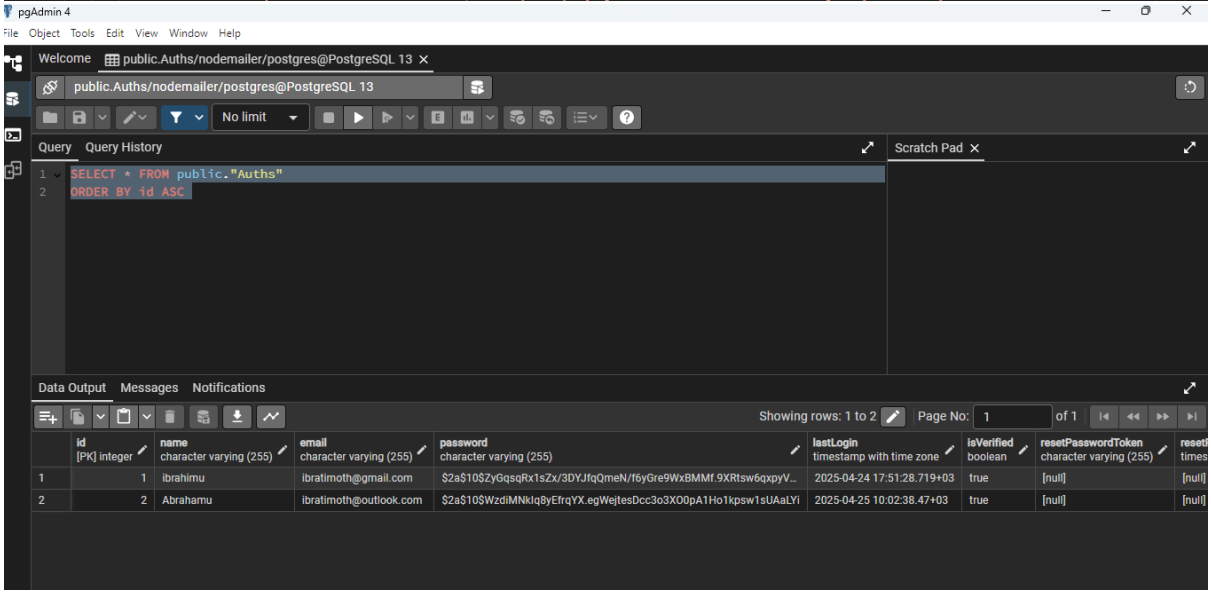

Home
View
Help
New mail
Delete
Archive
Report
Move to
Reply all
Mark all as read
Flag / Unflag
Discover groups
Favorites
Inbox 2
Sent Items
Focused
Other
Select
Filter
By Date
From
Subject
Received
AuthApp
Reset your password
Password Reset Hel...
9:55 AM


Outlook
Search
Meet Now
Home
View
Help
New mail
Delete
Archive
Report
Move to
Reply all
Read / Unread
Flag / Unflag
Discover groups
Favorites
Inbox 2
Sent Items
Drafts
Deleted Items
Folders
Inbox 2
Junk Email 2
Drafts
Sent Items
Deleted Items
Archive
Reset your password
Password Reset
Hello,
We received a request to reset your password. If you didn't make this request, please ignore this email.
To reset your password, click the button below:
Reset Password


POST
http://localhost:4003/api/auth/reset-password/c943d49f8dd8a8857b56635944795921 30f5fb1b
Send
Params
Authorization
Headers (9)
Body
Scripts
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"password": "Timotheo2000!"
}
Body
Cookies (2)
Headers (8)
Test Results
200 OK
3.53 s
334 B
{} JSON
Preview
Visualize
{
"success": true,
"message": "Password reset successful.",
"data": null
}


http://localhost:4003/api/auth/login
Save
Share
POST
http://localhost:4003/api/auth/login
Send
Params
Authorization
Headers (9)
Body
Scripts
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"email": "ibratimoth@outlook.com",
"password": "Timotheo2001!"
}
Body
Cookies (2)
Headers (8)
Test Results
400 Bad Request
264 ms
338 B
{} JSON
Preview
Visualize
{
"success": false,
"message": "Invalid credentials.",
"data": null
}


POST
http://localhost:4003/api/auth/login
Send
Params
Authorization
Headers (9)
Body
Scripts
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"email": "ibratimoth@outlook.com",
"password": "Timotheo2000!"
}
Body
Cookies (2)
Headers (10)
Test Results
200 OK
323 ms
1.27 KB
JSON
Preview
Visualize
{
"success": true,
"message": "Logged in successfully",
"data": {
"user": {
"id": 2,
"name": "Abrahamu",
"email": "ibratimoth@outlook.com",
"lastLogin": "2025-04-25T07:02:38.470Z",
"isVerified": true
},
"tokens": {
"accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VySWQiOjIsImlhdCI6MTc0NTU2NDU1OCwiZXhwIjoxNzQ1NTY1NDU4fQ.


pgAdmin 4
File
Object
Tools
Edit
View
Window
Help
Welcome
public.Auths/nodemailer/postgres@PostgreSQL 13
public.Auths/nodemailer/postgres@PostgreSQL 13
No limit
Query
Query History
Scratch Pad
SELECT * FROM public."Auths"
ORDER BY id ASC
Data Output
Messages
Notifications
Showing rows: 1 to 2
Page No: 1
of 1
id [PK] integer
name character varying (255)
email character varying (255)
password character varying (255)
lastLogin timestamp with time zone
isVerified boolean
resetPasswordToken character varying (255)
1 1 ibrahimu ibratimoth@gmail.com $2a$10$ZyGqsqRx1sZx/3DYJfqQmeN/f6yGre9WxBMMf.9XRtsw6qxpyV... 2025-04-24 17:51:28.719+03 true [null] [null]
2 2 Abrahamu ibratimoth@outlook.com $2a$10$WzdiMNklq8yEfrqYX.egWejtesDcc3o3XO0pA1Ho1kpsw1sUAaLYi 2025-04-25 10:02:38.47+03 true [null] [null]