API Testings

Authentication API's

Overview:

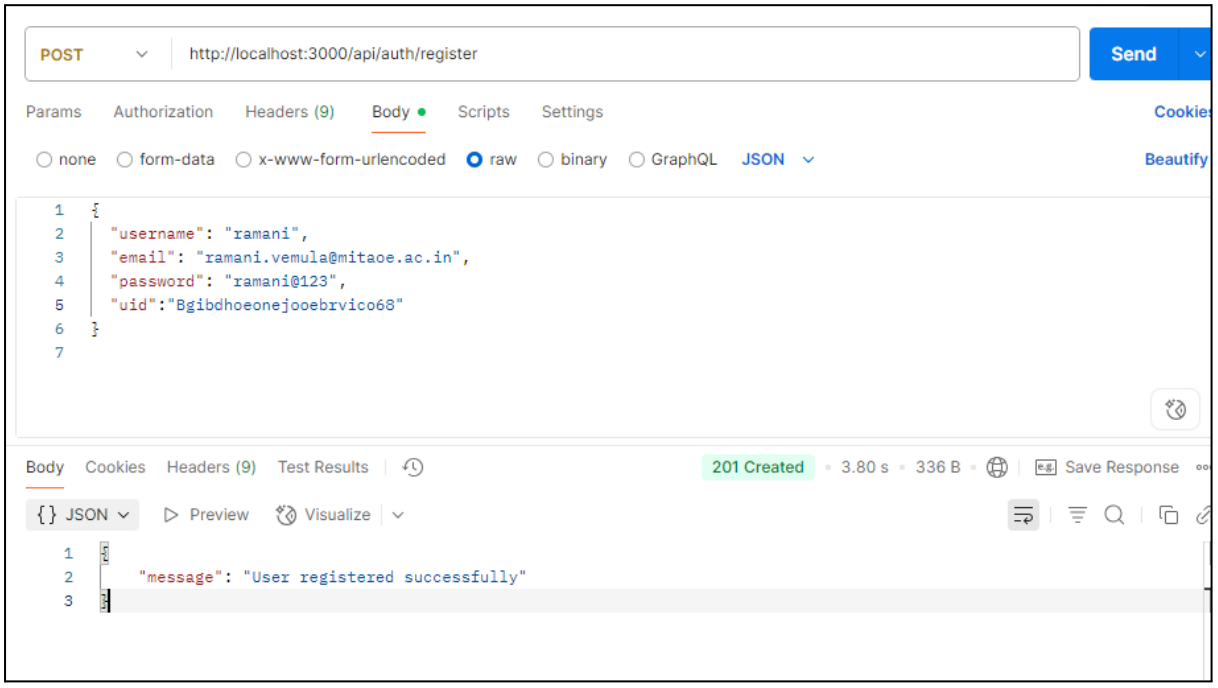
This API provides authentication features including user registration, login, OTP verification, password reset, and Google OAuth support.

Basic URL: http://localhost:3000/api/auth

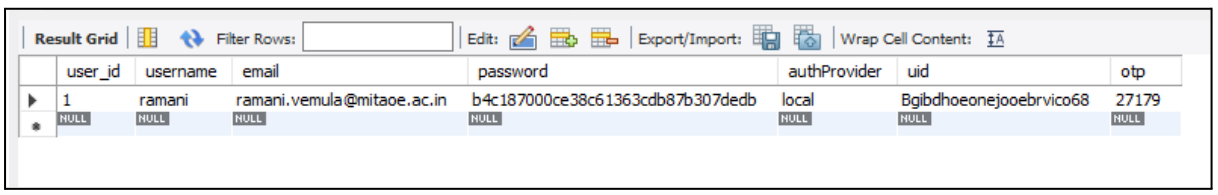
1. Register User through entering details

URL: **POST** http://localhost:3000/api/auth/register

Postman:



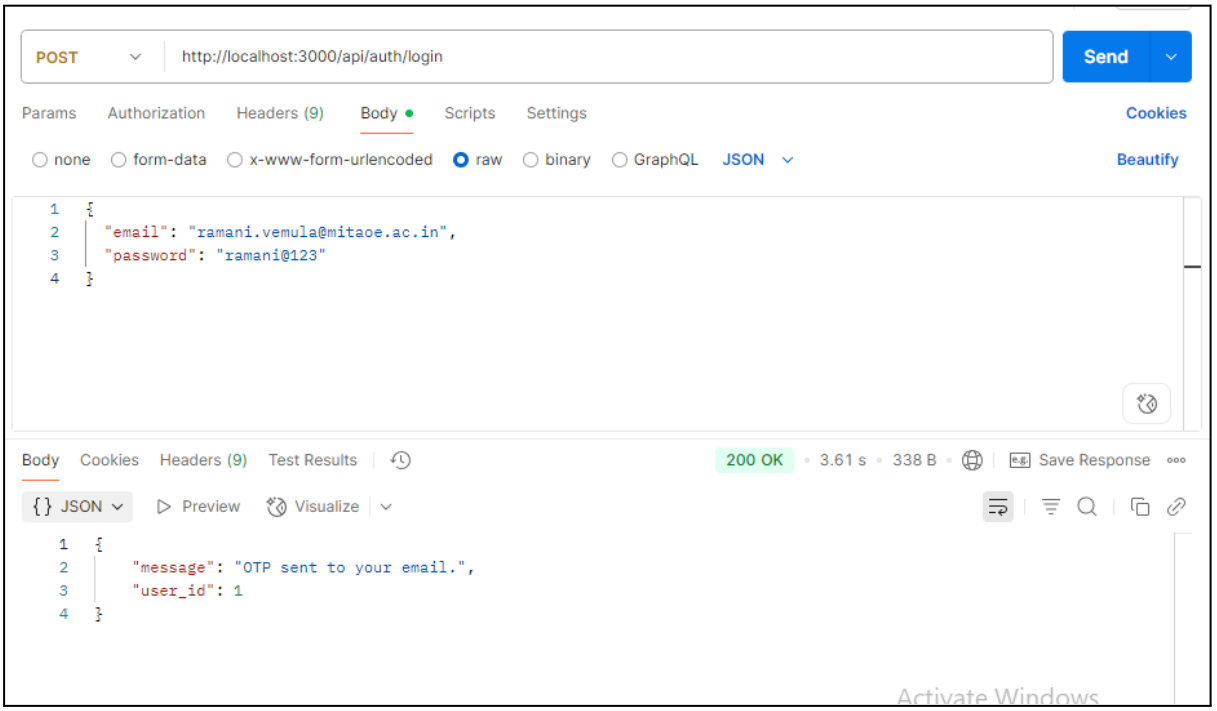
Database:



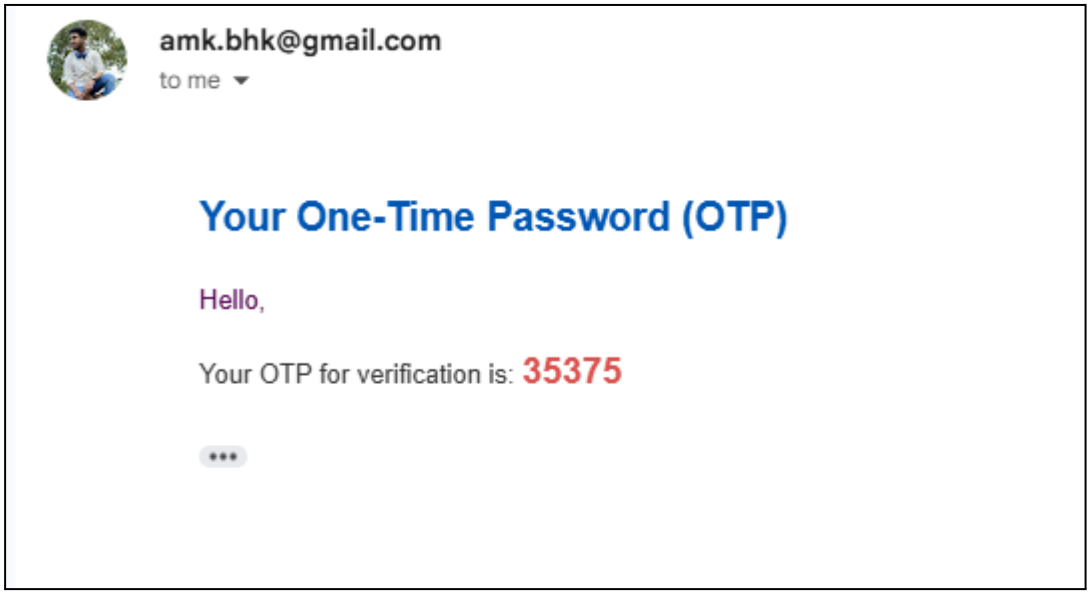
2. Login User through entering details

URL: **POST** http://localhost:3000/api/auth/login

Postman:



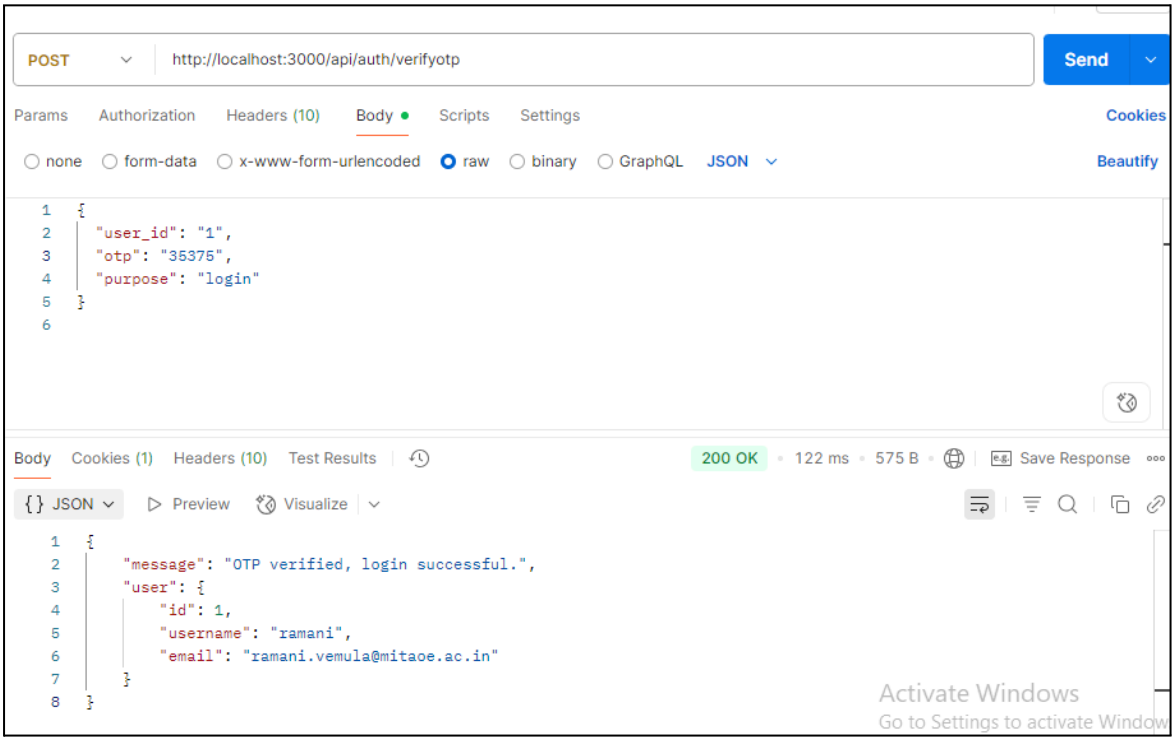
OTP will sent to email for verification:



3. Verifying account by entering the otp sent to mail:

URL: **POST** http://localhost:3000/api/auth/verifyotp

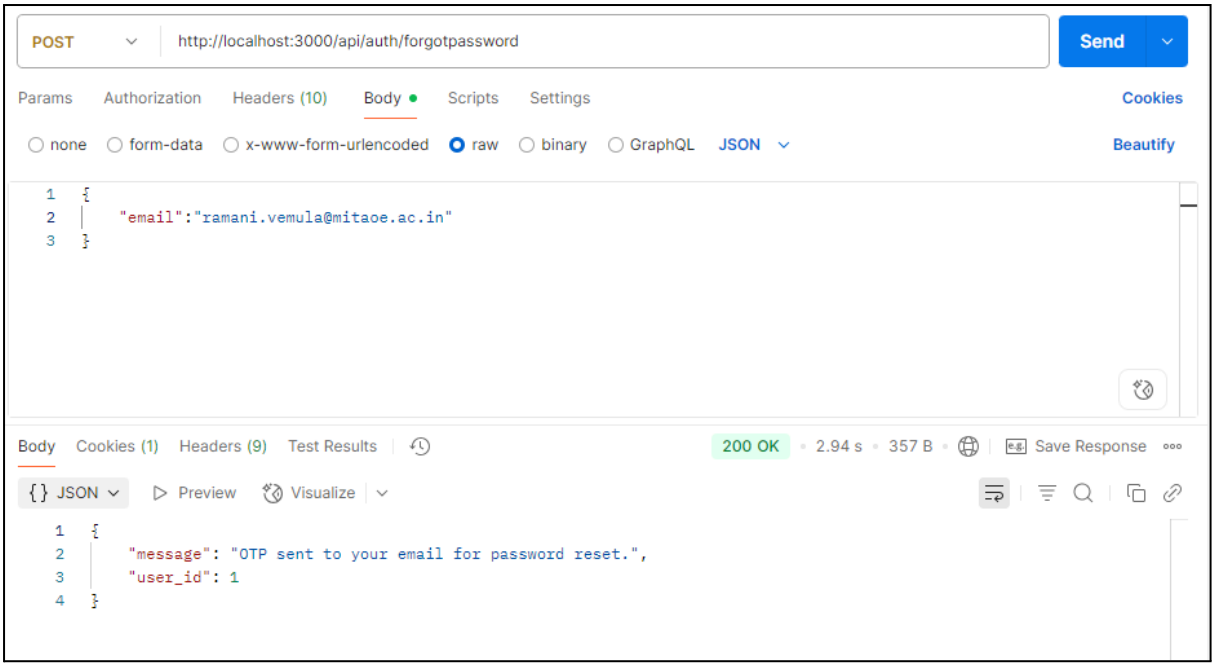
Postman:



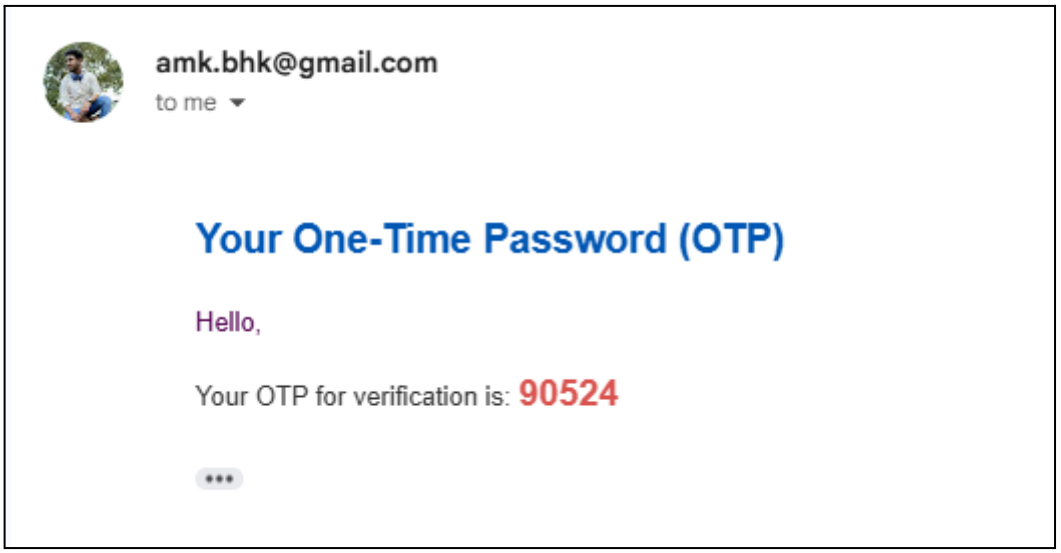
4. Forget Password

URL: **POST** http://localhost:3000/api/auth/forgetpassword

Postman:



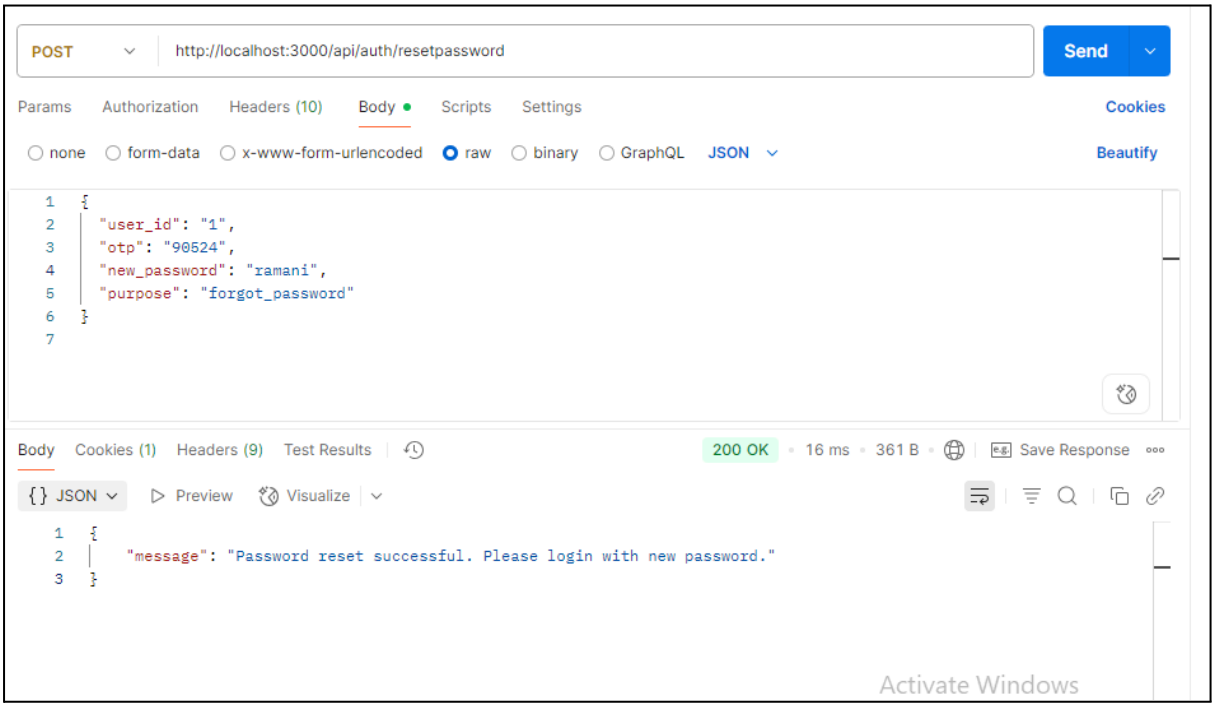
OTP will sent to email for verification:



5. Reset the password by entering OTP & New password

URL: **POST** http://localhost:3000/api/auth/resetpassword

Postman:

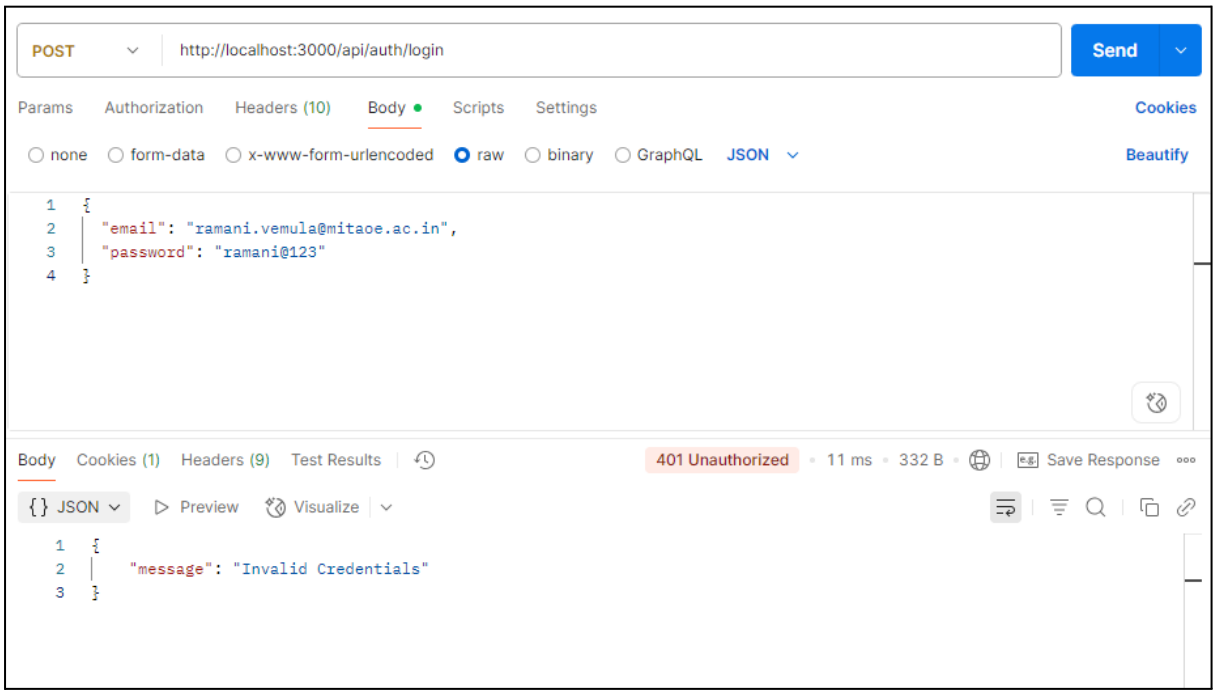


6. Login with previous and new password

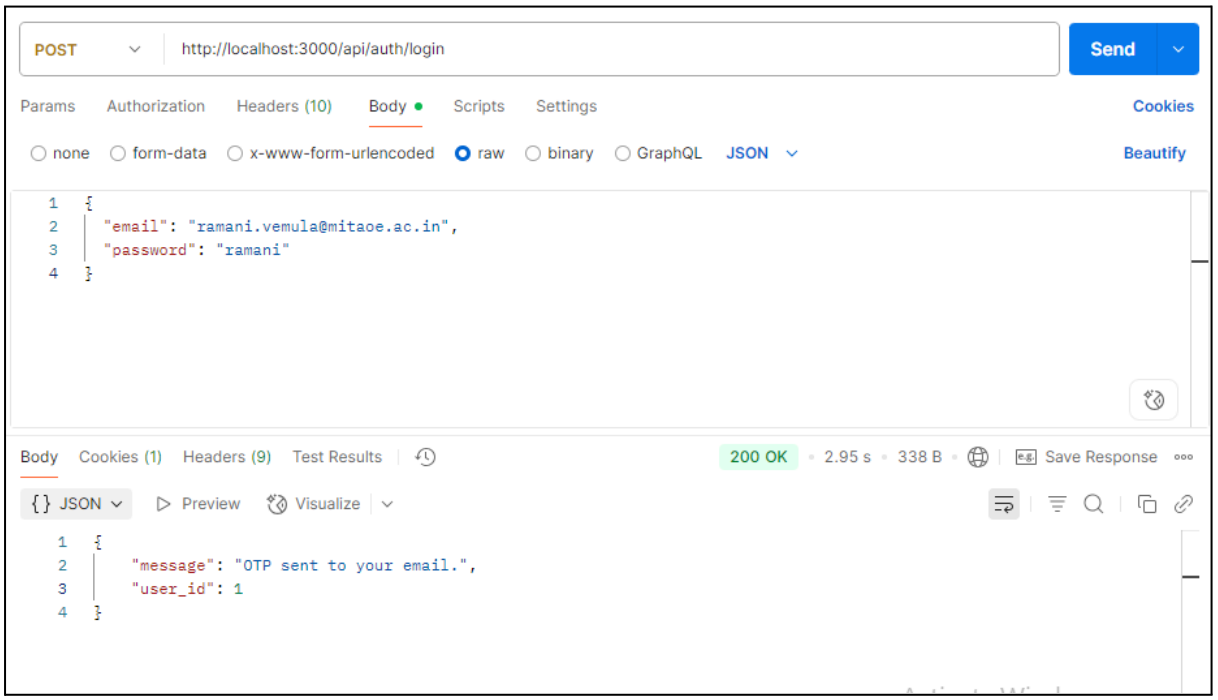
URL: **POST** http://localhost:3000/api/auth/login

Postman:

Old Password



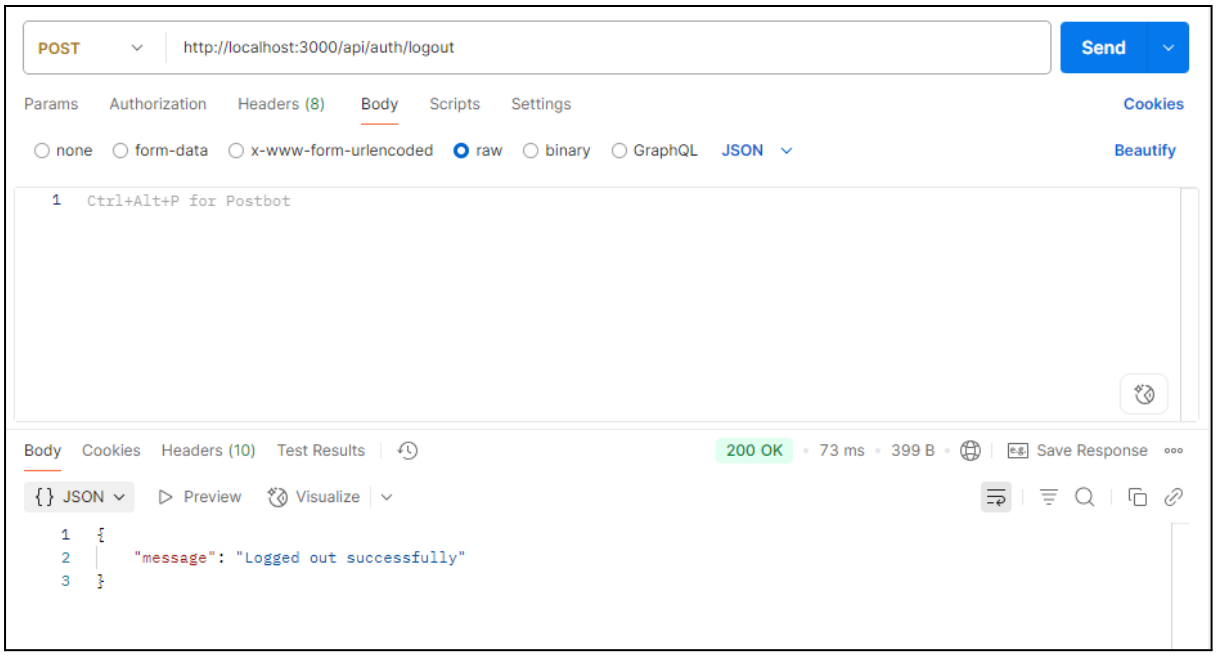
New password



7. Logout

URL: POST http://localhost:3000/api/auth/logout

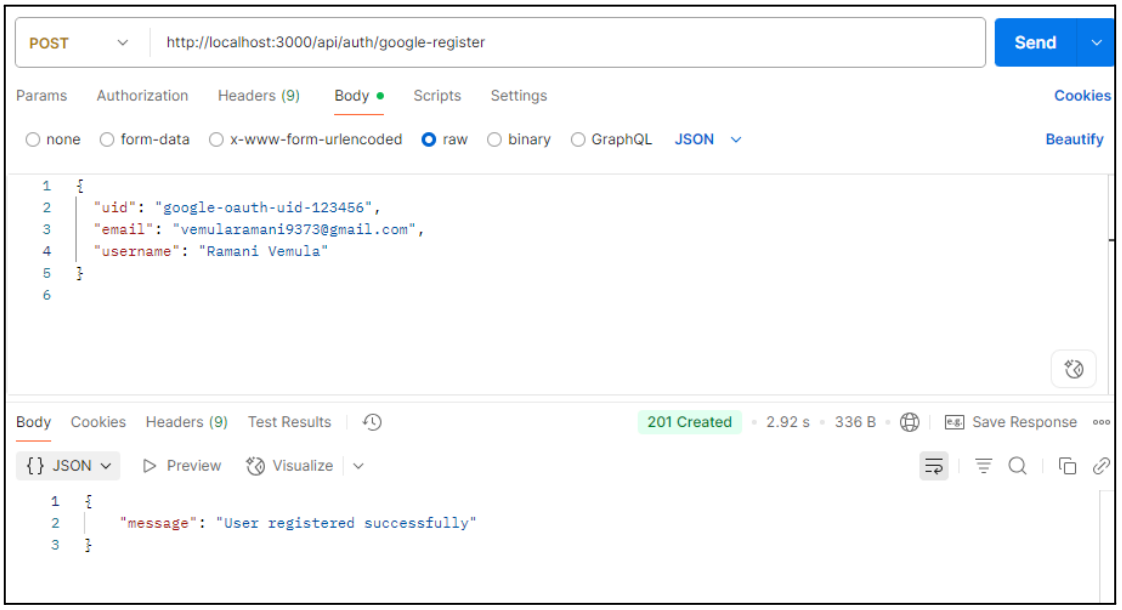
Postman:



8. Registering with the google

URL: POST http://localhost:3000/api/auth/google-register

Postman:



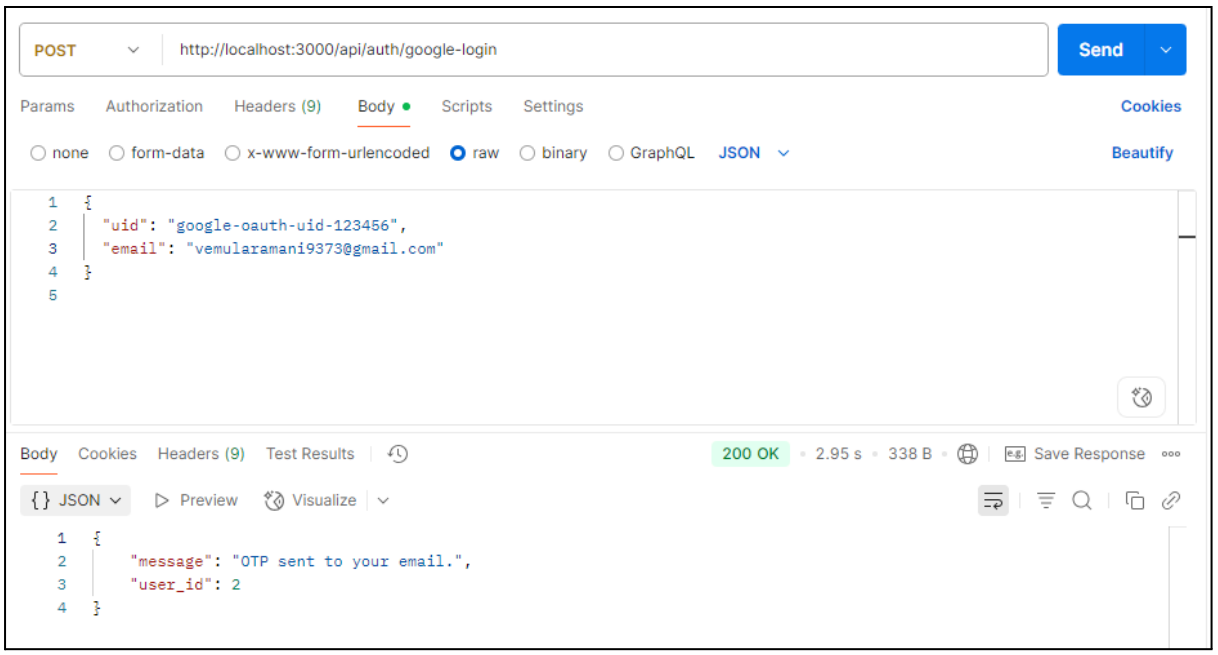
Database Schema:

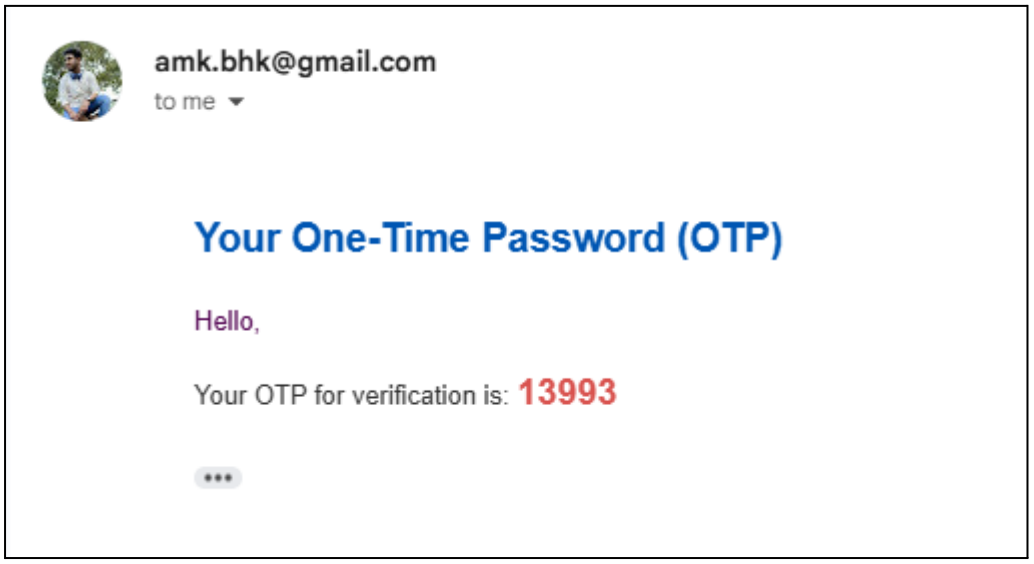
user_id	username	email	password	authProvider	uid	otp
1	ramani	ramani.vemula@mitaoe.ac.in	6b4995a929e2936ae07a4c0792385a93	local	Bgibdhoeonejooebrvico68	96958
2	Ramani Vemula	vemularamani9373@gmail.com		google	google-oauth-uid-123456	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL

9. Logging with the google

URL: **POST** http://localhost:3000/api/auth/google-login

Postman:

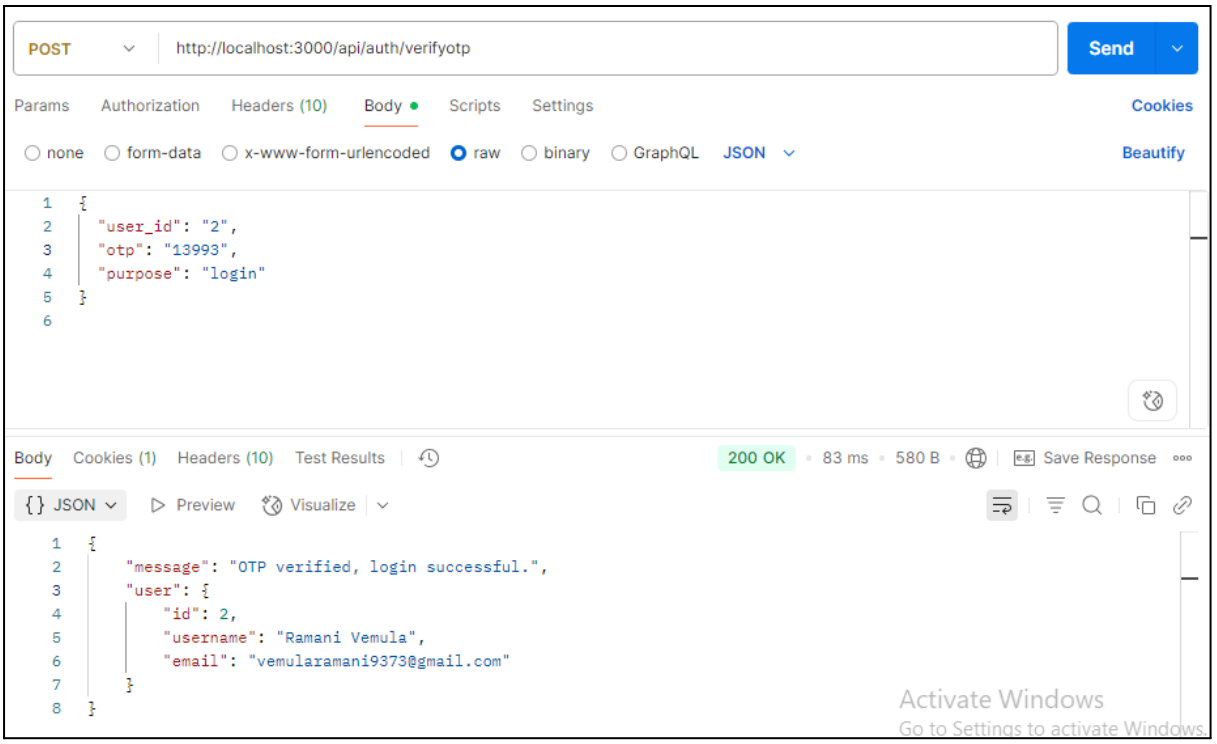




10. OTP Verification

URL: POST http://localhost:3000/api/auth/verifyotp

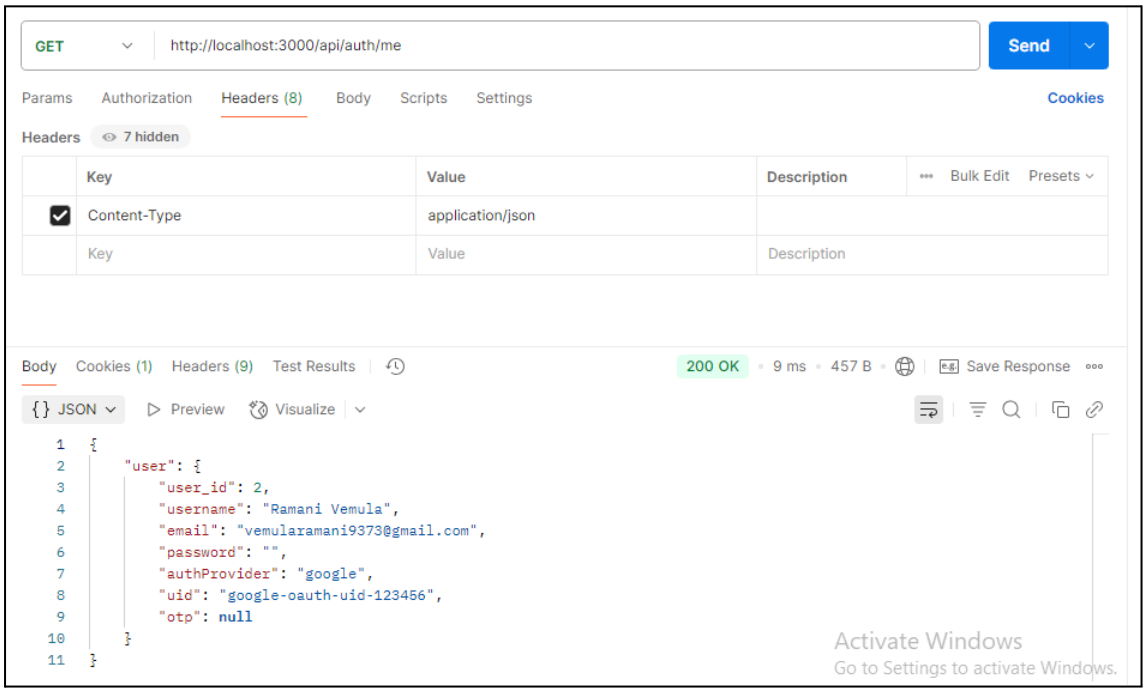
Postman:



11. Getting User Details

URL: GET http://localhost:3000/api/auth/me

Postman:



Model API's

Overview:

Models allow users to define and manage custom database structures by specifying a name and fields. The API supports creating, updating, viewing, and deleting models, and it automatically generates Sequelize code and stores metadata for each model.

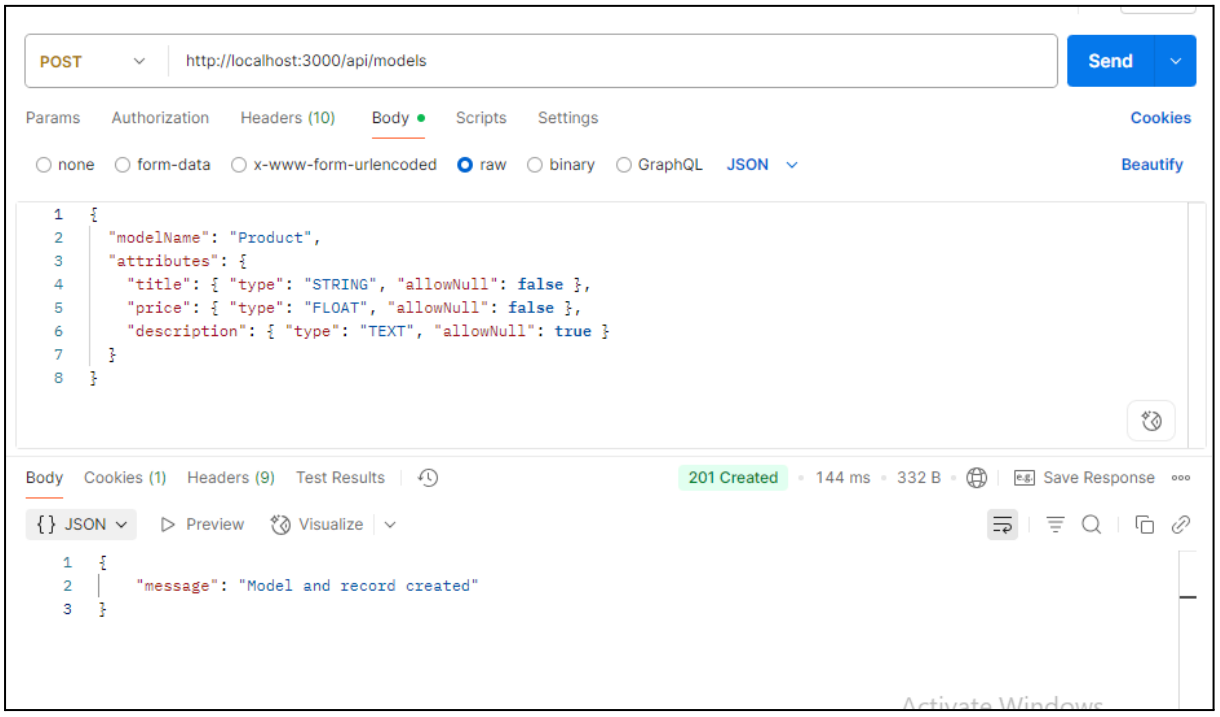
Basic URL: http://localhost:3000/api/models

1. Creating the model

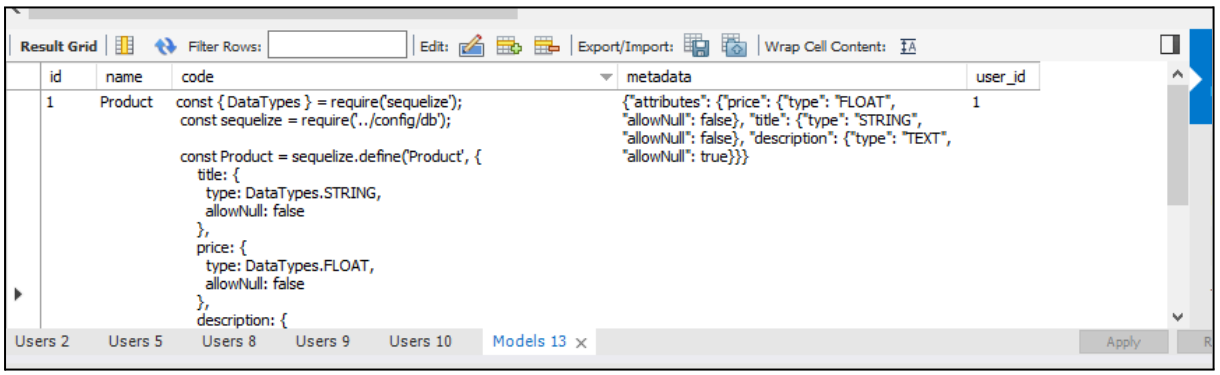
URL: **POST** http://localhost:3000/api/models

Postman:

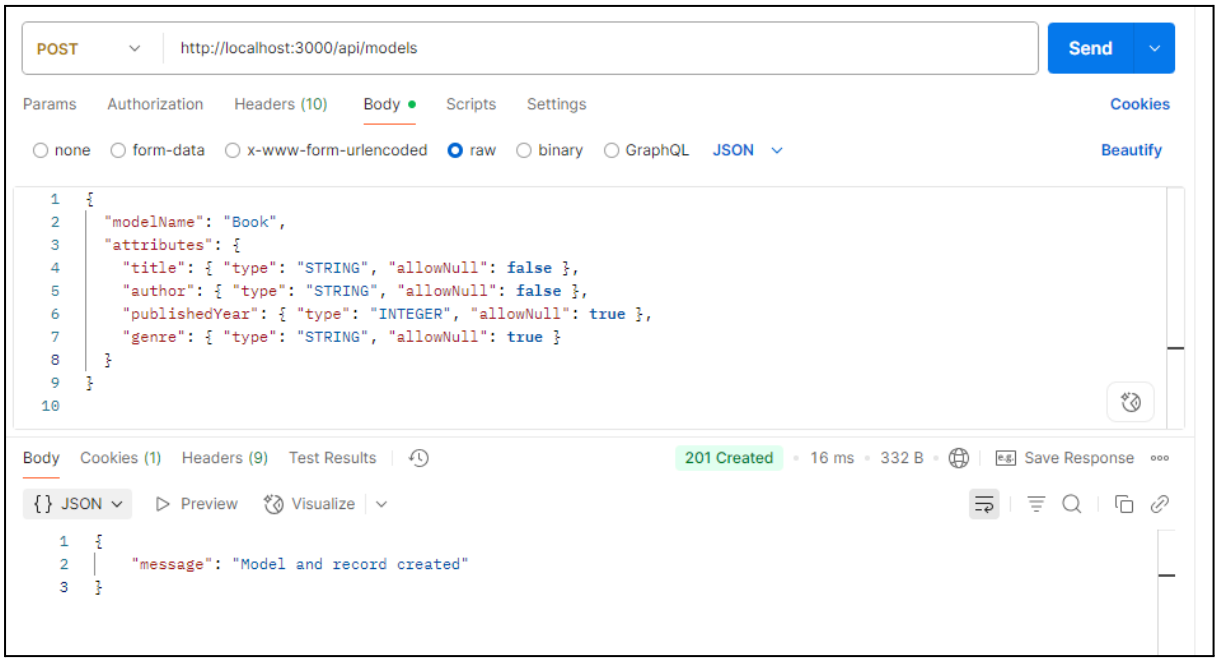
1. Product



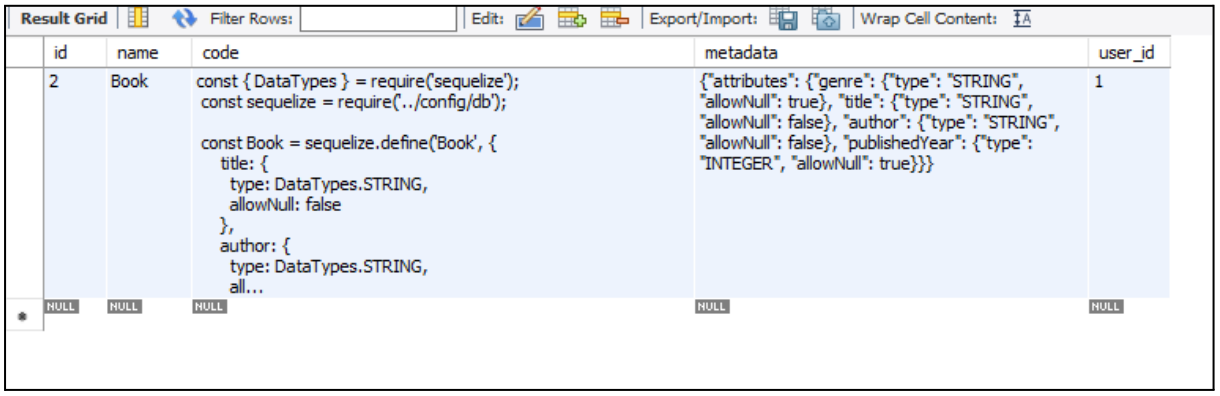
Database



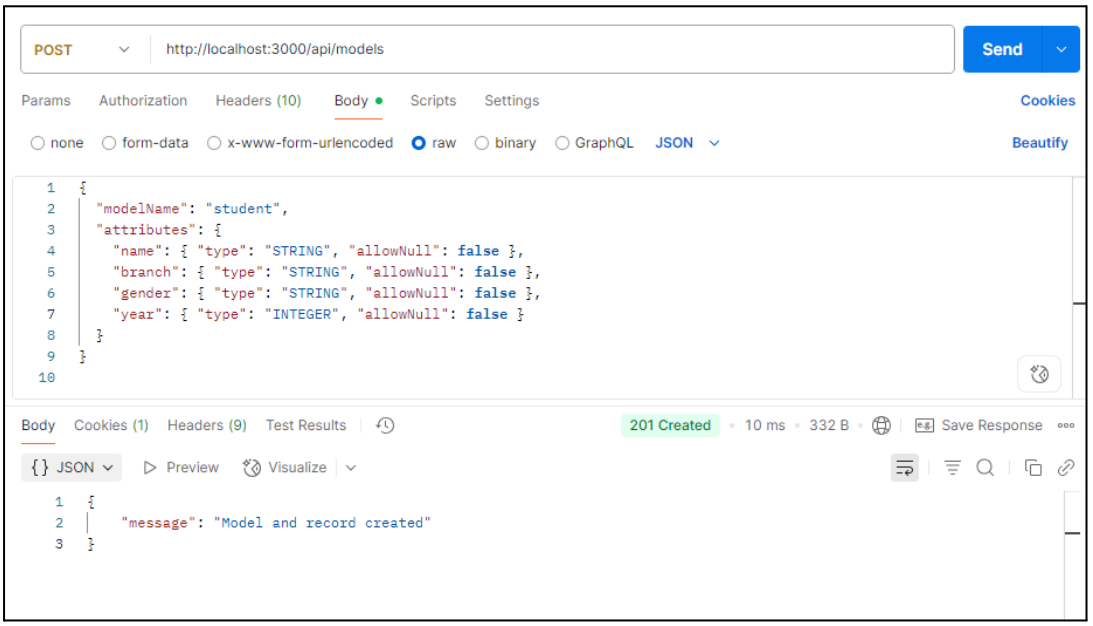
2. Book



Database



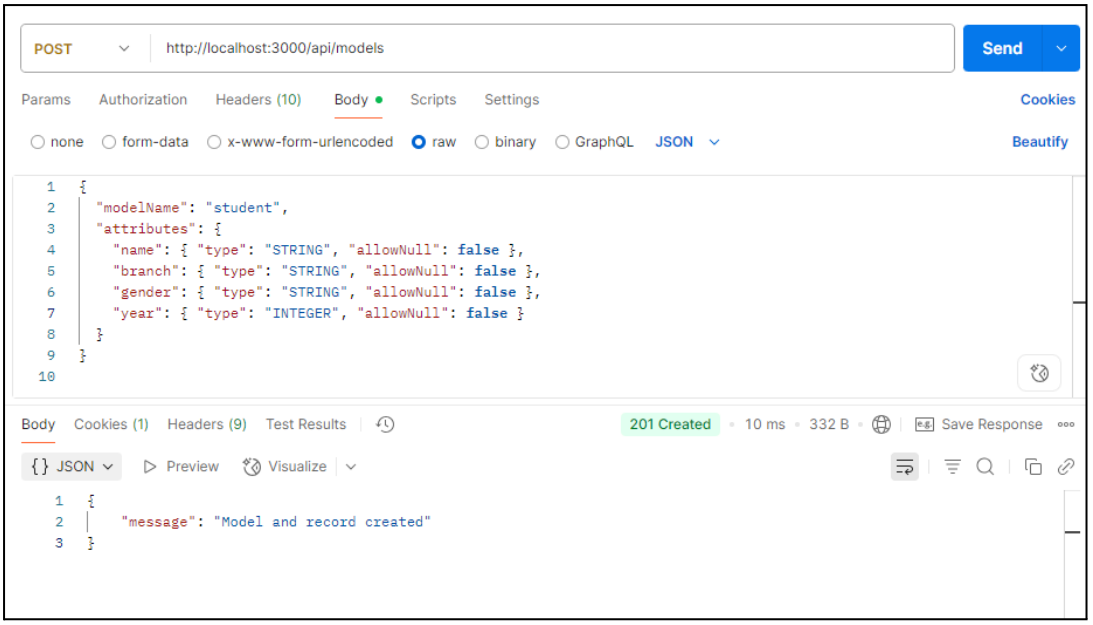
3. Student



Database

id	name	code	metadata	user_id
1	Product	const { DataTypes } = require('sequelize'); const sequelize = r...	{"attributes": {"price": {"type": "FLOAT", "allow...	1
2	Book	const { DataTypes } = require('sequelize'); const sequelize = r...	{"attributes": {"Pages": {"type": "INTEGER", "all...	1
4	student	const { DataTypes } = require('sequelize'); const sequelize = r...	{"attributes": {"name": {"type": "STRING", "allo...	1
NULL	NULL	NULL	NULL	NULL

After updation in code:



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

id	name	code	metadata	user_id

2	Book	const { DataTypes } = require('sequelize');
		const sequelize = require('../config/db');
		const Book = sequelize.define('Book', {
		title: {
		type: DataTypes.STRING,
		allowNull: false
		},
		author: {
		type: DataTypes.STRING,
		allo...
			{"attributes": {"Pages": {"type": "INTEGER", "allowNull": true}, "title": {"type": "STRING", "allowNull": false}, "author": {"type": "STRING", "allowNull": false}, "publishedYear": {"type": "INTEGER", "allowNull": true}}, "association": []}	1

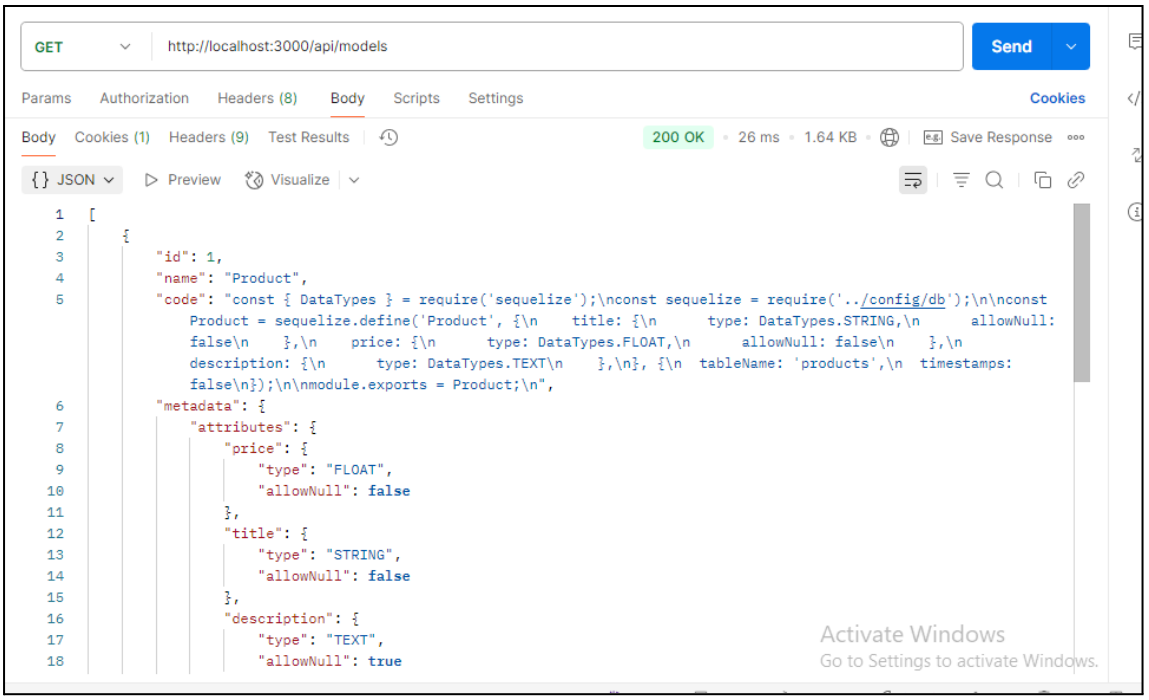
5	student	const { Model, DataTypes } = require('sequelize');
		const sequelize = require('../config/db');
		
		class student extends Model {}
		
		student.init({
		name: {
		type: DataTypes.STRING,
		allowNull: false
		},
		branch: {
		type: DataTypes.STRI...
			{"attributes": {"name": {"type": "STRING", "allowNull": false}, "year": {"type": "INTEGER", "allowNull": false}, "branch": {"type": "STRING", "allowNull": false}, "gender": {"type": "STRING", "allowNull": false}}}	1

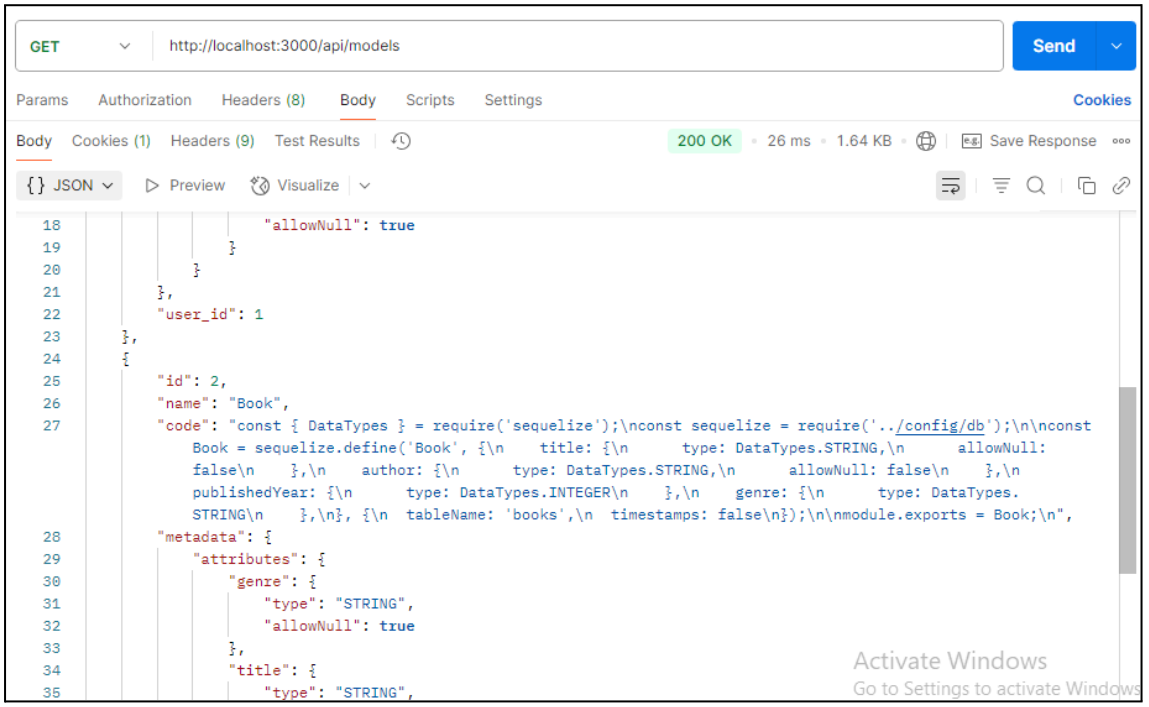
NULL	NULL	NULL	NULL	NULL

2. Getting all the models created by user_id : 1

URL: **GET** http://localhost:3000/api/models

Postman:

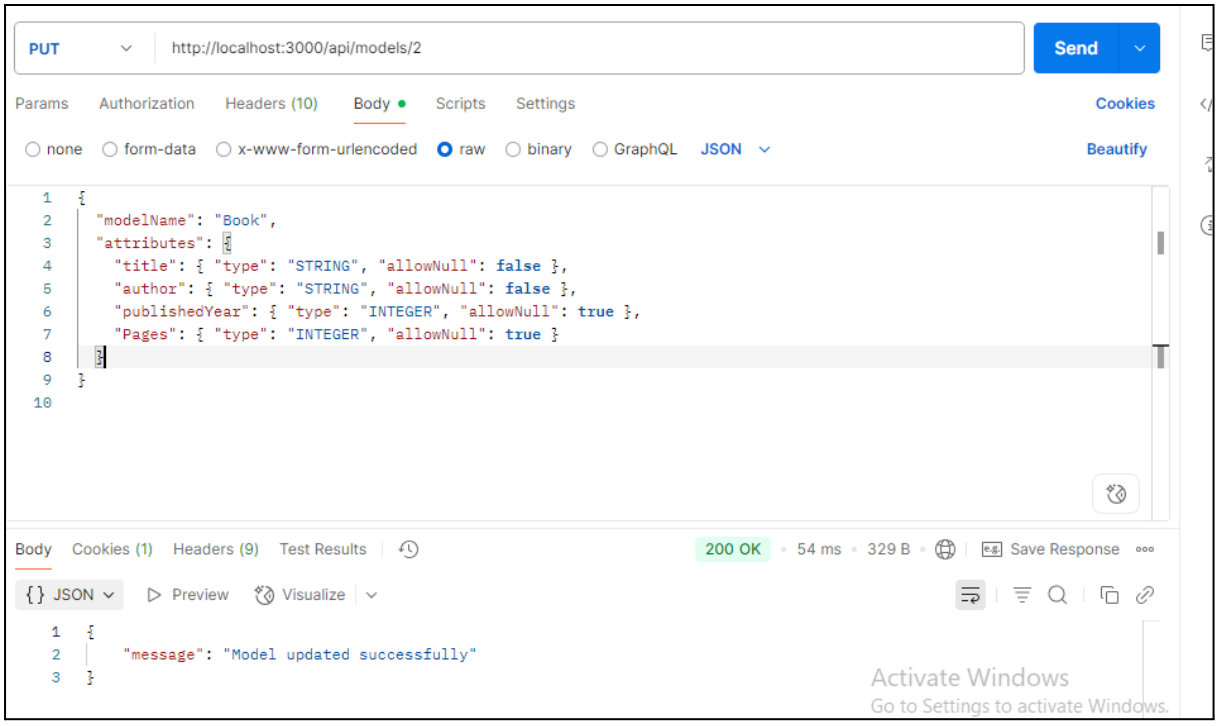




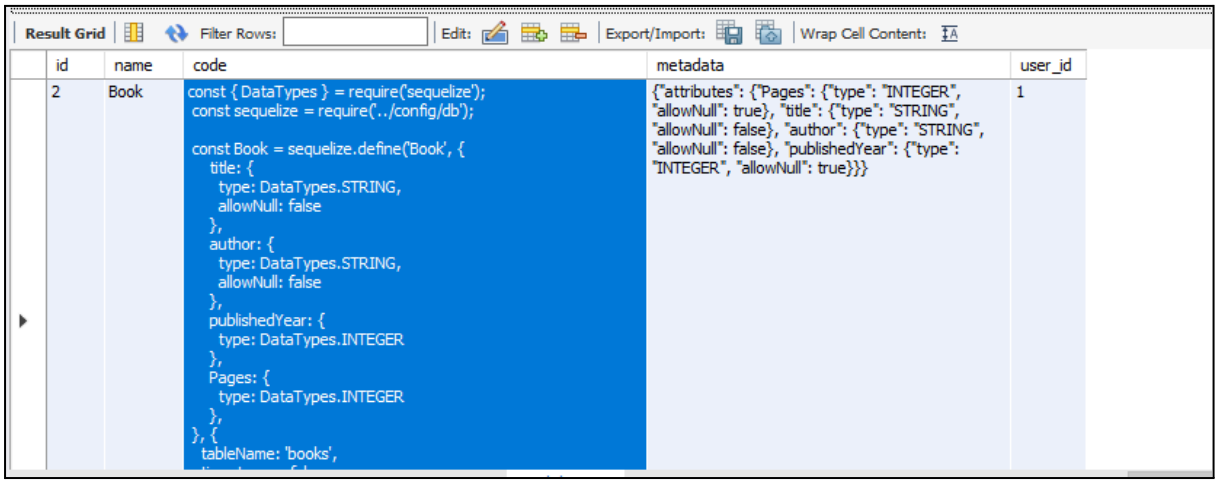
3. Updating the existing model → 2 [Book]

URL: **PUT** http://localhost:3000/api/models/2

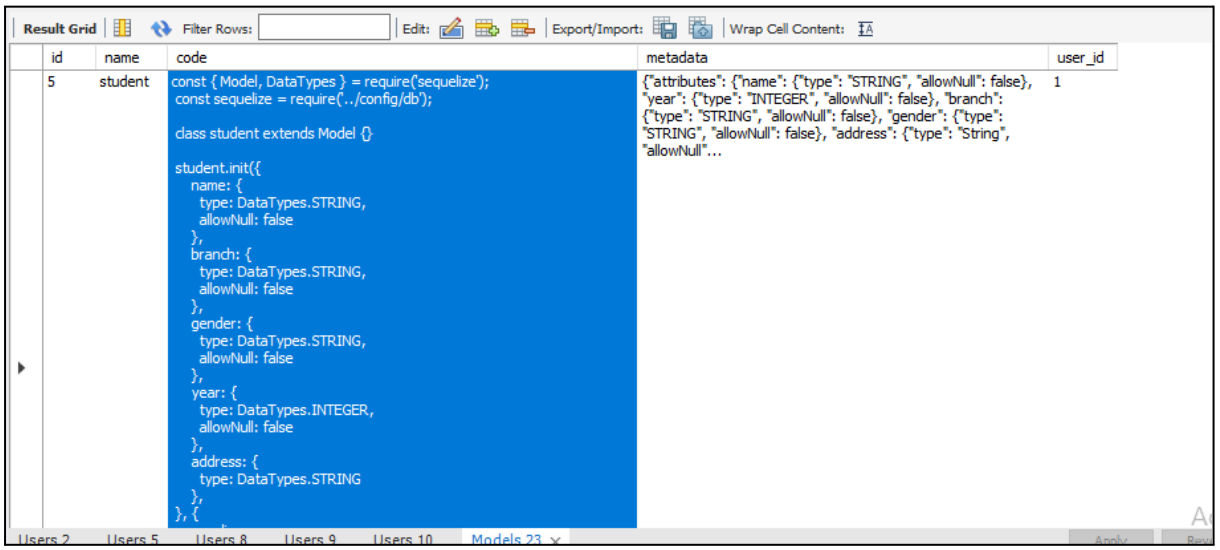
Postman:



Database



After updation in code:



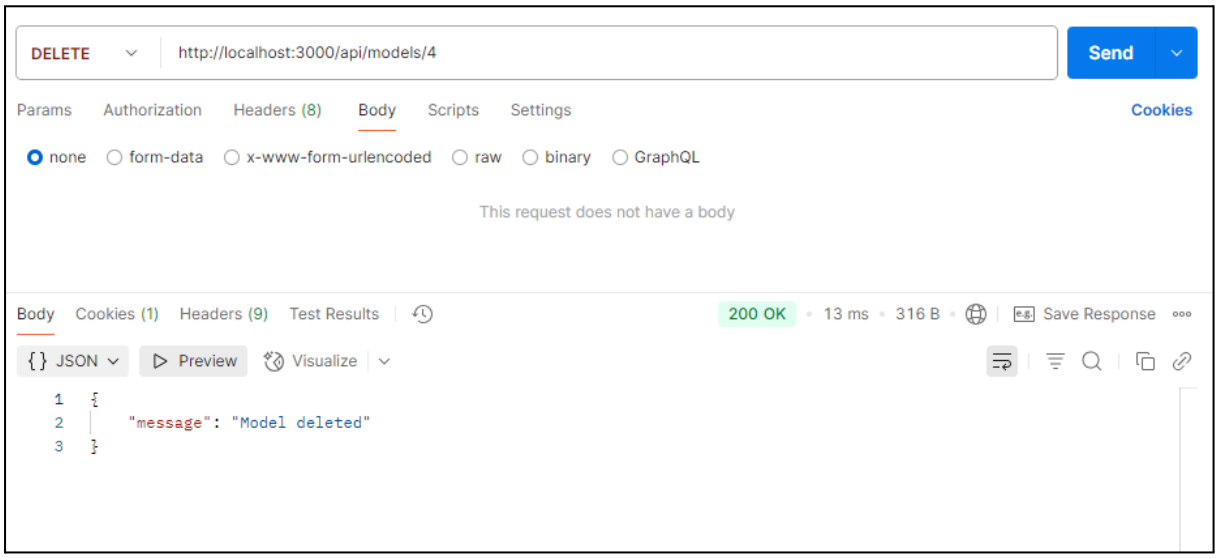
id	name	code	metadata	user_id

5	student	const { Model, DataTypes } = require('sequelize');
		const sequelize = require('../config/db');
		
		class student extends Model {}
		
		student.init({
		name: {
		type: DataTypes.STRING,
		allowNull: false
		},
		branch: {
		type: DataTypes.STRING,
		allowNull: false
		},
		gender: {
		type: DataTypes.STRING,
		allowNull: false
		},
		year: {
		type: DataTypes.INTEGER,
		allowNull: false
		},
		address: {
		type: DataTypes.STRING
		},
		}, {
			{"attributes": {"name": {"type": "STRING", "allowNull": false}, "year": {"type": "INTEGER", "allowNull": false}, "branch": {"type": "STRING", "allowNull": false}, "gender": {"type": "STRING", "allowNull": false}, "address": {"type": "String", "allowNull"...	1

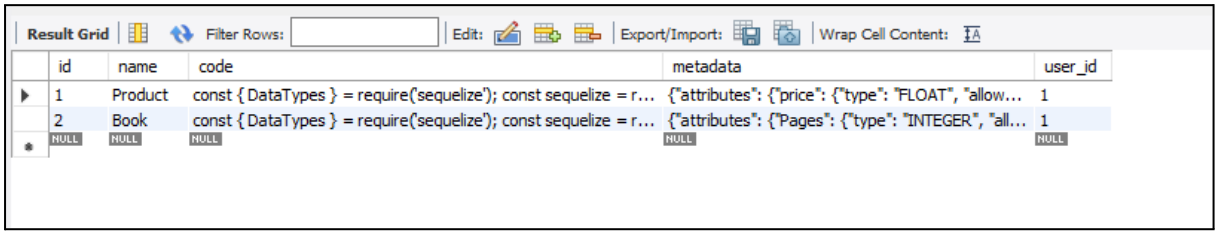
4. Deleting the existing model → 4 [Student]

URL: DELETE http://localhost:3000/api/models/4

Postman:



Database



id	name	code	metadata	user_id
1	Product	const { DataTypes } = require('sequelize'); const sequelize = r...	{"attributes": {"price": {"type": "FLOAT", "allow...	1
2	Book	const { DataTypes } = require('sequelize'); const sequelize = r...	{"attributes": {"Pages": {"type": "INTEGER", "all...	1
NULL	NULL	NULL	NULL	NULL

RELATIONSHIP API's

Overview:

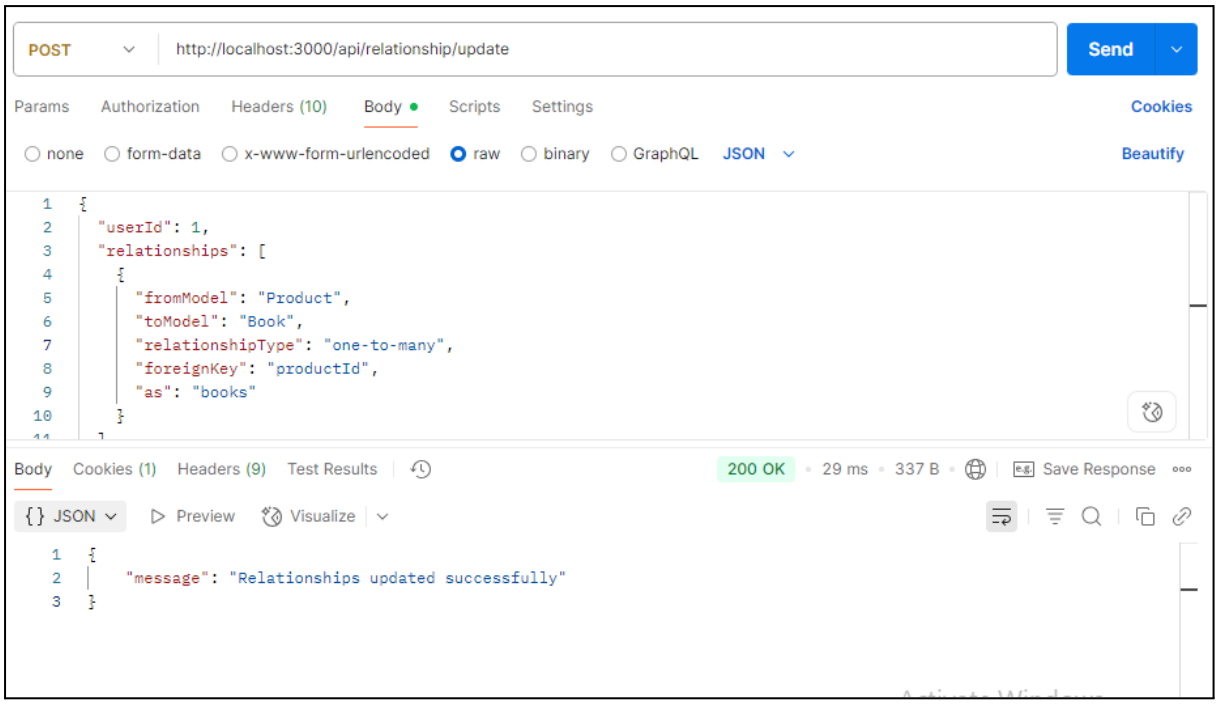
This module provides APIs to define, update, and delete associations between two models. The system ensures that both forward and reverse associations are stored and reflected in the Sequelize code and metadata, making models fully relational and manageable through the UI or programmatically.

Basic URL: http://localhost:3000/api/relationship

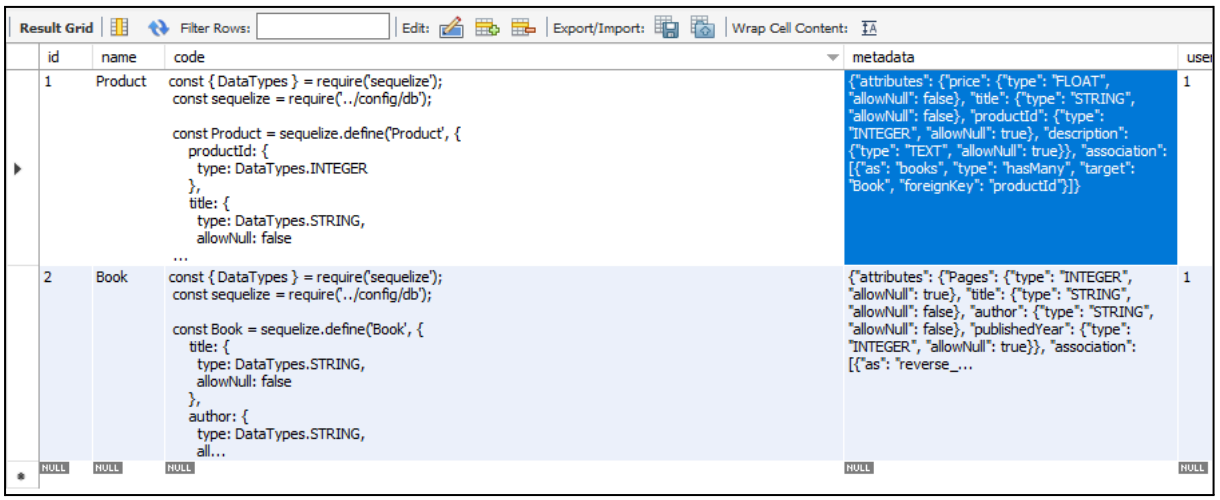
1. Updating the code by adding the relation between Product & book→ one-to-many

URL: **POST** http://localhost:3000/api/relationship/update

Postman:



Database → association added [hasMany]



id	name	code	metadata	user

1	Product	const { DataTypes } = require('sequelize');
		const sequelize = require('../config/db');
		
		const Product = sequelize.define('Product', {
		productId: {
		type: DataTypes.INTEGER
		},
		title: {
		type: DataTypes.STRING,
		allowNull: false
		...
			{"attributes": {"price": {"type": "FLOAT", "allowNull": false}, "title": {"type": "STRING", "allowNull": false}, "productId": {"type": "INTEGER", "allowNull": true}, "description": {"type": "TEXT", "allowNull": true}}, "association": [{"as": "books", "type": "hasMany", "target": "Book", "foreignKey": "productId"}]}	1

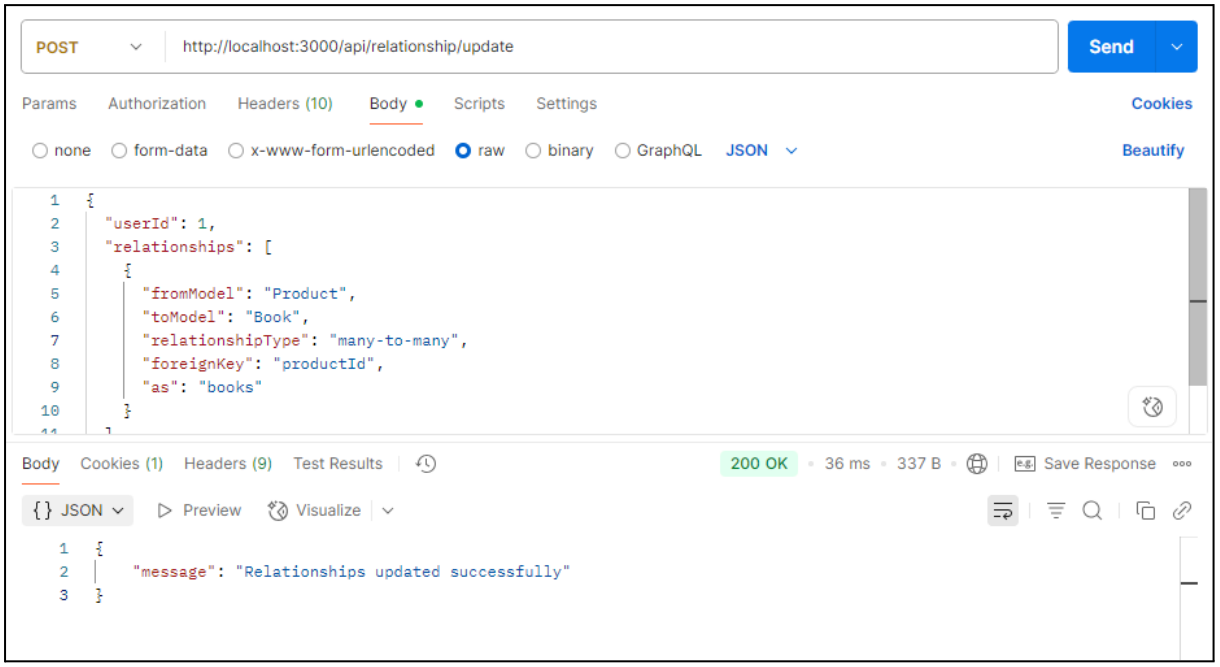
2	Book	const { DataTypes } = require('sequelize');
		const sequelize = require('../config/db');
		
		const Book = sequelize.define('Book', {
		title: {
		type: DataTypes.STRING,
		allowNull: false
		},
		author: {
		type: DataTypes.STRING,
		all...
			{"attributes": {"Pages": {"type": "INTEGER", "allowNull": true}, "title": {"type": "STRING", "allowNull": false}, "author": {"type": "STRING", "allowNull": false}, "publishedYear": {"type": "INTEGER", "allowNull": true}}, "association": [{"as": "reverse_...	1

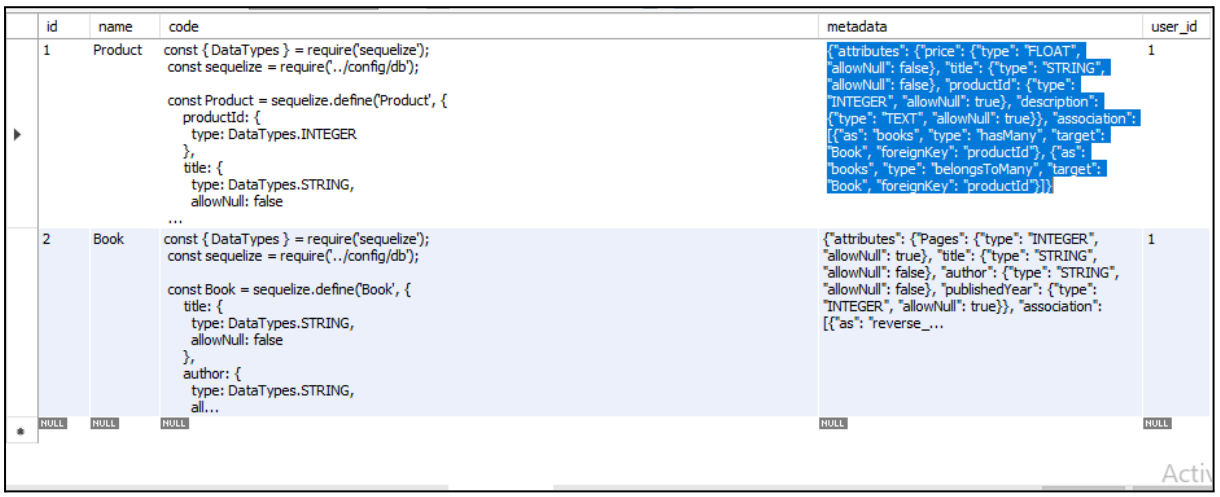
NULL	NULL	NULL	NULL	NULL

2. Updating the code by another relation between Product & book→ many-to-many

URL: **POST** http://localhost:3000/api/relationship/update

Postman:





id	name	code	metadata	user_id

1	Product	const { DataTypes } = require('sequelize');
		const sequelize = require('../config/db');
		
		const Product = sequelize.define('Product', {
		productId: {
		type: DataTypes.INTEGER
		},
		title: {
		type: DataTypes.STRING,
		allowNull: false
		...
			{"attributes": {"price": {"type": "FLOAT", "allowNull": false}, "title": {"type": "STRING", "allowNull": false}, "productId": {"type": "INTEGER", "allowNull": true}, "description": {"type": "TEXT", "allowNull": true}}, "association": [{"as": "books", "type": "hasMany", "target": "Book", "foreignKey": "productId"}, {"as": "books", "type": "belongsToMany", "target": "Book", "foreignKey": "productId"}]}	1

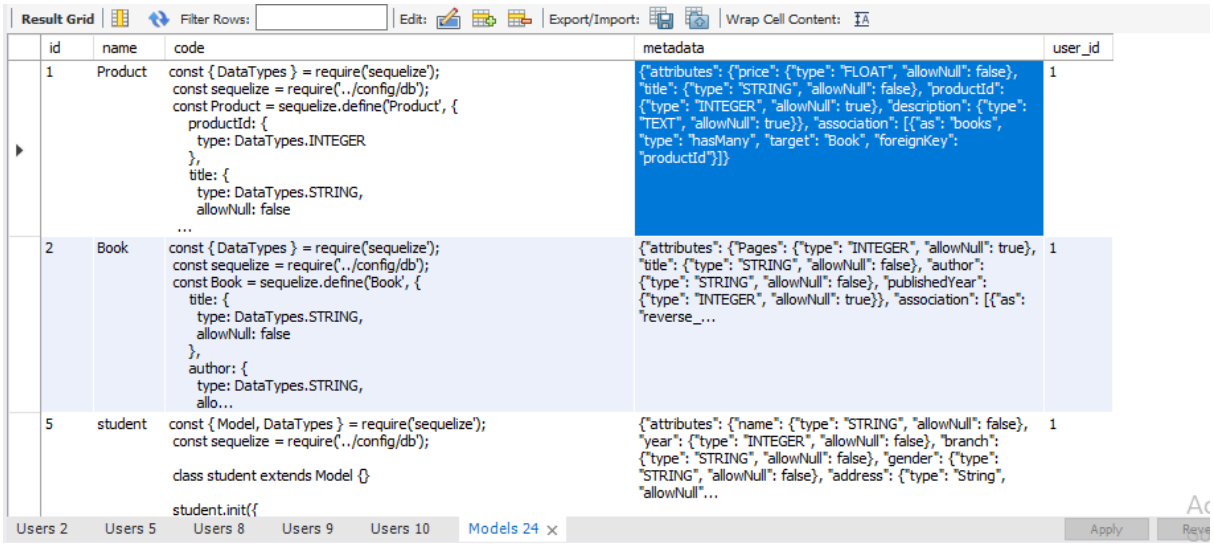
2	Book	const { DataTypes } = require('sequelize');
		const sequelize = require('../config/db');
		
		const Book = sequelize.define('Book', {
		title: {
		type: DataTypes.STRING,
		allowNull: false
		},
		author: {
		type: DataTypes.STRING,
		all...
			{"attributes": {"Pages": {"type": "INTEGER", "allowNull": true}, "title": {"type": "STRING", "allowNull": false}, "author": {"type": "STRING", "allowNull": false}, "publishedYear": {"type": "INTEGER", "allowNull": true}}, "association": [{"as": "reverse_...	1

NULL	NULL	NULL	NULL	NULL

ISSUE: **Duplicate Associations in Metadata When Updating Relationships**

→ Instead of replacing the existing one-to-many relationship, the system **added** a new many-to-many relationship

After Correcting the Issue:



Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

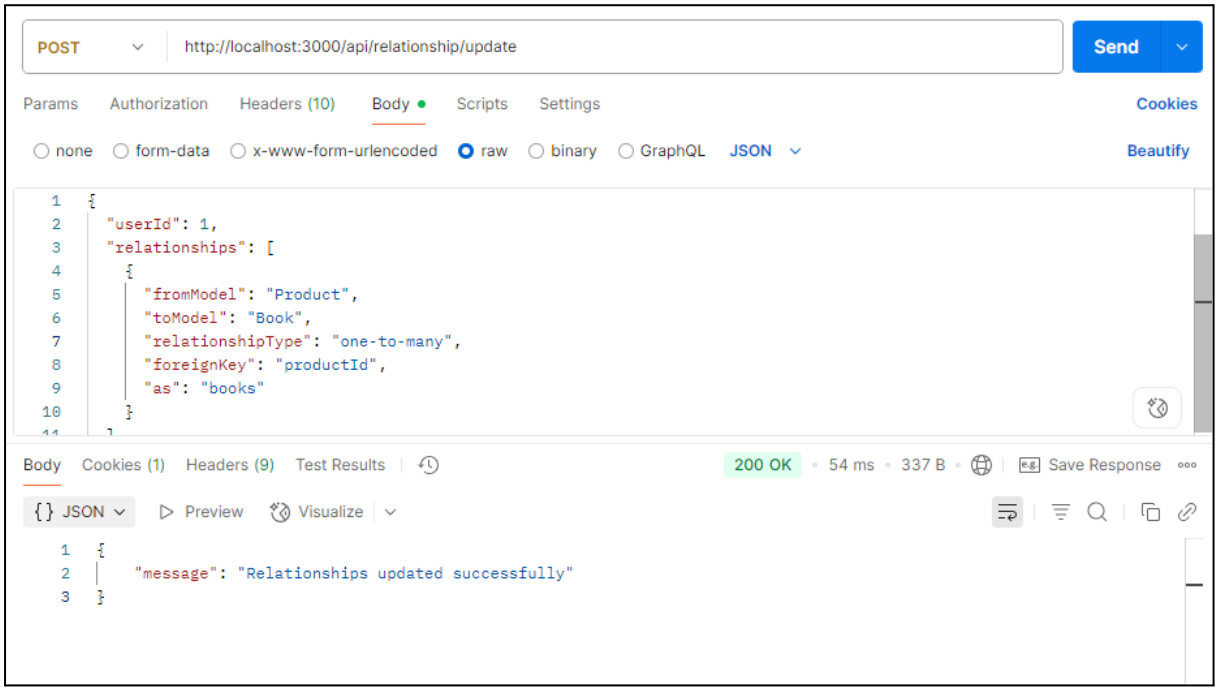
id	name	code	metadata	user_id

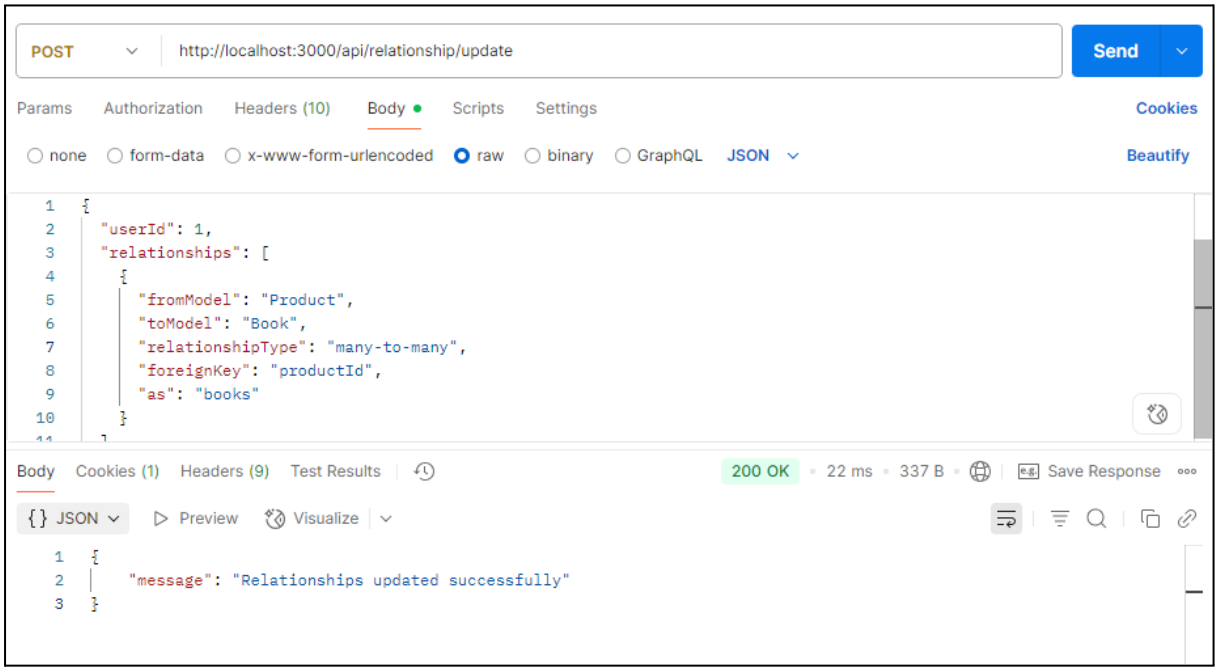
1	Product	const { DataTypes } = require('sequelize');
		const sequelize = require('../config/db');
		const Product = sequelize.define('Product', {
		productId: {
		type: DataTypes.INTEGER
		},
		title: {
		type: DataTypes.STRING,
		allowNull: false
		...
			{"attributes": {"price": {"type": "FLOAT", "allowNull": false}, "title": {"type": "STRING", "allowNull": false}, "productId": {"type": "INTEGER", "allowNull": true}, "description": {"type": "TEXT", "allowNull": true}}, "association": [{"as": "books", "type": "hasMany", "target": "Book", "foreignKey": "productId"}]}	1

2	Book	const { DataTypes } = require('sequelize');
		const sequelize = require('../config/db');
		const Book = sequelize.define('Book', {
		title: {
		type: DataTypes.STRING,
		allowNull: false
		},
		author: {
		type: DataTypes.STRING,
		allo...
			{"attributes": {"Pages": {"type": "INTEGER", "allowNull": true}, "title": {"type": "STRING", "allowNull": false}, "author": {"type": "STRING", "allowNull": false}, "publishedYear": {"type": "INTEGER", "allowNull": true}}, "association": [{"as": "reverse_...	1

5	student	const { Model, DataTypes } = require('sequelize');
		const sequelize = require('../config/db');
		
		class student extends Model {}
		
		student.init({
			{"attributes": {"name": {"type": "STRING", "allowNull": false}, "year": {"type": "INTEGER", "allowNull": false}, "branch": {"type": "STRING", "allowNull": false}, "gender": {"type": "STRING", "allowNull": false}, "address": {"type": "String", "allowNull"...	1

Users 2 | Users 5 | Users 8 | Users 9 | Users 10 | Models 24 | Apply

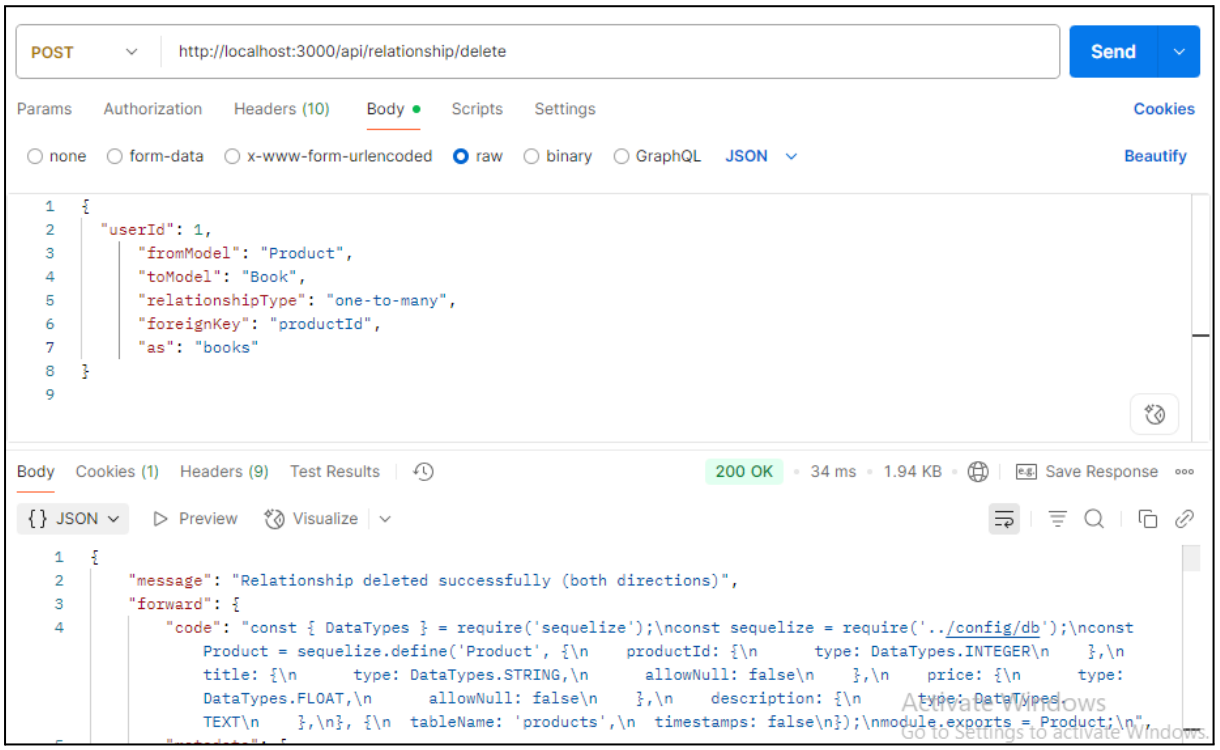




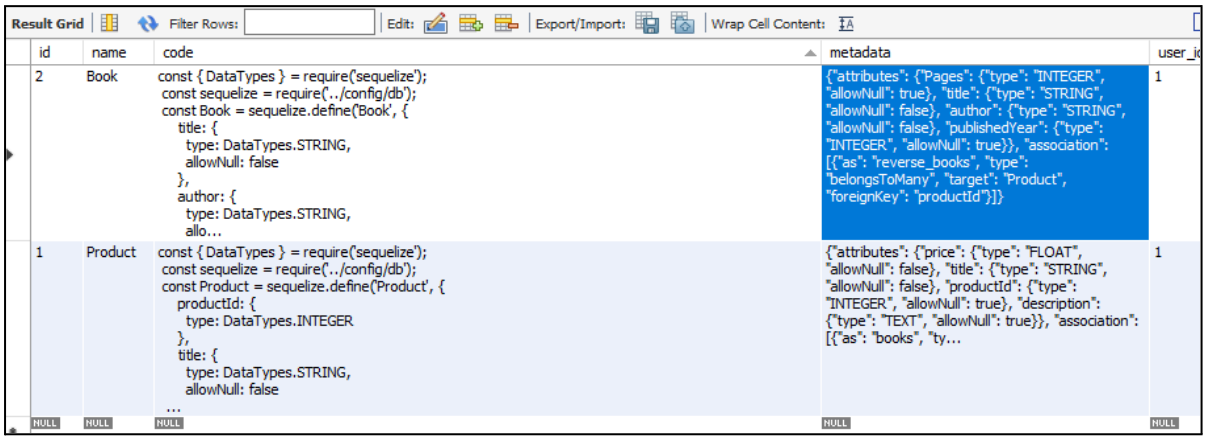
3. Delete the relation relation between Product & book→ one-to-many

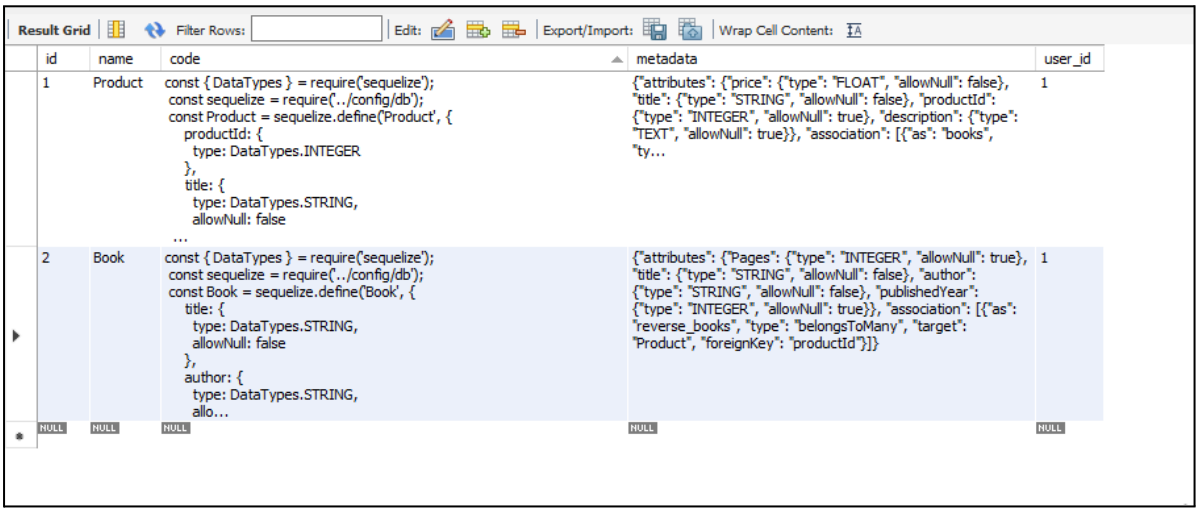
URL: **POST** http://localhost:3000/api/relationship/delete

Postman:



Database → relation deleted successfully from both side

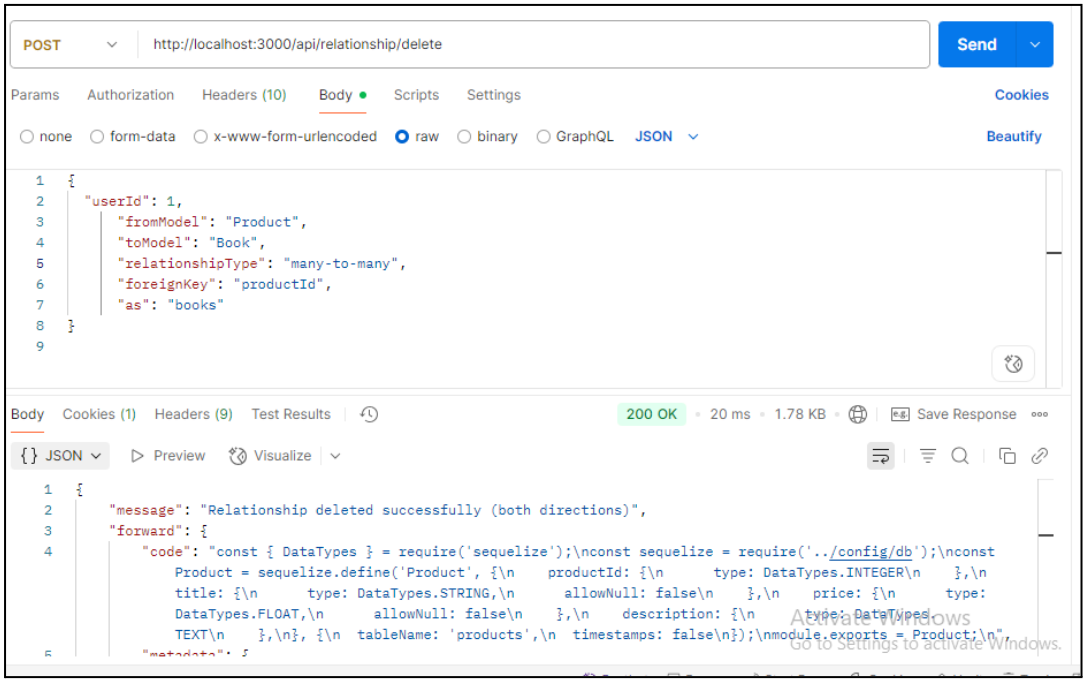




4. Delete the another relation between Product & book→ many-to-many

URL: **POST** http://localhost:3000/api/relationship/delete

Postman:



Database → relation deleted successfully from both side