# Node.js Authentication System Report

Pakinam Khaled

April 2025

**Abstract**

This report explains the full implementation of a Node.js-based authentication system. It includes password hashing using PBKDF2, JSON Web Token (JWT) generation and verification using HMAC-SHA256, and a complete Express server with protected routes. The implementation strictly uses Node.js built-in `crypto` module, as required by the assignment.

## 1 Project Structure

- **password-utils.js**: Handles password hashing and verification.
- **jwt-utils.js**: Handles JWT creation and validation.
- **server.js**: Express server with registration, login, and book management APIs.

## 2 Password Hashing (password-utils.js)

**hashPassword and verifyPassword functions:**

```
// password-utils.js
const crypto = require('crypto');

function hashPassword(password) {
  const salt = crypto.randomBytes(16).toString('hex');
  const hash = crypto.pbkdf2Sync(password, salt, 100000, 64, 'sha512').
     toString('hex');
  return `${salt}:${hash}`;
}

function verifyPassword(password, storedHash) {
  const [salt, originalHash] = storedHash.split(':');
  const hash = crypto.pbkdf2Sync(password, salt, 100000, 64, 'sha512').
     toString('hex');
  return hash === originalHash;
}

module.exports = { hashPassword, verifyPassword };
```

# 3 JWT Implementation (jwt-utils.js)

**signJWT and verifyJWT functions:**

```
// jwt-utils.js
const crypto = require('crypto');

function base64url(input) {
  return Buffer.from(JSON.stringify(input)).toString('base64')
    .replace(/=/g, '')
    .replace(/\+/g, '-')
    .replace(/\//g, '_');
}

function signJWT(payload, secret, expiresInSeconds = 3600) {
  const header = { alg: 'HS256', typ: 'JWT' };
  const exp = Math.floor(Date.now() / 1000) + expiresInSeconds;
  const fullPayload = { ...payload, exp };

  const encodedHeader = base64url(header);
  const encodedPayload = base64url(fullPayload);
  const signature = crypto
    .createHmac('sha256', secret)
    .update(`${encodedHeader}.${encodedPayload}`)
    .digest('base64')
    .replace(/=/g, '')
    .replace(/\+/g, '-')
    .replace(/\//g, '_');

  return `${encodedHeader}.${encodedPayload}.${signature}`;
}

function verifyJWT(token, secret) {
  const [encodedHeader, encodedPayload, receivedSignature] = token.
     split('.');

  const validSignature = crypto
    .createHmac('sha256', secret)
    .update(`${encodedHeader}.${encodedPayload}`)
    .digest('base64')
    .replace(/=/g, '')
    .replace(/\+/g, '-')
    .replace(/\//g, '_');

  if (validSignature !== receivedSignature) {
    throw new Error('Invalid signature');
  }

  const payload = JSON.parse(Buffer.from(encodedPayload, 'base64').
     toString());
  const now = Math.floor(Date.now() / 1000);

  if (payload.exp < now) {
    throw new Error('Token expired');
  }

  return payload;
}
```

```
module.exports = { signJWT, verifyJWT };
```

# 4 Express Server (server.js)

**Complete implementation:**

```
// server.js
const express = require('express');
const { hashPassword, verifyPassword } = require('./password-utils');
const { signJWT, verifyJWT } = require('./jwt-utils');

const app = express();
app.use(express.json());

const PORT = 3000;
const JWT_SECRET = 'my_secret';

const books = [
  { id: 1, title: '1984', author: 'George Orwell' },
  { id: 2, title: 'The Hobbit', author: 'J.R.R. Tolkien' },
];

const users = [];

function authenticate(req, res, next) {
  const authHeader = req.headers['authorization'];
  const token = authHeader?.split(' ')[1];

  if (!token) return res.status(401).json({ message: 'Token required'
     });

  try {
    const payload = verifyJWT(token, JWT_SECRET);
    req.user = payload;
    next();
  } catch (err) {
    res.status(403).json({ message: 'Invalid token', error: err.message
        });
  }
}

function authorizeAdmin(req, res, next) {
  if (req.user?.role !== 'admin')
    return res.status(403).json({ message: 'Admin role required' });

  next();
}

app.post('/register', (req, res) => {
  const { username, password, role } = req.body;

  if (!username || !password || !role)
    return res.status(400).json({ message: 'All fields required' });

  const existingUser = users.find((u) => u.username === username);
```

```
  if (existingUser)
    return res.status(409).json({ message: 'User already exists' });

  const hashedPassword = hashPassword(password);
  users.push({ username, password: hashedPassword, role });

  res.status(201).json({ message: 'User registered' });
});

app.post('/login', (req, res) => {
  const { username, password } = req.body;
  const user = users.find((u) => u.username === username);

  if (!user) return res.status(401).json({ message: 'Invalid
      credentials' });

  const isValid = verifyPassword(password, user.password);
  if (!isValid) return res.status(401).json({ message: 'Invalid
      credentials' });

  const token = signJWT({ username: user.username, role: user.role },
      JWT_SECRET);
  res.json({ token });
});

app.get('/books', (req, res) => {
  res.json(books);
});

app.get('/books/:id', (req, res) => {
  const book = books.find((b) => b.id === parseInt(req.params.id));
  if (!book) return res.status(404).json({ message: 'Book not found' })
      ;
  res.json(book);
});

app.post('/books', authenticate, authorizeAdmin, (req, res) => {
  const { title, author } = req.body;
  const newBook = { id: books.length + 1, title, author };
  books.push(newBook);
  res.status(201).json(newBook);
});

app.put('/books/:id', authenticate, authorizeAdmin, (req, res) => {
  const id = parseInt(req.params.id);
  const book = books.find((b) => b.id === id);
  if (!book) return res.status(404).json({ message: 'Book not found' })
      ;

  const { title, author } = req.body;
  book.title = title ?? book.title;
  book.author = author ?? book.author;
  res.json(book);
});

app.delete('/books/:id', authenticate, authorizeAdmin, (req, res) => {
  const id = parseInt(req.params.id);
  const index = books.findIndex((b) => b.id === id);
```

```
  if (index === -1) return res.status(404).json({ message: 'Book not
     found' });

  const deleted = books.splice(index, 1);
  res.json(deleted[0]);
});

app.listen(PORT, () => {
  console.log(`Server running on http://localhost:${PORT}`);
});
```

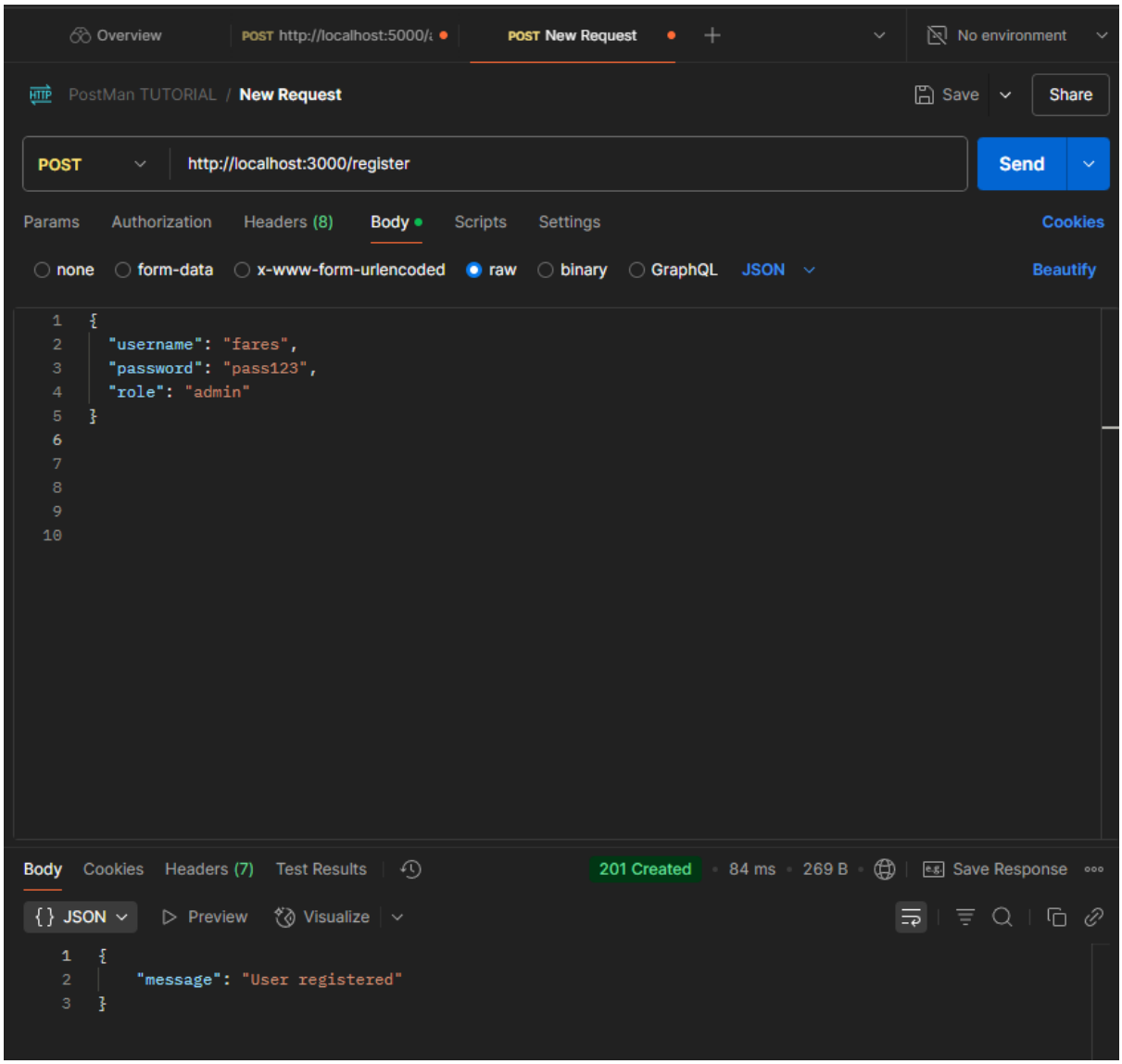
# 5 How to Run

1. Clone the project folder
2. Run `npm install`
3. Start the server with: `node server.js`

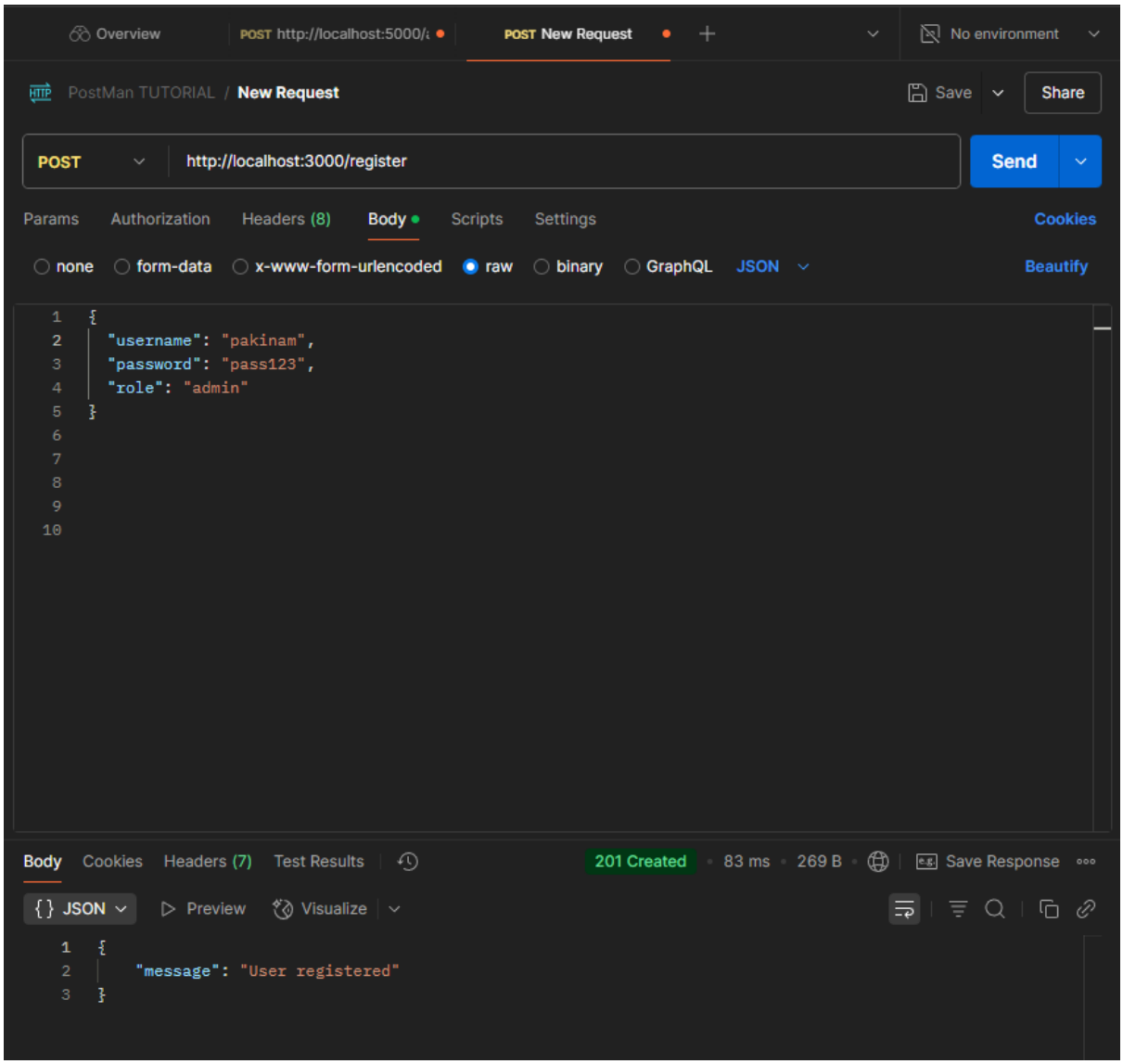
# 6 Postman Testing

- Register: `POST /register`
- Login: `POST /login`
- View books: `GET /books`
- Admin book actions: `POST, PUT, DELETE /books`

## Register Test



## Login Test


Overview
POST http://localhost:5000/
POST New Request
No environment
PostMan TUTORIAL / New Request
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
{
"username": "pakinam",
"password": "pass123",
"role": "admin"
}
Body
Cookies
Headers (7)
Test Results
201 Created
83 ms
269 B
Save Response
JSON
Preview
Visualize
{
"message": "User registered"
}

## Get Books

PostMan TUTORIAL / **New Request**

Save | Share

POST http://localhost:3000/login Send

Params | Authorization | Headers (8) | Body | Scripts | Settings | Cookies

none | form-data | x-www-form-urlencoded | raw | binary | GraphQL | JSON | Beautify

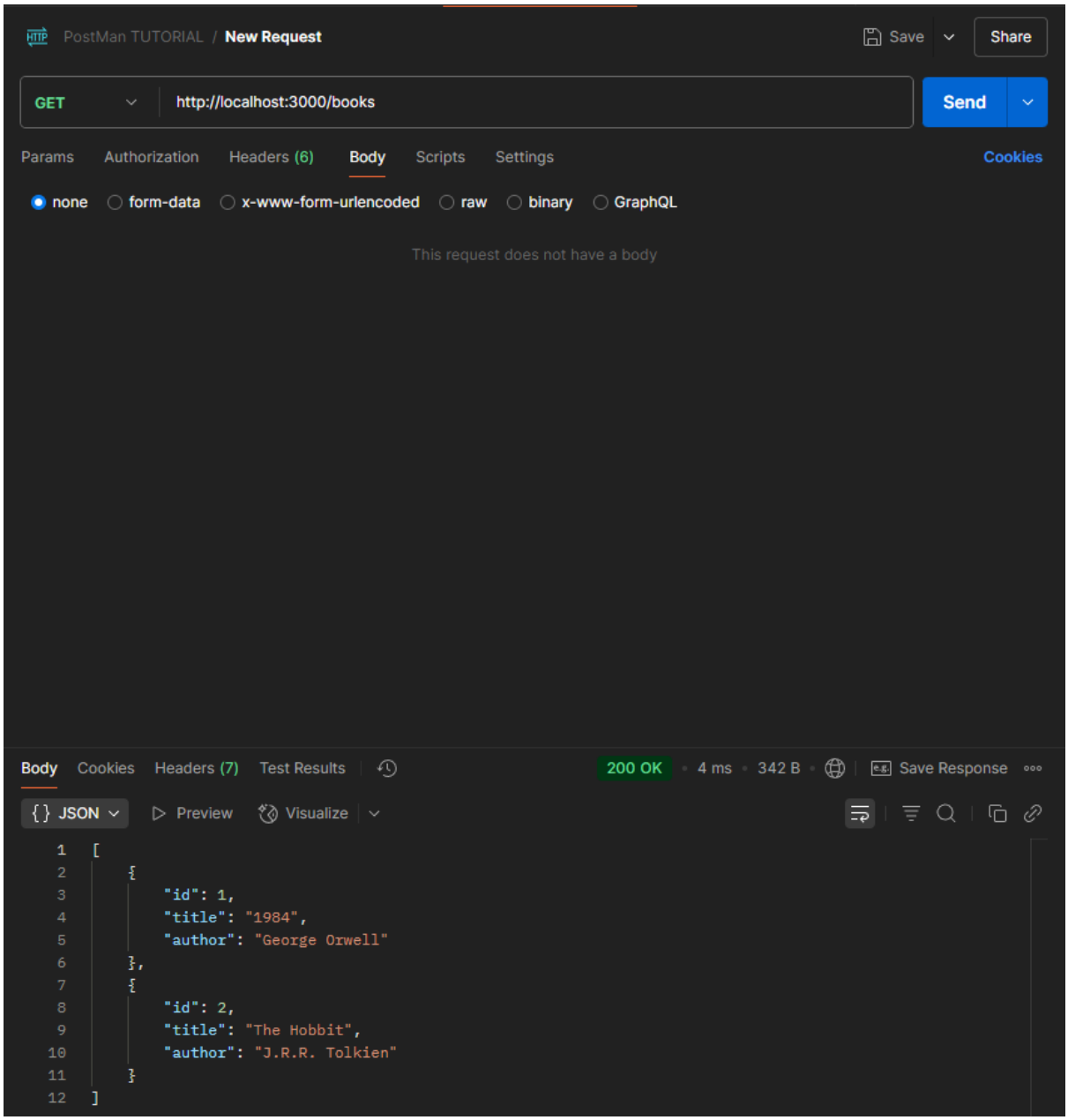
```
{
  "username": "pakinam",
  "password": "pass123"
}
```

Body | Cookies | Headers (7) | Test Results | 200 OK | 82 ms | 401 B | Save Response

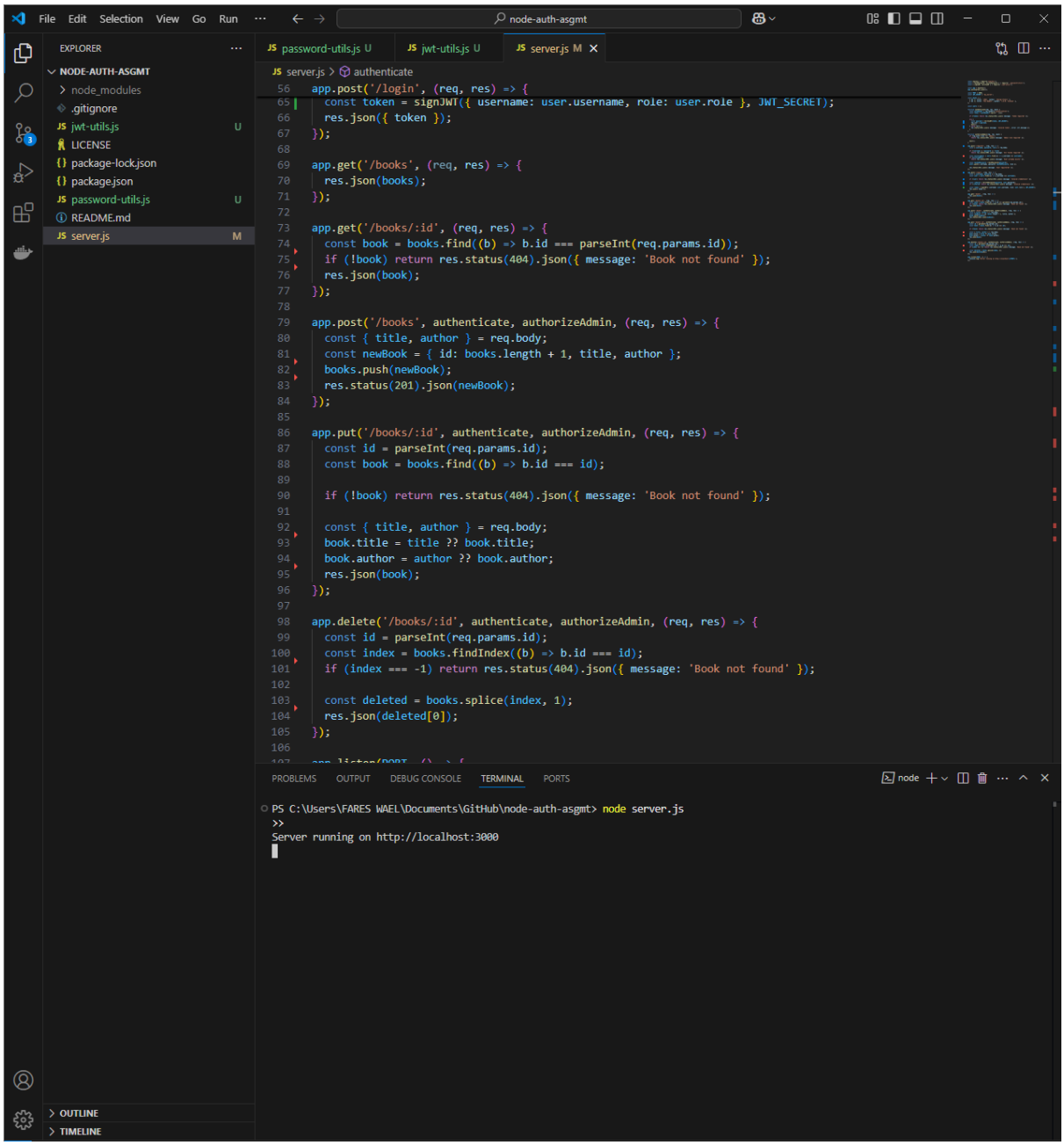
JSON | Preview | Visualize

```
{
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
        eyJ1c2VybmFtZSI6InBha2luYW0iLCJyb2xlIjoiYWRtaW4iLCJleHAiOjE3NDU1NzQ0ODJ9.
        toHsgO420VZmp1t1D2UMQyqlPn4LiedOeNIU7H36n1Y"
}
```

## Add Book


PostMan TUTORIAL / New Request
Save
Share
GET
http://localhost:3000/books
Send
Params
Authorization
Headers (6)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
This request does not have a body
Body
Cookies
Headers (7)
Test Results
200 OK
4 ms
342 B
Save Response
JSON
Preview
Visualize
[
{
"id": 1,
"title": "1984",
"author": "George Orwell"
},
{
"id": 2,
"title": "The Hobbit",
"author": "J.R.R. Tolkien"
}
]

## Running Server




# Conclusion

This system demonstrates secure password hashing and token-based access in Node.js, without any external libraries. It fulfills the full assignment scope.

**Made by Pakinam Khaled**