

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

The image shows the Postman application interface for a new request. The top bar indicates the request is a POST to `http://localhost:5000/`. The main area shows the request details for `POST http://localhost:3000/register`. The request body is set to raw JSON, containing the following data:

```
1 {
2   "username": "fares",
3   "password": "pass123",
4   "role": "admin"
5 }
```

The bottom section displays the response, which is a 201 Created status with a response time of 84 ms and a body size of 269 B. The response body is JSON, containing the following data:

```
1 {
2   "message": "User registered"
3 }
```

Login Test

The image shows the Postman application interface. At the top, there's a navigation bar with 'Overview', 'POST http://localhost:5000/', 'POST New Request', and 'No environment'. Below this, the 'New Request' tab is active, showing a 'POST' method and the URL 'http://localhost:3000/register'. A 'Send' button is visible. The 'Body' tab is selected, showing a JSON body:

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

. The bottom section shows the response: '201 Created' with a status of 83 ms and 269 B. The response body is JSON:

```
{  "message": "User registered"}
```

.

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

```
1 {
2   "username": "pakinam",
3   "password": "pass123",
4   "role": "admin"
5 }
```

Body | Cookies | Headers (7) | Test Results | 201 Created • 83 ms • 269 B • Save Response

{ } JSON | Preview | Visualize

```
1 {
2   "message": "User registered"
3 }
```

Get Books

PostMan TUTORIAL / New Request

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

```
1 {
2   "username": "pakinam",
3   "password": "pass123"
4 }
```

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

{ } JSON Preview Visualize

```
1 {
2   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6InBha2luYW0iLCJyb2x1IjoieYWRtaW4iLCJleHAiOjE3NDU1NzQ0ODJ9.toHsg0420VZmp1t1D2UMQyqlPn4Lied0eNIU7H36n1Y"
3 }
```

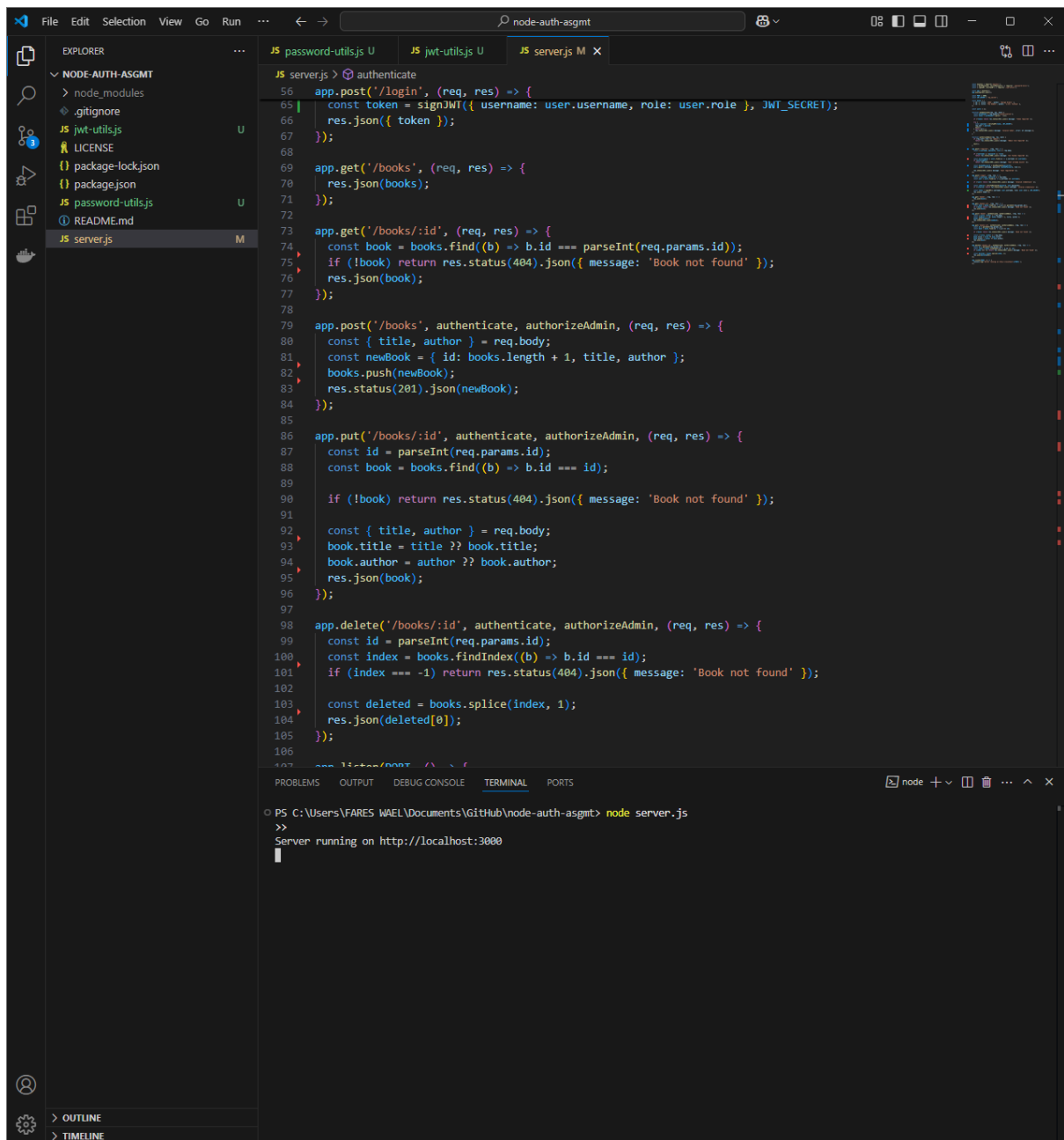

Add Book

The image shows the Postman application interface. At the top, the title bar reads "PostMan TUTORIAL / New Request". The main area is divided into tabs: "Params", "Authorization", "Headers (6)", "Body", "Scripts", and "Settings". The "Body" tab is selected, and the request type is "GET". The URL is "http://localhost:3000/books". The "Send" button is visible. Below the tabs, the "Body" section shows "none" selected, with a message "This request does not have a body".

The response section at the bottom shows a "200 OK" status, a response time of "4 ms", and a size of "342 B". The response is displayed in JSON format, showing a list of two books:

```
1 [
2   {
3     "id": 1,
4     "title": "1984",
5     "author": "George Orwell"
6   },
7   {
8     "id": 2,
9     "title": "The Hobbit",
10    "author": "J.R.R. Tolkien"
11  }
12 ]
```

Running Server



The screenshot shows the Visual Studio Code interface with a project named 'node-auth-asgmt'. The Explorer sidebar on the left shows the file structure, including 'node_modules', '.gitignore', 'LICENSE', 'package-lock.json', 'package.json', 'password-utils.js', 'README.md', and 'server.js'. The main editor displays the 'server.js' file, which contains the following code:

```
56 app.post('/login', (req, res) => {
57   const token = signJWT({ username: user.username, role: user.role }, JWT_SECRET);
58   res.json({ token });
59 });
60
61 app.get('/books', (req, res) => {
62   res.json(books);
63 });
64
65 app.get('/books/:id', (req, res) => {
66   const book = books.find((b) => b.id === parseInt(req.params.id));
67   if (!book) return res.status(404).json({ message: 'Book not found' });
68   res.json(book);
69 });
70
71 app.post('/books', authenticate, authorizeAdmin, (req, res) => {
72   const { title, author } = req.body;
73   const newBook = { id: books.length + 1, title, author };
74   books.push(newBook);
75   res.status(201).json(newBook);
76 });
77
78 app.put('/books/:id', authenticate, authorizeAdmin, (req, res) => {
79   const id = parseInt(req.params.id);
80   const book = books.find((b) => b.id === id);
81   if (!book) return res.status(404).json({ message: 'Book not found' });
82
83   const { title, author } = req.body;
84   book.title = title ?? book.title;
85   book.author = author ?? book.author;
86   res.json(book);
87 });
88
89 app.delete('/books/:id', authenticate, authorizeAdmin, (req, res) => {
90   const id = parseInt(req.params.id);
91   const index = books.findIndex((b) => b.id === id);
92   if (index === -1) return res.status(404).json({ message: 'Book not found' });
93
94   const deleted = books.splice(index, 1);
95   res.json(deleted[0]);
96 });
```

The bottom panel shows the TERMINAL output:

```
PS C:\Users\FARES WAEL\Documents\GitHub\node-auth-asgmt> node server.js
>>
Server running on http://localhost:3000
```

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