Authentication & Authorization

Rujuan Xing

Basic Problem

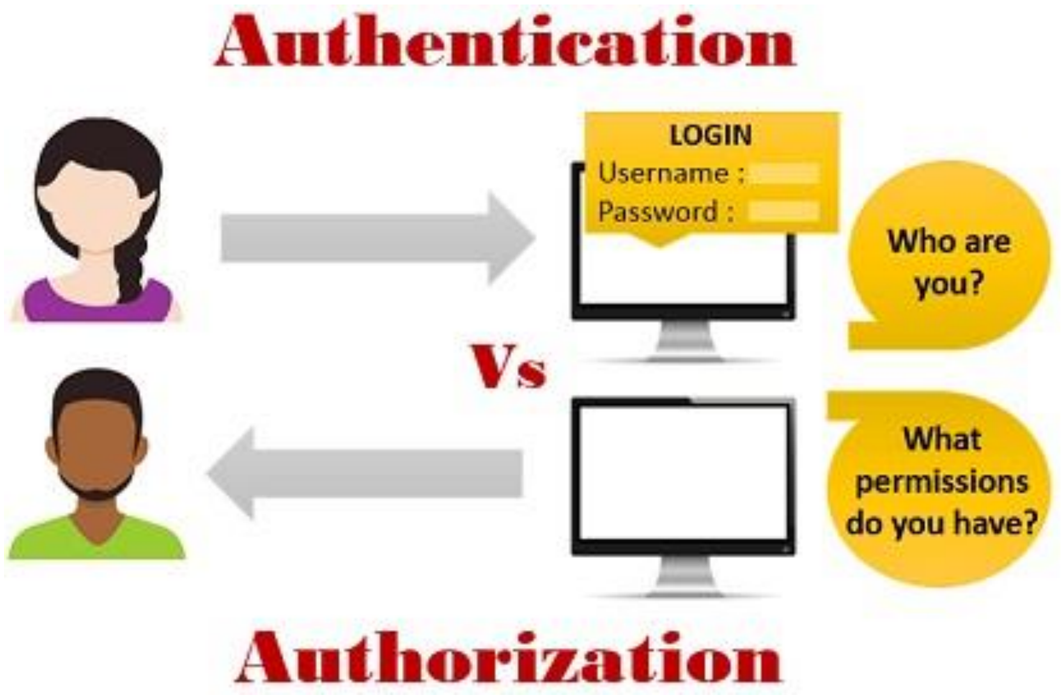
- How do you prove to someone that you are who you claim to be?

Authentication

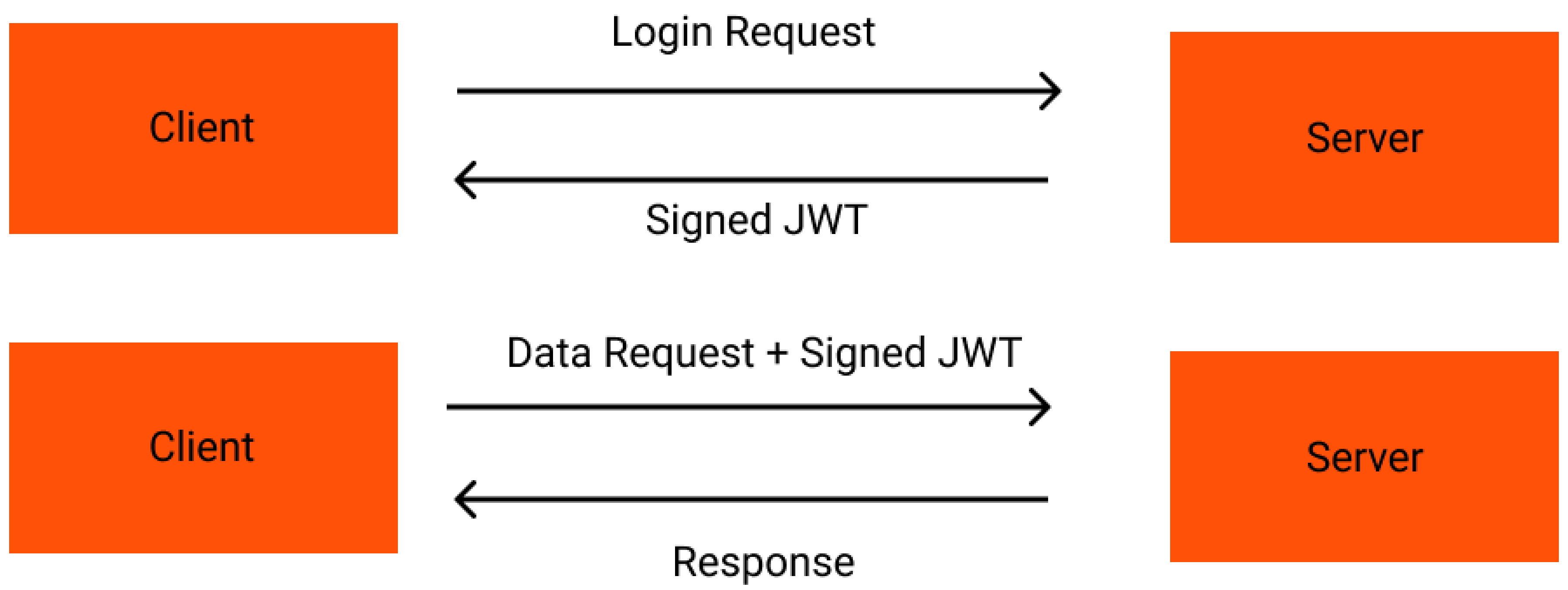
- Authentication is any process by which a system verifies the identity of a user who wishes to access it.
- Authentication may be implemented using Credentials, each of which is composed of a user Id and password. Alternately, Authentication may be implemented with Smart Cards, etc..

Authorization

- **Authorization** is the function of specifying access rights/privileges to resources, which is related to information security and computer security in general and to access control in particular.



Token-Based Authentication Systems



Token-based Authorization System

- Stateless: self contained
- Scalability: no need to store session in memory
- CSRF: no session being used
- Digitally-signed
- Decoupled

What is JSON Web Token?

- JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object.
- JWTs can be signed using a secret (with the **HMAC** algorithm) or a public/private key pair using **RSA**.
- This information can be verified and trusted because it is digitally signed.
- **Compact**: Because of their smaller size, JWTs can be sent through a URL, POST parameter, or inside an HTTP header. Additionally, the smaller size means transmission is fast.
  - Simply a string in the format of **header.payload.signature**
- **Self-contained**: The payload contains all the required information about the user, avoiding the need to query the database more than once.

JSON Web Token Structure

- JSON Web Tokens consist of three parts separated by dots (.), which are:
  - header
  - payload
  - signature

- Therefore, a JWT typically looks like the following:
  - xxxxx.yyyyy.zzzzz
  - eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c

JWT Header

- The header *typically* consists of two parts: the type of the token, which is JWT, and the hashing algorithm being used, such as HMAC SHA256 or RSA.
- For example:

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

- Then, this JSON is **Base64Url** encoded to form the first part of the JWT.

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c

HMAC SHA256 vs RSA SHA256 hashing algorithms

- HMAC SHA256: Symmetric Key cryptography, single shared private key. Faster, good between trusted parties.
  - A combination of a hashing function and one (secret) key that is shared between the two parties used to generate the hash that will serve as the signature.

- RSA SHA256: Asymmetric Key cryptography, public/private keys. Slower, good between untrusted parties.
  - The identity provider has a private (secret) key used to generate the signature, and the consumer of the JWT gets a public key to validate the signature.

JWT Payload

- The second part of the token is the payload, which contains the claims.
- **Claims** are statements about an entity (typically, the user) and additional metadata. There are three types of claims:
  - *Reserved/Registered*
    - These are a set of predefined claims which are not mandatory but recommended, to provide a set of useful, interoperable claims. Some of them are: **`iss`** (issuer), **`exp`** (expiration time), **`sub`** (subject), **`aud`** (audience), and **`others`**.
  - *Public*
    - These can be defined at will by those using JWTs. But to avoid collisions they should be defined in the IANA JSON Web Token Registry or be defined as a URI that contains a collision resistant namespace.
  - *Private*
    - These are the custom claims created to share information between parties that agree on using them and are neither registered or public claims.

JWT Payload

- For example:

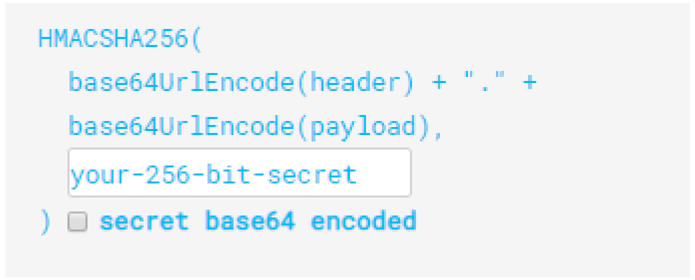
```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

- The payload is then **Base64Url** encoded to form the second part of the JSON Web Token.

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c

JWT Signature

- To create the signature part you have to take the encoded header, the encoded payload, a secret, the algorithm specified in the header, and sign that.
- The signature is used to verify that the sender of the JWT is who it says it is and to ensure that the message wasn't changed along the way.
- For example if you want to use the HMAC SHA256 algorithm, the signature will be created in the following way:

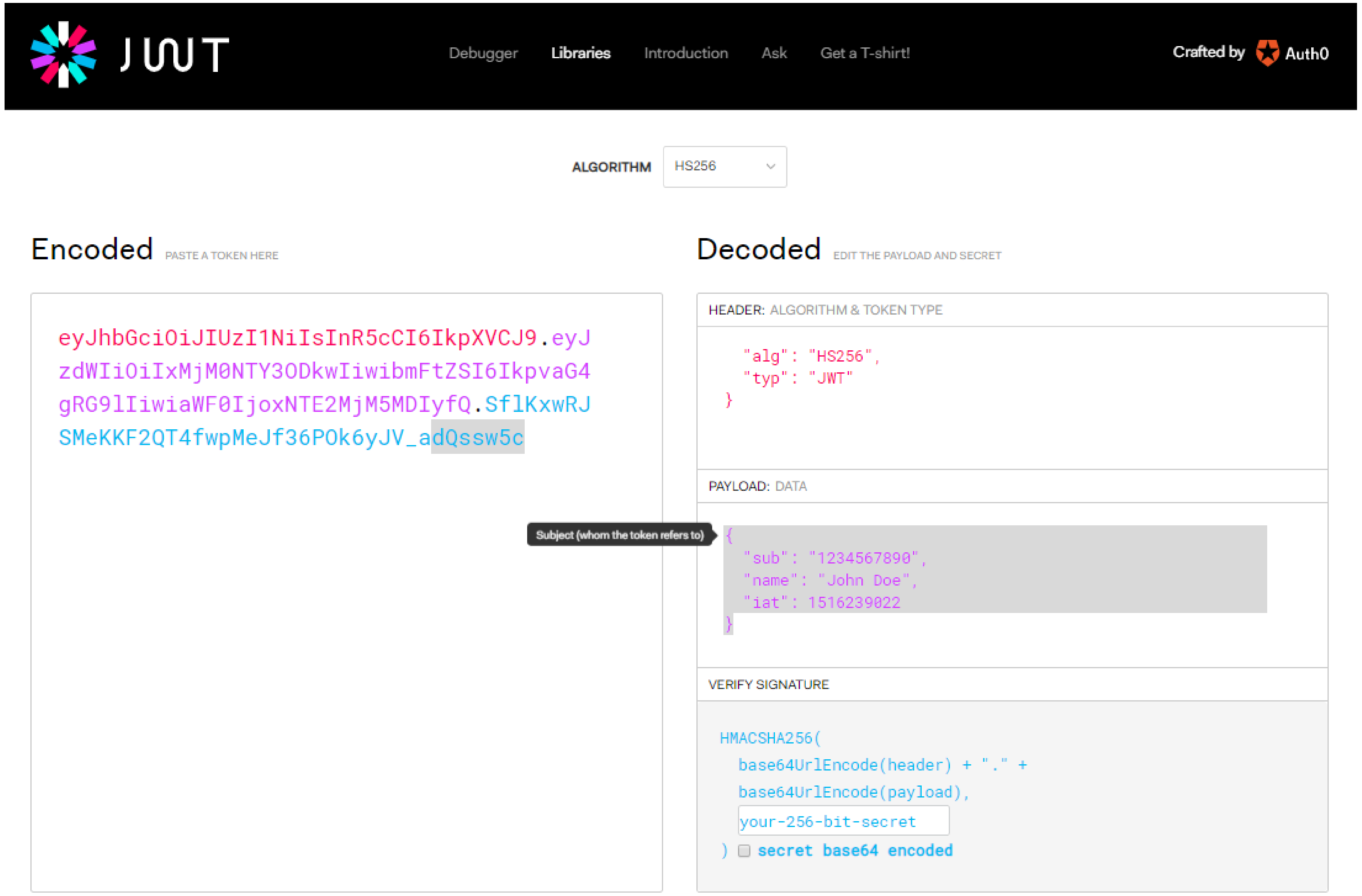


```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

- eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c

jwt.io

- JWT.IO allows you to decode, verify and generate JWT.



ALGORITHM HS256

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ
zdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJ
SMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

Subject (whom the token refers to)

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

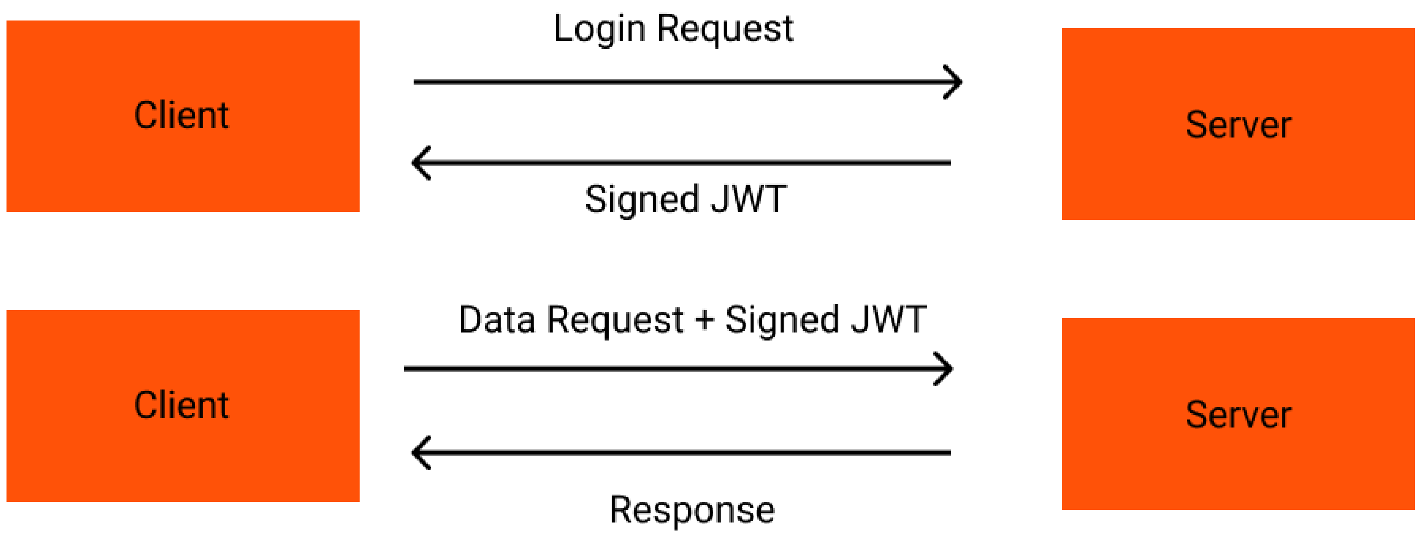
VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

How does JWT work?

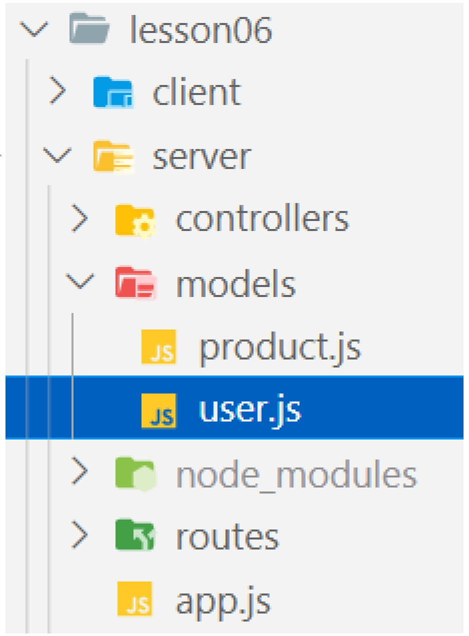
- In authentication, when the user successfully logs in using their credentials, a JSON Web Token will be returned and must be saved locally (typically in local storage, but cookies can be also used).
- Whenever the user wants to access a protected route or resource, the user agent should send the JWT, typically in the **Authorization** header using the **Bearer** schema. The content of the header should look like the following:

```
Authorization: Bearer <token>
```



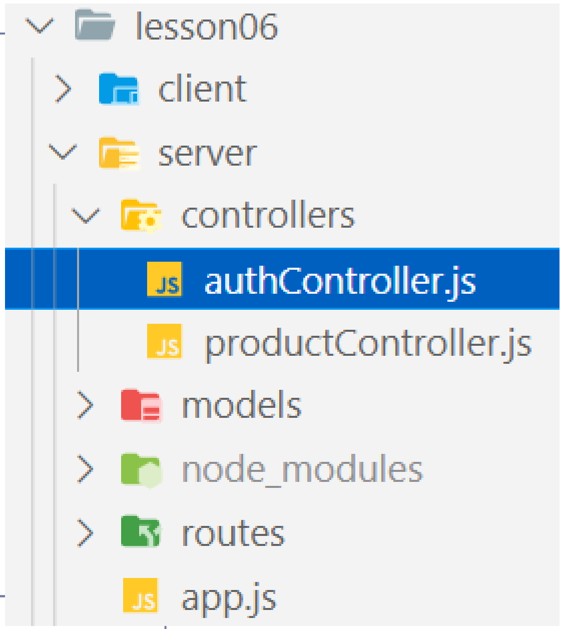
Auth Demo - Model

```
const users = [];

module.exports = class User {
    constructor(username, password, role) {
        this.username = username;
        this.password = password;
        this.role = role;
    }

    login() {
        return users.find(u => { return u.username === this.username && u.password === this.passw
ord });
    }

    static init() {
        users.push(new User('john', 'admin', 'admin'));
        users.push(new User('bella', 'member', 'member'));
    }
}
```



Auth Demo - Controller

```
const jwt = require('jsonwebtoken');

const User = require('../models/user');
const accessTokenSecret = "tina's shopping";

exports.login = (req, res, next) => {
    const user = new User(req.body.username, req.body.password, null).login();
    if (user) {
        const accessToken = jwt.sign({ username: user.username, role: user.role }, accessTokenSecret);
        res.json({ accessToken });
    } else {
        res.status(200).json({ 'error': 'username or password invalid' });
    }
}
```

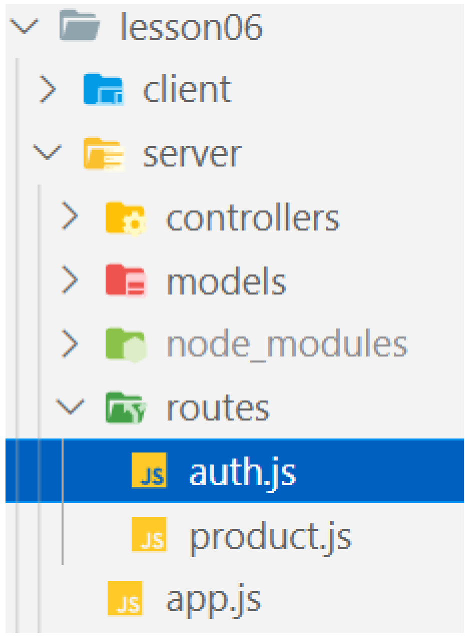


```
lesson06
  client
  server
    controllers
      authController.js
      productController.js
    models
    node_modules
    routes
    app.js
```

```
exports.authorize = (req, res, next) => {
    const authHeader = req.headers.authorization;
    if (authHeader) {
        const token = authHeader.split(' ')[1];
        jwt.verify(token, accessTokenSecret, (err, user) => {
            console.log(user);
            if (err) {
                return res.status(403).json({ "error": "Forbidden" });
            }
            req.user = user;
            next();
        });
    } else {
        res.status(401).json({ "error": "Unauthorized" });
    }
}
```

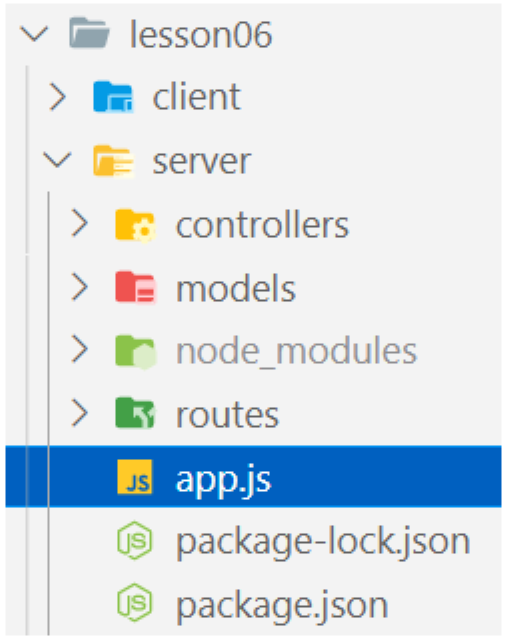
Auth Demo - Route

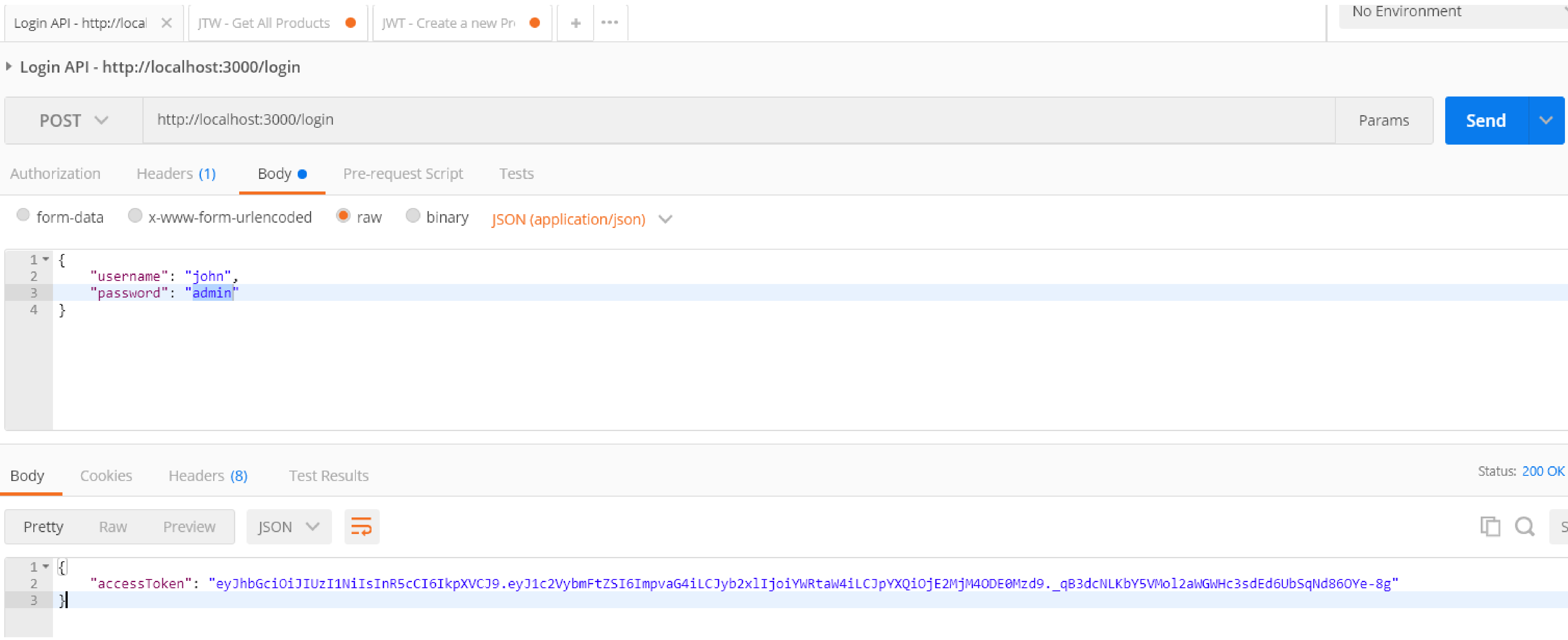
```
const express = require('express');
const authController = require('../controllers/authController');
const router = express.Router();

router.post('/login', authController.login);
router.use(authController.authorize);

module.exports = router;
```



Auth Demo – app.js

```
const authRouter = require('./routes/auth');
app.use(authRouter); //all urls access after authRouter needs JWT
app.use('/products', productRouter);
```




Login API - http://local | JTW - Get All Products | JWT - Create a new Pr

No Environment

Login API - http://localhost:3000/login

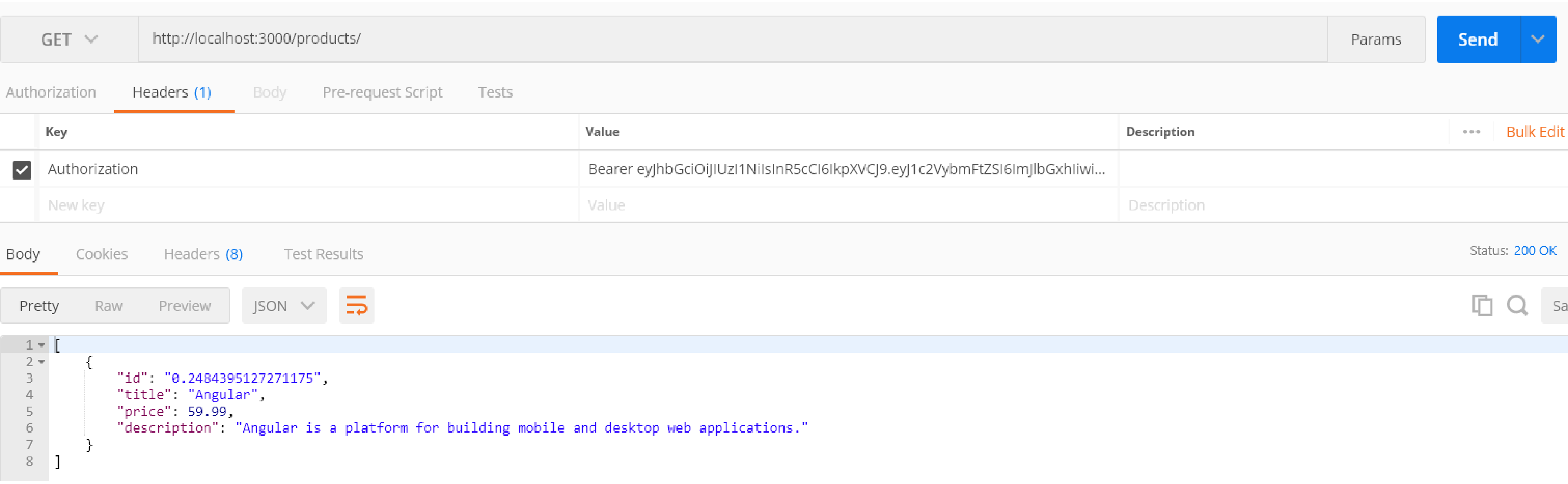
POST http://localhost:3000/login

Body: raw, JSON (application/json)

```
{
    "username": "john",
    "password": "admin"
}
```

Status: 200 OK

```
{
    "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImpvaG4iLCJyb2xlIjoiYWRtaW4iLCJpYXQiOjE2MjM4ODE0Mzd9._qB3dcNLKbY5VMol2aWGWHc3sdEd6UbSqNd86OYe-8g"
}
```



GET http://localhost:3000/products/

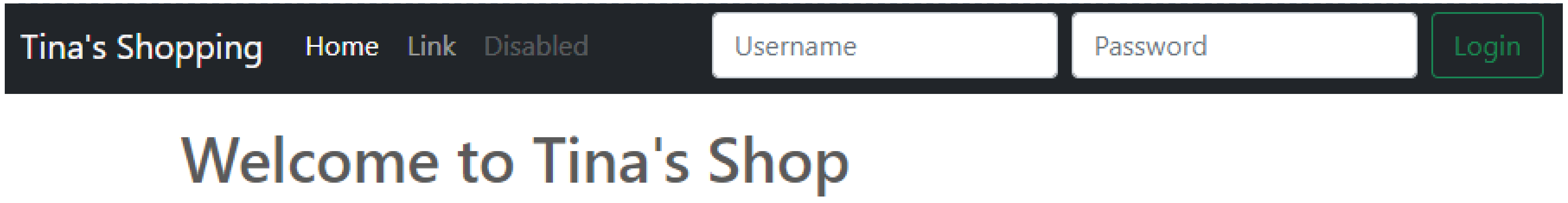
Headers (1)

	Key	Value	Description
✓	Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImJlbGxhIiwi...	

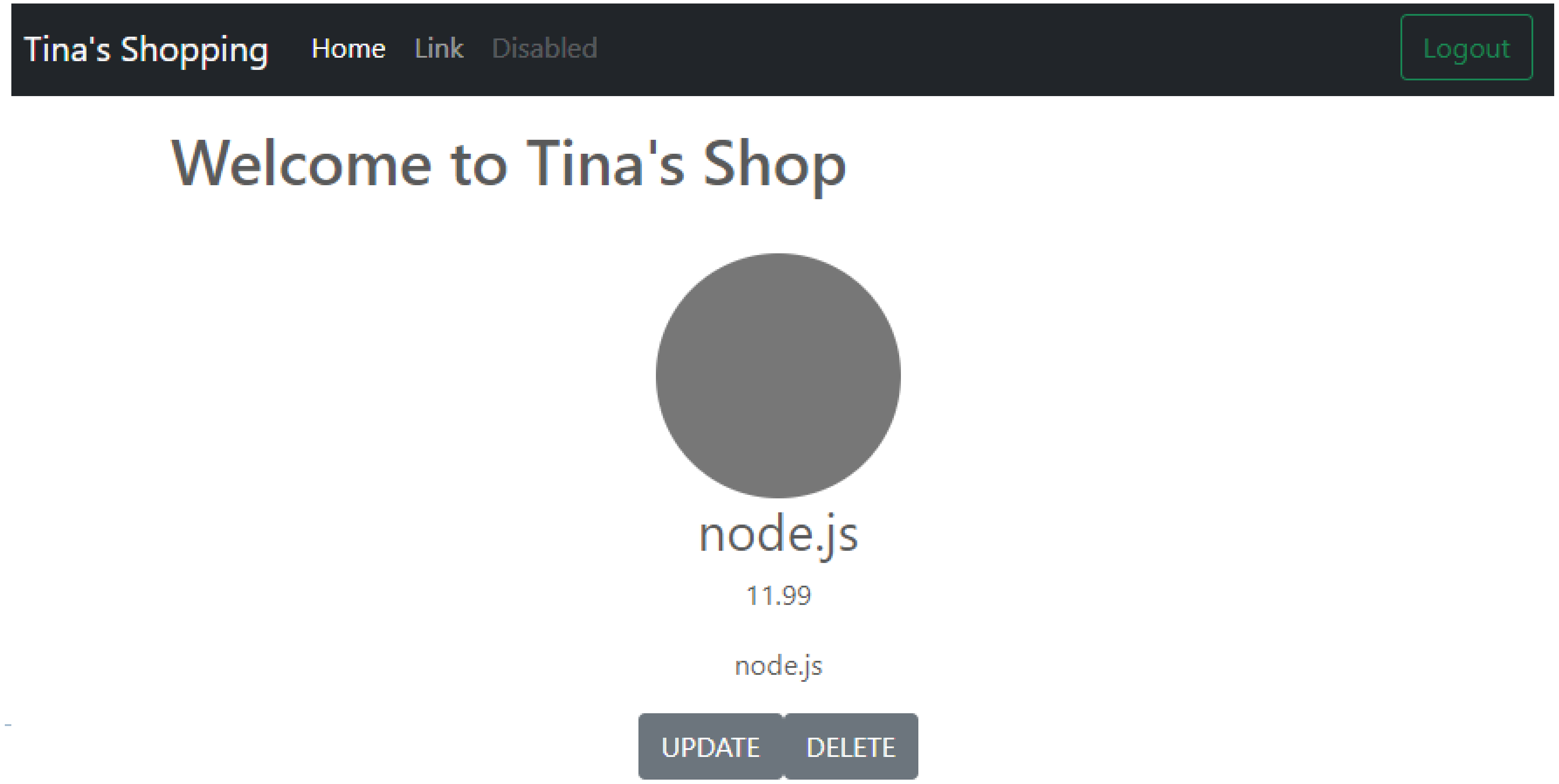
Status: 200 OK

```json
[
    {
        "id": "0.2484395127271175",
        "title": "Angular",
        "price": 59.99,
        "description": "Angular is a platform for building mobile and desktop web applications."
    }
]
```

Auth Demo - Client


Tina's Shopping Home Link Disabled Username Password Login

Welcome to Tina's Shop


Tina's Shopping Home Link Disabled Logout

Welcome to Tina's Shop

node.js

11.99

node.js

UPDATE DELETE

Resources

- https://stackabuse.com/authentication-and-authorization-with-jwts-in-express-js
- https://jwt.io/
- https://www.npmjs.com/package/jsonwebtoken