Backend 4.3_CW Exercises

understanding jwt

documentation

https://github.com/auth0/node-jsonwebtoken

important site for troubleshooting

https://jwt.io/

ex02: create a simple token

challenge

Create a simple JWT token without an expiration time.

understanding

- `jsonwebtoken` library is used to create a JWT token.
- The `jwt.sign` method takes two main arguments:
  - the payload (in this case, `{ foo: 'bar' }`)
  - the secret key used for signing.
- This produces a signed token that contains the payload data. The resulting token can be shared with clients or stored for later use.
- Keep in mind that this token *does not have an expiration time*, so it will remain valid until it's manually invalidated or revoked.

solution

```
const jwt = require('jsonwebtoken')
const secret = 'your-secret-goes-here' // Replace with your secret

const token = jwt.sign({ foo: 'bar' }, secret)
```

```
console.log({ token })
```

COPY

ex03: creating a token with expiry

challenge

Create a JWT token with a specified expiration time (e.g., 1 hour).

understanding

- Extend the previous example by adding an expiration time to the JWT token.
- The `{ expiresIn: '1h' }` option in the `jwt.sign` method specifies that the token should expire in one hour.
- This is useful for implementing token-based authentication with short-lived tokens.
- After the specified time period (1 hour in this case), the token will no longer be valid, and the user will need to re-authenticate.

solution

```
const jwt = require('jsonwebtoken')
const secret = 'your-secret-goes-here' // Replace with your secret

const token = jwt.sign({ foo: 'bar' }, secret, { expiresIn: '1h' })
console.log({ token })
```

COPY

ex04: verify the token

challenge

Verify a JWT token using your secret and print the decoded payload.

understanding

- In this exercise, the authenticity of a JWT token is verified using the `jwt.verify` method.
- If the token is valid and hasn't been tampered with, the method returns the decoded payload contained within the token.

solution

```
const decoded = jwt.verify(token, secret) // try with a different the secret as well
console.log({ decoded })
```

COPY

ex05: verify a token with error handling

challenge

Verify a JWT token with error handling, specifically handling the case where the token is incorrect or has expired.

understanding

- If the token is valid, the `try` block's code is executed, and the decoded payload is printed to the console.
- If the token is invalid (e.g., incorrect or expired), an error is thrown, and it's caught by the `catch` block, which prints an error message.

solution:

```
try {
  const decoded = jwt.verify(token, secret)
  console.log({ decoded })
} catch (error) {
  console.error('Token verification failed', error)
}
```

COPY

entire solution

https://replit.com/@tanaypratap/jwt-example#index.js

ex06: create a login route

challenge

- Create a POST route `/login` that accepts a username and password in the request body.
- Assume that the username and password are correct for this exercise.
- Generate a JWT token with a user ID and return it in the response along with the username.

solution

```
app.post('/login', (req, res) => {
  const { username, password } = req.body.user
  const token = jwt.sign({ userID: '1234' }, secret, { expiresIn: '24h' })
  res.json({ username, token })
})
```

COPY

ex07: user authentication

ex07.1 i**mplement JWT verification**

challenge

- Create a function called `verifyToken` that takes a JWT token and verifies it using the `jwt.verify` method.
- If the token is valid, the function should return the decoded payload (user ID).
- If the token is invalid, it should throw an error.

solution

```
const jwt = require('jsonwebtoken')
const secret = 'ksdnfaisodjfaiofj' // Replace with your secret

function verifyToken(token) {
  try {
    const decoded = jwt.verify(token, secret)
    return decoded
  } catch (error) {
    throw new Error('Invalid token')
  }
}
```

COPY

ex07.2: extracting user ID from decoded token

challenge

Create a function called `extractUserIDFromToken` that takes a decoded JWT token and returns the user ID contained in the payload.

solution

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

```
function extractUserIDFromToken(decodedToken) {
  if (decodedToken && decodedToken.userID) {
    return decodedToken.userID
  } else {
    throw new Error('Invalid or missing user ID in token')
  }
}
```

COPY

ex07.3: update `authVerify` middleware

challenge

- `authVerify` middleware will now use the `verifyToken` & the `extractUserIDFromToken` function to verify the JWT token present in the request's `Authorization` header.
- If the token is valid, set the `req.user` object with the user ID extracted from the decoded token and allow the request to proceed.
- If the token is invalid or missing, return a 401 status code with a JSON response containing the message "Unauthorised access, please add the token."

solution

```
const secret = 'ksdnfaisodjfaiofj'
const jwt = require('jsonwebtoken')

function authVerify(req, res, next) {
  const token = req.headers.authorization

  try {
    const decoded = verifyToken(token)
    const userID = extractUserIDFromToken(decoded)
    req.user = { userID }
    return next()
  } catch (error) {
    return res
      .status(401)
      .json({ message: 'Unauthorised access, please add the token' })
  }
}
```

COPY

ex10: implement user profile

challenge

- Create a GET route `/user/:userID` that requires authentication using the `authVerify` middleware.
- Retrieve the user's ID from the decoded token (`req.user.userID`) and compare it to the `userID` provided in the route parameters (`req.params.userID`).

- If they match, return a JSON response with user information.
- If they do not match, return a 403 status code with a JSON response containing the message "trying to access unauthorized."

solution

```
app.get('/user/:userID', authVerify, (req, res) => {
  const { userID } = req.user
  const { userID: requestedUser } = req.params

  if (userID !== requestedUser) {
    res.status(403).json({ message: 'Trying to access unauthorized' })
  }
  // Replace with actual logic to retrieve user information
  // UserAddress.findById({ userID }); // retrieving user address demo
  res.json({ name: 'Tanay', age: 31, pincode: '560102' })
})
```

COPY

entire solution

https://replit.com/@tanaypratap/BE43CW-auth-3

ex11: implement auth router

challenge

Would you put this middleware in all routes one by one?

How would you implement this for an entire router?

solution

```
/routers
        /public
                        products.router.js

        /protected
                        user.router.js
                        cart.router.js

// when you're using it

app.use("/user", authVerify, userRouter)
```

COPY

ex01: generate a secret

challenge

Generate a secret for signing and verifying JWTs.

understanding

Node.js contains built-in `crypto` module to generate a random secret key for signing and verifying JWTs.

The `crypto` module provides a secure way to generate random data, which is crucial for maintaining the security of your JWTs.

The generated secret is stored securely, such as in environment variables, to keep it secret.

solution

```
# In the terminal, generate a secret and store it in your .env file
node -e "console.log(require('crypto').randomBytes(256).toString('base64'));"
```