SQL ORM AND JWT AUTHENTICATION

FULL STACK SKILLS BOOTCAMP

SQL ORM AND JWT AUTHENTICATION

- **Lesson Overview:**
- In this lesson, we will be introduced to:

1. What is JWT
2. Creating and signing tokens
3. Middleware
4. Using tokens client-side
5. Session Authentication

WHAT IS A JWT?

- JWT stands for JSON Web Token.
- A compact, URL-safe way to represent claims between two parties.
- Typically used for authentication and authorization.
- Consists of three parts: Header, Payload, and Signature.

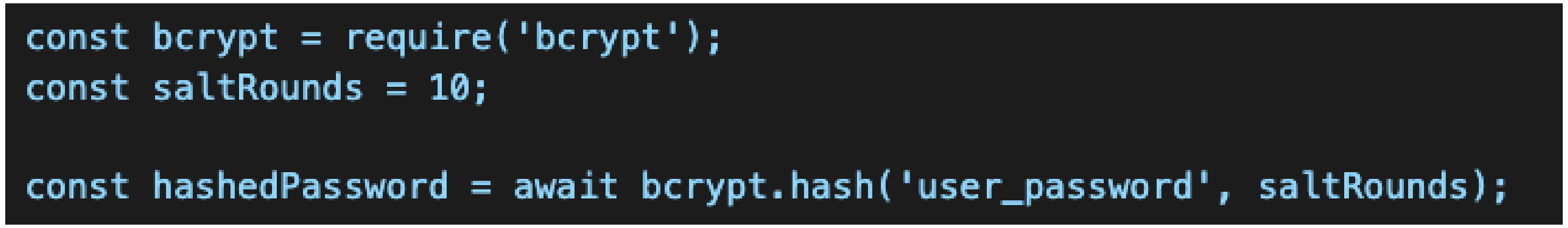
REAL-WORLD USE CASES FOR JWT

- Secure API authentication.
- User session management.
- Single Sign-On (SSO) implementations.
- Data exchange between services without requiring a database lookup

HANDLING PASSWORD ENCRYPTION IN SEQUELIZE WITH BCRYPT

- **Why Hash Passwords?**
- Protects user credentials.
- Prevents plaintext password storage.

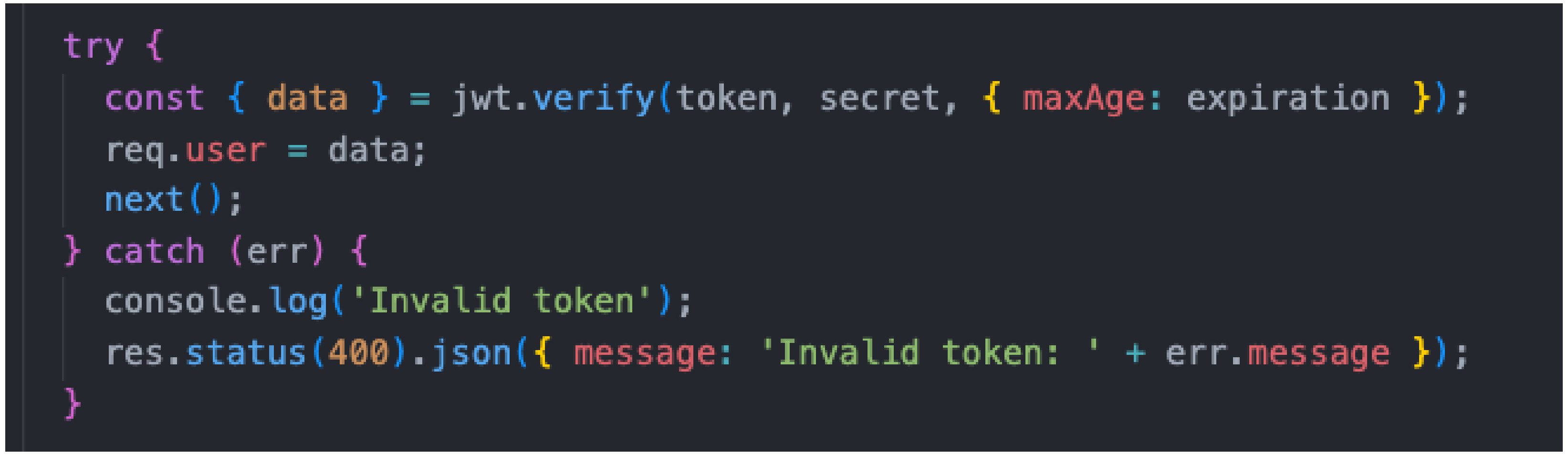
Using bcrypt to Hash Passwords:



```
const bcrypt = require('bcrypt');
const saltRounds = 10;

const hashedPassword = await bcrypt.hash('user_password', saltRounds);
```

HANDLING PASSWORD ENCRYPTION IN SEQUELIZE WITH BCRYPT

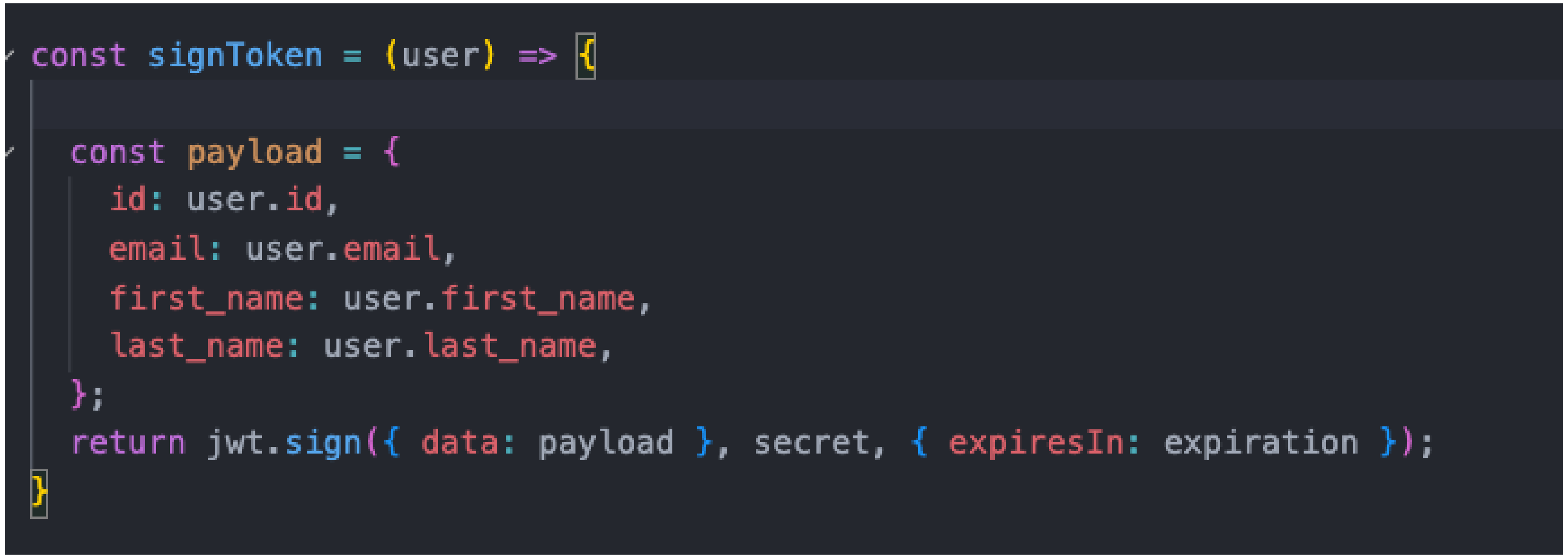
- **Verifying Passwords:**



```
try {
  const { data } = jwt.verify(token, secret, { maxAge: expiration });
  req.user = data;
  next();
} catch (err) {
  console.log('Invalid token');
  res.status(400).json({ message: 'Invalid token: ' + err.message });
}
```

SIGNING A JWT

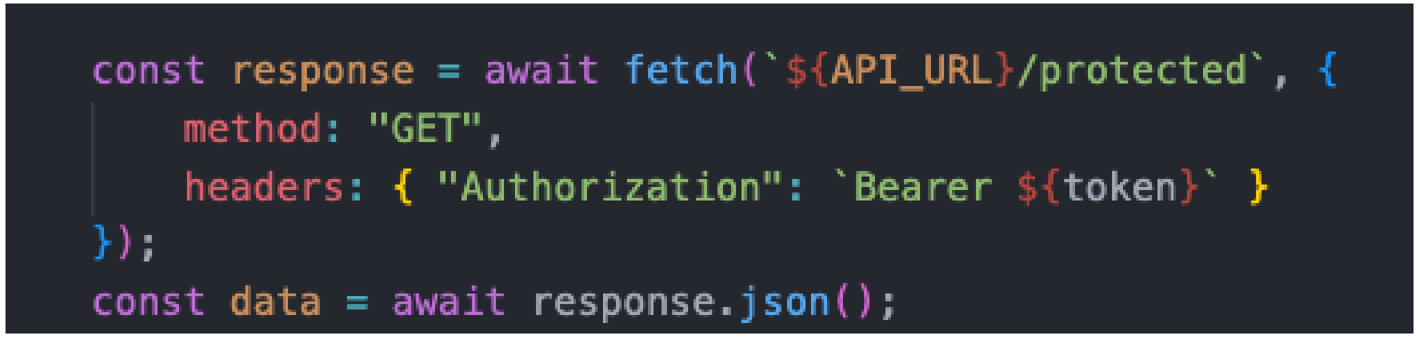
- **Generating a Token:**
- Tokens are created using a signing process



```
const signToken = (user) => {

  const payload = {
    id: user.id,
    email: user.email,
    first_name: user.first_name,
    last_name: user.last_name,
  };
  return jwt.sign({ data: payload }, secret, { expiresIn: expiration });
}
```

USING A JWT CLIENT-SIDE WITH JAVASCRIPT

- Store the JWT in localStorage or sessionStorage.
- Include the JWT in API requests via headers



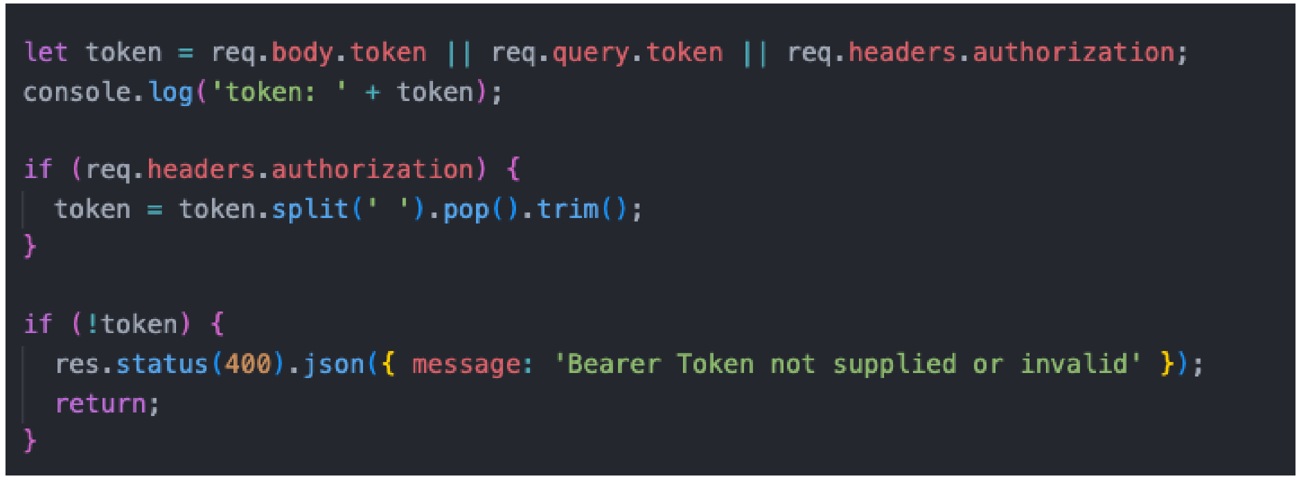
```
const response = await fetch(`${API_URL}/protected`, {
    method: "GET",
    headers: { "Authorization": `Bearer ${token}` }
});
const data = await response.json();
```

SESSION AUTHENTICATION

- JWT vs. Session-based authentication.
- Sessions require server-side storage.
- JWT authentication is stateless and scalable.

AUTH MIDDLEWARE

- Middleware to protect routes using JWT authentication.
- It intercepts a request and checks for a valid token before passing control to the route

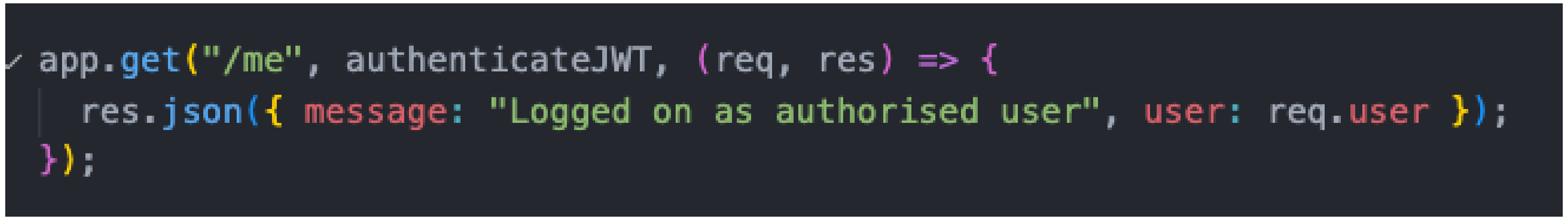


```
let token = req.body.token || req.query.token || req.headers.authorization;
console.log('token: ' + token);

if (req.headers.authorization) {
  token = token.split(' ').pop().trim();
}

if (!token) {
  res.status(400).json({ message: 'Bearer Token not supplied or invalid' });
  return;
}
```

PROTECTING ROUTES USING MIDDLEWARE

- **Protecting routes**

  The auth middleware function can be added as a parameter to the route
- The ensures the middleware function runs first
- The middleware function checks the JWT and allows the route to run or not



```
app.get("/me", authenticateJWT, (req, res) => {
  res.json({ message: "Logged on as authorised user", user: req.user });
});
```

REVIEW: THE AUTH LIFECYCLE

1. User signs up with a password (hashed using bcrypt).
2. User logs in and receives a JWT.
3. The client stores and uses the JWT for authentication.
4. Protected routes validate JWT via middleware.
5. JWT expires or user logs out, requiring re-authentication.

USEFUL RESOURCES

- JWT Official Docs
- Sequelize Documentation
- bcrypt Documentation
- Node.js Documentation

CONCLUSION & QUESTIONS

- JWT enables secure, scalable authentication.
- Sequelize and bcrypt enhance user security.
- Implementing authentication properly is crucial.

QUESTIONS?