4. Authentication and Security

Building a secure authentication system involves multiple aspects, including user authentication using JSON Web Tokens (JWT) and encrypting sensitive user data stored in the database. Below is a basic example using Node.js, Express.js, JWT for authentication, and the bcrypt library for password hashing.

1. Install required packages:

```bash
Npm install express jsonwebtoken bcrypt body-parser
```

2. Create an `app.js` file:

```javascript
Const express = require('express');
Const jwt = require('jsonwebtoken');
Const bcrypt = require('bcrypt');
Const bodyParser = require('body-parser');

Const app = express();
Const port = 3000;

// Replace this with your database connection logic
Const users = [];

App.use(bodyParser.json());

// Secret key for JWT (keep this secret and secure)
Const secretKey = 'your-secret-key';
```

```
// Middleware to verify JWT token
Function verifyToken(req, res, next) {
  Const token = req.headers['authorization'];

  If (!token) {
    Return res.status(403).json({ message: 'Unauthorized' });
  }

  Jwt.verify(token, secretKey, (err, decoded) => {
    If (err) {
      Return res.status(403).json({ message: 'Invalid token' });
    }

    Req.user = decoded.user;
    Next();
  });
}

// Route to register a new user
App.post('/register', async (req, res) => {
  Const { username, password } = req.body;

  // Hash the password before storing it in the database
  Const hashedPassword = await bcrypt.hash(password, 10);

  Users.push({ username, password: hashedPassword });

  Res.json({ message: 'User registered successfully' });
});
```

```
// Route to authenticate and get a JWT token
App.post('/login', async (req, res) => {
  Const { username, password } = req.body;

  Const user = users.find((u) => u.username === username);

  If (!user || !(await bcrypt.compare(password, user.password))) {
    Return res.status(401).json({ message: 'Invalid credentials' });
  }

  Const token = jwt.sign({ user: { username } }, secretKey, { expiresIn: '1h' });

  Res.json({ token });
});

// Protected route – requires a valid JWT token
App.get('/protected', verifyToken, (req, res) => {
  Res.json({ message: 'Protected route accessed successfully', user: req.user });
});

App.listen(port, () => {
  Console.log(`Server is running on http://localhost:${port}`);
});
```

In this example:

- The `/register` route allows users to register by hashing their password using `bcrypt` before storing it in memory (replace with your database logic).

- The `/login` route authenticates users by comparing the hashed password with the stored hash.

- The `/protected` route is protected by the `verifyToken` middleware, ensuring that only requests with a valid JWT token can access it.

Remember, this is a simplified example, and in a production environment, you should use a database for user storage, HTTPS for secure communication, and possibly additional security measures based on your application's requirements. Always keep your secret keys secure, and consider using environment variables for configuration.