Secure Coding Practices Implementation

This document demonstrates the implementation of secure coding practices in a Node.js/Express application, including input validation, output encoding, secure error handling, and authentication security.

1. Input Validation

The application implements robust input validation for both usernames and passwords:

Username Validation
```javascript
function isValidUsername(username) {
    const regex = /^[a-zA-Z0-9_]{3,20}$/;
    return regex.test(username);
}
```
- **Security Features**:
  - Only allows alphanumeric characters and underscores
  - Enforces length between 3-20 characters
  - Prevents SQL injection by restricting special characters

Password Validation
```javascript
function isValidPassword(password) {
    const regex = /^(?=.*[A-Za-z])(?=.*\d)[A-Za-z\d@$!%*?&]{8,}$/;
    return regex.test(password);
}
```
- **Security Features**:
  - Minimum 8 characters
  - Requires at least one letter and one number
  - Allows limited special characters (@$!%*?&)
  - Prevents weak passwords

Implementation in Registration Route:
```javascript
app.post('/register', (req, res) => {
    const { username, password } = req.body;

    if (!isValidUsername(username)) {
        return res.status(400).send('Invalid username format');
    }
    if (!isValidPassword(password)) {
```

```
        return res.status(400).send('Password requirements not met');
    }
    // ... rest of registration logic
});
```

2. Secure Authentication

The application implements secure password handling using bcrypt:

Password Hashing
```javascript
// During registration
bcrypt.hash(password, 10, (err, hash) => {
    if (err) throw err;
    users.push({ username, password: hash });
    res.send('User registered successfully!');
});
```

Secure Password Comparison
```javascript
app.post('/login', (req, res) => {
    const { username, password } = req.body;
    const user = users.find(u => u.username === username);

    if (!user) {
        return res.status(401).send('Invalid credentials');
    }

    bcrypt.compare(password, user.password, (err, isMatch) => {
        if (err) {
            console.error(err);
            return res.status(500).send('Error during authentication');
        }
        if (isMatch) {
            res.send('Login successful');
        } else {
            res.status(401).send('Invalid credentials');
        }
    });
});
```
- **Security Features**:

  - Uses bcrypt for secure password hashing (with salt rounds = 10)
  - Timing-safe comparison
  - Proper error handling
  - Generic error messages to prevent information leakage

3. Secure Error Handling

The application implements proper error handling middleware:

```javascript
// Error handling middleware (for uncaught errors)
app.use((err, req, res, next) => {
    console.error(err.stack);
    res.status(500).send('Something went wrong. Please try again later.');
});

// 404 handling for unrecognized routes
app.use((req, res, next) => {
    res.status(404).send('Page not found');
});
```
- **Security Features**:
  - Prevents stack traces from being exposed to users
  - Generic error messages
  - Proper logging of errors for debugging
  - 404 handling to prevent information disclosure about application structure

4. Verification of Implementation

The screenshots show successful testing of the security features:

![Testing Screenshot](Screenshot 2025-04-07 at 5.52.00 PM.png)
- Successful registration with valid credentials
- Successful login with correct credentials
- Proper error handling demonstrated
- No vulnerabilities found in dependencies (`npm audit` clean)

Security Best Practices Demonstrated

1. **Input Validation**: Strict regex patterns for usernames and passwords
2. **Password Security**: Hashing with bcrypt, proper salt rounds
3. **Error Handling**: No sensitive information leakage
4. **Secure Dependencies**: Regular auditing of packages
5. **Secure Configuration**: Environment variables for port configuration

6. **Generic Responses**: Prevents enumeration attacks in authentication

This implementation meets all the requirements for secure coding practices, demonstrating proper input validation, secure authentication, and safe error handling.

Search

Welcome | .gitignore | index.js | server.js

Users > sa55 > Documents > asdf > server.js > ...

```javascript
// Import required libraries
const express = require('express');
const bcrypt = require('bcryptjs');
const bodyParser = require('body-parser');

// Create an Express app
const app = express();

// Middleware for parsing incoming JSON requests
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));

let users = []; // In-memory user storage (for demonstration purposes)

// Username validation function (only alphanumeric and underscores)
function isValidUsername(username) {
  const regex = /^[a-zA-Z0-9_]{3,20}$/;
  return regex.test(username);
}

// Password validation function (at least 8 characters, with a letter and a numb
function isValidPassword(password) {
  const regex = /^(?=.*[A-Za-z])(?=.*\d)[A-Za-z\d@$!%*?&]{8,}$/;
  return regex.test(password);
}

// Registration route
app.post('/register', (req, res) => {
  const { username, password } = req.body;

  // Validate the username and password
  if (!isValidUsername(username)) {
    return res.status(400).send('Invalid username. It should be 3-20 characters
  }

  if (!isValidPassword(password)) {
    return res.status(400).send('Password must be at least 8 characters, with a
  }
```

0 0 | GitGuardian - Please authenticate | Ln 77, Col 16 | Spaces: 4 | UTF-8 | LF | JavaScript

Search

Welcome | .gitignore | index.js | server.js

Users > sa55 > Documents > asdf > server.js > ...

```javascript
app.post('/login', (req, res) => {
  const { username, password } = req.body;

  const user = users.find(u => u.username === username);
  if (!user) {
    return res.status(401).send('Invalid credentials');
  }

  // Compare entered password with the stored hashed password
  bcrypt.compare(password, user.password, (err, isMatch) => {
    if (err) {
      console.error(err);
      return res.status(500).send('Error during authentication');
    }

    if (isMatch) {
      res.send('Login successful');
    } else {
      res.status(401).send('Invalid credentials');
    }
  });
});

// Error handling middleware (for uncaught errors)
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).send('Something went wrong. Please try again later.');
});

// 404 handling for unrecognized routes
app.use((req, res, next) => {
  res.status(404).send('Page not found');
});

// Start the server
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

0 0 | GitGuardian - Please authenticate | Ln 77, Col 16 | Spaces: 4 | UTF-8 | LF | {} JavaScript

asdf — node server.js — 80×24

```
Last login: Mon Apr  7 16:04:13 on ttys000
sa55@SA55 ~ % cd Documents
sa55@SA55 Documents % cd asdf
sa55@SA55 asdf % npm install express bcryptjs body-parser


added 1 package, and audited 69 packages in 1s

14 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
sa55@SA55 asdf % touch server.js
sa55@SA55 asdf % node server.js
Server running on port 3000
sa55@SA55 asdf % node server.js
Server running on port 3000
```

asdf — -zsh — 80×24

```
found 0 vulnerabilities
sa55@SA55 asdf % touch index.js
sa55@SA55 asdf % code index.js
zsh: command not found: code
sa55@SA55 asdf % node -v
v22.14.0
sa55@SA55 asdf % which node

/usr/local/bin/node
sa55@SA55 asdf % node index.js

Server is running on http://localhost:3000
\^C
sa55@SA55 asdf % curl -X POST http://localhost:3000/register \
-H "Content-Type: application/json" \
-d '{"username": "john_doe", "password": "Password123"}'

User registered successfully!%
sa55@SA55 asdf % curl -X POST http://localhost:3000/login \
-H "Content-Type: application/json" \
-d '{"username": "john_doe", "password": "Password123"}'

Login successful%
sa55@SA55 asdf %
```