

Deliverable 2 Documentation: Student Portal Security Features

This document outlines the implementation of three key security features in the Student Portal project:

- HTTPS with Custom SSL Certificates
 - Two-Factor Authentication (2FA) via Email
 - Password Reset via Token
 - Session Management using express-session
-

1. HTTPS with Custom SSL Certificates

I enabled secure HTTPS communication between frontend and backend using locally generated SSL certificates:

- Created `server.key` and `server.cert` files in the `/backend/ssl/` directory.
- Used `https.createServer(sslOptions, app).listen(...)` to start the server.
- Configured the frontend to run on `https://localhost:3000` and the backend on `https://localhost:3443`.
- Updated `.env` and CORS setup to match HTTPS origins and allow cookies:

```
REACT_APP_API_URL=https://localhost:3000
```

```
app.use(cors({  
  origin: process.env.REACT_APP_API_URL,  
  credentials: true  
}));
```

✓ 2. Two-Factor Authentication (2FA) via Email

Purpose

To add an extra layer of security during user login by requiring a 6-digit code sent to the user's email after password verification.

Flow

1. **User logs in** with email and password.
2. The server verifies the email and password.
3. If valid, the server:
 - Generates a **6-digit numeric code** (e.g. `123456`).
 - Sets an **expiration time** for the code (10 minutes from generation).
 - Stores the code and expiry in the database (`two_factor_code`, `two_factor_expires` columns in `users` table).
4. Sends the code via **nodemailer** to the user's email.
5. Frontend navigates to `/two-factor` route and waits for user input.
6. User submits the 6-digit code.
7. Server checks:
 - If the code matches the stored `two_factor_code`.
 - If the current time is before `two_factor_expires`.
8. If both checks pass:
 - Server clears the 2FA fields.
 - Sets up a session for the user (`req.session.userId = user.id`).
 - User is redirected to the dashboard.

Tech Stack

- Backend: Node.js + Express
 - Email Service: Nodemailer (Gmail)
 - Session Storage: express-session
 - Frontend: React
-

3. Password Reset with Token + Expiration

Purpose

Allow users to reset their password via email, securely using tokens.

Flow

1. User clicks on "Forgot Password?".
2. Enters their registered email.
3. Server:
 - Verifies email exists.
 - Generates a **UUID token** using `uuidv4()`.
 - Sets **expiration time** (1 hour).
 - Stores token and expiry in a new table `password_reset_tokens` with `user_id`, `reset_token`, and `reset_token_expires`.

Sends a password reset link with the token to user's email:

`https://localhost:3000/reset-password?token=abc123...`

- 4.
5. User clicks the link and is redirected to the frontend reset form.

6. Submits new password + token.
7. Server:
 - Verifies the token exists and is not expired.
 - If valid:
 - Hashes new password with `bcrypt`.
 - Updates password in `users` table.
 - Deletes the used token from `password_reset_tokens`.

Tech Stack

- `uuid`: For secure unique tokens
 - `bcryptjs`: For hashing the new password
 - `nodemailer`: For sending reset links
-



4. Session-Based Authentication (express-session)

Purpose

To maintain secure, server-side sessions after successful login and 2FA verification.

Flow

After successful 2FA, server creates a session:

```
req.session.userId = user.id;
```

- 1.
2. Session data is stored in memory (or can be moved to Redis).
3. The session ID is sent to the browser as a **cookie** named `connect.sid`.

4. On every request to protected routes (e.g. `/api/dashboard-data`), the server checks if `req.session.userId` exists.
5. If not, user is unauthorized.

On logout, session is destroyed:

```
req.session.destroy();  
res.clearCookie("connect.sid");
```

6.

Configuration

```
app.use(session({  
  secret: process.env.SESSION_SECRET,  
  resave: false,  
  saveUninitialized: false,  
  cookie: {  
    secure: true,    // HTTPS only  
    httpOnly: true, // Inaccessible from JS  
    maxAge: 1000 * 60 * 60 // 1 hour  
  }  
}));
```

Notes

- Sessions are required for authentication checks in `/api/dashboard-data`.
 - Used with `credentials: 'include'` in frontend fetch calls.
-

Summary

This implementation significantly improves the security of the Student Portal:

- HTTPS with Custom SSL Certificates.
- Protects against unauthorized logins with 2FA.
- Provides secure password recovery.
- Maintains authenticated sessions across secure routes.

Future enhancements could include:

- Session storage in Redis
- Rate limiting login attempts
- Email verification for new users