Penetration Testing Report – Mobile App (Flutter + Node.js)

1. Executive Summary

This report documents the results of a security assessment conducted on the mobile application "Salkah," developed using Flutter for the frontend and Node.js for the backend, with a MySQL database hosted on Railway.

Summary of Key Findings:

- Sensitive APIs lacked authentication and authorization mechanisms.
- Communication was not encrypted, risking data exposure.
- No enforcement of password strength or CAPTCHA, increasing brute-force risk.
- JWT session management was not implemented.
- Email verification was not required during signup.

Overall Risk Level: HIGH

Immediate remediation is advised for vulnerabilities affecting authentication, authorization, and transport security.

2. Scope

Frontend: Flutter (Signup, Login, Live Tracking)

Backend: Node.js with Express.js

Database: MySQL hosted on Railway

APIs Reviewed:

- /signup
- /signin
- /send_tracking_request
- /check_tracking_requests

- /update_tracking_request
- /send_location
- /get_latest_location
- /get_price

---

3. Methodology

The following testing methods were applied:

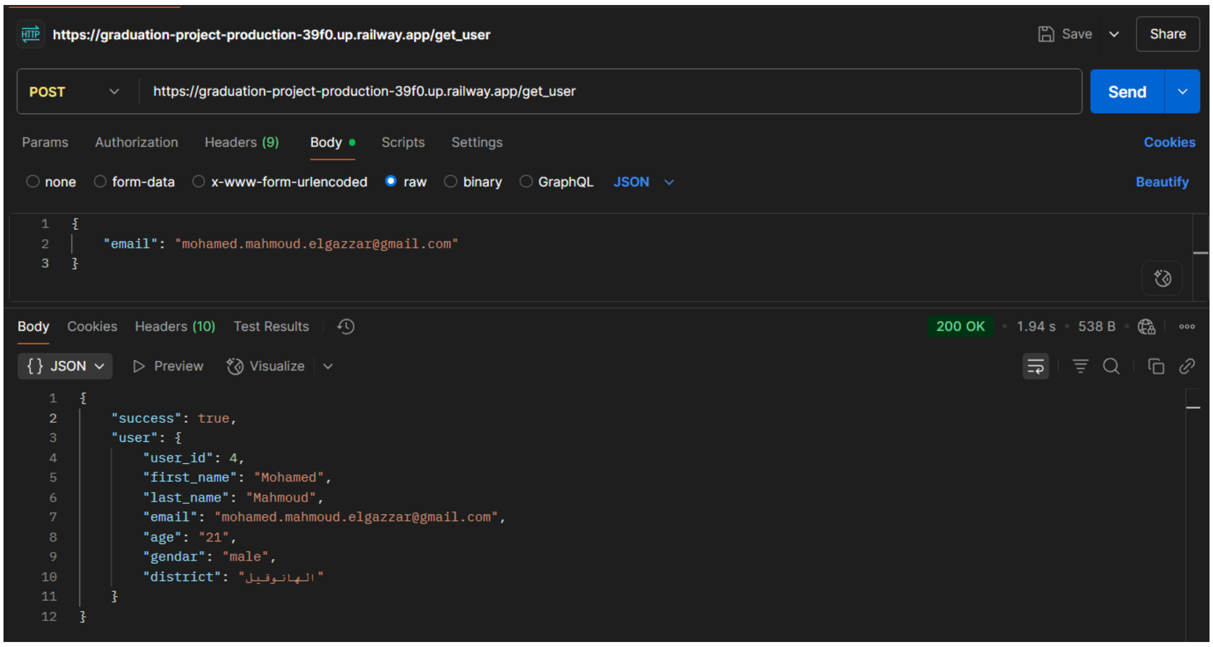
- **White-box Penetration Testing:** Full access to source code and internal API logic.
- **Static Code Analysis:** Reviewed Flutter and Node.js source code.
- **Manual API Testing:** Using Postman to craft and send requests with/without tokens.
- **Token and Session Analysis:** Focused on JWT and authentication flows.
- **OWASP Mobile Top 10 Mapping:** Findings were mapped against OWASP Mobile Top 10 threats.

---

4. Findings

4.1 Lack of Authentication on Sensitive APIs

- **Severity:** Critical
- **Description:** Several endpoints, such as `/send_location` and `/get_latest_location`, were accessible without authentication.
- **Impact:** Unauthorized data manipulation and tracking.
- **OWASP Reference:** M6: Insecure Authorization
- **Layer:** Application Layer (OSI Layer 7)
- **Fix:** JWT-based middleware was implemented to protect all user-specific routes.



POST https://graduation-project-production-39f0.up.railway.app/get_user

```
{
    "email": "mohamed.mahmoud.elgazzar@gmail.com"
}
```

Body | Cookies | Headers (10) | Test Results — 200 OK • 1.94 s • 538 B

```
{
    "success": true,
    "user": {
        "user_id": 4,
        "first_name": "Mohamed",
        "last_name": "Mahmoud",
        "email": "mohamed.mahmoud.elgazzar@gmail.com",
        "age": "21",
        "gendar": "male",
        "district": "الهانوفيل"
    }
}
```

4.2 Open CORS Policy

- **Severity:** High
- **Description:** CORS policy was open to all origins, enabling possible CSRF attacks.
- **Impact:** Cross-origin abuse, unauthorized data exposure from user sessions.
- **OWASP Reference:** M3: Insecure Communication
- **Layer:** Application Layer
- **Fix:** CORS policy restricted to frontend domain.

4.3 JWT Library Present but Not Used

- **Severity:** High
- **Description:** JWT library was installed but not utilized for authentication.
- **Impact:** No secure session control.
- **OWASP Reference:** M4: Insecure Authentication
- **Layer:** Application Layer
- **Fix:** JWT issued on login and verified via middleware.

4.4 No HTTPS Enforcement

- **Severity:** High
- **Description:** HTTP used in requests and server setup.
- **Impact:** Plaintext transmission of sensitive data.
- **OWASP Reference:** M3: Insecure Communication
- **Layer:** Transport Layer (OSI Layer 4)
- **Fix:** HTTPS enforced on Railway and Flutter HTTP clients.

4.5 No Rate Limiting or CAPTCHA

- **Severity:** Medium
- **Description:** `/signin` and `/signup` allowed unlimited attempts.
- **Impact:** Brute-force and credential stuffing attacks.
- **OWASP Reference:** M1, M4
- **Layer:** Application Layer
- **Fix:** `express-rate-limit` and reCAPTCHA added to protect endpoints.

4.6 Weak Password Policy

- **Severity:** Medium
- **Description:** No password complexity requirements enforced.
- **Impact:** Increased likelihood of account compromise.
- **OWASP Reference:** M4: Insecure Authentication
- **Layer:** Application Layer
- **Fix:** Frontend password strength meter + backend regex validation.

4.7 No Email Verification Post Signup

- **Severity:** Informational
- **Description:** Accounts became active without email verification.
- **Impact:** System could be abused with fake/spam accounts.

- **OWASP Reference:** M1: Improper Account Management
- **Layer:** Application Layer
- **Fix:** Verification link sent to email, and access restricted unless `is_verified = true`.

---

5. Recommendations

To improve and maintain the app's security posture:

- Implement and maintain JWT authentication on all sensitive APIs.
- Restrict CORS policy to trusted origins only.
- Enforce HTTPS across all layers.
- Add CAPTCHA and rate limiting to public authentication endpoints.
- Validate password strength both client- and server-side.
- Require email verification for account activation.
- Log and monitor authentication attempts and sensitive requests.

---

6. Conclusion

The application had critical vulnerabilities that could be exploited to compromise user privacy, session integrity, and system functionality. All identified issues were resolved and validated during the same testing cycle. Future efforts should include regular penetration testing, dependency management, and progressive enhancement of authentication methods (e.g., 2FA).

Appendix A: Tools Used

- Postman (Manual API Testing)
- Railway Logs (Request Inspection)
- JWT.io (Token Debugging)
- dotenv (Environment Management)
- express-rate-limit (Brute-force Protection)
- Google reCAPTCHA

Appendix B: Timeline

Assessment Date: May 2025

Duration: Estimated 2 business days

Fixes Applied: Immediately after findings

Tested By: Mohamed Mahmoud Elgazzar (Developer & Security Tester)