To ensure the security and integrity of user data, the following security mechanisms and best practices were implemented:

Authentication, Authorization & Data Protection

- Authentication: We use JSON Web Tokens (JWT) to manage user sessions securely. Tokens are stored in HTTP-only cookies or sent in headers to prevent client-side script access.
- Authorization: Role-based access control (RBAC) is enforced for users and admins to ensure proper permission boundaries across endpoints.
- Password Handling: Passwords are hashed using `bcrypt` with a strong salt factor before storage. Plaintext passwords are never saved or logged.
- Sensitive Data: Personally identifiable information (PII) such as email, address, and phone number are transmitted only over HTTPS and stored securely in the PostgreSQL database with access restrictions at both the application and database level.

Security Best Practices Implemented

- HTTPS: All communications are served over HTTPS (with automatic HTTP to HTTPS redirection in production) to protect against network-based attacks.
- Input Validation: Both client and server-side validations are applied to ensure that only expected and safe data is processed (e.g., for login, registration, order submission, contact forms).
- Rate Limiting: Brute-force protection was added using rate-limiting middleware to limit login attempts and API abuse.
- Helmet.js: We use Helmet middleware in our Express.js server to set appropriate HTTP headers to prevent common web vulnerabilities.
- CORS Protection: Configured CORS policy to allow only whitelisted frontend domains to interact with the backend API.

Security Testing Results

- SQL Injection (SQLi): All SQL queries use parameterized statements with the `pg` module to eliminate the possibility of injection attacks. Manual tests using common payloads returned safe and correct responses.
- Cross-Site Scripting (XSS): All user input is sanitized, and dynamic content is safely rendered using escaping functions on the frontend. Tests using `<script>alert('xss')</script>` were safely neutralized.

- Cross-Site Request Forgery (CSRF): Actions requiring authentication (e.g., order placement, profile updates) are protected using token validation, and state-changing actions are restricted to authenticated users only.
- Dependency Vulnerabilities: We ran `npm audit`, and the results confirmed:

  This indicates all used packages are currently free from known security issues.