**Security Report for Fitness Application**

OWASP Top 10 Security Risks

1. Broken Access Control

* Mitigation:
  + Role-based access control (RBAC) is implemented to ensure users can only access data they are authorized for. Users have specific roles such as CUSTOMER, ADMIN, etc.
  + Access to API endpoints is restricted using annotations like @PreAuthorize in Spring Security.
* Backend Code Example:

@PreAuthorize("hasRole('CUSTOMER') and #id == authentication.principal.id")

@GetMapping("/user/{id}/workouts")

public List<WorkoutDTO> getUserWorkouts(@PathVariable Long id) {

return workoutService.getWorkoutsByUserId(id).stream()

.map(workoutMapper::toDto)

.collect(Collectors.toList());

}

* Why it works:
  + The @PreAuthorize annotation ensures the endpoint is accessible only to users with the CUSTOMER role and whose ID matches the authenticated user's ID.
  + This prevents CUSTOMER A from accessing or modifying data belonging to CUSTOMER B.

2. Injection

* Mitigation:
  + All SQL queries are executed using JPA and Hibernate, which leverage parameterized queries to prevent SQL injection.
  + Direct SQL statements in the code use prepared statements, which automatically escape inputs.
* Backend Code Example:

@Repository

public interface UserRepository extends JpaRepository<User, Long> {

@Query("SELECT u FROM User u WHERE u.username = :username")

Optional<User> findByUsername(@Param("username") String username);

}

* Why it works:
  + The @Query annotation with @Param ensures that user input is parameterized, preventing malicious SQL injection attempts.

3. Security Misconfiguration

* Mitigation:
  + All sensitive configurations such as database credentials and JWT secrets are stored in environment variables.
  + Headers like X-Content-Type-Options, Strict-Transport-Security, and X-Frame-Options are enforced via Nginx or Spring Security.

4. Cryptographic Failures

* Mitigation:
  + Passwords are hashed using BCrypt before storing them in the database.
  + SecureRandom is used for generating tokens.

5. Insecure Design

* Mitigation:
  + Security requirements are considered during design phases. Examples include role-based access and secure API endpoints.

6. Vulnerable and Outdated Components

* Mitigation:
  + Dependencies are managed via Maven, and tools like Dependabot are used to track and update outdated dependencies.

7. Identification and Authentication Failures

* Mitigation:
  + Spring Security's built-in authentication mechanisms are used.
  + JWT tokens are implemented for session management with proper expiration times and signature validation.

8. Software and Data Integrity Failures

* Mitigation:
  + Code integrity is enforced through a CI/CD pipeline, and SonarQube is used for static code analysis.

9. Security Logging and Monitoring Failures

* Mitigation:
  + Logs are generated for critical actions such as authentication and data access using tools like SLF4J and Logback.
  + Unusual activities trigger alerts via monitoring tools such as Prometheus and Grafana.

10. Server-Side Request Forgery (SSRF)

* Mitigation:
  + Outbound requests are restricted using firewalls, and only necessary domains are whitelisted.