Security Risk

A1:2017-Injection

Preventing injection requires keeping data separate from commands and queries.
* The preferred option is to use a safe API, which avoids the use of the interpreter entirely or provides a par
interface, or migrate to use Object Relational Mapping Tools (ORMs).
* Use positive or "whitelist" server-side input validation. This is not a complete defense as many application
characters, such as text areas or APIs for mobile applications.
* For any residual dynamic queries, escape special characters using the specific escape syntax for that interp
* Use LIMIT and other SQL controls within queries to prevent mass disclosure of records in case of SQL inject

A2:2017-Broken Authentication

* Where possible, implement multi-factor authentication to prevent automated, credential stuffing, brute fo
credential re-use attacks.
* Do not ship or deploy with any default credentials, particularly for admin users.

* Implement weak-password checks, such as testing new or changed passwords against a list of the top 100(
passwords.* Align password length, complexity and rotation policies with NIST 800-63 B's guidelines in secti
Memorized Secrets or other modern, evidence based password policies.

* Ensure registration, credential recovery, and API pathways are hardened against account enumeration att
same messages for all outcomes.

* Use a server-side, secure, built-in session manager that generates a new random session ID with high entr
Session IDs should not be in the URL, be securely stored and invalidated after logout, idle, and absolute time

A3:2017-Sensitive Data Exposure

* Classify data processed, stored or transmitted by an application. Identify which data is sensitive according
regulatory requirements, or business needs.
* Apply controls as per the classification.
* Don't store sensitive data unnecessarily. Discard it as soon as possible or use PCI DSS compliant tokenizati
truncation. Data that is not retained cannot be stolen.

* Make sure to encrypt all sensitive data at rest.
* Ensure up-to-date and strong standard algorithms, protocols, and keys are in place; use proper key manag
passwords using strong adaptive and salted hashing functions with a work factor (delay factor), such as Argo
or PBKDF2.
* Verify independently the effectiveness of configuration and settings.

* Encrypt all data in transit with secure protocols such as TLS with perfect forward secrecy (PFS) ciphers, ciph
by the server, and secure parameters. Enforce encryption using directives like HTTP Strict Transport Security
* Disable caching for response that contain sensitive data.

A4:2017-XML External Entities (XXE)

* Whenever possible, use less complex data formats such as JSON, and avoiding serialization of sensitive dat
* Patch or upgrade all XML processors and libraries in use by the application or on the underlying operating
dependency checkers. Update SOAP to SOAP 1.2 or higher.

A5:2017-Broken Access Control

Access control is only effective if enforced in trusted server-side code or server-less API, where the attacker
access control check or metadata.
* With the exception of public resources, deny by default.
* Implement access control mechanisms once and re-use them throughout the application, including minim
* Model access controls should enforce record ownership, rather than accepting that the user can create, re
delete any record.

* Unique application business limit requirements should be enforced by domain models.
* Disable web server directory listing and ensure file metadata (e.g. .git) and backup files are not present wi
* Log access control failures, alert admins when appropriate (e.g. repeated failures).
* Rate limit API and controller access to minimize the harm from automated attack tooling.
* JWT tokens should be invalidated on the server after logout.
Developers and QA staff should include functional access control unit and integration tests.

A6:2017-Security Misconfiguration

A7:2017-Cross-Site Scripting (XSS)

A8:2017-Insecure Deserialization

A9:2017-Using Components with Known Vulnerabilities

A10:2017-Insufficient Logging & Monitoring

	Meets the criteria	Description
	YES	fully meets the criteria
ameterized s require special reter. tion.	YES	By using Hibernate ORM and Spring Data JPA methods with parameterized queries prevents SQL injections.
	Partially	partially meets the criteria
rce, and stolen	NO	
	YES	New admins are assigned from database or from other admins.
0 worst on 5.1.1 for	Partially	The password is validated by length.
acks by using the	NO	
opy after login. outs.	YES	By using JWT tokens, Session IDs are not a security concern.
	Partially	partially meets the criteria
to privacy laws, on or even	NO	
ement.* Store n2, scrypt, bcrypt	YES	Sensitive data is encrypted by using bcrypt hashing algorithm and is also stored encrypted in the database. bcrypt is a password-hashing algorithm incorporating a salt to protect against rainbow table attacks. bcrypt is resistant to brute-force search attacks
er prioritization (HSTS).	NO	

	YES	fully meets the criteria
ta. system. Use	YES	By using JSON instead of XML the application is not vulnerable to XXE attacks. Serialization of sensitive data is avoided. By using REST web services the vulnerabilities of SOAP are not applicable to the application.
	Partially	partially meets the criteria
cannot modify the izing CORS usage. ad, update, or	YES	Authorization is required for all non public resources, access to such pages is denied by default for non authenticated users.
thin web roots.	NO	
	N/A	not applicable to the application
	NO	doesn't meet the criteria
	NO	doesn't meet the criteria
	N/A	The time since the software has been developed is not sufficient enought to be determined if it gets updates frequently or if it uses outdated components and methodologies.
	NO	doesn't meet the criteria

a