Spring Boot 3 - REST API Security

Spring Boot REST API Security Overview

- **Spring Security Model**
  - Defines a **framework** for **security**
  - Implemented using **Servlet filters** in the **background**
  - Two methods of securing an app → **Declarative** and **Programmatic**
- **Spring Security with Servlet Filters**
  - **Servlet Filters** are used to **pre-process / post-process web requests**
  - **Servlet Filters** can **route web requests based** on **security logic**
  - **Spring** provides a bulk of **security functionality** with **servlet filters**
- **Security Concepts**
  - **Authentication** → Check **User ID** and **Password** with **credentials** stored in the **application** or **database**
  - **Authorization** → Determine if the **user** has an **authorized role**
- **Declarative Security**
  - Define the **application's security constraints** in **configuration**
    - All **Java config** → **@Configuration**
  - Provides **separation of concerns** between **application code** and **security**
- **Programmatic Security**
  - **Spring Security** provides an **API** for **custom application coding**
  - Provides **greater customization** for specific **app requirements**
- **Enabling Spring Security**
  - Add **spring-boot-starter-security** to **pom.xml**
  - **Automatically secures all endpoints**
- **Secured Endpoints**
  - **Spring** prompts for a login
  - Credentials
    - **Username** → Defaults to **admin**

        - **Password** → Located in the **console logs**
- **Spring Security Configuration**
    - Can **override default username** and **generated password** in application.properties
- **Authentication and Authorization**
    - In-memory
    - JDBC
    - LDAP
    - Custom / Pluggable
    - etc.

Spring Boot REST API Security - Basic Configuration

- **Development Process**
    - Create **Spring Security Configuration** → **@Configuration**
    - Add **users**, **passwords**, and **roles**
- **Spring Security Password Storage**

ID	Description
noop	Plain-text passwords
bcrypt	BCrypt password hashing
. . .	. . .

Spring Boot REST API Security - Restrict URLs Based on Roles

- **Restricting Access to Roles**
    - General **syntax**
        - **requestMatchers(<< add HTTP METHOD to match on >>, << add path to match on >>) .hasRole(<< authorized role >>);**
        - **.hasRole(XXX)** → **Single** role

        - **.hasAnyRole(<< list of authorized roles >>)** → Comma-delimited list
- **Cross-Site Request Forgery (CSRF)**
    - **Spring Security** can **protect** against **CSRF attacks**
    - **Embed** additional **authentication data / token** into **all HTML forms**
    - On **subsequent requests**, the **web app** will **verify** the **token before processing**
    - The **primary use case** is for **traditional web applications**
- **When** to use **CSRF Protection?**
    - Per the **Spring Security** team's recommendations
        - Use **CSRF protection** for any **normal browser web requests**
        - **Traditional web apps** with **HTML forms** to **add/modify data**
    - May want to **disable CSRF** for **non-browser clients**

Spring Boot REST API Security - JDBC Authentication - Plain Text

- **Database Support in Spring Security**
    - **Spring Security** can read user **account info** from a **database**
    - By default, you have to follow **Spring Security's predefined table schemas**
        - Requires **users** and **authorities** tables
- **Customize Database Access** with **Spring Security**
    - Can also **customize table schemas**
    - Useful if you have **custom tables** specific to your **project**
    - You'll also be **responsible** for **developing** the **code** to **access** the **data**
        - **JDBC**, **JPA/Hibernate**, etc.
- **Development Process**
    - Develop **SQL Script** to **initialize database tables**
    - **Add database support** to the **Maven POM file**
    - Create **JDBC properties file**
    - Update **Spring Security Configuration** to use **JDBC**

Spring Boot REST API Security - BCrypt Encryption

- **Password Storage - Best Practice**
  - The **best practice** s to **store passwords** in an **encrypted format**
- **Spring Security Team Recommendation**
  - **Spring Security** recommends using the popular **bcrypt** algorithm
  - **bcrypt**
    - Performs **one-way encrypted hashing**
    - Adds a random **salt** to the password for additional proection
    - Includes **support** to defeat **brute-force attacks**
- **Development Process**
  - Run **SQL Script** containing **encrypted passwords**
    - Modify **DDL** for **password field** → **Length** should be **68**
      - **{bcrypt}** → **8**
      - **encodedPassword** → **60**
- **Spring Security Login Process**
  - **Retrieve password** from the **database** for the user
  - Read the **encoding algorithm ID** (bcyrpt, etc.)
  - For the case of **bcrypt**, **encrypt** the **plain-text password** from the **login form** (using a salt from the database password)
  - **Compare** the **encrypted password** from the **login form** with the **encrypted password** from the **database**
  - If there's a **match**, login **successful**
  - If there's **no match**, login **unsuccessful**

Spring Boot REST API Security - JDBC Authentication - Custom Tables

- **Custom Tables**
  - Tell **Spring** how to **query** your **custom tables**
  - Provide **query** to **find user** by **name**
  - Provide **query** to **find authorities/roles** by **username**
- **Development Process**

- **Create** our **custom tables** with **SQL**
- Update **Spring Security Configuration**
  - Provide **query** to find **user** by **username**
  - Provide **query** to **find authorities/roles** by **username**