# Install STS

Spring STS Installation

* Ensure that Java 8 is present with any of the following commands
  + Javac -version
  + Java -version
* Download Spring Tool Suite
  + Download from [here](https://spring.io/tools)
  + Select Spring Tools 4 Eclipse
* Create a suitable installation directory
  + Example
    - C:\Softwares\STS
    - Copy the “spring-tool-suite-4-4.15.1.RELEASE-...jar” to the above directory
    - Run the command to install STS
      * “**java -jar spring-tool-suite-4-4.15.1.RELEASE.jar**”
* Launch STS
  + Specify workspace directory
* Settings
  + Windows -> Preferences -> Maven
    - Check the option “Download Repo index updates on startup”
* Restart STS

# Project Setup & Creation & Welcome Page

* Create Spring Starter Project
  + Page 1
    - Type - Maven
    - Packaging - Jar
    - Java Version - 8
    - Group
      * “com.greatlearning”
    - Artifact
      * studentmanagement
    - Version
      * 1.0
    - Package
      * “com.greatlearning.studentmanagement”
  + Page 2
    - Dependencies
      * Spring Web
      * Spring Boot DevTools
    - Go to Next and Finish
* POM.xml
  + In addition to existing dependencies, also include the following dependency within the <dependencies> tag
    - <dependency>
    - <groupId>org.apache.tomcat.embed</groupId>
    - <artifactId>tomcat-embed-jasper</artifactId>
    - <scope>provided</scope>
    - </dependency>
* File Changes
  + application.properties
    - # Application context name
    - server.servlet.context-path=/studentmanagement
    - # Spring MVC related
    - spring.mvc.view.prefix=/WEB-INF/views/
    - spring.mvc.view.suffix=.jsp
  + Create new folders
    - Create the following directory under “main” folder
      * “webapp/WEB-INF/views”
  + WelcomeController (controller)
    - Create the subpackage “controller”
    - Code
      * @Controller
      * public class WelcomeController{
      * @RequestMapping("/")
      * public String handleWelcomePage() {
      * return "welcome";
      * }
      * }
  + welcome.jsp (webapp/WEB-INF/views)
    - <html>
    - <head>
    - <title>Spring Boot - Student Management</title>
    - </head>
    - <body>
    - Spring Boot - Welcome to Student Management
    - </body>
    - </html>
* Run the project
  + Right click the project -> Run as -> Spring Boot Application
* Access the project
  + http://localhost:8080/studentmanagement/

# Listing Page

* Pom.xml changes
  + <dependency>
  + <groupId>org.springframework.boot</groupId>
  + <artifactId>spring-boot-starter-data-jpa</artifactId>
  + </dependency>
  + <dependency>
  + <groupId>mysql</groupId>
  + <artifactId>mysql-connector-java</artifactId>
  + <scope>runtime</scope>
  + </dependency>
  + <dependency>
  + <groupId>javax.servlet</groupId>
  + <artifactId>jstl</artifactId>
  + </dependency>

<dependency>

* + <groupId>org.springframework.security</groupId>
  + <artifactId>spring-security-taglibs</artifactId>
  + <version>5.2.2.RELEASE</version>

</dependency>

* application.properties
  + spring.jpa.hibernate.ddl-auto=update
  + spring.datasource.url=jdbc:mysql://localhost:3306/lib-mgmt-4-spring-boot
  + spring.datasource.username=test\_user
  + spring.datasource.password=test@12345
  + spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect
* Create the Database
  + Database alone and not the tables
* Student Entity
  + Entire file
* Student Repository interface
  + No need to add any methods for this list functionality
* Student Service interface
  + Addition of listAll method
* Student Service Impl class
  + Implementation of listAll method that makes use of StudentRepository dependency
* students-lister.jsp
  + Include all the file contents
  + Exclude the following
    - Search form & Add button
    - Table
      * Remove of Action and its corresponding row
* StudentsController
  + Addition of the method listStudents
* Access URL
  + [http://localhost:8080/library4/students/list](http://localhost:8080/library4/books/list)
* Student Records creation
  + Go and check the database to see the Book table being created
  + Add the following records
    - Insert into book (id, name, author, category) values (1,
    - "Lord of the Rings", "Tolkien, J.R.", "Adventure Fantasy");
    - Insert into book (id, name, author, category) values (2,
    - "Harry Potter", "Rowling, J.K.", "Adventure Fantasy");
    - Insert into book (id, name, author, category) values (3,
    - "The Hours", "Cunnningham, Michael", "Family Drama");

# Add Student

* Entity
  + No changes
* Repository
  + No changes
* StudentService
  + Addition of the save method
* StudentServiceImpl
  + Implementation for the save method
* StudentController
  + Addition of the handleBeginAdd method
  + Addition of the handleSaveStudent method
* students-lister.jsp
  + Addition of <a> tag for ‘Add’ button
    - Make sure that the context path is updated (1 place)
* student-details-form.jsp
  + Addition of the entire content
  + Make sure to update the context path (2 places)

# Update Book

* Entity
  + No changes
* Repository
  + No changes
* StudentService
  + Add findById method
* StudentServiceImpl
  + Implementation for findById method
* StudentController
  + Implementation for handleBeginUpdate method
  + Update code for handleSaveBook method
* students-lister.jsp
  + Addition of 5th element within the table
    - Action element
    - Corresponding <td> element
      * Make sure that the context path is updated

# Delete Student

* Entity
  + No changes
* Repository
  + No changes
* StudentService
  + Addition of deleteById method
* StudentServiceImpl
  + Implementation of deleteById implementation
* StudentController
  + Addition of handleDelete method
* student-lister.jsp
  + Addition of 5th element within the table
    - Action element
    - Corresponding <td> element

# Security Features

## Context

* Terminologies
  + Authentication
    - Application asking the question “Who are you” and the user providing credentials (username / password) to the application
  + Authorization
    - Application checking “whether whatever the user requested for” is allowed or not?
  + Principal
    - Represents the user who has been identified and verified through the process of Authentication
    - Currently logged in user
  + Authority / Granted Authority
    - Represents one or more permissions
    - Example
      * The user ‘user-1’ tries to list down the books and the application will allow the ‘user-1’ to do so, only if the user is granted authority / permission to do it
  + Role
    - A group of authority / permission
    - Example
      * Admin\_User
        + Can list, add, update and delete
      * Normal\_User
        + Can only list and add

## Default Spring Boot Security Features

* Step 1
  + POM.xml changes
    - <dependency>
      * <groupId>org.springframework.boot</groupId>
    - <artifactId>spring-boot-starter-security</artifactId>
    - </dependency>
  + Note
    - Console
      * Watch out in the console for ‘Security related / password’ output
    - The end points / API / URLs are protected
      * Access to “/library4/books/list” prompts for login
    - Spring Boot provides default login page
    - Submitting wrong credentials (example - “test/test”) and validations are triggered as part of response as well
    - Submit will trigger an action to “/library4/login”
    - User Credentials
      * “user / <console\_password>”
* Step 2
  + To login to the application
    - Option 1
      * Use the following credentials
        + Username - “user”
        + Password - “<value available in the console>”
    - Option 2
      * Update application.properties with the following values and login to the application
        + spring.security.user.name = mysecretuser
        + spring.security.user.password = mysecretpassword

## Adding Custom Security

* Step 1
  + Update pom.xml with the following dependency
    - <dependency>
      * <groupId>org.springframework.boot</groupId>
    - <artifactId>spring-boot-starter-security</artifactId>
    - </dependency>
* Step 2
  + Remove the following entries in application.properties, if they are present
    - “spring.security.user.name”
    - “spring.security.user.password”
* Step 3
  + Configure Authentication
    - The method WebSecurityConfigurerAdapter.configure(AuthenticationManagerBuilder) needs to be overridden to configure custom authentication
    - Create a class ‘LibrarySecurityAdapter’ that extends ‘WebSecurityConfigurerAdapter’ and place it in the package ‘com.greatlearning.librarymanagement.security’
    - Option 1
      * Configuring InMemoryAuthentication
      * AuthenticationBuilder
        + Specify the type of authentication

Example

In-Memory authentication

Code

auth.inMemoryAuthentication()

This opens up the method chain

A call to and() ends the chaining

* + - * + Supply the username, password and roles

“withUser(“user-1”).password(“mypassword”).roles(“NORMAL\_USER”)

Note - the role will not be getting used in this scenario

* + - * Add @EnableWebSecurity annotation
      * Password Encoding
        + Plain text password should not be used
        + Define @Bean PasswordEncoder and Spring will do the mapping automatically
        + Return an instance of BCryptPasswordEncoder()
        + Encode the password using the “BCrypt Online tool”

<https://bcrypt-generator.com/>

* + - Option 2
      * User & Role Additions
        + Java Entity Additions

Addition of Role class

Addition of User class

ManyToMany

One user can have many roles

One role can have many users

CascadeType.ALL

The propagation nature is applicable from parent to all child entities (Persist, Remove, Refresh)

JoinTable

* + - * + LibraryUserDetails (security package)

Copy the entire content

* + - * + UserRepository class

Copy the entire content

* + - * + LibraryUserDetailsServiceImpl

A flavour of class that implements ‘org.springframework.security.core.userdetails.UserDetailsService’ must be available, hence this class

The method loadUserByUsername() needs to be overridden and the return type must be ‘org.springframework.security.core.userdetails.UserDetails’

Hence the custom class ‘MyUserDetails’ is created

The implementation of loadUserByUsername has to do

Retrieving the user object from database

Hence the UserRepository class is written

Converts the library-user to a ‘user that spring can understand’

* + - * Population of Data
        + Usage of brypt (<https://bcrypt-generator.com/>) for encrypting data

Data

“mypassword”

$2a$12$ncVKTp7LOrPWiHqyEDginOO/4o4abR9rvbYbv5nX1wejkjMWHdAfm

* + - * + Role table records

insert into roles (role\_id, name) values (1, 'NORMAL\_USER');

insert into roles (role\_id, name) values (2, 'ADMIN\_USER');

* + - * + User table records

insert into users (user\_id, username, password) values (

1, 'varun', '$2a$12$ncVKTp7LOrPWiHqyEDginOO/4o4abR9rvbYbv5nX1wejkjMWHdAfm');

insert into users (user\_id, username, password) values (

2, 'sanjay', '$2a$12$ncVKTp7LOrPWiHqyEDginOO/4o4abR9rvbYbv5nX1wejkjMWHdAfm');

* + - * + Users\_Roles table records

insert into users\_roles (user\_id, role\_id) values (

(select user\_id from users where username = 'varun'),

(select role\_id from roles where name = 'ADMIN\_USER')

);

insert into users\_roles (user\_id, role\_id) values (

(select user\_id from users where username = 'sanjay'),

(select role\_id from roles where name = 'NORMAL\_USER')

);

* Step 4
  + Configure Authorization
    - Define the Authorization rules
      * Mapping between the API / Endpoint to the roles

| **API / Endpoint** | **Roles allowed for access** | **Comments** |
| --- | --- | --- |
| /book/save | NORMAL\_USER  ADMIN\_USER |  |
| /book/add-begin | NORMAL\_USER  ADMIN\_USER |  |
| /books/403 | NORMAL\_USER  ADMIN\_USER |  |
| /book/update-begin | ADMIN\_USER |  |
| /book/delete | ADMIN\_USER |  |
| /login | Anyone |  |

* + - The method WebSecurityConfigurerAdapter.configure(HttpSecurity) needs to be overridden to configure custom authorization
    - Mapping
      * The URL path needs to be mapped for the role
    - On HttpSecurity, do the following
      * httpSecurity.authorizeRequests()
        + Opens up the chain
        + This follows the pattern

Wherein the URL pattern is specified through antMatchers() followed by hasAuthority() or hasAnyAuthority()

* + - * To allow access on any API
        + Use permitAll()
      * Watch out for the values of user types
        + NORMAL\_USER
        + ADMIN\_USER
    - Define Authentication Error page - 403.jsp
      * Usage of the key “msg”
    - Add the accessDenied() in BooksController
      * Note
        + View name set to “403”
        + The key “msg” has the relevant message
  + Note
    - Make use of Incognito window for test purposes
    - If the logged-in user is persisted for a URL access, make use of the following URL to logout and to re-trigger a login
      * http://localhost:8080/library5/logout
* Step 5 - Logout link
  + students-lister.jsp
    - To have the logout link
      * The implementation handled by Spring Boot itself
      * The context path to be updated as well

# References

| **Item** | **Reference** |
| --- | --- |
|  |  |