Spring Security - Remember Me

Last Updated : 23 Jul, 2025

In order to remember the users on the computer while logging into a website, via **Spring Security**, we can maintain that by using a "Remember Me" option. **Spring Security** provides a **"Remember Me"** feature to allow users to stay logged in even after closing the browser**.** sends a cookie to the browser. This will have a specific time limit, and till that time, the cookie is valid, and in that timeframe, for the given URL, automatic login is possible. Let us see via a sample application code and the working functionality.

Step-by-Step Implementation

Spring Security provides two approaches for implementing remember-me:

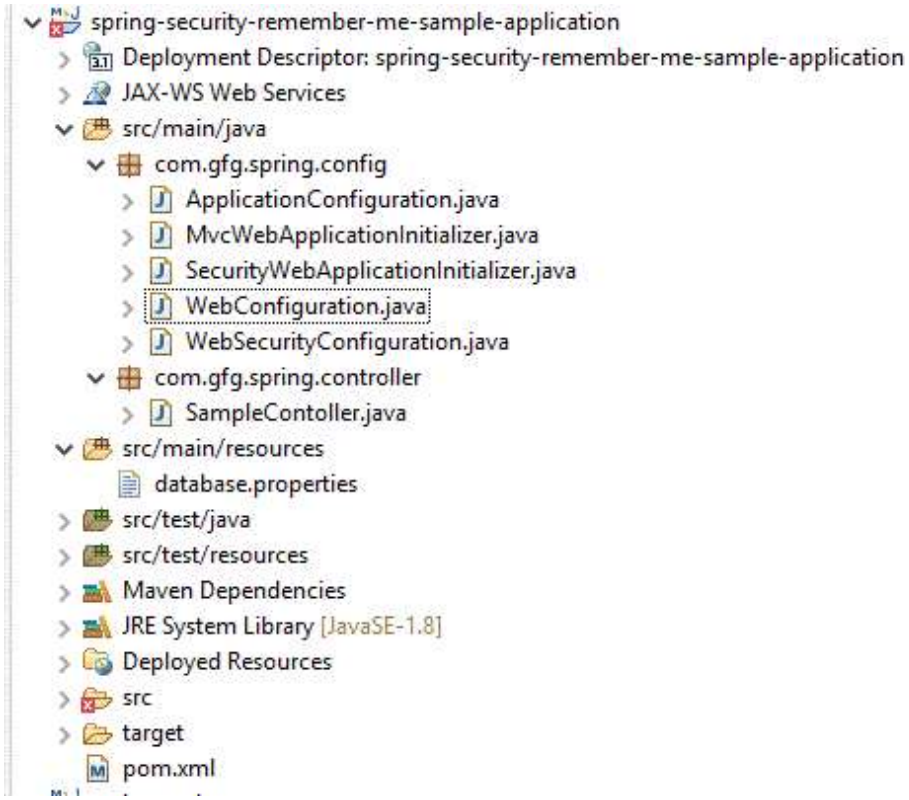
- Hash-Based Token Approach
- Persistent Token Approach

In our example, we are taking the **Persistent Token Approach** in which a database or other persistent storage mechanism is used, and it is helpful to store the generated tokens.

Step 1: Create a Spring Boot Application

First, we need to create a Spring Boot application. Then we need to import our project to the IDE (STS or IntelliJ IDEA) and create packages and classes as shown in the project structure image below.

Project Structure:



This is a maven-driven project. Let us see the pom.xml

```
<project xmlns="https://maven.apache.org/POM/4.0.0"
         xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="https://maven.apache.org/POM/4.0.0
                               https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.gfg.springsecurity</groupId>
  <artifactId>spring-security-remember-me-sample-application</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>spring-security-remember-me-sample-application</name>
  <packaging>war</packaging>
  <properties>
    <!-- Updated to Java 17 as required by Spring Boot 3.x -->
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
    <failOnMissingWebXml>false</failOnMissingWebXml>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
      <version>3.2.2</version>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-security</artifactId>
      <version>3.2.2</version>
    </dependency>
```

```
    <dependency>
      <groupId>org.springframework.security</groupId>
      <artifactId>spring-security-web</artifactId>
      <version>6.2.2</version>
    </dependency>
    <dependency>
      <groupId>org.springframework.security</groupId>
      <artifactId>spring-security-config</artifactId>
      <version>6.2.2</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-jdbc</artifactId>
      <version>6.1.2</version>
    </dependency>
    <dependency>
      <groupId>jakarta.servlet</groupId>
      <artifactId>jakarta.servlet-api</artifactId>
      <version>6.0.0</version>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <groupId>jakarta.servlet.jsp</groupId>
      <artifactId>jakarta.servlet.jsp-api</artifactId>
      <version>3.0.0</version>
      <scope>provided</scope>
    </dependency>
    <dependency>
      <groupId>org.glassfish.web</groupId>
      <artifactId>jakarta.servlet.jsp.jstl</artifactId>
      <version>2.0.0</version>
    </dependency>
    <dependency>
      <groupId>org.apache.commons</groupId>
      <artifactId>commons-dbcp2</artifactId>
      <version>2.9.0</version>
    </dependency>
    <dependency>
      <groupId>com.mysql</groupId>
      <artifactId>mysql-connector-j</artifactId>
      <version>8.2.0</version>
    </dependency>
  </dependencies>
  <build>
    <plugins>
      <!-- Updated Jetty plugin version -->
      <plugin>
        <groupId>org.eclipse.jetty</groupId>
        <artifactId>jetty-maven-plugin</artifactId>
        <version>11.0.17</version>
      </plugin>
      <!-- Required Maven WAR plugin -->
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-war-plugin</artifactId>
        <version>3.4.0</version>
```

```
      </plugin>
    </plugins>
  </build>
</project>
```

Step 2: Database Configuration

On the database side, we are using MySQL. Hence, those configurations are given in pom.xml. To test the application for remembering, we need to have some sample tables in the database. Let the sample database is 'geeksforgeeks'. sample tables are 'users' and 'authorities'. For 'remember me' authentication, we need 'persistent_logins' as well for storing the generated tokens

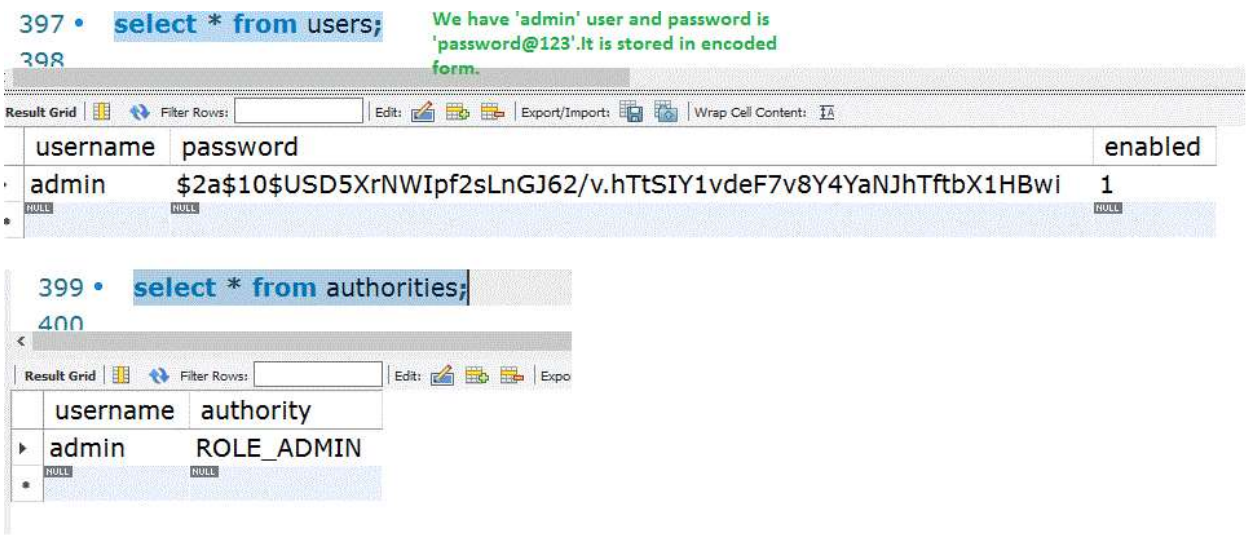
```
use geeksforgeeks;
-- users
create table users(
    username varchar(50) not null primary key,
    password varchar(100) not null,
    enabled boolean not null
);
-- authorities
create table authorities(
    username varchar(50) not null,
    authority varchar(50) not null,
    constraint fk_authorities_users foreign key(username) references
users(username)
);
create unique index ix_auth_username on geekAuthorities
(username,authority);

create table persistent_logins(
    username varchar(50) not null,
    series varchar(64) primary key,
    token varchar(64) not null,
    last_used timestamp not null
);
```

Step 3: Insert Sample Data

Let us insert a few data into the users and authorities table for testing purposes

-- Let us create a user with admin and password as password@123
-- While storing into the database let us store as encoded password with BCryptPasswordEncoder
-- For password@123, it will be $2a$10$USD5XrNWIpf2sLnGJ62/v.hTtSIY1vdeF7v8Y4YaNJhTftbX1HBwi
insert into users(username,password,enabled)

values('admin','$2a$10$hbxecwitQQ.dDT4JOFzQAulNySFwEpaFLw38jda6Td.Y/cOiRzDFu',true);
insert into authorities(username,authority)
values('admin','ROLE_ADMIN');



To get the encoded password, by using a sample code, we can get it:

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

//.....
BCryptPasswordEncoder passwordEncoder = new BCryptPasswordEncoder();
// Specify the required password here
```

```
String password = "password@123";
String encodedPassword = passwordEncoder.encode(password);
//---
```

Step 4: Configure Database Connectivity

Now let us check the database connectivity information. It is available under the **src/main/resources** folder.

database.properties:

spring.datasource.url=jdbc:mysql://localhost:3306/your_database?useSSL=false&allowPublicKeyRetrieval=true&serverTimezone=UTC

spring.datasource.username=root

spring.datasource.password=yourpassword

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

Step 5: Application Configuration

Let us start by creating the **@Configuration** class and inside that define the **@Bean** method for DataSource.

ApplicationConfiguration.java:

```
import javax.sql.DataSource;

import org.apache.commons.dbcp2.BasicDataSource;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;
import org.springframework.core.env.Environment;

@Configuration
@PropertySource("classpath:database.properties")
public class ApplicationConfiguration {
```

```
  @Autowired
  private Environment environment;

  @Bean
  public DataSource getDataSource() {
    BasicDataSource basicDataSource = new BasicDataSource();

basicDataSource.setDriverClassName(environment.getProperty("mysql.driver"));
    basicDataSource.setUrl(environment.getProperty("mysql.jdbcUrl"));
    basicDataSource.setUsername(environment.getProperty("mysql.username"));
    basicDataSource.setPassword(environment.getProperty("mysql.password"));
    return basicDataSource;
  }
}
```

Step 6: Spring Security Configuration

- Creation of a springSecurityFilterChain Servlet Filter is needed for protecting and validating all URLs by creating a @Configuration class.
- Need to register the springSecurityFilterChain filter with war.
- rememberMe() method of the HttpSecurity class is required to enable remember-me authentication
- Invoke the tokenRepository() method with the PersistentTokenRepository argument and store the generated tokens in the database table

WebSecurityConfiguration.java:

```
import jakarta.sql.DataSource;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.authentication.configuration.A
uthenticationManagerConfigurer;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.provisioning.InMemoryUserDetailsManager;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
```

```
import
org.springframework.security.web.authentication.rememberme.JdbcTokenRepositor
yImpl;
import
org.springframework.security.web.authentication.rememberme.PersistentTokenRep
ository;

@Configuration
public class WebSecurityConfiguration {

  private final DataSource dataSource;

  public WebSecurityConfiguration(DataSource dataSource) {
    this.dataSource = dataSource;
  }

  @Bean
  public SecurityFilterChain securityFilterChain(HttpSecurity http) throws
Exception {
    http.authorizeHttpRequests(auth -> auth
        .requestMatchers("/login**").permitAll()
        .anyRequest().hasAnyRole("ADMIN", "USER")
    )
    .formLogin(form -> form
        .loginPage("/login")
        .loginProcessingUrl("/loginAction")
        .permitAll()
    )
    .logout(logout -> logout.logoutSuccessUrl("/login").permitAll())
    .rememberMe(rememberMe -> rememberMe
        .rememberMeParameter("remember-me")
        .tokenRepository(persistentTokenRepository(dataSource))
    )
    .csrf(csrf -> csrf.disable());

    return http.build();
  }

  @Bean
  public UserDetailsService userDetailsService() {
    UserDetails user = User.withUsername("user")
        .password(passwordEncoder().encode("password"))
        .roles("USER")
        .build();
    return new InMemoryUserDetailsManager(user);
  }

  @Bean
  public PasswordEncoder passwordEncoder() {
    return new BCryptPasswordEncoder();
  }

  @Bean
  public PersistentTokenRepository persistentTokenRepository(DataSource
dataSource) {
    JdbcTokenRepositoryImpl repo = new JdbcTokenRepositoryImpl();
```

```
    repo.setDataSource(dataSource);
    return repo;
}

}
```

Note: The encoded password must be stored in the database.

Step 7: Spring MVC Configuration

Spring MVC Configuration is getting done by using JSP Views for the view part.

WebConfiguration.java:

```
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import
org.springframework.web.servlet.config.annotation.ViewControllerRegistry;
import
org.springframework.web.servlet.config.annotation.ViewResolverRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
@EnableWebMvc
@ComponentScan(basePackages = { "com.gfg.spring.controller" })
public class WebConfiguration implements WebMvcConfigurer {

  @Override
  public void configureViewResolvers(ViewResolverRegistry registry) {
    registry.jsp().prefix("/WEB-INF/views/").suffix(".jsp");
  }

  @Override
  public void addViewControllers(ViewControllerRegistry registry) {
    registry.addViewController("/login").setViewName("login");
  }
}
```

Step 8: Servlet Initialization

Servlet container Initialization and configuration can be seen by using,

MvcWebApplicationInitializer.java:

```
import
org.springframework.web.servlet.support.AbstractAnnotationConfigDispatcherSer
vletInitializer;

public class MvcWebApplicationInitializer
      extends AbstractAnnotationConfigDispatcherServletInitializer {

  // Load database and spring security configuration
  @Override
  protected Class<?>[] getRootConfigClasses() {
    return new Class[] { ApplicationConfiguration.class,
WebSecurityConfiguration.class };
  }

  // Load spring web configuration
  @Override
  protected Class<?>[] getServletConfigClasses() {
    return new Class[] { WebConfiguration.class };
  }

  @Override
  protected String[] getServletMappings() {
    return new String[] { "/" };
  }

}
```

Step 9: Create Controller

Let us see the controller class to handle requests and display messages.

SampleContoller.java:

```
import java.security.Principal;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class SampleContoller {
  @GetMapping("/")
  public String index(Model model, Principal principal) {
    model.addAttribute("message", "Welcome! You are logged by using " +
```

```
principal.getName() + " username");
    return "index";
  }
}
```

Step 10: Create JSP Views

Let us see the JSP view part now

login.jsp:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@taglib uri="http://www.springframework.org/tags" prefix="spring"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-
1">
        <title>Spring Security</title>
        <style>
            table#sample tr:nth-child(odd) {
            background-color: #0074ab ;
            color: yellow ;
            }
            table#sample tr:nth-child(even) {
            background-color: #e6f7ff ;
            }
            table#sample {
            border-collapse: collapse ;
            }
            table#sample td {
            padding: 5px ;
            }
            table#sample caption {
            font-style: italic ;
            background-color: black ;
            color: white ;
            }
        </style>
    </head>
    <body background="#FFFFFF">
        <center>
            <h1>Spring Security - Remember Me Example</h1>
            <h4>Sample Login Form</h4>
            <form action='<spring:url value="/loginAction"/>' method="post">
                <table id = "sample">
                    <tr>
```

```
                <td>Username</td>
                <td><input type="text" name="username"></td>
            </tr>
            <tr>
                <td>Password</td>
                <td><input type="password" name="password"></td>
            </tr>
            <tr>
                <td><input type="checkbox" name="remember-me"></td>
                <td>Remember me on this Computer</td>
            </tr>
            <tr>
                <td align="center" colspan="2"><button
type="submit">Login</button></td>
            </tr>
        </table>
    </form>
    <br/>
  </center>
 </body>
</html>
```

index.jsp:

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html>
<html>
 <head>
  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-
1">
  <title>Spring Security</title>
 </head>
 <body>
  <h1>Spring Security - Remember Me Example</h1>
  <h2>${message}</h2>
  <form action="/logout" method="post">
   <input value="Logout" type="submit">
  </form>
 </body>
</html>
```

Step 11: Build and Run the Application

As this is the maven project, first let us build the application from the command prompt as follows:

mvn clean install

Output:

```
D:\Priya\geeksforgeeks\spring-security-remember-me-example\spring-security-remember-me-example>mvn clean install
[INFO] Scanning for projects...
[INFO]
[INFO] --< com.gfg.springsecurity:spring-security-remember-me-sample-application >--
[INFO] Building spring-security-remember-me-sample-application 0.0.1-SNAPSHOT
[INFO] --------------------------------[ war ]---------------------------------
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ spring-security-remember-me-sample-application ---
[INFO] Deleting D:\Priya\geeksforgeeks\spring-security-remember-me-example\spring-security-remember-me-example\targ
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ spring-security-remember-me-sample-applicatio
ly) to copy filtered resources, i.e. build is platform dependent!
[INFO] Copying 1 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ spring-security-remember-me-sample-application ---
[INFO] Changes detected - recompiling the module!
[WARNING] File encoding has not been set, using platform encoding Cp1252, i.e. build is platform dependent!
[INFO] Compiling 6 source files to D:\Priya\geeksforgeeks\spring-security-remember-me-example\spring-security-remem
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ spring-security-remember-me-sample-ap
[WARNING] Using platform encoding (Cp1252 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ spring-security-remember-me-sample-applica
```

Run the application by using below command:

mvn jetty:run

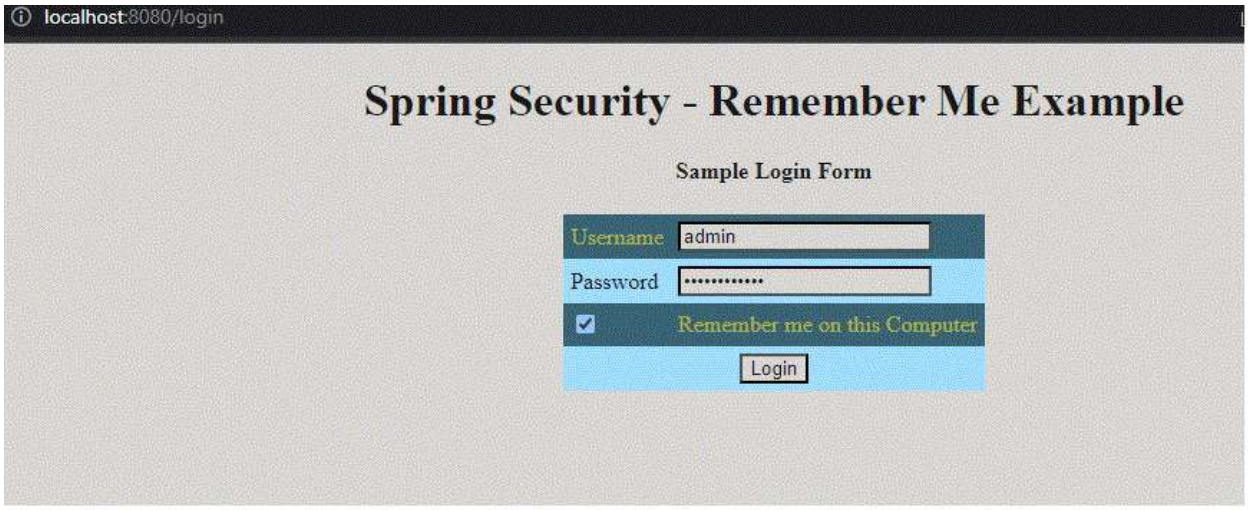
Output:

```
D:\Priya\geeksforgeeks\spring-security-remember-me-example\spring-security-remember-me-example>mvn jetty:run
[INFO] Scanning for projects...
[INFO]
[INFO] --< com.gfg.springsecurity:spring-security-remember-me-sample-application >--
[INFO] Building spring-security-remember-me-sample-application 0.0.1-SNAPSHOT
[INFO] --------------------------------[ war ]---------------------------------
[INFO]
[INFO] >>> jetty-maven-plugin:9.4.8.v20171121:run (default-cli) > test-compile @ spring-security-remember-me-sample
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ spring-security-remember-me-sample-applicatio
ly) to copy filtered resources, i.e. build is platform dependent!
[INFO] Copying 1 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ spring-security-remember-me-sample-application ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- [INFO] Root WebApplicationContext: initialization completed in 1318 ms
[WARNING] [INFO] Initializing Spring FrameworkServlet 'dispatcher'
[INFO] Co [INFO] FrameworkServlet 'dispatcher': initialization started
[INFO]    [INFO] Refreshing WebApplicationContext for namespace 'dispatcher-servlet': startup date [Fri Jun 10
[INFO] --- [INFO] Registering annotated classes: [class com.gfg.spring.config.WebConfiguration]
Not       [INFO] JSR-330 'javax.inject.Inject' annotation found and supported for autowiring
          [INFO] Mapped "{[/],methods=[GET]}" onto public java.lang.String com.gfg.spring.controller.SampleCon
          [INFO] Mapped URL path [/login] onto handler of type [class org.springframework.web.servlet.mvc.Para
          [INFO] Looking for @ControllerAdvice: WebApplicationContext for namespace 'dispatcher-servlet': star
          Context
          [INFO] FrameworkServlet 'dispatcher': initialization completed in 561 ms
          [INFO] Started o.e.j.m.p.JettyWebAppContext@68c87fc3{/,file:///D:/Priya/geeksforgeeks/spring-securit
          ebapp/,AVAILABLE}{file:///D:/Priya/geeksforgeeks/spring-security-remember-me-example/spring-security
          [INFO] Started ServerConnector@7af0affa{HTTP/1.1,[http/1.1]}{0.0.0.0:8080}
          [INFO] Started @7322ms
```

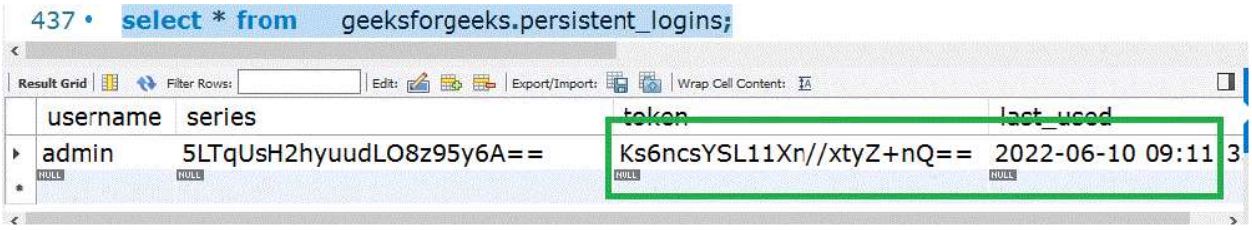
We can see that jetty server has started at port 8080

Step 12: Test the Remember-Me Feature

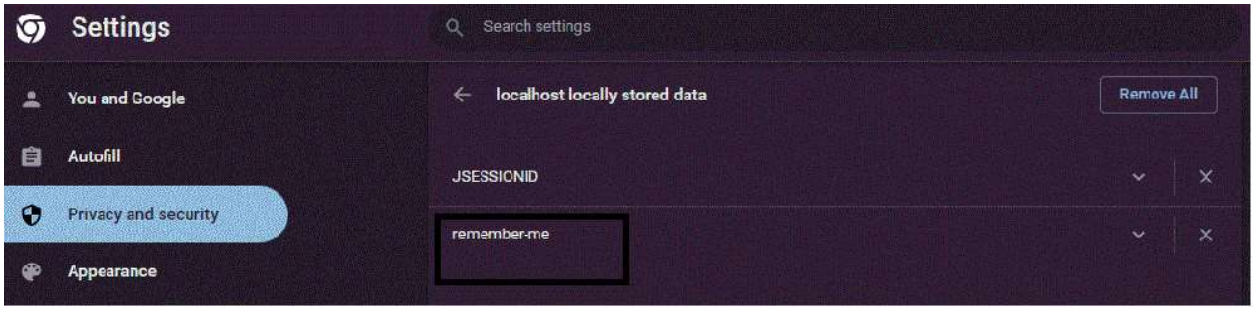
Let us test now by hitting **http://localhost:8080/**

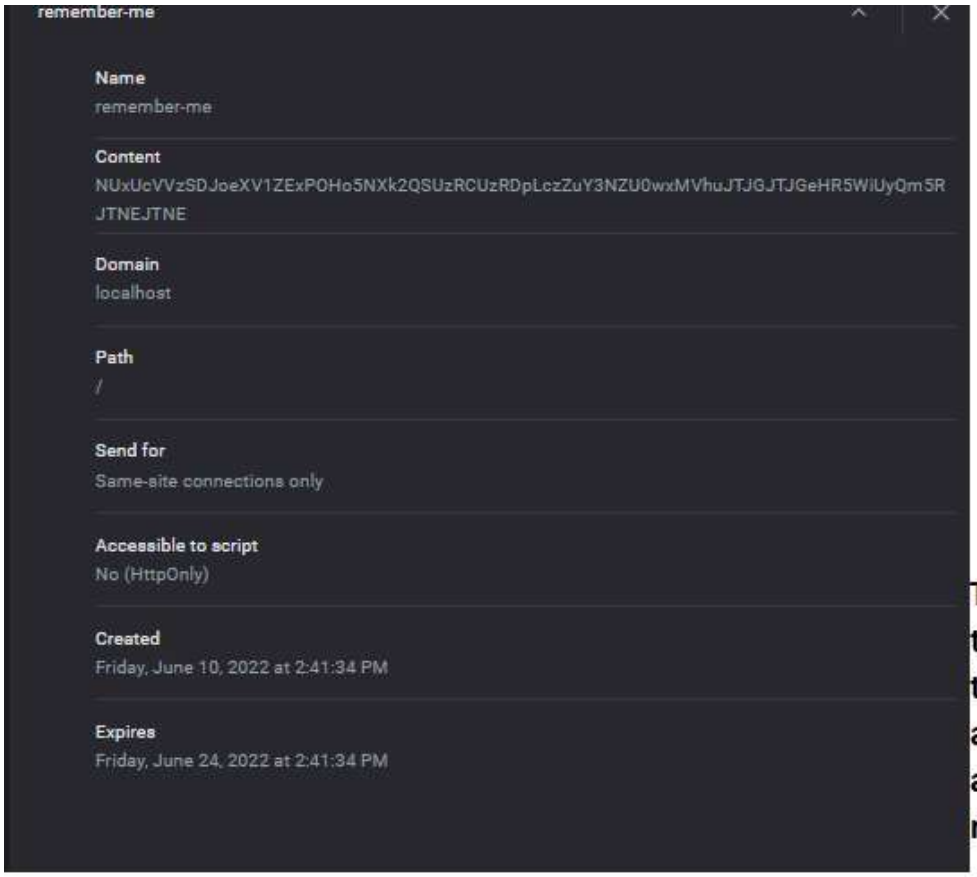


admin/password@123 has to be given as credentials. As it is the user available in the user's table and that password is kept in an encoded way. As the remember me option is selected, in the database, we can see an entry under 'persistent_logins'



At the same time, we can check the same under cookies as well. As the chrome browser is used, let us check that via chrome browser settings options





Till the expiry time, this cookie is available and it will be remembered in

As of now, the credentials are stored in cookies and also in DB due to the persistence mechanism, after we close the browser and open it again, we can able to see the index page when we hit **http://localhost:8080**. This is the advantage of remember me option. This will be possible till the expiry time. Moreover cookies data should not be cleared. Here for our example, till June 24, the cookies are not cleared, we can see the 'Remember Me' functionality enabled.