**Spring Boot – Adding Pagination And Sorting In Home Page Continues**

**Updated SeedData.java**

package org.studyeasy.SpringBlog.config;

import java.time.LocalDate;

import java.util.HashSet;

import java.util.List;

import java.util.Set;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.stereotype.Component;

import org.studyeasy.SpringBlog.models.Account;

import org.studyeasy.SpringBlog.models.Authority;

import org.studyeasy.SpringBlog.models.Post;

import org.studyeasy.SpringBlog.services.AccountService;

import org.studyeasy.SpringBlog.services.AuthorityService;

import org.studyeasy.SpringBlog.services.PostService;

import org.studyeasy.SpringBlog.util.constants.Privillages;

import org.studyeasy.SpringBlog.util.constants.Roles;

@Component

public class SeedData implements CommandLineRunner{

    @Autowired

    private PostService postService;

    @Autowired

    private AccountService accountService;

    @Autowired

    private AuthorityService authorityService;

    @Override

    public void run(String... args) throws Exception {

       for(Privillages auth: Privillages.values()){

            Authority authority = new Authority();

            authority.setId(auth.getId());

            authority.setName(auth.getPrivillage());

            authorityService.save(authority);

       }

       Account account01 = new Account();

       Account account02 = new Account();

       Account account03 = new Account();

       Account account04 = new Account();

       account01.setEmail("user@user.com");

       account01.setPassword("pass987");

       account01.setFirstname("User");

       account01.setLastname("lastname");

       account01.setAge(25);

       account01.setDate\_of\_birth(LocalDate.parse("1990-01-01"));

       account01.setGender("Male");

       account02.setEmail("admin@studyeasy.org");

       account02.setPassword("pass987");

       account02.setFirstname("Admin");

       account02.setLastname("lastname");

       account02.setRole(Roles.ADMIN.getRole());

       account02.setAge(25);

       account02.setDate\_of\_birth(LocalDate.parse("1990-01-01"));

       account02.setGender("Famale");

       account03.setEmail("editor@editor.com");

       account03.setPassword("pass987");

       account03.setFirstname("Editor");

       account03.setLastname("lastname");

       account03.setRole(Roles.EDITOR.getRole());

       account03.setAge(55);

       account03.setDate\_of\_birth(LocalDate.parse("1975-01-01"));

       account03.setGender("Male");

       account04.setEmail("super\_editor@editor.com");

       account04.setPassword("pass987");

       account04.setFirstname("Editor");

       account04.setLastname("lastname");

       account04.setRole(Roles.EDITOR.getRole());

       account04.setAge(40);

       account04.setDate\_of\_birth(LocalDate.parse("1980-01-01"));

       account04.setGender("Female");

       Set<Authority> authorities = new HashSet<>();

       authorityService.findById(Privillages.ACCESS\_ADMIN\_PANEL.getId()).ifPresent(authorities::add);

       authorityService.findById(Privillages.RESET\_ANY\_USER\_PASSWORD.getId()).ifPresent(authorities::add);

       account04.setAuthorities(authorities);

       accountService.save(account01);

       accountService.save(account02);

       accountService.save(account03);

       accountService.save(account04);

       List<Post> posts = postService.findAll();

       if (posts.size() == 0){

            Post post01 = new Post();

            post01.setTitle("About Git");

            post01.setBody("""

               Git (/ɡɪt/)[8] is a distributed version control system: tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development. Its goals include speed, data integrity, and support for distributed, non-linear workflows (thousands of parallel branches running on different systems).[9][10][11]

               Git was originally authored by Linus Torvalds in 2005 for development of the Linux kernel, with other kernel developers contributing to its initial development.[12] Since 2005, Junio Hamano has been the core maintainer. As with most other distributed version control systems, and unlike most client–server systems, every Git directory on every computer is a full-fledged repository with complete history and full version-tracking abilities, independent of network access or a central server.[13] Git is free and open-source software distributed under the GPL-2.0-only license.

            """);

            post01.setAccount(account01);

            postService.save(post01);

            Post post02 = new Post();

            post02.setTitle("Spring Boot Model–view–controller framework");

            post02.setBody("""

              <h3><strong>Model–view–controller framework</strong></h3>

              <p><a href="https://en.wikipedia.org/wiki/File:Spring5JuergenHoeller2.jpg"><img src="https://upload.wikimedia.org/wikipedia/commons/thumb/7/7b/Spring5JuergenHoeller2.jpg/220px-Spring5JuergenHoeller2.jpg" alt="" srcset="//upload.wikimedia.org/wikipedia/commons/thumb/7/7b/Spring5JuergenHoeller2.jpg/330px-Spring5JuergenHoeller2.jpg 1.5x, //upload.wikimedia.org/wikipedia/commons/thumb/7/7b/Spring5JuergenHoeller2.jpg/440px-Spring5JuergenHoeller2.jpg 2x" sizes="100vw" width="220"></a></p><p>&nbsp;</p><p>Spring MVC/Web Reactive presentation given by Jürgen Höller</p><p>The Spring Framework features its own <a href="https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller">model–view–controller</a> (MVC) <a href="https://en.wikipedia.org/wiki/Web\_application\_framework">web application framework</a>, which was not originally planned. The Spring developers decided to write their own Web framework as a reaction to what they perceived as the poor design of the (then) popular <a href="https://en.wikipedia.org/wiki/Jakarta\_Struts">Jakarta Struts</a> Web framework,<a href="https://en.wikipedia.org/wiki/Spring\_Framework#cite\_note-21">[21]</a> as well as deficiencies in other available frameworks. In particular, they felt there was insufficient separation between the presentation and request handling layers, and between the request handling layer and the model.<a href="https://en.wikipedia.org/wiki/Spring\_Framework#cite\_note-22">[22]</a></p><p>Like Struts, Spring MVC is a request-based framework. The framework defines <a href="https://en.wikipedia.org/wiki/Strategy\_pattern">strategy</a> interfaces for all of the responsibilities that must be handled by a modern request-based framework. The goal of each interface is to be simple and clear so that it's easy for Spring MVC users to write their own implementations, if they so choose. MVC paves the way for cleaner front end code. All interfaces are tightly coupled to the <a href="https://en.wikipedia.org/wiki/Java\_Servlet">Servlet API</a>. This tight coupling to the Servlet API is seen by some as a failure on the part of the Spring developers to offer a high-level abstraction for Web-based applications[<a href="https://en.wikipedia.org/wiki/Wikipedia:Citation\_needed"><i>citation needed</i></a>]. However, this coupling makes sure that the features of the Servlet API remain available to developers while also offering a high abstraction framework to ease working with it.</p><p>The DispatcherServlet class is the <a href="https://en.wikipedia.org/wiki/Front\_controller">front controller</a><a href="https://en.wikipedia.org/wiki/Spring\_Framework#cite\_note-23">[23]</a> of the framework and is responsible for delegating control to the various interfaces during the execution phases of an <a href="https://en.wikipedia.org/wiki/Hypertext\_Transfer\_Protocol">HTTP request</a>.</p><p>The most important interfaces defined by Spring MVC, and their responsibilities, are listed below:</p><ul><li>Controller: comes between Model and View to manage incoming requests and redirect to proper response. Controller will map the http request to corresponding methods. It acts as a gate that directs the incoming information. It switches between going into model or view.</li><li>HandlerAdapter: execution of objects that handle incoming requests</li><li>HandlerInterceptor: interception of incoming requests comparable, but not equal to Servlet filters (use is optional and not controlled by DispatcherServlet).</li><li>HandlerMapping: selecting objects that handle incoming requests (handlers) based on any attribute or condition internal or external to those requests</li><li>LocaleResolver: resolving and optionally saving of the <a href="https://en.wikipedia.org/wiki/Locale\_(computer\_software)">locale</a> of an individual user</li><li>MultipartResolver: facilitate working with file uploads by wrapping incoming requests</li><li>View: responsible for returning a response to the client. Some requests may go straight to view without going to the model part; others may go through all three.</li><li>ViewResolver: selecting a View based on a logical name for the view (use is not strictly required)</li></ul><p>Each strategy interface above has an important responsibility in the overall framework. The abstractions offered by these interfaces are powerful, so to allow for a set of variations in their implementations, Spring MVC ships with implementations of all these interfaces and together offers a feature set on top of the Servlet API. However, developers and vendors are free to write other implementations. Spring MVC uses the Java java.util.Map interface as a data-oriented abstraction for the Model where keys are expected to be string values.</p><p>The ease of testing the implementations of these interfaces seems one important advantage of the high level of abstraction offered by Spring MVC. DispatcherServlet is tightly coupled to the Spring inversion of control container for configuring the web layers of applications. However, web applications can use other parts of the Spring Framework—including the container—and choose not to use Spring MVC.</p>

            """);

            post02.setAccount(account02);

            postService.save(post02);

            Post post03 = new Post();

            post03.setTitle("third post");

            post03.setBody("""

               Git (/ɡɪt/)[8] is a distributed version control system: tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development. Its goals include speed, data integrity, and support for distributed, non-linear workflows (thousands of parallel branches running on different systems).[9][10][11]

               Git was originally authored by Linus Torvalds in 2005 for development of the Linux kernel, with other kernel developers contributing to its initial development.[12] Since 2005, Junio Hamano has been the core maintainer. As with most other distributed version control systems, and unlike most client–server systems, every Git directory on every computer is a full-fledged repository with complete history and full version-tracking abilities, independent of network access or a central server.[13] Git is free and open-source software distributed under the GPL-2.0-only license.

            """);

            post03.setAccount(account01);

            postService.save(post03);

            Post post04 = new Post();

            post04.setTitle("Fouth post");

            post04.setBody("""

              <h3><strong>Model–view–controller framework</strong></h3>

              <p><a href="https://en.wikipedia.org/wiki/File:Spring5JuergenHoeller2.jpg"><img src="https://upload.wikimedia.org/wikipedia/commons/thumb/7/7b/Spring5JuergenHoeller2.jpg/220px-Spring5JuergenHoeller2.jpg" alt="" srcset="//upload.wikimedia.org/wikipedia/commons/thumb/7/7b/Spring5JuergenHoeller2.jpg/330px-Spring5JuergenHoeller2.jpg 1.5x, //upload.wikimedia.org/wikipedia/commons/thumb/7/7b/Spring5JuergenHoeller2.jpg/440px-Spring5JuergenHoeller2.jpg 2x" sizes="100vw" width="220"></a></p><p>&nbsp;</p><p>Spring MVC/Web Reactive presentation given by Jürgen Höller</p><p>The Spring Framework features its own <a href="https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller">model–view–controller</a> (MVC) <a href="https://en.wikipedia.org/wiki/Web\_application\_framework">web application framework</a>, which was not originally planned. The Spring developers decided to write their own Web framework as a reaction to what they perceived as the poor design of the (then) popular <a href="https://en.wikipedia.org/wiki/Jakarta\_Struts">Jakarta Struts</a> Web framework,<a href="https://en.wikipedia.org/wiki/Spring\_Framework#cite\_note-21">[21]</a> as well as deficiencies in other available frameworks. In particular, they felt there was insufficient separation between the presentation and request handling layers, and between the request handling layer and the model.<a href="https://en.wikipedia.org/wiki/Spring\_Framework#cite\_note-22">[22]</a></p><p>Like Struts, Spring MVC is a request-based framework. The framework defines <a href="https://en.wikipedia.org/wiki/Strategy\_pattern">strategy</a> interfaces for all of the responsibilities that must be handled by a modern request-based framework. The goal of each interface is to be simple and clear so that it's easy for Spring MVC users to write their own implementations, if they so choose. MVC paves the way for cleaner front end code. All interfaces are tightly coupled to the <a href="https://en.wikipedia.org/wiki/Java\_Servlet">Servlet API</a>. This tight coupling to the Servlet API is seen by some as a failure on the part of the Spring developers to offer a high-level abstraction for Web-based applications[<a href="https://en.wikipedia.org/wiki/Wikipedia:Citation\_needed"><i>citation needed</i></a>]. However, this coupling makes sure that the features of the Servlet API remain available to developers while also offering a high abstraction framework to ease working with it.</p><p>The DispatcherServlet class is the <a href="https://en.wikipedia.org/wiki/Front\_controller">front controller</a><a href="https://en.wikipedia.org/wiki/Spring\_Framework#cite\_note-23">[23]</a> of the framework and is responsible for delegating control to the various interfaces during the execution phases of an <a href="https://en.wikipedia.org/wiki/Hypertext\_Transfer\_Protocol">HTTP request</a>.</p><p>The most important interfaces defined by Spring MVC, and their responsibilities, are listed below:</p><ul><li>Controller: comes between Model and View to manage incoming requests and redirect to proper response. Controller will map the http request to corresponding methods. It acts as a gate that directs the incoming information. It switches between going into model or view.</li><li>HandlerAdapter: execution of objects that handle incoming requests</li><li>HandlerInterceptor: interception of incoming requests comparable, but not equal to Servlet filters (use is optional and not controlled by DispatcherServlet).</li><li>HandlerMapping: selecting objects that handle incoming requests (handlers) based on any attribute or condition internal or external to those requests</li><li>LocaleResolver: resolving and optionally saving of the <a href="https://en.wikipedia.org/wiki/Locale\_(computer\_software)">locale</a> of an individual user</li><li>MultipartResolver: facilitate working with file uploads by wrapping incoming requests</li><li>View: responsible for returning a response to the client. Some requests may go straight to view without going to the model part; others may go through all three.</li><li>ViewResolver: selecting a View based on a logical name for the view (use is not strictly required)</li></ul><p>Each strategy interface above has an important responsibility in the overall framework. The abstractions offered by these interfaces are powerful, so to allow for a set of variations in their implementations, Spring MVC ships with implementations of all these interfaces and together offers a feature set on top of the Servlet API. However, developers and vendors are free to write other implementations. Spring MVC uses the Java java.util.Map interface as a data-oriented abstraction for the Model where keys are expected to be string values.</p><p>The ease of testing the implementations of these interfaces seems one important advantage of the high level of abstraction offered by Spring MVC. DispatcherServlet is tightly coupled to the Spring inversion of control container for configuring the web layers of applications. However, web applications can use other parts of the Spring Framework—including the container—and choose not to use Spring MVC.</p>

            """);

            post04.setAccount(account02);

            postService.save(post04);

            Post post05 = new Post();

            post05.setTitle("Fifth post");

            post05.setBody("""

               Git (/ɡɪt/)[8] is a distributed version control system: tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development. Its goals include speed, data integrity, and support for distributed, non-linear workflows (thousands of parallel branches running on different systems).[9][10][11]

               Git was originally authored by Linus Torvalds in 2005 for development of the Linux kernel, with other kernel developers contributing to its initial development.[12] Since 2005, Junio Hamano has been the core maintainer. As with most other distributed version control systems, and unlike most client–server systems, every Git directory on every computer is a full-fledged repository with complete history and full version-tracking abilities, independent of network access or a central server.[13] Git is free and open-source software distributed under the GPL-2.0-only license.

            """);

            post05.setAccount(account01);

            postService.save(post05);

            Post post06 = new Post();

            post06.setTitle("Sixth post");

            post06.setBody("""

              <h3><strong>Model–view–controller framework</strong></h3>

              <p><a href="https://en.wikipedia.org/wiki/File:Spring5JuergenHoeller2.jpg"><img src="https://upload.wikimedia.org/wikipedia/commons/thumb/7/7b/Spring5JuergenHoeller2.jpg/220px-Spring5JuergenHoeller2.jpg" alt="" srcset="//upload.wikimedia.org/wikipedia/commons/thumb/7/7b/Spring5JuergenHoeller2.jpg/330px-Spring5JuergenHoeller2.jpg 1.5x, //upload.wikimedia.org/wikipedia/commons/thumb/7/7b/Spring5JuergenHoeller2.jpg/440px-Spring5JuergenHoeller2.jpg 2x" sizes="100vw" width="220"></a></p><p>&nbsp;</p><p>Spring MVC/Web Reactive presentation given by Jürgen Höller</p><p>The Spring Framework features its own <a href="https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller">model–view–controller</a> (MVC) <a href="https://en.wikipedia.org/wiki/Web\_application\_framework">web application framework</a>, which was not originally planned. The Spring developers decided to write their own Web framework as a reaction to what they perceived as the poor design of the (then) popular <a href="https://en.wikipedia.org/wiki/Jakarta\_Struts">Jakarta Struts</a> Web framework,<a href="https://en.wikipedia.org/wiki/Spring\_Framework#cite\_note-21">[21]</a> as well as deficiencies in other available frameworks. In particular, they felt there was insufficient separation between the presentation and request handling layers, and between the request handling layer and the model.<a href="https://en.wikipedia.org/wiki/Spring\_Framework#cite\_note-22">[22]</a></p><p>Like Struts, Spring MVC is a request-based framework. The framework defines <a href="https://en.wikipedia.org/wiki/Strategy\_pattern">strategy</a> interfaces for all of the responsibilities that must be handled by a modern request-based framework. The goal of each interface is to be simple and clear so that it's easy for Spring MVC users to write their own implementations, if they so choose. MVC paves the way for cleaner front end code. All interfaces are tightly coupled to the <a href="https://en.wikipedia.org/wiki/Java\_Servlet">Servlet API</a>. This tight coupling to the Servlet API is seen by some as a failure on the part of the Spring developers to offer a high-level abstraction for Web-based applications[<a href="https://en.wikipedia.org/wiki/Wikipedia:Citation\_needed"><i>citation needed</i></a>]. However, this coupling makes sure that the features of the Servlet API remain available to developers while also offering a high abstraction framework to ease working with it.</p><p>The DispatcherServlet class is the <a href="https://en.wikipedia.org/wiki/Front\_controller">front controller</a><a href="https://en.wikipedia.org/wiki/Spring\_Framework#cite\_note-23">[23]</a> of the framework and is responsible for delegating control to the various interfaces during the execution phases of an <a href="https://en.wikipedia.org/wiki/Hypertext\_Transfer\_Protocol">HTTP request</a>.</p><p>The most important interfaces defined by Spring MVC, and their responsibilities, are listed below:</p><ul><li>Controller: comes between Model and View to manage incoming requests and redirect to proper response. Controller will map the http request to corresponding methods. It acts as a gate that directs the incoming information. It switches between going into model or view.</li><li>HandlerAdapter: execution of objects that handle incoming requests</li><li>HandlerInterceptor: interception of incoming requests comparable, but not equal to Servlet filters (use is optional and not controlled by DispatcherServlet).</li><li>HandlerMapping: selecting objects that handle incoming requests (handlers) based on any attribute or condition internal or external to those requests</li><li>LocaleResolver: resolving and optionally saving of the <a href="https://en.wikipedia.org/wiki/Locale\_(computer\_software)">locale</a> of an individual user</li><li>MultipartResolver: facilitate working with file uploads by wrapping incoming requests</li><li>View: responsible for returning a response to the client. Some requests may go straight to view without going to the model part; others may go through all three.</li><li>ViewResolver: selecting a View based on a logical name for the view (use is not strictly required)</li></ul><p>Each strategy interface above has an important responsibility in the overall framework. The abstractions offered by these interfaces are powerful, so to allow for a set of variations in their implementations, Spring MVC ships with implementations of all these interfaces and together offers a feature set on top of the Servlet API. However, developers and vendors are free to write other implementations. Spring MVC uses the Java java.util.Map interface as a data-oriented abstraction for the Model where keys are expected to be string values.</p><p>The ease of testing the implementations of these interfaces seems one important advantage of the high level of abstraction offered by Spring MVC. DispatcherServlet is tightly coupled to the Spring inversion of control container for configuring the web layers of applications. However, web applications can use other parts of the Spring Framework—including the container—and choose not to use Spring MVC.</p>

            """);

            post06.setAccount(account02);

            postService.save(post06);

       }

    }

}

**In PostService.java**

public List<Post> getAll(){

        return postRepository.findAll();

    }

**Change to**

public List<Post> findAll(){

        return postRepository.findAll();

    }

**In HomeController.java**

@GetMapping("/")

    public String home(Model model){

        List<Post> posts = postService.getAll();

        model.addAttribute("posts", posts);

        return "home\_views/home";

    }

**Change to**

@GetMapping("/")

    public String home(Model model){

        List<Post> posts = postService.findAll();

        model.addAttribute("posts", posts);

        return "home\_views/home";

    }

**In home.html -> Modify code -> add below code**

<!-- about section - added code - navbar -->

  <form action="/" , method="get">

    <nav class="navbar navbar-expand-lg navbar-light bg-light">

      <span class="navbar-brand">Sort by: </span>

      <ul class="navbar-nav">

        <li class="nav-item active" style="padding-right: 20px;">

          <div class="dropdown">

            <select class="form-select" aria-label="Default select example" name="sort\_by">

              <option value="createdAt">Created date</option>

              <option value="updatedAt">Updated date</option>

            </select>

          </div>

        </li>

        <span class="navbar-brand">Per page: </span>

        <ul class="navbar-nav">

          <li class="nav-item active" style="padding-right: 20px;">

            <div class="dropdown">

              <select class="form-select" aria-label="Default select example" name="per\_page">

                <option selected value="2">2</option>

                <option value="5">5</option>

                <option value="10">10</option>

                <option value="15">15</option>

              </select>

            </div>

          </li>

          <li class="nav-item active" style="padding-left: 20px;">

            <div class="input-group-prepend">

              <button class="btn btn-primary" type="submit" id="dropdownMenuButton">

                Apply filter(s)

              </button>

            </div>

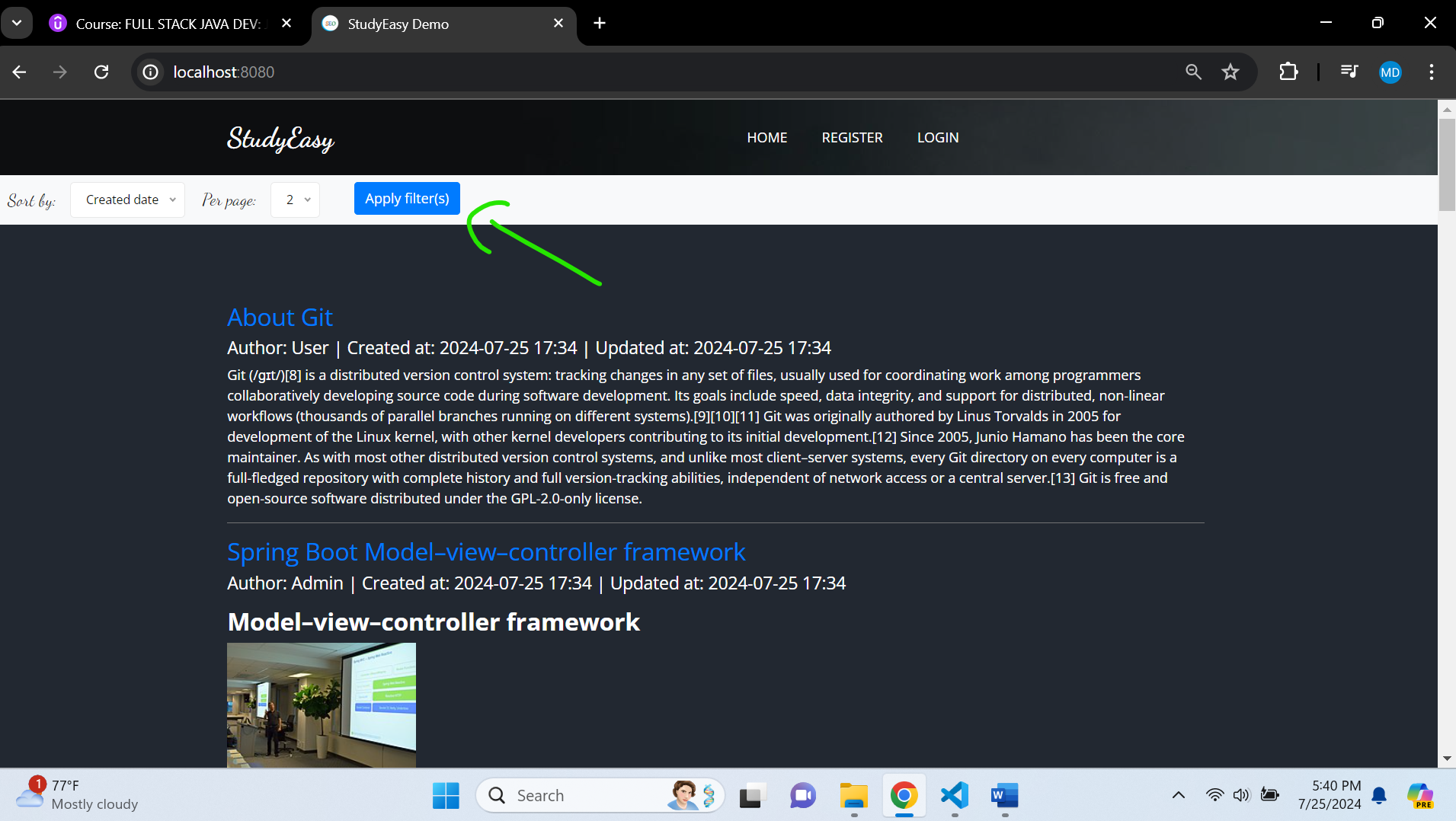
          </li>

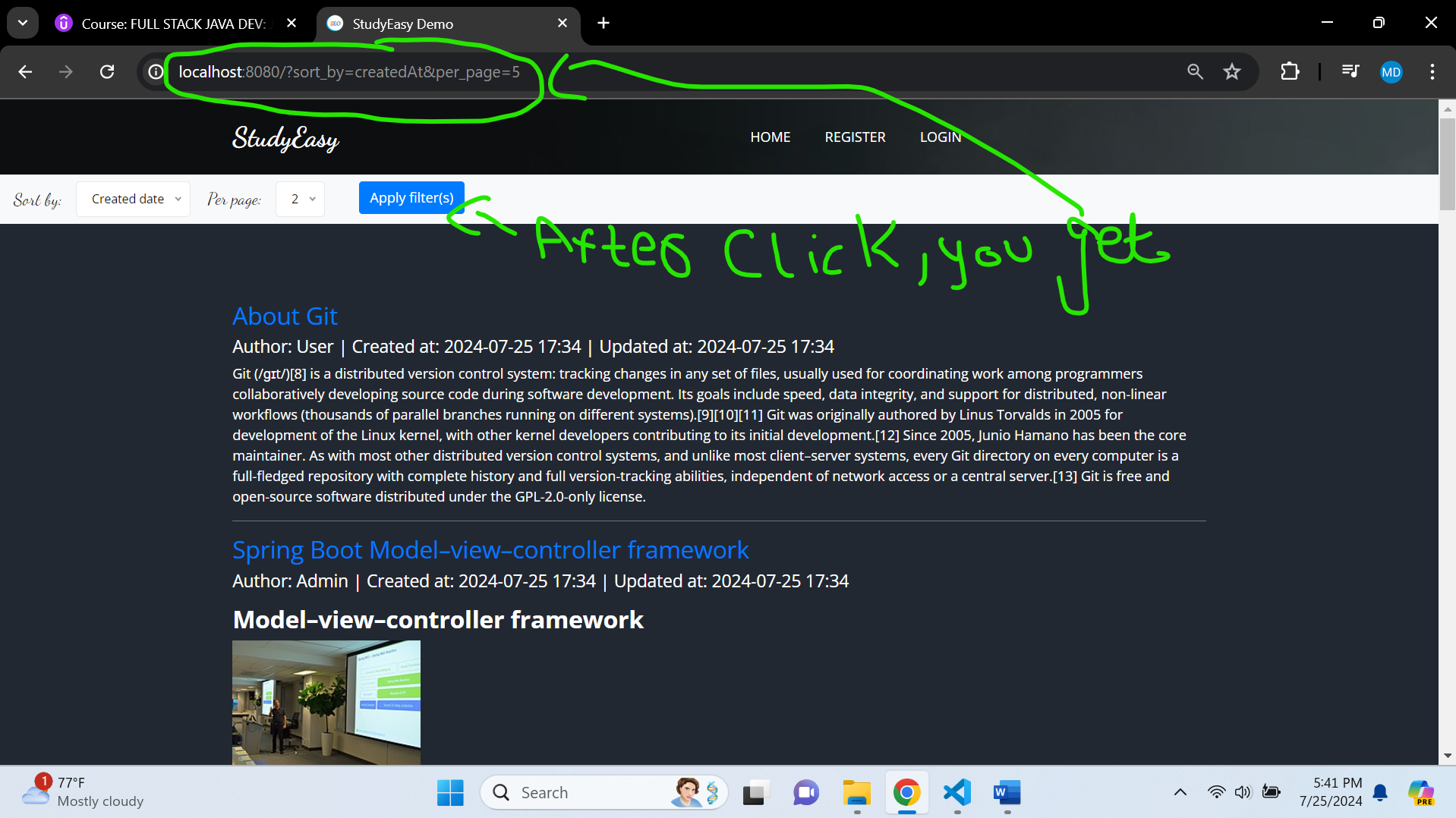
        </ul>

    </nav>

  </form>

**Output**

****

****

**Overloading findAll() to get data for the paginated data in PostService.java**

Offset – It means which page we are in.

PageSize – How many records there should be in particular page.

 //Paginated Page Service

 public Page<Post> findAll(int offset, int pageSize, String field){

   return postRepository.findAll(PageRequest.of(offset, pageSize).withSort(Direction.ASC, field));

    }

-------------------------------------------------------XXXX--------------------------------------------

Updated HomeController -> home()

  @GetMapping("/")

    public String home(Model model,  @RequestParam(required = false, name = "sort\_by", defaultValue = "createdAt") String sort\_by,

      @RequestParam(required = false, name="per\_page", defaultValue = "2") String per\_page,

      @RequestParam(required = false, name="page", defaultValue = "1") String page

    ) {

        Page<Post> posts\_on\_page = postService.findAll(Integer.parseInt(page)-1, Integer.parseInt(per\_page), sort\_by);

        int total\_pages = posts\_on\_page.getTotalPages();

        List<Integer> pages = new ArrayList<>();

        if (total\_pages > 0){

            pages = IntStream.rangeClosed(0, total\_pages-1)

            .boxed().collect(Collectors.toList());

        }

        List<String> links = new ArrayList<>();

        if(pages != null){

            for(int link : pages){

                String active = "";

                if(link == posts\_on\_page.getNumber()){

                    active = "active";

                }

                String \_temp\_link = "/?per\_page="+per\_page+"&page="+(link+1)+"&sort\_by="+sort\_by;

                links.add("<li class=\"page-item "+active+"\"><a href=\""+\_temp\_link+"\" class='page-link'>"+(link+1)+"</a></li>");

            }

            model.addAttribute("links", links);

        }

        model.addAttribute("posts", posts\_on\_page);

        return "home\_views/home";

    }

Add this code in home.html

   <!--Pagination Code-->

    <nav th:if="${links ne null}" aria-label="...">

      <ul class="pagination justify-content-center">

        <th:block th:each="link: ${links}">

          <li class="page-item">

            <th:block th:utext="${link}">link</th:block>

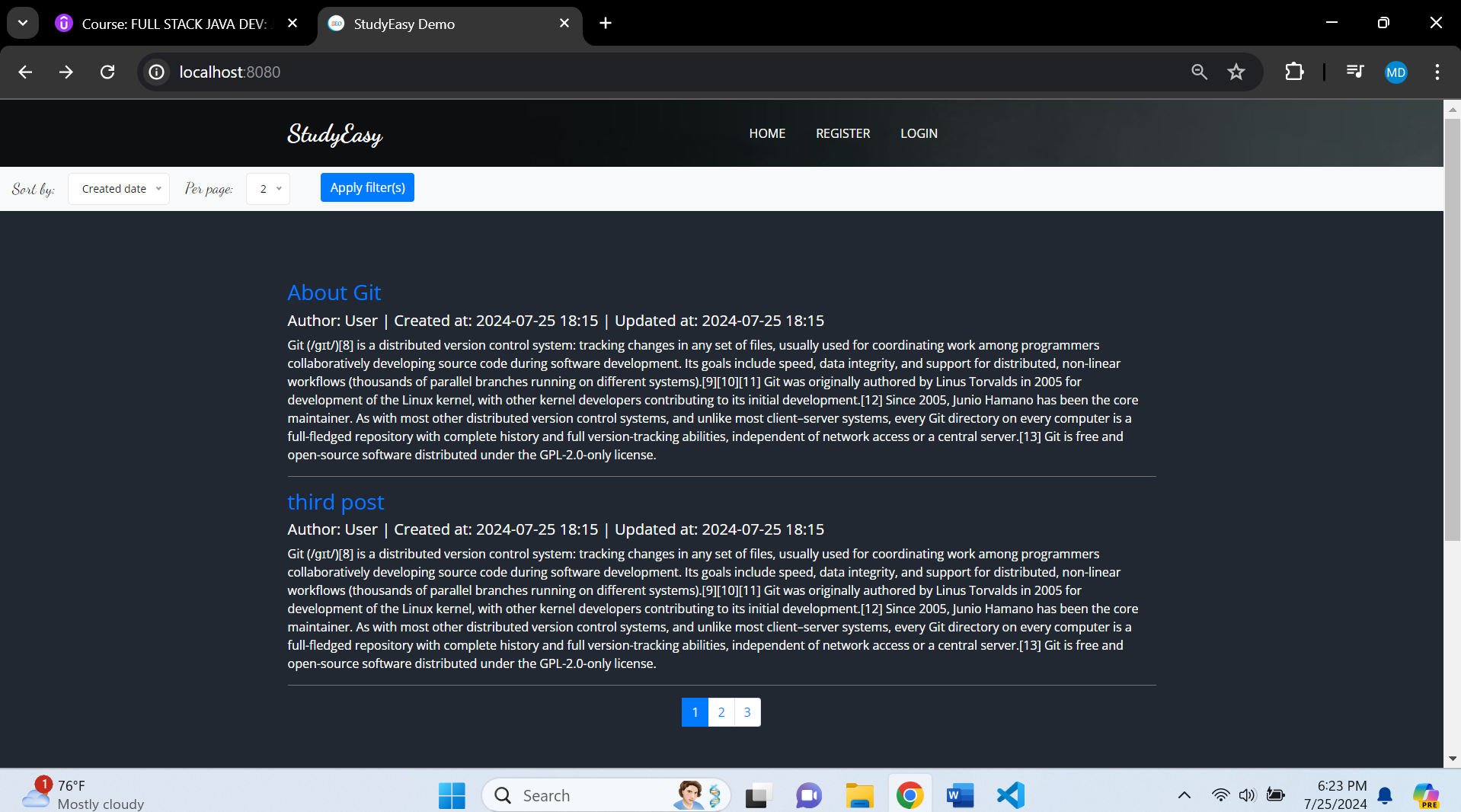
          </li>

        </th:block>

      </ul>

    </nav>

Output



Status : working ; Output :success