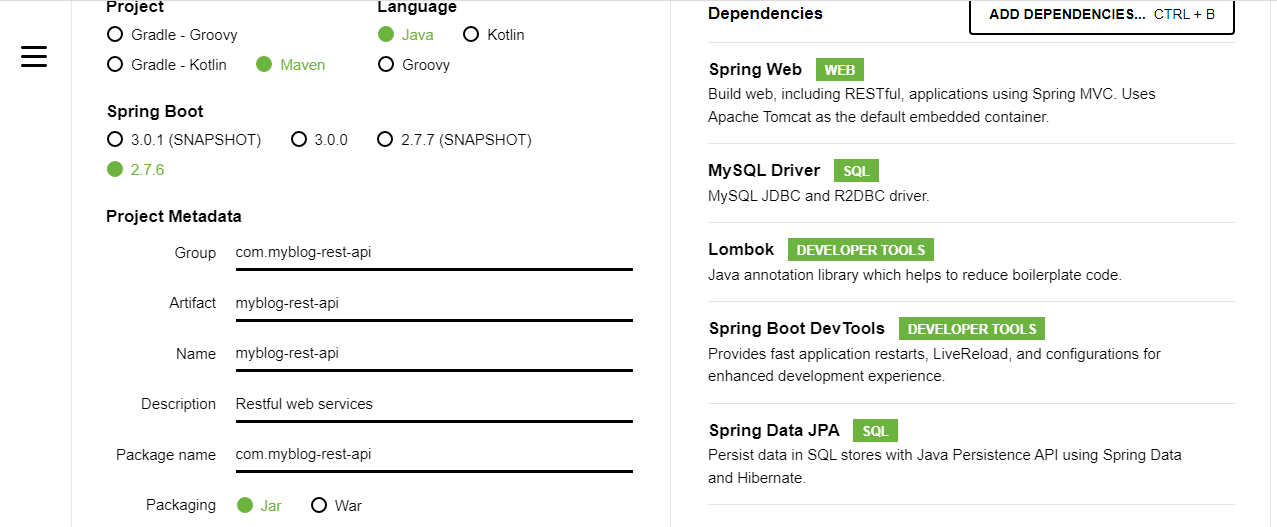
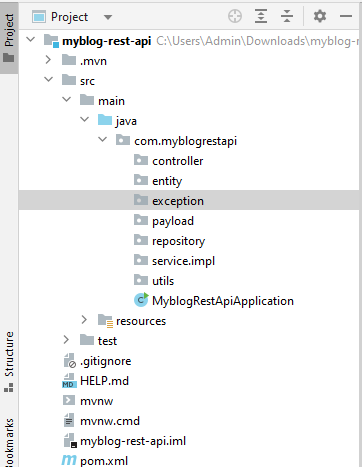
**Developing restful web services in spring boot**

1. **Create Spring boot project with following dependencies:**



1. Create Following Project Structure in IntelliJ Idea



Step 3: Create POST Entity Class

import lombok.AllArgsConstructor;

import lombok.Data;

import lombok.NoArgsConstructor;

import javax.persistence.\*;

@Data

@AllArgsConstructor

@NoArgsConstructor

@Entity

@Table

(

name = "posts", uniqueConstraints = {@UniqueConstraint(columnNames = {"title"})}

)

public class Post {

@Id

@GeneratedValue( strategy = GenerationType.IDENTITY) )

private Long id;

@Column(name = "title", nullable = false)

private String title;

@Column(name = "description", nullable = false)

private String description;

@Column(name = "content", nullable = false)

private String content;

}

Step 3: Update application.properties file

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

spring.datasource.username = root

spring.datasource.password = root

# hibernate properties

spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5InnoDBDialect

# Hibernate ddl auto

spring.jpa.hibernate.ddl-auto = update

Step 4: Create Post Repository Layer:

import org.springframework.data.jpa.repository.JpaRepository;

public interface PostRepository extends JpaRepository<Post, Long> {

}

Step 5: Create Payload PostDto class

import lombok.Data;

@Data

public class PostDto {

private long id;

private String title;

private String description;

private String content;

}

Step 6: Create PostService Interface

import java.util.List;

public interface PostService {

PostDto createPost(PostDto postDto);

}

Step 7: Create PostServiceImpl class

@Service

public class PostServiceImpl implements PostService {

private PostRepository postRepository;

public PostServiceImpl(PostRepository postRepository) {

this.postRepository = postRepository;

}

@Override

public PostDto createPost(PostDto postDto) {

// convert DTO to entity

Post post = mapToEntity(postDto);

Post newPost = postRepository.save(post);

// convert entity to DTO

PostDto postResponse = mapToDTO(newPost);

return postResponse;

}

// convert Entity into DTO

private PostDto mapToDTO(Post post){

PostDto postDto = new PostDto();

postDto.setId(post.getId());

postDto.setTitle(post.getTitle());

postDto.setDescription(post.getDescription());

postDto.setContent(post.getContent());

return postDto;

}

// convert DTO to entity

private Post mapToEntity(PostDto postDto){

Post post = new Post();

post.setTitle(postDto.getTitle());

post.setDescription(postDto.getDescription());

post.setContent(postDto.getContent());

return post;

}

}

Step 8: Create PostController Class:

@RestController

@RequestMapping("/api/posts")

public class PostController {

private PostService postService;

public PostController(PostService postService) {

this.postService = postService;

}

// create blog post rest api

@PostMapping

public ResponseEntity<PostDto> createPost(@RequestBody PostDto postDto){

return new ResponseEntity<>(postService.createPost(postDto), HttpStatus.CREATED);

}

}

Step 9: Create Exception class

import org.springframework.http.HttpStatus;

import org.springframework.web.bind.annotation.ResponseStatus;

@ResponseStatus(value = HttpStatus.NOT\_FOUND)

public class ResourceNotFoundException extends RuntimeException{

private String resourceName;

private String fieldName;

private long fieldValue;

public ResourceNotFoundException(String resourceName, String fieldName, long fieldValue) {

super(String.format("%s not found with %s : '%s'", resourceName, fieldName, fieldValue)); // Post not found with id : 1

this.resourceName = resourceName;

this.fieldName = fieldName;

this.fieldValue = fieldValue;

}

public String getResourceName() {

return resourceName;

}

public String getFieldName() {

return fieldName;

}

public long getFieldValue() {

return fieldValue;

}

}

Step 10: Create GetMapping in controller layer:

import java.util.List;

@RestController

@RequestMapping("/api/posts")

public class PostController {

private PostService postService;

public PostController(PostService postService) {

this.postService = postService;

}

// create blog post rest api

@PostMapping

public ResponseEntity<PostDto> createPost(@RequestBody PostDto postDto){

return new ResponseEntity<>(postService.createPost(postDto), HttpStatus.CREATED);

}

// get all posts rest api

@GetMapping

public List<PostDto> getAllPosts(){

return postService.getAllPosts();

}

}

Step 11: Update PostService interface:

import com.springboot.blog.payload.PostDto;

import java.util.List;

public interface PostService {

PostDto createPost(PostDto postDto);

List<PostDto> getAllPosts();

}

Step 12: Update PostServiceImpl class:

import com.springboot.blog.entity.Post;

import com.springboot.blog.exception.ResourceNotFoundException;

import com.springboot.blog.payload.PostDto;

import com.springboot.blog.repository.PostRepository;

import com.springboot.blog.service.PostService;

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

import org.springframework.stereotype.Service;

import java.util.List;

import java.util.stream.Collectors;

@Service

public class PostServiceImpl implements PostService {

private PostRepository postRepository;

public PostServiceImpl(PostRepository postRepository) {

this.postRepository = postRepository;

}

@Override

public PostDto createPost(PostDto postDto) {

// convert DTO to entity

Post post = mapToEntity(postDto);

Post newPost = postRepository.save(post);

// convert entity to DTO

PostDto postResponse = mapToDTO(newPost);

return postResponse;

}

@Override

public List<PostDto> getAllPosts() {

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

return posts.stream().map(post -> mapToDTO(post)).collect(Collectors.toList());

}

// convert Entity into DTO

private PostDto mapToDTO(Post post){

PostDto postDto = new PostDto();

postDto.setId(post.getId());

postDto.setTitle(post.getTitle());

postDto.setDescription(post.getDescription());

postDto.setContent(post.getContent());

return postDto;

}

// convert DTO to entity

private Post mapToEntity(PostDto postDto){

Post post = new Post();

post.setTitle(postDto.getTitle());

post.setDescription(postDto.getDescription());

post.setContent(postDto.getContent());

return post;

}

}

Step 13: Create DeleteMapping By Id:

import com.springboot.blog.payload.PostDto;

import com.springboot.blog.service.PostService;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/api/posts")

public class PostController {

private PostService postService;

public PostController(PostService postService) {

this.postService = postService;

}

// create blog post rest api

@PostMapping

public ResponseEntity<PostDto> createPost(@RequestBody PostDto postDto){

return new ResponseEntity<>(postService.createPost(postDto), HttpStatus.CREATED);

}

// get all posts rest api

@GetMapping

public List<PostDto> getAllPosts(){

return postService.getAllPosts();

}

// get post by id

@GetMapping("/{id}")

public ResponseEntity<PostDto> getPostById(@PathVariable(name = "id") long id){

return ResponseEntity.ok(postService.getPostById(id));

}

}

Step 14: Update PostServiceImpl interface:

import com.springboot.blog.payload.PostDto;

import java.util.List;

public interface PostService {

PostDto createPost(PostDto postDto);

List<PostDto> getAllPosts();

PostDto getPostById(long id);

}

Step 15: Update PostServiceImpl class

import com.springboot.blog.entity.Post;

import com.springboot.blog.exception.ResourceNotFoundException;

import com.springboot.blog.payload.PostDto;

import com.springboot.blog.repository.PostRepository;

import com.springboot.blog.service.PostService;

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

import org.springframework.stereotype.Service;

import java.util.List;

import java.util.stream.Collectors;

@Service

public class PostServiceImpl implements PostService {

private PostRepository postRepository;

public PostServiceImpl(PostRepository postRepository) {

this.postRepository = postRepository;

}

@Override

public PostDto createPost(PostDto postDto) {

// convert DTO to entity

Post post = mapToEntity(postDto);

Post newPost = postRepository.save(post);

// convert entity to DTO

PostDto postResponse = mapToDTO(newPost);

return postResponse;

}

@Override

public List<PostDto> getAllPosts() {

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

return posts.stream().map(post -> mapToDTO(post)).collect(Collectors.toList());

}

@Override

public PostDto getPostById(long id) {

Post post = postRepository.findById(id).orElseThrow(() -> new ResourceNotFoundException("Post", "id", id));

return mapToDTO(post);

}

// convert Entity into DTO

private PostDto mapToDTO(Post post){

PostDto postDto = new PostDto();

postDto.setId(post.getId());

postDto.setTitle(post.getTitle());

postDto.setDescription(post.getDescription());

postDto.setContent(post.getContent());

return postDto;

}

// convert DTO to entity

private Post mapToEntity(PostDto postDto){

Post post = new Post();

post.setTitle(postDto.getTitle());

post.setDescription(postDto.getDescription());

post.setContent(postDto.getContent());

return post;

}

}

Step 16: Create UpdateMapping Controller

import com.springboot.blog.payload.PostDto;

import com.springboot.blog.service.PostService;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/api/posts")

public class PostController {

private PostService postService;

public PostController(PostService postService) {

this.postService = postService;

}

// create blog post rest api

@PostMapping

public ResponseEntity<PostDto> createPost(@RequestBody PostDto postDto){

return new ResponseEntity<>(postService.createPost(postDto), HttpStatus.CREATED);

}

// get all posts rest api

@GetMapping

public List<PostDto> getAllPosts(){

return postService.getAllPosts();

}

// get post by id

@GetMapping("/{id}")

public ResponseEntity<PostDto> getPostById(@PathVariable(name = "id") long id){

return ResponseEntity.ok(postService.getPostById(id));

}

// update post by id rest api

@PutMapping("/{id}")

public ResponseEntity<PostDto> updatePost(@RequestBody PostDto postDto, @PathVariable(name = "id") long id){

PostDto postResponse = postService.updatePost(postDto, id);

return new ResponseEntity<>(postResponse, HttpStatus.OK);

}

}

Step 17: Update PostService Interface:

import com.springboot.blog.payload.PostDto;

import java.util.List;

public interface PostService {

PostDto createPost(PostDto postDto);

List<PostDto> getAllPosts();

PostDto getPostById(long id);

PostDto updatePost(PostDto postDto, long id);

}

Step 18: Update PostServiceImpl class:

import com.springboot.blog.entity.Post;

import com.springboot.blog.exception.ResourceNotFoundException;

import com.springboot.blog.payload.PostDto;

import com.springboot.blog.repository.PostRepository;

import com.springboot.blog.service.PostService;

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

import org.springframework.stereotype.Service;

import java.util.List;

import java.util.stream.Collectors;

@Service

public class PostServiceImpl implements PostService {

private PostRepository postRepository;

public PostServiceImpl(PostRepository postRepository) {

this.postRepository = postRepository;

}

@Override

public PostDto createPost(PostDto postDto) {

// convert DTO to entity

Post post = mapToEntity(postDto);

Post newPost = postRepository.save(post);

// convert entity to DTO

PostDto postResponse = mapToDTO(newPost);

return postResponse;

}

@Override

public List<PostDto> getAllPosts() {

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

return posts.stream().map(post -> mapToDTO(post)).collect(Collectors.toList());

}

@Override

public PostDto getPostById(long id) {

Post post = postRepository.findById(id).orElseThrow(() -> new ResourceNotFoundException("Post", "id", id));

return mapToDTO(post);

}

@Override

public PostDto updatePost(PostDto postDto, long id) {

// get post by id from the database

Post post = postRepository.findById(id).orElseThrow(() -> new ResourceNotFoundException("Post", "id", id));

post.setTitle(postDto.getTitle());

post.setDescription(postDto.getDescription());

post.setContent(postDto.getContent());

Post updatedPost = postRepository.save(post);

return mapToDTO(updatedPost);

}

// convert Entity into DTO

private PostDto mapToDTO(Post post){

PostDto postDto = new PostDto();

postDto.setId(post.getId());

postDto.setTitle(post.getTitle());

postDto.setDescription(post.getDescription());

postDto.setContent(post.getContent());

return postDto;

}

// convert DTO to entity

private Post mapToEntity(PostDto postDto){

Post post = new Post();

post.setTitle(postDto.getTitle());

post.setDescription(postDto.getDescription());

post.setContent(postDto.getContent());

return post;

}

}

Step 19: Create DeleteMapping controller:

import com.springboot.blog.payload.PostDto;

import com.springboot.blog.service.PostService;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/api/posts")

public class PostController {

private PostService postService;

public PostController(PostService postService) {

this.postService = postService;

}

// create blog post rest api

@PostMapping

public ResponseEntity<PostDto> createPost(@RequestBody PostDto postDto){

return new ResponseEntity<>(postService.createPost(postDto), HttpStatus.CREATED);

}

// get all posts rest api

@GetMapping

public List<PostDto> getAllPosts(){

return postService.getAllPosts();

}

// get post by id

@GetMapping("/{id}")

public ResponseEntity<PostDto> getPostById(@PathVariable(name = "id") long id){

return ResponseEntity.ok(postService.getPostById(id));

}

// update post by id rest api

@PutMapping("/{id}")

public ResponseEntity<PostDto> updatePost(@RequestBody PostDto postDto, @PathVariable(name = "id") long id){

PostDto postResponse = postService.updatePost(postDto, id);

return new ResponseEntity<>(postResponse, HttpStatus.OK);

}

// delete post rest api

@DeleteMapping("/{id}")

public ResponseEntity<String> deletePost(@PathVariable(name = "id") long id){

postService.deletePostById(id);

return new ResponseEntity<>("Post entity deleted successfully.", HttpStatus.OK);

}

}

Step 20: Update PostService Interface:

import com.springboot.blog.payload.PostDto;

import java.util.List;

public interface PostService {

PostDto createPost(PostDto postDto);

List<PostDto> getAllPosts();

PostDto getPostById(long id);

PostDto updatePost(PostDto postDto, long id);

void deletePostById(long id);

}

Step 21: Create PostServiceImpl class:

import com.springboot.blog.entity.Post;

import com.springboot.blog.exception.ResourceNotFoundException;

import com.springboot.blog.payload.PostDto;

import com.springboot.blog.repository.PostRepository;

import com.springboot.blog.service.PostService;

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

import org.springframework.stereotype.Service;

import java.util.List;

import java.util.stream.Collectors;

@Service

public class PostServiceImpl implements PostService {

private PostRepository postRepository;

public PostServiceImpl(PostRepository postRepository) {

this.postRepository = postRepository;

}

@Override

public PostDto createPost(PostDto postDto) {

// convert DTO to entity

Post post = mapToEntity(postDto);

Post newPost = postRepository.save(post);

// convert entity to DTO

PostDto postResponse = mapToDTO(newPost);

return postResponse;

}

@Override

public List<PostDto> getAllPosts() {

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

return posts.stream().map(post -> mapToDTO(post)).collect(Collectors.toList());

}

@Override

public PostDto getPostById(long id) {

Post post = postRepository.findById(id).orElseThrow(() -> new ResourceNotFoundException("Post", "id", id));

return mapToDTO(post);

}

@Override

public PostDto updatePost(PostDto postDto, long id) {

// get post by id from the database

Post post = postRepository.findById(id).orElseThrow(() -> new ResourceNotFoundException("Post", "id", id));

post.setTitle(postDto.getTitle());

post.setDescription(postDto.getDescription());

post.setContent(postDto.getContent());

Post updatedPost = postRepository.save(post);

return mapToDTO(updatedPost);

}

@Override

public void deletePostById(long id) {

// get post by id from the database

Post post = postRepository.findById(id).orElseThrow(() -> new ResourceNotFoundException("Post", "id", id));

postRepository.delete(post);

}

// convert Entity into DTO

private PostDto mapToDTO(Post post){

PostDto postDto = new PostDto();

postDto.setId(post.getId());

postDto.setTitle(post.getTitle());

postDto.setDescription(post.getDescription());

postDto.setContent(post.getContent());

return postDto;

}

// convert DTO to entity

private Post mapToEntity(PostDto postDto){

Post post = new Post();

post.setTitle(postDto.getTitle());

post.setDescription(postDto.getDescription());

post.setContent(postDto.getContent());

return post;

}

}

**Pagination and Sorting in rest API**

**Step 1: Update Post Controller Class:**

import com.springboot.blog.payload.PostDto;

import com.springboot.blog.payload.PostResponse;

import com.springboot.blog.service.PostService;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/api/posts")

public class PostController {

private PostService postService;

public PostController(PostService postService) {

this.postService = postService;

}

// create blog post rest api

@PostMapping

public ResponseEntity<PostDto> createPost(@RequestBody PostDto postDto){

return new ResponseEntity<>(postService.createPost(postDto), HttpStatus.CREATED);

}

// get all posts rest api

**@GetMapping**

**public PostResponse getAllPosts(**

**@RequestParam(value = "pageNo", defaultValue = "0", required = false) int pageNo,**

**@RequestParam(value = "pageSize", defaultValue = "10", required = false) int pageSize,**

**@RequestParam(value = "sortBy", defaultValue = "id", required = false) String sortBy,**

**@RequestParam(value = "sortDir", defaultValue = "asc", required = false) String sortDir**

**){**

**return postService.getAllPosts(pageNo, pageSize, sortBy, sortDir);**

**}**

// get post by id

@GetMapping("/{id}")

public ResponseEntity<PostDto> getPostById(@PathVariable(name = "id") long id){

return ResponseEntity.ok(postService.getPostById(id));

}

// update post by id rest api

@PutMapping("/{id}")

public ResponseEntity<PostDto> updatePost(@RequestBody PostDto postDto, @PathVariable(name = "id") long id){

PostDto postResponse = postService.updatePost(postDto, id);

return new ResponseEntity<>(postResponse, HttpStatus.OK);

}

// delete post rest api

@DeleteMapping("/{id}")

public ResponseEntity<String> deletePost(@PathVariable(name = "id") long id){

postService.deletePostById(id);

return new ResponseEntity<>("Post entity deleted successfully.", HttpStatus.OK);

}

}

**Step 2: Update PostService interface”:**

import com.springboot.blog.payload.PostDto;

import com.springboot.blog.payload.PostResponse;

import java.util.List;

public interface PostService {

PostDto createPost(PostDto postDto);

**PostResponse getAllPosts(int pageNo, int pageSize, String sortBy, String sortDir);**

PostDto getPostById(long id);

PostDto updatePost(PostDto postDto, long id);

void deletePostById(long id);

}

**Step 3: Update PostServiceImpl class:**

import com.springboot.blog.entity.Post;

import com.springboot.blog.exception.ResourceNotFoundException;

import com.springboot.blog.payload.PostDto;

import com.springboot.blog.payload.PostResponse;

import com.springboot.blog.repository.PostRepository;

import com.springboot.blog.service.PostService;

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

import org.springframework.data.domain.Page;

import org.springframework.data.domain.PageRequest;

import org.springframework.data.domain.Pageable;

import org.springframework.data.domain.Sort;

import org.springframework.stereotype.Service;

import java.util.List;

import java.util.stream.Collectors;

@Service

public class PostServiceImpl implements PostService {

private PostRepository postRepository;

public PostServiceImpl(PostRepository postRepository) {

this.postRepository = postRepository;

}

@Override

public PostDto createPost(PostDto postDto) {

// convert DTO to entity

Post post = mapToEntity(postDto);

Post newPost = postRepository.save(post);

// convert entity to DTO

PostDto postResponse = mapToDTO(newPost);

return postResponse;

}

**@Override**

**public PostResponse getAllPosts(int pageNo, int pageSize, String sortBy, String sortDir) {**

**Sort sort = sortDir.equalsIgnoreCase(Sort.Direction.ASC.name()) ? Sort.by(sortBy).ascending()**

**: Sort.by(sortBy).descending();**

**// create Pageable instance**

**Pageable pageable = PageRequest.of(pageNo, pageSize, sort);**

**Page<Post> posts = postRepository.findAll(pageable);**

**// get content for page object**

**List<Post> listOfPosts = posts.getContent();**

**List<PostDto> content= listOfPosts.stream().map(post -> mapToDTO(post)).collect(Collectors.toList());**

**PostResponse postResponse = new PostResponse();**

**postResponse.setContent(content);**

**postResponse.setPageNo(posts.getNumber());**

**postResponse.setPageSize(posts.getSize());**

**postResponse.setTotalElements(posts.getTotalElements());**

**postResponse.setTotalPages(posts.getTotalPages());**

**postResponse.setLast(posts.isLast());**

**return postResponse;**

**}**

@Override

public PostDto getPostById(long id) {

Post post = postRepository.findById(id).orElseThrow(() -> new ResourceNotFoundException("Post", "id", id));

return mapToDTO(post);

}

@Override

public PostDto updatePost(PostDto postDto, long id) {

// get post by id from the database

Post post = postRepository.findById(id).orElseThrow(() -> new ResourceNotFoundException("Post", "id", id));

post.setTitle(postDto.getTitle());

post.setDescription(postDto.getDescription());

post.setContent(postDto.getContent());

Post updatedPost = postRepository.save(post);

return mapToDTO(updatedPost);

}

@Override

public void deletePostById(long id) {

// get post by id from the database

Post post = postRepository.findById(id).orElseThrow(() -> new ResourceNotFoundException("Post", "id", id));

postRepository.delete(post);

}

// convert Entity into DTO

private PostDto mapToDTO(Post post){

PostDto postDto = new PostDto();

postDto.setId(post.getId());

postDto.setTitle(post.getTitle());

postDto.setDescription(post.getDescription());

postDto.setContent(post.getContent());

return postDto;

}

// convert DTO to entity

private Post mapToEntity(PostDto postDto){

Post post = new Post();

post.setTitle(postDto.getTitle());

post.setDescription(postDto.getDescription());

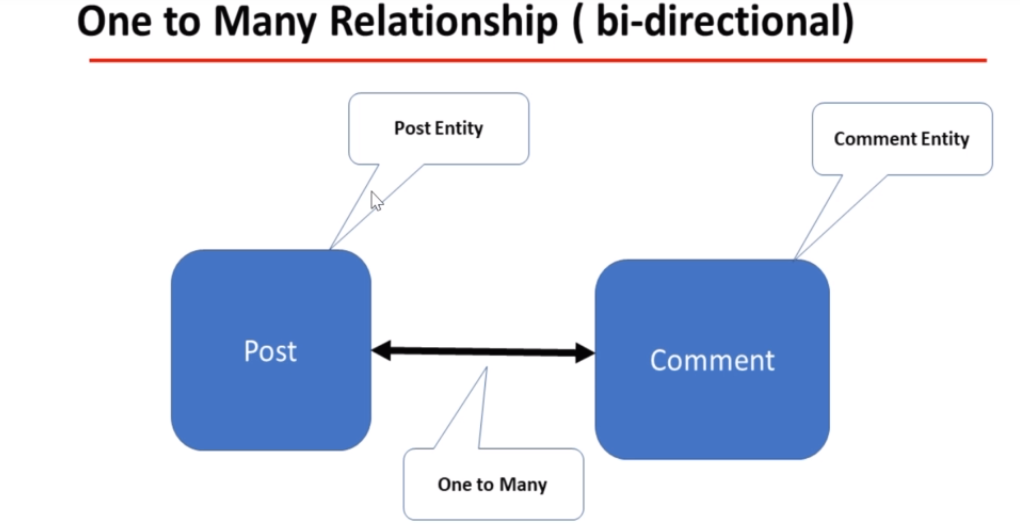
post.setContent(postDto.getContent());

return post;

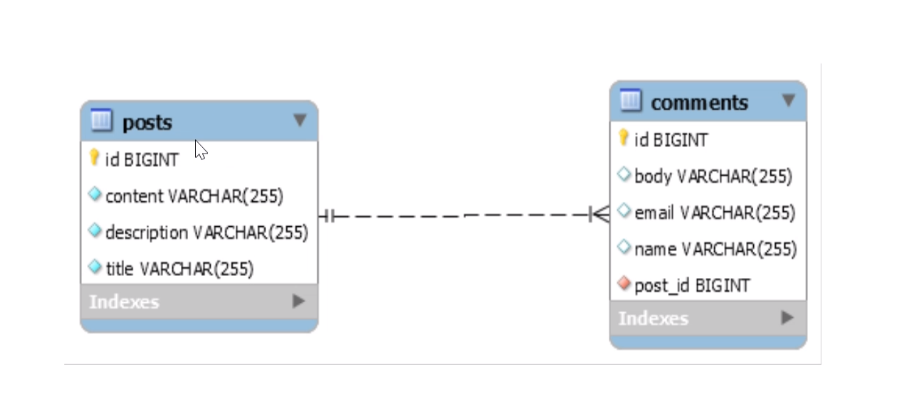
}

}

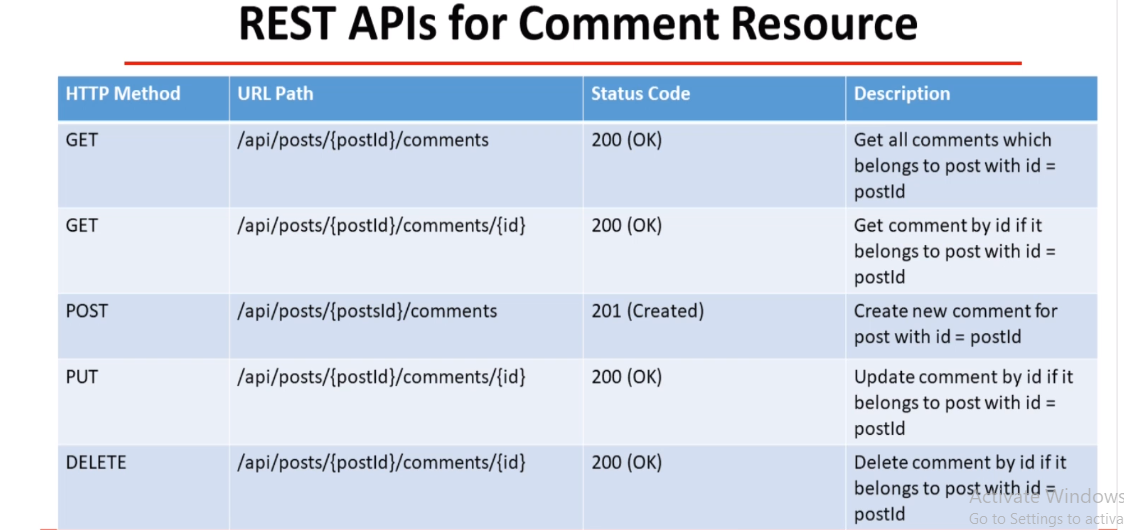
**Create Comments API Later**



**ER(Entity Relationship Diagram)**

****

**URL Documentation with status code:**

****

**Step 1: Create Comment Entity Class and do oneTomany bidirectional mapping**

**import lombok.AllArgsConstructor;**

**import lombok.Data;**

**import lombok.NoArgsConstructor;**

**import javax.persistence.\*;**

**@Data**

**@AllArgsConstructor**

**@NoArgsConstructor**

**@Entity**

**@Table(name = "comments")**

**public class Comment {**

**@Id**

**@GeneratedValue(strategy = GenerationType.IDENTITY)**

**private long id;**

**private String name;**

**private String email;**

**private String body;**

**@ManyToOne(fetch = FetchType.LAZY)**

**@JoinColumn(name = "post\_id", nullable = false)**

**private Post post;**

**}**

**Step 2: Update Post Entity Class:**

**import lombok.\*;**

**import javax.persistence.\*;**

**import java.util.HashSet;**

**import java.util.Set;**

**@Getter**

**@Setter**

**@AllArgsConstructor**

**@NoArgsConstructor**

**@Entity**

**@Table(**

**name = "posts", uniqueConstraints = {@UniqueConstraint(columnNames = {"title"})}**

**)**

**public class Post {**

**@Id**

**@GeneratedValue(**

**strategy = GenerationType.IDENTITY**

**)**

**private Long id;**

**@Column(name = "title", nullable = false)**

**private String title;**

**@Column(name = "description", nullable = false)**

**private String description;**

**@Column(name = "content", nullable = false)**

**private String content;**

**@OneToMany(mappedBy = "post", cascade = CascadeType.ALL, orphanRemoval = true)**

**private Set<Comment> comments = new HashSet<>();**

**}**

**Step 3: Create CommentDto class**

**@Data**

**public class CommentDto {**

**private long id;**

**private String name;**

**private String email;**

**private String body;**

**}**

**Step 4: Create CommentService Interface:**

**import java.util.List;**

**public interface CommentService {**

**CommentDto createComment(long postId, CommentDto commentDto);**

**}**

**Step 5: Create CommentServiceImpl class:**

**@Service**

**public class CommentServiceImpl implements CommentService {**

**private CommentRepository commentRepository;**

**private PostRepository postRepository;**

**private ModelMapper mapper;**

**public CommentServiceImpl(CommentRepository commentRepository,**

**PostRepository postRepository, ModelMapper mapper) {**

**this.commentRepository = commentRepository;**

**this.postRepository = postRepository;**

**this.mapper = mapper;**

**}**

**@Override**

**public CommentDto createComment(long postId, CommentDto commentDto) {**

**Comment comment = mapToEntity(commentDto);**

**// retrieve post entity by id**

**Post post = postRepository.findById(postId).orElseThrow(**

**() -> new ResourceNotFoundException("Post", "id", postId));**

**// set post to comment entity**

**comment.setPost(post);**

**// comment entity to DB**

**Comment newComment = commentRepository.save(comment);**

**return mapToDTO(newComment);**

**}**

**private CommentDto mapToDTO(Comment comment){**

**CommentDto commentDto = mapper.map(comment, CommentDto.class);**

**CommentDto commentDto = new CommentDto();**

**commentDto.setId(comment.getId());**

**commentDto.setName(comment.getName());**

**commentDto.setEmail(comment.getEmail());**

**commentDto.setBody(comment.getBody());**

**return commentDto;**

**}**

**private Comment mapToEntity(CommentDto commentDto){**

**Comment comment = mapper.map(commentDto, Comment.class);**

**Comment comment = new Comment();**

**comment.setId(commentDto.getId());**

**comment.setName(commentDto.getName());**

**comment.setEmail(commentDto.getEmail());**

**comment.setBody(commentDto.getBody());**

**return comment;**

**}**

**}**

**Step 6: Create RestController CommentController Class:**

**@RestController**

**@RequestMapping("/api/")**

**public class CommentController {**

**private CommentService commentService;**

**public CommentController(CommentService commentService) {**

**this.commentService = commentService;**

**}**

**@PostMapping("/posts/{postId}/comments")**

**public ResponseEntity<CommentDto> createComment(@PathVariable(value = "postId") long postId,**

**@RequestBody CommentDto commentDto){**

**return new ResponseEntity<>(commentService.createComment(postId, commentDto), HttpStatus.CREATED);**

**}**

**}**

**Get All Comments By PostId**

**Step 1: Update CommentRepository as shown below:**

**import org.springframework.data.jpa.repository.JpaRepository;**

**import java.util.List;**

**public interface CommentRepository extends JpaRepository<Comment, Long> {**

**List<Comment> findByPostId(long postId);**

**}**

**Step 2: Update CommentService Interface:**

**import java.util.List;**

**public interface CommentService {**

**CommentDto createComment(long postId, CommentDto commentDto);**

**List<CommentDto> getCommentsByPostId(long postId);**

**}**

**Step 3: Update CommentServiceImpl Class:**

**@Service**

**public class CommentServiceImpl implements CommentService {**

**private CommentRepository commentRepository;**

**private PostRepository postRepository;**

**private ModelMapper mapper;**

**public CommentServiceImpl(CommentRepository commentRepository, PostRepository postRepository, ModelMapper mapper) {**

**this.commentRepository = commentRepository;**

**this.postRepository = postRepository;**

**this.mapper = mapper;**

**}**

**@Override**

**public CommentDto createComment(long postId, CommentDto commentDto) {**

**Comment comment = mapToEntity(commentDto);**

**// retrieve post entity by id**

**Post post = postRepository.findById(postId).orElseThrow(**

**() -> new ResourceNotFoundException("Post", "id", postId));**

**// set post to comment entity**

**comment.setPost(post);**

**// comment entity to DB**

**Comment newComment = commentRepository.save(comment);**

**return mapToDTO(newComment);**

**}**

**@Override**

**public List<CommentDto> getCommentsByPostId(long postId) {**

**// retrieve comments by postId**

**List<Comment> comments = commentRepository.findByPostId(postId);**

**// convert list of comment entities to list of comment dto's**

**return comments.stream().map(comment -> mapToDTO(comment)).collect(Collectors.toList());**

**}**

**private CommentDto mapToDTO(Comment comment){**

**CommentDto commentDto = mapper.map(comment, CommentDto.class);**

**CommentDto commentDto = new CommentDto();**

**commentDto.setId(comment.getId());**

**commentDto.setName(comment.getName());**

**commentDto.setEmail(comment.getEmail());**

**commentDto.setBody(comment.getBody());**

**return commentDto;**

**}**

**private Comment mapToEntity(CommentDto commentDto){**

**Comment comment = mapper.map(commentDto, Comment.class);**

**Comment comment = new Comment();**

**comment.setId(commentDto.getId());**

**comment.setName(commentDto.getName());**

**comment.setEmail(commentDto.getEmail());**

**comment.setBody(commentDto.getBody());**

**return comment;**

**}**

**}**

**Step 4: Create handler method in CommentController Layer:**

**@RestController**

**@RequestMapping("/api/")**

**public class CommentController {**

**private CommentService commentService;**

**public CommentController(CommentService commentService) {**

**this.commentService = commentService;**

**}**

**@PostMapping("/posts/{postId}/comments")**

**public ResponseEntity<CommentDto> createComment(@PathVariable(value = "postId") long postId, @RequestBody CommentDto commentDto){**

**return new ResponseEntity<>(commentService.createComment(postId, commentDto), HttpStatus.CREATED);**

**}**

**@GetMapping("/posts/{postId}/comments")**

**public List<CommentDto> getCommentsByPostId(@PathVariable(value = "postId") Long postId){**

**return commentService.getCommentsByPostId(postId);**

**}**

**}**

**Get Comment By CommentId**

**Step 1: Update CommentService interface:**

**import java.util.List;**

**public interface CommentService {**

**CommentDto createComment(long postId, CommentDto commentDto);**

**List<CommentDto> getCommentsByPostId(long postId);**

**CommentDto getCommentById(Long postId, Long commentId);**

**}**

**Step 2: Create BlogApi Exception class:**

**import org.springframework.http.HttpStatus;**

**public class BlogAPIException extends RuntimeException {**

**private HttpStatus status;**

**private String message;**

**public BlogAPIException(HttpStatus status, String message) {**

**this.status = status;**

**this.message = message;**

**}**

**public BlogAPIException(String message, HttpStatus status, String message1) {**

**super(message);**

**this.status = status;**

**this.message = message1;**

**}**

**public HttpStatus getStatus() {**

**return status;**

**}**

**@Override**

**public String getMessage() {**

**return message;**

**}**

**}**

**Step 3: Update CommentServiceImpl class:**

**@Service**

**public class CommentServiceImpl implements CommentService {**

**private CommentRepository commentRepository;**

**private PostRepository postRepository;**

**private ModelMapper mapper;**

**public CommentServiceImpl(CommentRepository commentRepository, PostRepository postRepository, ModelMapper mapper) {**

**this.commentRepository = commentRepository;**

**this.postRepository = postRepository;**

**this.mapper = mapper;**

**}**

**@Override**

**public CommentDto createComment(long postId, CommentDto commentDto) {**

**Comment comment = mapToEntity(commentDto);**

**// retrieve post entity by id**

**Post post = postRepository.findById(postId).orElseThrow(**

**() -> new ResourceNotFoundException("Post", "id", postId));**

**// set post to comment entity**

**comment.setPost(post);**

**// comment entity to DB**

**Comment newComment = commentRepository.save(comment);**

**return mapToDTO(newComment);**

**}**

**@Override**

**public List<CommentDto> getCommentsByPostId(long postId) {**

**// retrieve comments by postId**

**List<Comment> comments = commentRepository.findByPostId(postId);**

**// convert list of comment entities to list of comment dto's**

**return comments.stream().map(comment -> mapToDTO(comment)).collect(Collectors.toList());**

**}**

**@Override**

**public CommentDto getCommentById(Long postId, Long commentId) {**

**// retrieve post entity by id**

**Post post = postRepository.findById(postId).orElseThrow(**

**() -> new ResourceNotFoundException("Post", "id", postId));**

**// retrieve comment by id**

**Comment comment = commentRepository.findById(commentId).orElseThrow(() ->**

**new ResourceNotFoundException("Comment", "id", commentId));**

**if(!comment.getPost().getId().equals(post.getId())){**

**throw new BlogAPIException(HttpStatus.BAD\_REQUEST, "Comment does not belong to post");**

**}**

**return mapToDTO(comment);**

**}**

**private CommentDto mapToDTO(Comment comment){**

**CommentDto commentDto = mapper.map(comment, CommentDto.class);**

**CommentDto commentDto = new CommentDto();**

**commentDto.setId(comment.getId());**

**commentDto.setName(comment.getName());**

**commentDto.setEmail(comment.getEmail());**

**commentDto.setBody(comment.getBody());**

**return commentDto;**

**}**

**private Comment mapToEntity(CommentDto commentDto){**

**Comment comment = mapper.map(commentDto, Comment.class);**

**Comment comment = new Comment();**

**comment.setId(commentDto.getId());**

**comment.setName(commentDto.getName());**

**comment.setEmail(commentDto.getEmail());**

**comment.setBody(commentDto.getBody());**

**return comment;**

**}**

**}**

**Step 4: Update CommentController class:**

**@RestController**

**@RequestMapping("/api/")**

**public class CommentController {**

**private CommentService commentService;**

**public CommentController(CommentService commentService) {**

**this.commentService = commentService;**

**}**

**@PostMapping("/posts/{postId}/comments")**

**public ResponseEntity<CommentDto> createComment(@PathVariable(value = "postId") long postId,**

**@RequestBody CommentDto commentDto){**

**return new ResponseEntity<>(commentService.createComment(postId, commentDto), HttpStatus.CREATED);**

**}**

**@GetMapping("/posts/{postId}/comments")**

**public List<CommentDto> getCommentsByPostId(@PathVariable(value = "postId") Long postId){**

**return commentService.getCommentsByPostId(postId);**

**}**

**@GetMapping("/posts/{postId}/comments/{id}")**

**public ResponseEntity<CommentDto> getCommentById(@PathVariable(value = "postId") Long postId,**

**@PathVariable(value = "id") Long commentId){**

**CommentDto commentDto = commentService.getCommentById(postId, commentId);**

**return new ResponseEntity<>(commentDto, HttpStatus.OK);**

**}**

**}**

**Developing Update Comment Rest API**

Rest api url: [http://localhost:8080/api/posts/{postId}/comments{id}](http://localhost:8080/api/posts/%7bpostId%7d/comments%7bid%7d)

Step 1: Update CommentController with following handler method:

**@PutMapping("/posts/{postId}/comments/{id}")**

**public ResponseEntity<CommentDto> updateComment(@PathVariable(value = "postId") Long postId,**

**@PathVariable(value = "id") Long commentId,**

**@RequestBody CommentDto commentDto){**

**CommentDto updatedComment = commentService.updateComment(postId, commentId, commentDto);**

**return new ResponseEntity<>(updatedComment, HttpStatus.OK);**

**}**

**Step 2: Update CommentService Interface:**

**import java.util.List;**

**public interface CommentService {**

**CommentDto createComment(long postId, CommentDto commentDto);**

**List<CommentDto> getCommentsByPostId(long postId);**

**CommentDto getCommentById(Long postId, Long commentId);**

**CommentDto updateComment(Long postId, long commentId, CommentDto commentRequest);**

**}**

**Step 3: Update CommentServiceImpl class:**

**@Override**

**public CommentDto updateComment(Long postId, long commentId, CommentDto commentRequest) {**

**// retrieve post entity by id**

**Post post = postRepository.findById(postId).orElseThrow(**

**() -> new ResourceNotFoundException("Post", "id", postId));**

**// retrieve comment by id**

**Comment comment = commentRepository.findById(commentId).orElseThrow(() ->**

**new ResourceNotFoundException("Comment", "id", commentId));**

**if(!comment.getPost().getId().equals(post.getId())){**

**throw new BlogAPIException(HttpStatus.BAD\_REQUEST, "Comment does not belongs to post");**

**}**

**comment.setName(commentRequest.getName());**

**comment.setEmail(commentRequest.getEmail());**

**comment.setBody(commentRequest.getBody());**

**Comment updatedComment = commentRepository.save(comment);**

**return mapToDTO(updatedComment);**

**}**

**Perform Testing in PostMan:**

**Delete Comment Feature**

**URL: http://localhost:8080/api/posts/{postId}/comments/{id}**

**Step 1: Update CommentController Class:**

**@DeleteMapping("/posts/{postId}/comments/{id}")**

**public ResponseEntity<String> deleteComment(@PathVariable(value = "postId") Long postId,**

**@PathVariable(value = "id") Long commentId){**

**commentService.deleteComment(postId, commentId);**

**return new ResponseEntity<>("Comment deleted successfully", HttpStatus.OK);**

**}**

**Step 2: Update CommentService Interface**

**import java.util.List;**

**public interface CommentService {**

**CommentDto createComment(long postId, CommentDto commentDto);**

**List<CommentDto> getCommentsByPostId(long postId);**

**CommentDto getCommentById(Long postId, Long commentId);**

**CommentDto updateComment(Long postId, long commentId, CommentDto commentRequest);**

**void deleteComment(Long postId, Long commentId);**

**}**

**Step 3: Update CommentServiceImpl class**

**@Override**

**public void deleteComment(Long postId, Long commentId) {**

**// retrieve post entity by id**

**Post post = postRepository.findById(postId).orElseThrow(**

**() -> new ResourceNotFoundException("Post", "id", postId));**

**// retrieve comment by id**

**Comment comment = commentRepository.findById(commentId).orElseThrow(() ->**

**new ResourceNotFoundException("Comment", "id", commentId));**

**if(!comment.getPost().getId().equals(post.getId())){**

**throw new BlogAPIException(HttpStatus.BAD\_REQUEST, "Comment does not belongs to post");**

**}**

**commentRepository.delete(comment);**

**}**

**ModelMapper library or MapStruct**

**Step 1: Add the following dependency:**

**<!-- https://mvnrepository.com/artifact/org.modelmapper/modelmapper -->**

**<dependency>**

**<groupId>org.modelmapper</groupId>**

**<artifactId>modelmapper</artifactId>**

**<version>2.3.9</version>**

**</dependency>**

**Step 2: Update PostServiceImpl class as shown below:**

**@Service**

**public class PostServiceImpl implements PostService {**

**private PostRepository postRepository;**

**private ModelMapper mapper;**

**public PostServiceImpl(PostRepository postRepository, ModelMapper mapper) {**

**this.postRepository = postRepository;**

**this.mapper = mapper;**

**}**

**@Override**

**public PostDto createPost(PostDto postDto) {**

**// convert DTO to entity**

**Post post = mapToEntity(postDto);**

**Post newPost = postRepository.save(post);**

**// convert entity to DTO**

**PostDto postResponse = mapToDTO(newPost);**

**return postResponse;**

**}**

**@Override**

**public PostResponse getAllPosts(int pageNo, int pageSize, String sortBy, String sortDir) {**

**Sort sort = sortDir.equalsIgnoreCase(Sort.Direction.ASC.name()) ? Sort.by(sortBy).ascending()**

**: Sort.by(sortBy).descending();**

**// create Pageable instance**

**Pageable pageable = PageRequest.of(pageNo, pageSize, sort);**

**Page<Post> posts = postRepository.findAll(pageable);**

**// get content for page object**

**List<Post> listOfPosts = posts.getContent();**

**List<PostDto> content= listOfPosts.stream().map(post -> mapToDTO(post)).collect(Collectors.toList());**

**PostResponse postResponse = new PostResponse();**

**postResponse.setContent(content);**

**postResponse.setPageNo(posts.getNumber());**

**postResponse.setPageSize(posts.getSize());**

**postResponse.setTotalElements(posts.getTotalElements());**

**postResponse.setTotalPages(posts.getTotalPages());**

**postResponse.setLast(posts.isLast());**

**return postResponse;**

**}**

**@Override**

**public PostDto getPostById(long id) {**

**Post post = postRepository.findById(id).orElseThrow(() -> new ResourceNotFoundException("Post", "id", id));**

**return mapToDTO(post);**

**}**

**@Override**

**public PostDto updatePost(PostDto postDto, long id) {**

**// get post by id from the database**

**Post post = postRepository.findById(id).orElseThrow(() -> new ResourceNotFoundException("Post", "id", id));**

**post.setTitle(postDto.getTitle());**

**post.setDescription(postDto.getDescription());**

**post.setContent(postDto.getContent());**

**Post updatedPost = postRepository.save(post);**

**return mapToDTO(updatedPost);**

**}**

**@Override**

**public void deletePostById(long id) {**

**// get post by id from the database**

**Post post = postRepository.findById(id).orElseThrow(() -> new ResourceNotFoundException("Post", "id", id));**

**postRepository.delete(post);**

**}**

**// convert Entity into DTO**

**private PostDto mapToDTO(Post post){**

**PostDto postDto = mapper.map(post, PostDto.class);**

**// PostDto postDto = new PostDto();**

**// postDto.setId(post.getId());**

**// postDto.setTitle(post.getTitle());**

**// postDto.setDescription(post.getDescription());**

**// postDto.setContent(post.getContent());**

**return postDto;**

**}**

**// convert DTO to entity**

**private Post mapToEntity(PostDto postDto){**

**Post post = mapper.map(postDto, Post.class);**

**// Post post = new Post();**

**// post.setTitle(postDto.getTitle());**

**// post.setDescription(postDto.getDescription());**

**// post.setContent(postDto.getContent());**

**return post;**

**}**

**}**

**Step 3: Update CommentServiceImpl class:**

**@Service**

**public class CommentServiceImpl implements CommentService {**

**private CommentRepository commentRepository;**

**private PostRepository postRepository;**

**private ModelMapper mapper;**

**public CommentServiceImpl(CommentRepository commentRepository, PostRepository postRepository, ModelMapper mapper) {**

**this.commentRepository = commentRepository;**

**this.postRepository = postRepository;**

**this.mapper = mapper;**

**}**

**@Override**

**public CommentDto createComment(long postId, CommentDto commentDto) {**

**Comment comment = mapToEntity(commentDto);**

**// retrieve post entity by id**

**Post post = postRepository.findById(postId).orElseThrow(**

**() -> new ResourceNotFoundException("Post", "id", postId));**

**// set post to comment entity**

**comment.setPost(post);**

**// comment entity to DB**

**Comment newComment = commentRepository.save(comment);**

**return mapToDTO(newComment);**

**}**

**@Override**

**public List<CommentDto> getCommentsByPostId(long postId) {**

**// retrieve comments by postId**

**List<Comment> comments = commentRepository.findByPostId(postId);**

**// convert list of comment entities to list of comment dto's**

**return comments.stream().map(comment -> mapToDTO(comment)).collect(Collectors.toList());**

**}**

**@Override**

**public CommentDto getCommentById(Long postId, Long commentId) {**

**// retrieve post entity by id**

**Post post = postRepository.findById(postId).orElseThrow(**

**() -> new ResourceNotFoundException("Post", "id", postId));**

**// retrieve comment by id**

**Comment comment = commentRepository.findById(commentId).orElseThrow(() ->**

**new ResourceNotFoundException("Comment", "id", commentId));**

**if(!comment.getPost().getId().equals(post.getId())){**

**throw new BlogAPIException(HttpStatus.BAD\_REQUEST, "Comment does not belong to post");**

**}**

**return mapToDTO(comment);**

**}**

**@Override**

**public CommentDto updateComment(Long postId, long commentId, CommentDto commentRequest) {**

**// retrieve post entity by id**

**Post post = postRepository.findById(postId).orElseThrow(**

**() -> new ResourceNotFoundException("Post", "id", postId));**

**// retrieve comment by id**

**Comment comment = commentRepository.findById(commentId).orElseThrow(() ->**

**new ResourceNotFoundException("Comment", "id", commentId));**

**if(!comment.getPost().getId().equals(post.getId())){**

**throw new BlogAPIException(HttpStatus.BAD\_REQUEST, "Comment does not belongs to post");**

**}**

**comment.setName(commentRequest.getName());**

**comment.setEmail(commentRequest.getEmail());**

**comment.setBody(commentRequest.getBody());**

**Comment updatedComment = commentRepository.save(comment);**

**return mapToDTO(updatedComment);**

**}**

**@Override**

**public void deleteComment(Long postId, Long commentId) {**

**// retrieve post entity by id**

**Post post = postRepository.findById(postId).orElseThrow(**

**() -> new ResourceNotFoundException("Post", "id", postId));**

**// retrieve comment by id**

**Comment comment = commentRepository.findById(commentId).orElseThrow(() ->**

**new ResourceNotFoundException("Comment", "id", commentId));**

**if(!comment.getPost().getId().equals(post.getId())){**

**throw new BlogAPIException(HttpStatus.BAD\_REQUEST, "Comment does not belongs to post");**

**}**

**commentRepository.delete(comment);**

**}**

**private CommentDto mapToDTO(Comment comment){**

**CommentDto commentDto = mapper.map(comment, CommentDto.class);**

**// CommentDto commentDto = new CommentDto();**

**// commentDto.setId(comment.getId());**

**// commentDto.setName(comment.getName());**

**// commentDto.setEmail(comment.getEmail());**

**// commentDto.setBody(comment.getBody());**

**return commentDto;**

**}**

**private Comment mapToEntity(CommentDto commentDto){**

**Comment comment = mapper.map(commentDto, Comment.class);**

**// Comment comment = new Comment();**

**// comment.setId(commentDto.getId());**

**// comment.setName(commentDto.getName());**

**// comment.setEmail(commentDto.getEmail());**

**// comment.setBody(commentDto.getBody());**

**return comment;**

**}**

**}**

**Exception Handling – Specific Exception & Global Exception**

**Step 1: Create ErrorDetails class in payload package**

**import java.util.Date;**

**public class ErrorDetails {**

**private Date timestamp;**

**private String message;**

**private String details;**

**public ErrorDetails(Date timestamp, String message, String details) {**

**this.timestamp = timestamp;**

**this.message = message;**

**this.details = details;**

**}**

**public Date getTimestamp() {**

**return timestamp;**

**}**

**public String getMessage() {**

**return message;**

**}**

**public String getDetails() {**

**return details;**

**}**

**}**

**Step 2: Create GlobalExceptionHandler class in exceptionpackage**

**import com.springboot.blog.payload.ErrorDetails;**

**import org.springframework.http.HttpHeaders;**

**import org.springframework.http.HttpStatus;**

**import org.springframework.http.ResponseEntity;**

**import org.springframework.validation.FieldError;**

**import org.springframework.web.bind.MethodArgumentNotValidException;**

**import org.springframework.web.bind.annotation.ControllerAdvice;**

**import org.springframework.web.bind.annotation.ExceptionHandler;**

**import org.springframework.web.context.request.WebRequest;**

**import org.springframework.web.servlet.mvc.method.annotation.ResponseEntityExceptionHandler;**

**import java.util.Date;**

**import java.util.HashMap;**

**import java.util.Map;**

**@ControllerAdvice**

**public class GlobalExceptionHandler extends ResponseEntityExceptionHandler {**

**// handle specific exceptions**

**@ExceptionHandler(ResourceNotFoundException.class)**

**public ResponseEntity<ErrorDetails> handleResourceNotFoundException(ResourceNotFoundException exception,**

**WebRequest webRequest){**

**ErrorDetails errorDetails = new ErrorDetails(new Date(), exception.getMessage(),**

**webRequest.getDescription(false));**

**return new ResponseEntity<>(errorDetails, HttpStatus.NOT\_FOUND);**

**}**

**@ExceptionHandler(BlogAPIException.class)**

**public ResponseEntity<ErrorDetails> handleBlogAPIException(BlogAPIException exception,**

**WebRequest webRequest){**

**ErrorDetails errorDetails = new ErrorDetails(new Date(), exception.getMessage(),**

**webRequest.getDescription(false));**

**return new ResponseEntity<>(errorDetails, HttpStatus.BAD\_REQUEST);**

**}**

**// global exceptions**

**@ExceptionHandler(Exception.class)**

**public ResponseEntity<ErrorDetails> handleGlobalException(Exception exception,**

**WebRequest webRequest){**

**ErrorDetails errorDetails = new ErrorDetails(new Date(), exception.getMessage(),**

**webRequest.getDescription(false));**

**return new ResponseEntity<>(errorDetails, HttpStatus.INTERNAL\_SERVER\_ERROR);**

**}**

**}**

**Spring Validations**

**Step 1: Add dependency in pom.xml file**

**<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-validation -->**

**<dependency>**

**<groupId>org.springframework.boot</groupId>**

**<artifactId>spring-boot-starter-validation</artifactId>**

**</dependency>**

**Step 2: Add Validation annotations in DTO classes**

**package com.springboot.blog.payload;**

**import io.swagger.annotations.ApiModel;**

**import io.swagger.annotations.ApiModelProperty;**

**import lombok.Data;**

**import javax.validation.constraints.NotEmpty;**

**import javax.validation.constraints.Size;**

**import java.util.Set;**

**@ApiModel(description = "Post model information")**

**@Data**

**public class PostDto {**

**private long id;**

**// title should not be null or empty**

**// title should have at least 2 characters**

**@NotEmpty**

**@Size(min = 2, message = "Post title should have at least 2 characters")**

**private String title;**

**// post description should be not null or empty**

**// post description should have at least 10 characters**

**@NotEmpty**

**@Size(min = 10, message = "Post description should have at least 10 characters")**

**private String description;**

**// post content should not be null or empty**

**@NotEmpty**

**private String content;**

**private Set<CommentDto> comments;**

**}**

**Step 3: Add @Valid annotation in controller class:**

**import com.springboot.blog.payload.PostDto;**

**import com.springboot.blog.payload.PostResponse;**

**import com.springboot.blog.service.PostService;**

**import com.springboot.blog.utils.AppConstants;**

**import io.swagger.annotations.Api;**

**import io.swagger.annotations.ApiOperation;**

**import io.swagger.annotations.ApiResponses;**

**import org.springframework.http.HttpStatus;**

**import org.springframework.http.ResponseEntity;**

**import org.springframework.security.access.prepost.PreAuthorize;**

**import org.springframework.web.bind.annotation.\*;**

**import javax.validation.Valid;**

**@RestController**

**@RequestMapping()**

**public class PostController {**

**private PostService postService;**

**public PostController(PostService postService) {**

**this.postService = postService;**

**}**

**// create blog post rest api**

**@PostMapping("/api/v1/posts")**

**public ResponseEntity<PostDto> createPost(@Valid @RequestBody PostDto postDto){**

**return new ResponseEntity<>(postService.createPost(postDto), HttpStatus.CREATED);**

**}**

**// get all posts rest api**

**@GetMapping("/api/v1/posts")**

**public PostResponse getAllPosts(**

**@RequestParam(value = "pageNo", defaultValue = AppConstants.DEFAULT\_PAGE\_NUMBER, required = false) int pageNo,**

**@RequestParam(value = "pageSize", defaultValue = AppConstants.DEFAULT\_PAGE\_SIZE, required = false) int pageSize,**

**@RequestParam(value = "sortBy", defaultValue = AppConstants.DEFAULT\_SORT\_BY, required = false) String sortBy,**

**@RequestParam(value = "sortDir", defaultValue = AppConstants.DEFAULT\_SORT\_DIRECTION, required = false) String sortDir**

**){**

**return postService.getAllPosts(pageNo, pageSize, sortBy, sortDir);**

**}**

**// get post by id**

**@GetMapping(value = "/api/v1/posts/{id}")**

**public ResponseEntity<PostDto> getPostByIdV1(@PathVariable(name = "id") long id){**

**return ResponseEntity.ok(postService.getPostById(id));**

**}**

**// update post by id rest api**

**@PutMapping("/api/v1/posts/{id}")**

**public ResponseEntity<PostDto> updatePost(@Valid @RequestBody PostDto postDto, @PathVariable(name = "id") long id){**

**PostDto postResponse = postService.updatePost(postDto, id);**

**return new ResponseEntity<>(postResponse, HttpStatus.OK);**

**}**

**// delete post rest api**

**@DeleteMapping("/api/v1/posts/{id}")**

**public ResponseEntity<String> deletePost(@PathVariable(name = "id") long id){**

**postService.deletePostById(id);**

**return new ResponseEntity<>("Post entity deleted successfully.", HttpStatus.OK);**

**}**

**}**

**Spring Security**

**Step 1: Add Spring Dependency Jar**

[<dependency>](file:///C:\Users\Admin\Downloads\springboot-blog-rest-api%20(2)\springboot-blog-rest-api\pom.xml)

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

**Step 2: All Links of rest api are now secured**

**Step 3: Update application.properties file**

**Spring.security.user.name=pankaj**

**Spring.security.user.password=password**

**Spring.security.user.roles=ADMIN**

**Step 4: Implementing basic authentication**

**Develop config package**

**Step 5: Develop SecurityConfig class and Extend WebSecurityConfigurerAdapter**

**@Configuration**

**@EnableWebSecurity**

**public class SecurityConfig extends WebSecurityConfigurerAdapter {**

**@Override**

**protected void configure(HttpSecurity http) throws Exception {**

**http**

**.csrf().disable()**

**.authorizeRequests()**

**.anyRequest()**

**.authenticated()**

**.and()**

**.httpBasic();**

**}**

**}**

**In memory Authentication**

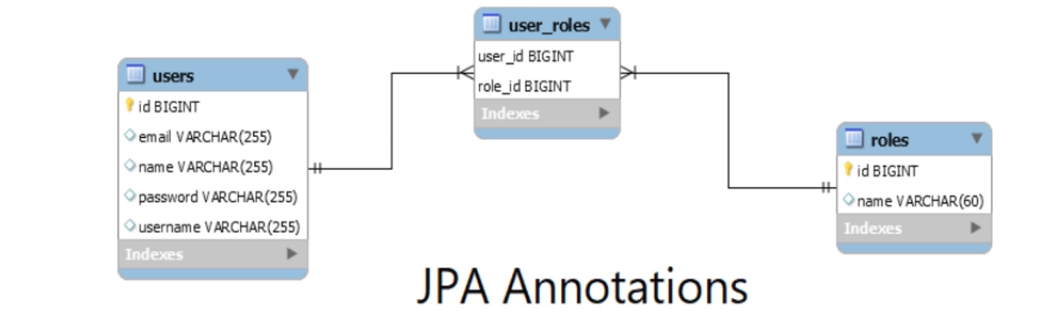
**Step 1: Update SecurityConfig class as shown below:**

package com.springboot.blog.config;  
  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.http.HttpMethod;  
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;  
import org.springframework.security.config.annotation.web.builders.HttpSecurity;  
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;  
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;  
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.crypto.bcrypt.BCryptPasswordEncoder;  
import org.springframework.security.crypto.password.PasswordEncoder;  
import org.springframework.security.provisioning.InMemoryUserDetailsManager;  
  
  
@Configuration  
@EnableWebSecurity  
@EnableGlobalMethodSecurity(prePostEnabled = true)  
public class SecurityConfig extends WebSecurityConfigurerAdapter {  
  
  
 @Bean  
 PasswordEncoder passwordEncoder(){  
 return new BCryptPasswordEncoder();  
 }  
  
 @Override  
 protected void configure(HttpSecurity http) throws Exception {  
 http  
 .csrf().disable()  
 .authorizeRequests()  
 .antMatchers(HttpMethod.*GET*, "/api/\*\*").permitAll()  
 .anyRequest()  
 .authenticated()  
 .and()  
 .httpBasic();  
 }  
  
  
  
 @Override  
 @Bean  
 protected UserDetailsService userDetailsService() {  
 UserDetails ramesh = User.*builder*().username("pankaj").password(passwordEncoder()  
 .encode("password")).roles("USER").build();  
 UserDetails admin = User.*builder*().username("admin").password(passwordEncoder()  
 .encode("admin")).roles("ADMIN").build();  
 return new InMemoryUserDetailsManager(ramesh, admin);  
 }  
}

**Step 2: Add @PreAuthorize(“hasRole(‘ADMIN’)”) Annotation in controller layer**

package com.springboot.blog.controller;  
  
import com.springboot.blog.payload.PostDto;  
import com.springboot.blog.payload.PostResponse;  
import com.springboot.blog.service.PostService;  
import com.springboot.blog.utils.AppConstants;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.security.access.prepost.PreAuthorize;  
import org.springframework.web.bind.annotation.\*;  
  
import javax.validation.Valid;  
import java.util.List;  
  
@RestController  
@RequestMapping("/api/posts")  
public class PostController {  
  
 private PostService postService;  
  
 public PostController(PostService postService) {  
 this.postService = postService;  
 }  
  
 @PreAuthorize("hasRole('ADMIN')")  
 *// create blog post rest api* @PostMapping  
 public ResponseEntity<PostDto> createPost(@Valid @RequestBody PostDto postDto){  
 return new ResponseEntity<>(postService.createPost(postDto), HttpStatus.*CREATED*);  
 }  
  
 *// get all posts rest api* @GetMapping  
 public PostResponse getAllPosts(  
 @RequestParam(value = "pageNo", defaultValue = AppConstants.*DEFAULT\_PAGE\_NUMBER*, required = false) int pageNo,  
 @RequestParam(value = "pageSize", defaultValue = AppConstants.*DEFAULT\_PAGE\_SIZE*, required = false) int pageSize,  
 @RequestParam(value = "sortBy", defaultValue = AppConstants.*DEFAULT\_SORT\_BY*, required = false) String sortBy,  
 @RequestParam(value = "sortDir", defaultValue = AppConstants.*DEFAULT\_SORT\_DIRECTION*, required = false) String sortDir  
 ){  
 return postService.getAllPosts(pageNo, pageSize, sortBy, sortDir);  
 }  
  
 *// get post by id* @GetMapping("/{id}")  
 public ResponseEntity<PostDto> getPostById(@PathVariable(name = "id") long id){  
 return ResponseEntity.*ok*(postService.getPostById(id));  
 }  
  
 @PreAuthorize("hasRole('ADMIN')")  
 *// update post by id rest api* @PutMapping("/{id}")  
 public ResponseEntity<PostDto> updatePost(@Valid @RequestBody PostDto postDto, @PathVariable(name = "id") long id){  
  
 PostDto postResponse = postService.updatePost(postDto, id);  
  
 return new ResponseEntity<>(postResponse, HttpStatus.*OK*);  
 }  
  
 @PreAuthorize("hasRole('ADMIN')")  
 *// delete post rest api* @DeleteMapping("/{id}")  
 public ResponseEntity<String> deletePost(@PathVariable(name = "id") long id){  
  
 postService.deletePostById(id);  
  
 return new ResponseEntity<>("Post entity deleted successfully.", HttpStatus.*OK*);  
 }  
}

**Create JPA Entities User & Role**

****

**Step 1: Create user table:**

package com.springboot.blog.entity;  
  
import lombok.Data;  
  
import javax.persistence.\*;  
import java.util.Set;  
  
@Data  
@Entity  
@Table(name = "users", uniqueConstraints = {  
 @UniqueConstraint(columnNames = {"username"}),  
 @UniqueConstraint(columnNames = {"email"})  
})  
public class User {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private long id;  
 private String name;  
 private String username;  
 private String email;  
 private String password;  
  
 @ManyToMany(fetch = FetchType.*EAGER*, cascade = CascadeType.*ALL*)  
 @JoinTable(name = "user\_roles",  
 joinColumns = @JoinColumn(name = "user\_id", referencedColumnName = "id"),  
 inverseJoinColumns = @JoinColumn(name = "role\_id", referencedColumnName = "id"))  
 private Set<Role> roles;  
}

**Step 2: Create Role Entity Class:**

package com.springboot.blog.entity;  
  
import lombok.Getter;  
import lombok.Setter;  
  
import javax.persistence.\*;  
  
@Setter  
@Getter  
@Entity  
@Table(name = "roles")  
public class Role {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private long id;  
  
 @Column(length = 60)  
 private String name;  
}

**Create Repository Layer**

**Step 1: Create UserRepository Layer**

package com.springboot.blog.repository;  
import com.springboot.blog.entity.User;  
import org.springframework.data.domain.Example;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
import java.util.Optional;  
  
public interface UserRepository extends JpaRepository<User, Long> {  
 Optional<User> findByEmail(String email);  
 Optional<User> findByUsernameOrEmail(String username, String email);  
 Optional<User> findByUsername(String username);  
 Boolean existsByUsername(String username);  
 Boolean existsByEmail(String email);  
}

**Step 2: Create RoleRepository Layer**

import com.springboot.blog.entity.Role;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
import java.util.Optional;  
  
public interface RoleRepository extends JpaRepository<Role, Long> {  
 Optional<Role> findByName(String name);  
}

**UserDetailsService Implementation**

**Step 1: Create CustomUserDetailsService class in security package**

package com.springboot.blog.security;  
  
  
import com.springboot.blog.entity.Role;  
import com.springboot.blog.entity.User;  
import com.springboot.blog.repository.UserRepository;  
import org.springframework.security.core.GrantedAuthority;  
import org.springframework.security.core.authority.SimpleGrantedAuthority;  
import org.springframework.security.core.userdetails.UserDetails;  
import org.springframework.security.core.userdetails.UserDetailsService;  
import org.springframework.security.core.userdetails.UsernameNotFoundException;  
import org.springframework.stereotype.Service;  
  
import java.util.Collection;  
import java.util.Set;  
import java.util.stream.Collectors;  
  
@Service  
public class CustomUserDetailsService implements UserDetailsService {  
  
 private UserRepository userRepository;  
  
 public CustomUserDetailsService(UserRepository userRepository) {  
 this.userRepository = userRepository;  
 }  
  
 @Override  
 public UserDetails loadUserByUsername(String usernameOrEmail) throws UsernameNotFoundException {  
 User user = userRepository.findByUsernameOrEmail(usernameOrEmail, usernameOrEmail)  
 .orElseThrow(() ->  
 new UsernameNotFoundException("User not found with username or email:" + usernameOrEmail));  
 return new org.springframework.security.core.userdetails.User(user.getEmail(),  
 user.getPassword(), mapRolesToAuthorities(user.getRoles()));  
 }  
  
 private Collection< ? extends GrantedAuthority> mapRolesToAuthorities(Set<Role> roles){  
 return roles.stream().map(role -> new SimpleGrantedAuthority(role.getName())).collect(Collectors.*toList*());  
 }  
}

**Step 2: Update SecurityConfig File as shown below:**

package com.springboot.blog.config;  
  
import com.springboot.blog.security.CustomUserDetailsService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.http.HttpMethod;  
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;  
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;  
import org.springframework.security.config.annotation.web.builders.HttpSecurity;  
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;  
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;  
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.crypto.bcrypt.BCryptPasswordEncoder;  
import org.springframework.security.crypto.password.PasswordEncoder;  
import org.springframework.security.provisioning.InMemoryUserDetailsManager;  
  
@Configuration  
@EnableWebSecurity  
@EnableGlobalMethodSecurity(prePostEnabled = true)  
public class SecurityConfig extends WebSecurityConfigurerAdapter {  
  
 @Autowired  
 private CustomUserDetailsService userDetailsService;  
  
 @Bean  
 PasswordEncoder passwordEncoder(){  
 return new BCryptPasswordEncoder();  
 }  
  
 @Override  
 protected void configure(HttpSecurity http) throws Exception {  
 http  
 .csrf().disable()  
 .authorizeRequests()  
 .antMatchers(HttpMethod.*GET*, "/api/\*\*").permitAll()  
 .anyRequest()  
 .authenticated()  
 .and()  
 .httpBasic();  
 }  
  
 @Override  
 protected void configure(AuthenticationManagerBuilder auth) throws Exception {  
 auth.userDetailsService(userDetailsService)  
 .passwordEncoder(passwordEncoder());  
 }  
  
 *// @Override  
// @Bean  
// protected UserDetailsService userDetailsService() {  
// UserDetails ramesh = User.builder().username("ramesh").password(passwordEncoder()  
// .encode("password")).roles("USER").build();  
// UserDetails admin = User.builder().username("admin").password(passwordEncoder()  
// .encode("admin")).roles("ADMIN").build();  
// return new InMemoryUserDetailsManager(ramesh, admin);  
// }*}

**Developing Signin Rest API**

**Step 1: Create LoginDto class in payload package:**

import lombok.Data;

@Data

public class LoginDto {

private String usernameOrEmail;

private String password;

}

Step 2: Create AuthController class in controller package:

import com.springboot.blog.payload.LoginDto;  
import com.springboot.blog.repository.UserRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.security.authentication.AuthenticationManager;  
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;  
import org.springframework.security.core.Authentication;  
import org.springframework.security.core.context.SecurityContextHolder;  
import org.springframework.web.bind.annotation.PostMapping;  
import org.springframework.web.bind.annotation.RequestBody;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
@RequestMapping("/api/auth")  
public class AuthController {  
  
 @Autowired  
 private AuthenticationManager authenticationManager;  
  
 @PostMapping("/signin")  
 public ResponseEntity<String> authenticateUser(@RequestBody LoginDto loginDto){  
 Authentication authentication = authenticationManager.authenticate(  
 new UsernamePasswordAuthenticationToken(loginDto.getUsernameOrEmail(), loginDto.getPassword())  
 );  
 SecurityContextHolder.*getContext*().setAuthentication(authentication);  
 return new ResponseEntity<>("User signed-in successfully!.", HttpStatus.*OK*);  
 }  
}

Step 3: Update SecurityConfig File:

import com.springboot.blog.security.CustomUserDetailsService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.http.HttpMethod;  
import org.springframework.security.authentication.AuthenticationManager;  
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;  
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;  
import org.springframework.security.config.annotation.web.builders.HttpSecurity;  
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;  
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;  
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.crypto.bcrypt.BCryptPasswordEncoder;  
import org.springframework.security.crypto.password.PasswordEncoder;  
import org.springframework.security.provisioning.InMemoryUserDetailsManager;  
  
@Configuration  
@EnableWebSecurity  
@EnableGlobalMethodSecurity(prePostEnabled = true)  
public class SecurityConfig extends WebSecurityConfigurerAdapter {  
  
 @Autowired  
 private CustomUserDetailsService userDetailsService;  
  
 @Bean  
 PasswordEncoder passwordEncoder(){  
 return new BCryptPasswordEncoder();  
 }  
  
 @Override  
 @Bean  
 public AuthenticationManager authenticationManagerBean() throws Exception {  
 return super.authenticationManagerBean();  
 }  
  
 @Override  
 protected void configure(HttpSecurity http) throws Exception {  
 http  
 .csrf().disable()  
 .authorizeRequests()  
 .antMatchers(HttpMethod.*GET*, "/api/\*\*").permitAll()  
 .antMatchers("/api/auth/\*\*").permitAll()  
 .anyRequest()  
 .authenticated()  
 .and()  
 .httpBasic();  
 }  
  
 @Override  
 protected void configure(AuthenticationManagerBuilder auth) throws Exception {  
 auth.userDetailsService(userDetailsService)  
 .passwordEncoder(passwordEncoder());  
 }  
  
 *// @Override  
// @Bean  
// protected UserDetailsService userDetailsService() {  
// UserDetails ramesh = User.builder().username("ramesh").password(passwordEncoder()  
// .encode("password")).roles("USER").build();  
// UserDetails admin = User.builder().username("admin").password(passwordEncoder()  
// .encode("admin")).roles("ADMIN").build();  
// return new InMemoryUserDetailsManager(ramesh, admin);  
// }*}

Developing SignUp Feature Rest API

Step 1: Update AuthController class as shown below

package com.springboot.blog.controller;  
  
import com.springboot.blog.entity.Role;  
import com.springboot.blog.entity.User;  
import com.springboot.blog.payload.LoginDto;  
import com.springboot.blog.payload.SignUpDto;  
import com.springboot.blog.repository.RoleRepository;  
import com.springboot.blog.repository.UserRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.security.authentication.AuthenticationManager;  
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;  
import org.springframework.security.core.Authentication;  
import org.springframework.security.core.context.SecurityContextHolder;  
import org.springframework.security.crypto.password.PasswordEncoder;  
import org.springframework.web.bind.annotation.PostMapping;  
import org.springframework.web.bind.annotation.RequestBody;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
import java.util.Collections;  
  
@RestController  
@RequestMapping("/api/auth")  
public class AuthController {  
  
 @Autowired  
 private AuthenticationManager authenticationManager;  
  
 @Autowired  
 private UserRepository userRepository;  
  
 @Autowired  
 private RoleRepository roleRepository;  
  
 @Autowired  
 private PasswordEncoder passwordEncoder;  
  
 @PostMapping("/signin")  
 public ResponseEntity<String> authenticateUser(@RequestBody LoginDto loginDto){  
 Authentication authentication = authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(  
 loginDto.getUsernameOrEmail(), loginDto.getPassword()));  
  
 SecurityContextHolder.*getContext*().setAuthentication(authentication);  
 return new ResponseEntity<>("User signed-in successfully!.", HttpStatus.*OK*);  
 }  
  
 @PostMapping("/signup")  
 public ResponseEntity<?> registerUser(@RequestBody SignUpDto signUpDto){  
  
 *// add check for username exists in a DB* if(userRepository.existsByUsername(signUpDto.getUsername())){  
 return new ResponseEntity<>("Username is already taken!", HttpStatus.*BAD\_REQUEST*);  
 }  
  
 *// add check for email exists in DB* if(userRepository.existsByEmail(signUpDto.getEmail())){  
 return new ResponseEntity<>("Email is already taken!", HttpStatus.*BAD\_REQUEST*);  
 }  
  
 *// create user object* User user = new User();  
 user.setName(signUpDto.getName());  
 user.setUsername(signUpDto.getUsername());  
 user.setEmail(signUpDto.getEmail());  
 user.setPassword(passwordEncoder.encode(signUpDto.getPassword()));  
  
 Role roles = roleRepository.findByName("ROLE\_ADMIN").get();  
 user.setRoles(Collections.*singleton*(roles));  
  
 userRepository.save(user);  
  
 return new ResponseEntity<>("User registered successfully", HttpStatus.*OK*);  
  
 }  
}

Step 2: Develop SignUpDto payload class:

import lombok.Data;  
  
@Data  
public class SignUpDto {  
 private String name;  
 private String username;  
 private String email;  
 private String password;  
}

**Developing JWT Token**

**For JWT Token add the following dependency:**

[<dependency>](file:///C:\Users\Admin\Downloads\springboot-blog-rest-api%20(5)\springboot-blog-rest-api\pom.xml)

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.1</version>

</dependency>

**Step 1: In security package create JwtAuthenticationEntryPoint**

**import org.springframework.security.core.AuthenticationException;**

**import org.springframework.security.web.AuthenticationEntryPoint;**

**import org.springframework.stereotype.Component;**

**import javax.servlet.ServletException;**

**import javax.servlet.http.HttpServletRequest;**

**import javax.servlet.http.HttpServletResponse;**

**import java.io.IOException;**

**@Component**

**public class JwtAuthenticationEntryPoint implements AuthenticationEntryPoint {**

**@Override**

**public void commence(HttpServletRequest request,**

**HttpServletResponse response,**

**AuthenticationException authException) throws IOException, ServletException {**

**response.sendError(HttpServletResponse.SC\_UNAUTHORIZED, authException.getMessage());**

**}**

**}**

**Step 2: Update application.properties file:**

**## App Properties**

**app.jwt-secret= JWTSecretKey**

**app.jwt-expiration-milliseconds = 604800000**

**Step 3: Develop JwtAuthenticationFilter class in security package:**

**package com.springboot.blog.security;**

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

**import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;**

**import org.springframework.security.core.context.SecurityContextHolder;**

**import org.springframework.security.core.userdetails.UserDetails;**

**import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;**

**import org.springframework.util.StringUtils;**

**import org.springframework.web.filter.OncePerRequestFilter;**

**import javax.servlet.FilterChain;**

**import javax.servlet.ServletException;**

**import javax.servlet.http.HttpServletRequest;**

**import javax.servlet.http.HttpServletResponse;**

**import java.io.IOException;**

**public class JwtAuthenticationFilter extends OncePerRequestFilter {**

**// inject dependencies**

**@Autowired**

**private JwtTokenProvider tokenProvider;**

**@Autowired**

**private CustomUserDetailsService customUserDetailsService;**

**@Override**

**protected void doFilterInternal(HttpServletRequest request,**

**HttpServletResponse response,**

**FilterChain filterChain) throws ServletException, IOException {**

**// get JWT (token) from http request**

**String token = getJWTfromRequest(request);**

**// validate token**

**if(StringUtils.hasText(token) && tokenProvider.validateToken(token)){**

**// get username from token**

**String username = tokenProvider.getUsernameFromJWT(token);**

**// load user associated with token**

**UserDetails userDetails = customUserDetailsService.loadUserByUsername(username);**

**UsernamePasswordAuthenticationToken authenticationToken = new UsernamePasswordAuthenticationToken(**

**userDetails, null, userDetails.getAuthorities()**

**);**

**authenticationToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));**

**// set spring security**

**SecurityContextHolder.getContext().setAuthentication(authenticationToken);**

**}**

**filterChain.doFilter(request, response);**

**}**

**// Bearer <accessToken>**

**private String getJWTfromRequest(HttpServletRequest request){**

**String bearerToken = request.getHeader("Authorization");**

**if(StringUtils.hasText(bearerToken) && bearerToken.startsWith("Bearer ")){**

**return bearerToken.substring(7, bearerToken.length());**

**}**

**return null;**

**}**

**}**

**Step 4: Develop JwtTokenProvider class in security package:**

**package com.springboot.blog.security;**

**import com.springboot.blog.exception.BlogAPIException;**

**import io.jsonwebtoken.\*;**

**import org.springframework.beans.factory.annotation.Value;**

**import org.springframework.http.HttpStatus;**

**import org.springframework.security.core.Authentication;**

**import org.springframework.stereotype.Component;**

**import java.util.Date;**

**@Component**

**public class JwtTokenProvider {**

**@Value("${app.jwt-secret}")**

**private String jwtSecret;**

**@Value("${app.jwt-expiration-milliseconds}")**

**private int jwtExpirationInMs;**

**// generate token**

**public String generateToken(Authentication authentication){**

**String username = authentication.getName();**

**Date currentDate = new Date();**

**Date expireDate = new Date(currentDate.getTime() + jwtExpirationInMs);**

**String token = Jwts.builder()**

**.setSubject(username)**

**.setIssuedAt(new Date())**

**.setExpiration(expireDate)**

**.signWith(SignatureAlgorithm.HS512, jwtSecret)**

**.compact();**

**return token;**

**}**

**// get username from the token**

**public String getUsernameFromJWT(String token){**

**Claims claims = Jwts.parser()**

**.setSigningKey(jwtSecret)**

**.parseClaimsJws(token)**

**.getBody();**

**return claims.getSubject();**

**}**

**// validate JWT token**

**public boolean validateToken(String token){**

**try{**

**Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(token);**

**return true;**

**}catch (SignatureException ex){**

**throw new BlogAPIException(HttpStatus.BAD\_REQUEST, "Invalid JWT signature");**

**} catch (MalformedJwtException ex) {**

**throw new BlogAPIException(HttpStatus.BAD\_REQUEST, "Invalid JWT token");**

**} catch (ExpiredJwtException ex) {**

**throw new BlogAPIException(HttpStatus.BAD\_REQUEST, "Expired JWT token");**

**} catch (UnsupportedJwtException ex) {**

**throw new BlogAPIException(HttpStatus.BAD\_REQUEST, "Unsupported JWT token");**

**} catch (IllegalArgumentException ex) {**

**throw new BlogAPIException(HttpStatus.BAD\_REQUEST, "JWT claims string is empty.");**

**}**

**}**

**}**

**Step 4: Update AuthController class:**

**import com.springboot.blog.entity.Role;**

**import com.springboot.blog.entity.User;**

**import com.springboot.blog.payload.JWTAuthResponse;**

**import com.springboot.blog.payload.LoginDto;**

**import com.springboot.blog.payload.SignUpDto;**

**import com.springboot.blog.repository.RoleRepository;**

**import com.springboot.blog.repository.UserRepository;**

**import com.springboot.blog.security.JwtTokenProvider;**

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

**import org.springframework.http.HttpStatus;**

**import org.springframework.http.ResponseEntity;**

**import org.springframework.security.authentication.AuthenticationManager;**

**import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;**

**import org.springframework.security.core.Authentication;**

**import org.springframework.security.core.context.SecurityContextHolder;**

**import org.springframework.security.crypto.password.PasswordEncoder;**

**import org.springframework.web.bind.annotation.PostMapping;**

**import org.springframework.web.bind.annotation.RequestBody;**

**import org.springframework.web.bind.annotation.RequestMapping;**

**import org.springframework.web.bind.annotation.RestController;**

**import java.util.Collections;**

**@RestController**

**@RequestMapping("/api/auth")**

**public class AuthController {**

**@Autowired**

**private AuthenticationManager authenticationManager;**

**@Autowired**

**private UserRepository userRepository;**

**@Autowired**

**private RoleRepository roleRepository;**

**@Autowired**

**private PasswordEncoder passwordEncoder;**

**@Autowired**

**private JwtTokenProvider tokenProvider;**

**@PostMapping("/signin")**

**public ResponseEntity<JWTAuthResponse> authenticateUser(@RequestBody LoginDto loginDto){**

**Authentication authentication = authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(**

**loginDto.getUsernameOrEmail(), loginDto.getPassword()));**

**SecurityContextHolder.getContext().setAuthentication(authentication);**

**// get token form tokenProvider**

**String token = tokenProvider.generateToken(authentication);**

**return ResponseEntity.ok(new JWTAuthResponse(token));**

**}**

**@PostMapping("/signup")**

**public ResponseEntity<?> registerUser(@RequestBody SignUpDto signUpDto){**

**// add check for username exists in a DB**

**if(userRepository.existsByUsername(signUpDto.getUsername())){**

**return new ResponseEntity<>("Username is already taken!", HttpStatus.BAD\_REQUEST);**

**}**

**// add check for email exists in DB**

**if(userRepository.existsByEmail(signUpDto.getEmail())){**

**return new ResponseEntity<>("Email is already taken!", HttpStatus.BAD\_REQUEST);**

**}**

**// create user object**

**User user = new User();**

**user.setName(signUpDto.getName());**

**user.setUsername(signUpDto.getUsername());**

**user.setEmail(signUpDto.getEmail());**

**user.setPassword(passwordEncoder.encode(signUpDto.getPassword()));**

**Role roles = roleRepository.findByName("ROLE\_ADMIN").get();**

**user.setRoles(Collections.singleton(roles));**

**userRepository.save(user);**

**return new ResponseEntity<>("User registered successfully", HttpStatus.OK);**

**}**

**}**

**Step 5: Create payload class JWTAuthResponse**

**public class JWTAuthResponse {**

**private String accessToken;**

**private String tokenType = "Bearer";**

**public JWTAuthResponse(String accessToken) {**

**this.accessToken = accessToken;**

**}**

**public void setAccessToken(String accessToken) {**

**this.accessToken = accessToken;**

**}**

**public void setTokenType(String tokenType) {**

**this.tokenType = tokenType;**

**}**

**public String getAccessToken() {**

**return accessToken;**

**}**

**public String getTokenType() {**

**return tokenType;**

**}**

**}**