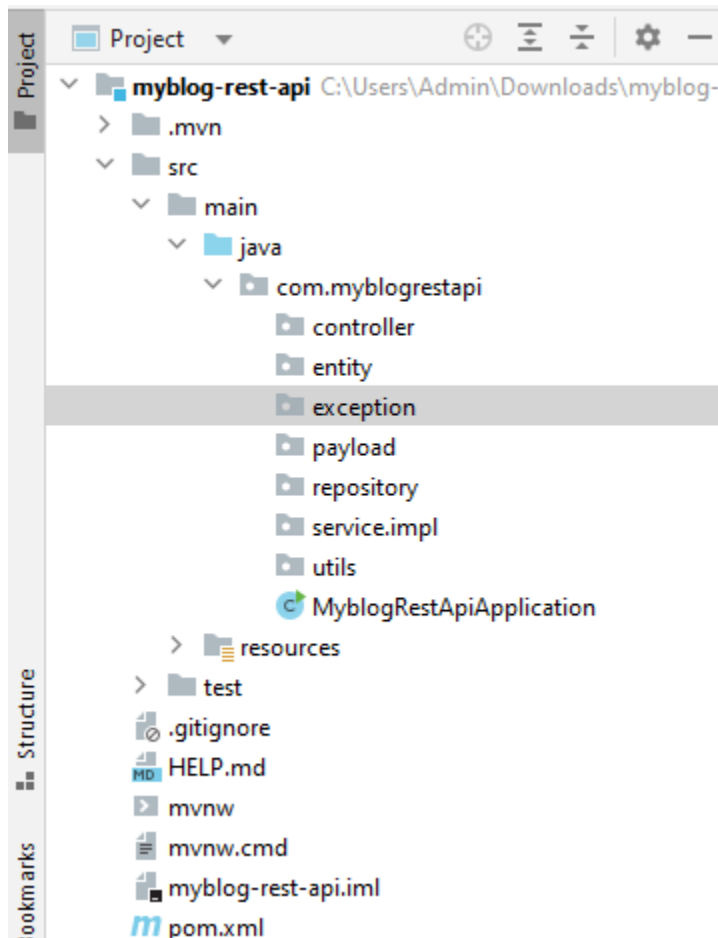


Developing restful web services in spring boot

1. Create Spring boot project with following dependencies:

Project <input type="radio"/> Gradle - Groovy <input type="radio"/> Gradle - Kotlin <input checked="" type="radio"/> Maven Spring Boot <input type="radio"/> 3.0.1 (SNAPSHOT) <input type="radio"/> 3.0.0 <input type="radio"/> 2.7.7 (SNAPSHOT) <input checked="" type="radio"/> 2.7.6 Project Metadata Group <input type="text" value="com.myblog-rest-api"/> Artifact <input type="text" value="myblog-rest-api"/> Name <input type="text" value="myblog-rest-api"/> Description <input type="text" value="Restful web services"/> Package name <input type="text" value="com.myblog-rest-api"/> Packaging <input checked="" type="radio"/> Jar <input type="radio"/> War	Language <input checked="" type="radio"/> Java <input type="radio"/> Kotlin <input type="radio"/> Groovy Dependencies ADD DEPENDENCIES... CTRL + B Spring Web WEB Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container. MySQL Driver SQL MySQL JDBC and R2DBC driver. Lombok DEVELOPER TOOLS Java annotation library which helps to reduce boilerplate code. Spring Boot DevTools DEVELOPER TOOLS Provides fast application restarts, LiveReload, and configurations for enhanced development experience. Spring Data JPA SQL Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.
---	---

2. 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 GetMapping 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 UpdatePost 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);
}

```

```

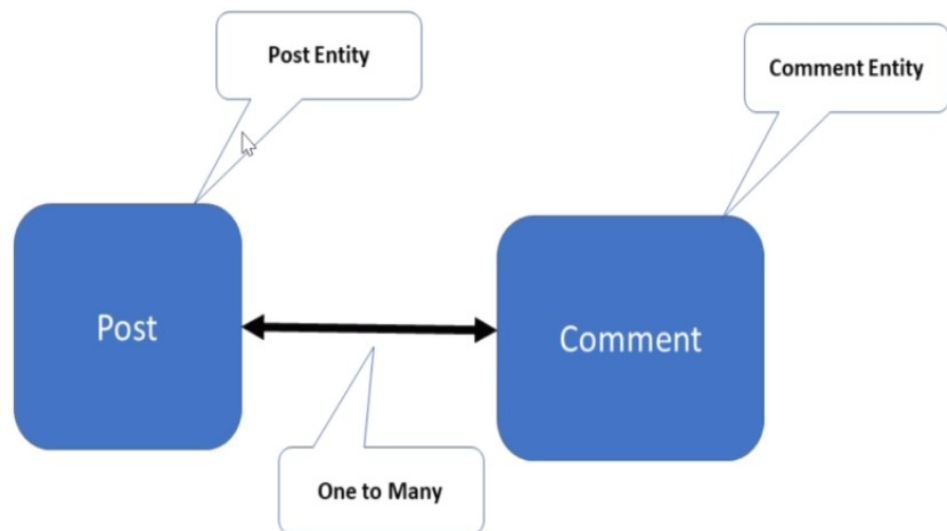
// convert Entity into DTO
private PostDto mapToDTO(Post post){
    PostDto postDto = new PostDto();
    postDto.setId(post.getId());
    postDto.setTitle(post.getTitle());
    postDto.setDescription(post.getDescription());
    postDto.setContent(post.getContent());
    return postDto;
}

// 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;
}
}

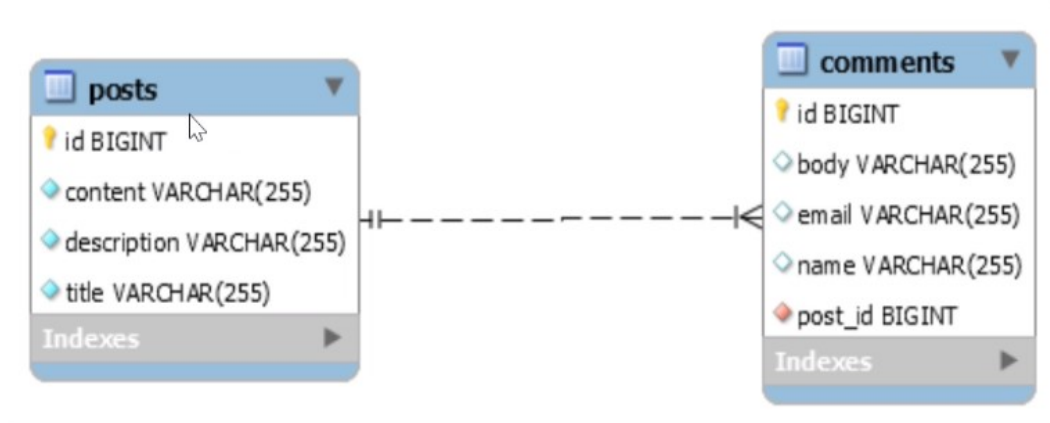
```

Create Comments API Later

One to Many Relationship (bi-directional)



ER(Entity Relationship Diagram)



URL Documentation with status code:

REST APIs for Comment Resource

HTTP Method	URL Path	Status Code	Description
GET	/api/posts/{postId}/comments	200 (OK)	Get all comments which belongs to post with id = postId
GET	/api/posts/{postId}/comments/{id}	200 (OK)	Get comment by id if it belongs to post with id = postId
POST	/api/posts/{postId}/comments	201 (Created)	Create new comment for post with id = postId
PUT	/api/posts/{postId}/comments/{id}	200 (OK)	Update comment by id if it belongs to post with id = postId
DELETE	/api/posts/{postId}/comments/{id}	200 (OK)	Delete comment by id if it belongs to post with id = postId

Step 1: Create Comment Entity Class and do oneToMan many 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 getCommentByld(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.findById(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}>

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.findById(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;  
    }  
}
```



```

        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){


        if(result.hasErrors()){
            return new ResponseEntity<>(result.getFieldError().getDefaultMessage(),
            HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }
}
```

```

        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>
```

```
<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.EnableGlob
alMethodSecurity;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecu
rity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityCo
nfigurerAdapter;
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 user =
            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);
}

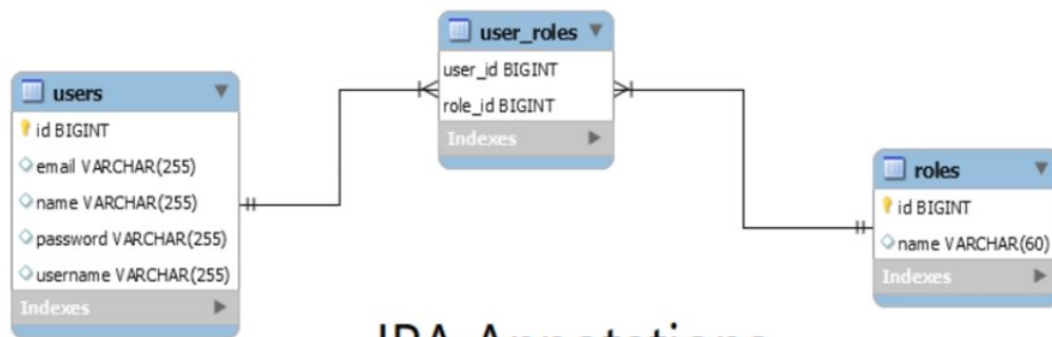
@PreAuthorize("hasRole('ADMIN')")
// delete post rest api
@DeleteMapping("/{id}")
public ResponseEntity<String> deletePost(@PathVariable(name = "id") long
id) {

    postService.deletePostById(id);

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

```

Create JPA Entities User & Role



JPA Annotations

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.Authent
icationManagerBuilder;
import
org.springframework.security.config.annotation.method.configuration.EnableGlob
alMethodSecurity;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecu
rity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityCo
nfigurerAdapter;
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.Authent
icationManagerBuilder;
import
org.springframework.security.config.annotation.method.configuration.EnableGlob
alMethodSecurity;
import
org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecu
rity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityCo
nfigurerAdapter;
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>

<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,
```

```

        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;
}

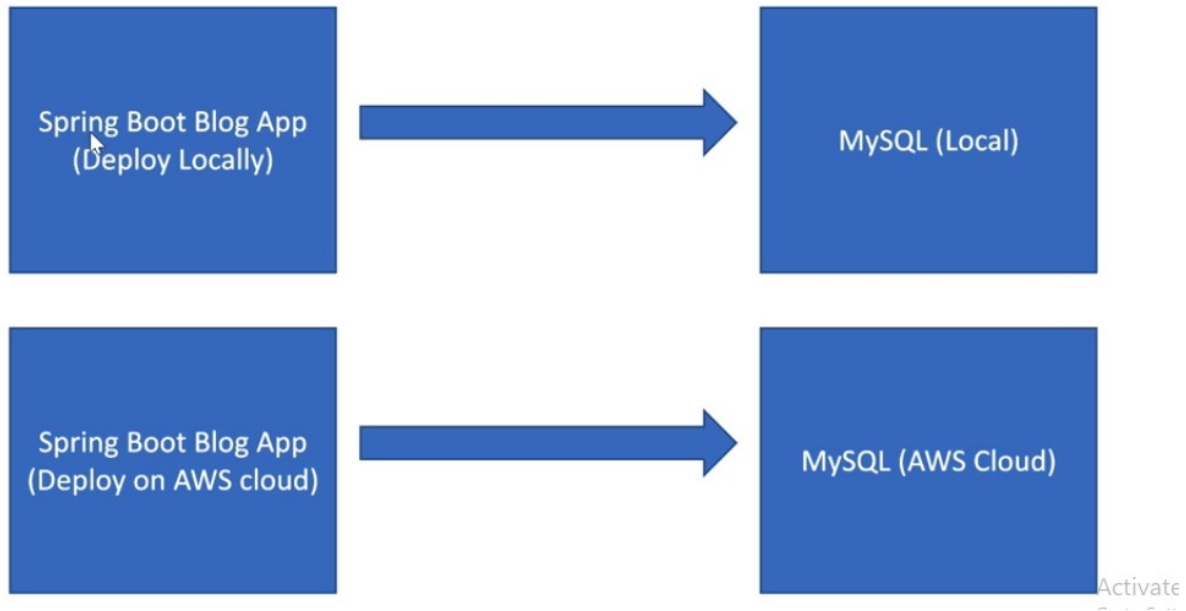
public String getAccessToken() {
    return accessToken;
}

public String getTokenType() {
    return tokenType;
}
}
```

Deployment Of Spring Boot Application Amazon AWS Cloud:

Some Of the important cloud service provider

1. AWS
2. Heroku
3. Google Cloud
4. Microsoft Azure
5. Oracle
6. IBM Cloud



Important AWS Services every java developer should be aware of:

1. **Amazon EC2** - Amazon Elastic Compute Cloud (EC2) is a web service that provides resizable computing capacity in the cloud. It allows users to rent virtual machines (VMs), known as instances, which can be used to run a variety of different operating systems and applications. With EC2, users can easily scale their computing resources up or down as needed, paying only for the resources they actually use. This makes it an ideal service for applications that have varying compute needs, such as web servers, batch processing, and big data processing. EC2 also provides a variety of different instance types, each optimized for different types of workloads, such as compute-optimized, memory-optimized, and storage-optimized instances. Additionally, EC2 also provides features such as load balancing, auto-scaling, and virtual private cloud (VPC) to give users more control and security over their instances

2. **AWS Elastic Beanstalk** -

Amazon Elastic Beanstalk is a fully managed service offered by AWS that makes it easy to deploy, run, and scale web applications and services. It supports several programming languages including Java, .NET, PHP, Node.js, Python, Ruby, and Go. Elastic Beanstalk handles the provisioning of the infrastructure resources, load balancing, and automatic scaling, allowing developers to focus on writing code for their application. The service also includes monitoring and logging features, so developers can easily track the performance and troubleshoot issues.

Elastic Beanstalk provides a simple, unified user interface to deploy and manage web applications, as well as a command-line interface and APIs for more

advanced users. It integrates with other AWS services such as Amazon RDS, Amazon S3, Amazon SNS, and AWS ElastiCache. Elastic Beanstalk also provides a feature called "platform versions" that allows developers to choose a specific version of the language runtime, web server, and other software components to use with their application.

3. AMAZON RDS -

Amazon Relational Database Service (RDS) is a web service that makes it easy to set up, operate, and scale a relational database in the cloud. RDS supports several popular database engines including MySQL, PostgreSQL, Oracle, Microsoft SQL Server, and Amazon Aurora.

RDS automates many of the time-consuming tasks typically associated with managing a relational database, such as provisioning, patching, backup, and recovery. It also provides features such as automatic failover, read replicas, and a point-in-time restore, which help to improve the availability and durability of the database. In addition, RDS allows users to easily scale the resources allocated to a database up or down as needed, and it also offers a variety of different instance types optimized for different types of workloads.

RDS also provides a feature called "Multi-AZ Deployments" that allows the user to create a primary DB instance and synchronously replicate the data to a standby instance in a different availability zone (AZ) for failover capabilities. This provides an automatic failover to the standby instance in the event of a planned or unplanned outage of the primary instance.

4. **S3 Service** - Amazon S3 (Simple Storage Service) is a cloud-based object storage service offered by Amazon Web Services (AWS). It allows users to store and retrieve any amount of data, at any time, from anywhere on the internet. S3 provides a simple web services interface that can be used to store and retrieve any amount of data, at any time, from anywhere on the web. It is designed for storing and retrieving large amounts of data, such as photos, videos, and backups. S3 is widely used for a variety of applications including, cloud storage, backup and archiving, big data analytics, disaster recovery, and more.

5. Amazon Route 53 -

Amazon Route 53 is a highly available and scalable Domain Name System (DNS) web service offered by AWS. It translates human-friendly domain names, such as www.example.com, into the IP addresses, such as 192.0.2.1, that computers use to identify each other on the internet. Route 53 is designed to give developers and businesses a reliable and cost-effective way to route end users to internet applications.

Route 53 provides a variety of different routing types, such as simple routing, which routes traffic to a single resource, such as a web server, and complex

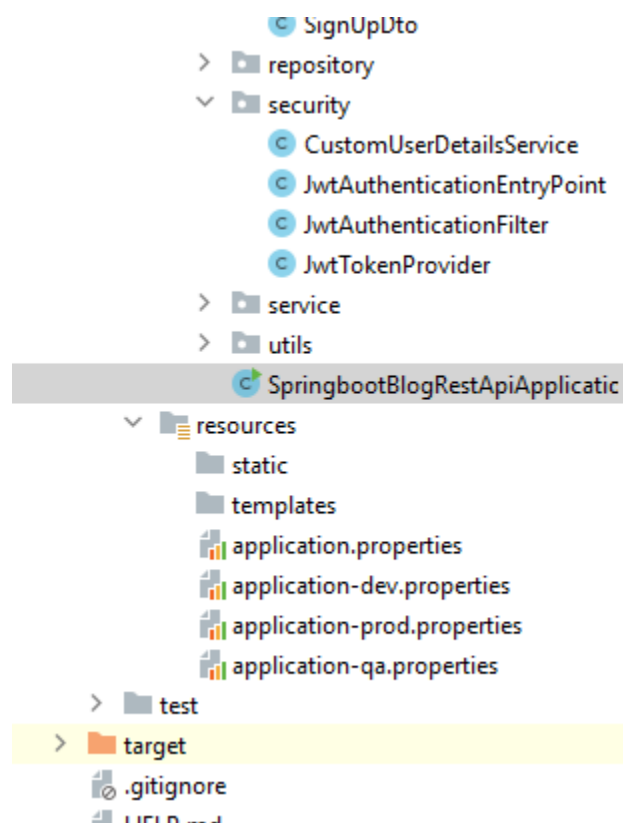
routing, which allows you to route traffic based on factors such as the geographic location of your users, the health of your resources, and the routing policies that you specify.

Route 53 also provides a feature called "Health Check", that allows the user to monitor the health of their resources, such as web servers, and route traffic to healthy resources. It also integrates with other AWS services such as Amazon CloudFront, Elastic Load Balancing, and AWS Elastic Beanstalk.

It also provides a feature called "Traffic Flow" that allows the user to create a visual representation of their routing policies and test how the traffic will be routed before it's updated.

Using Profiles In Spring Boot Application

Step 1: Create Following Properties file:



application.properties file content:

```
#spring.datasource.url = jdbc:mysql://localhost:3306/myblog
#spring.datasource.username = root
#spring.datasource.password = test

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

# Hibernate ddl auto (create, create-drop, validate, update)
#spring.jpa.hibernate.ddl-auto = update

# App Properties
app.jwt-secret= JWTSecretKey
app.jwt-expiration-milliseconds = 604800000

spring.profiles.active=prod
```

application-dev.properties content:

```
spring.datasource.url = jdbc:mysql://localhost:3306/myblog
spring.datasource.username = root
spring.datasource.password = test

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

# Hibernate ddl auto (create, create-drop, validate, update)
spring.jpa.hibernate.ddl-auto = update
```

application-qa.properties content:

```
spring.datasource.url = jdbc:mysql://localhost:3306/myblog
spring.datasource.username = root
spring.datasource.password = test

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

# Hibernate ddl auto (create, create-drop, validate, update)
spring.jpa.hibernate.ddl-auto = update
```

application-prod.properties content:

```
spring.datasource.url = jdbc:mysql://localhost:3306/myblog
spring.datasource.username = root
spring.datasource.password = test

# hibernate properties
```



```

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

# Hibernate ddl auto (create, create-drop, validate, update)
spring.jpa.hibernate.ddl-auto = update

```

Step 2: Create default (Meta Data) in tables

Manually Enter Data into Roles Table Using Command Line Runner

```

package com.springboot.blog;

import com.springboot.blog.entity.Role;
import com.springboot.blog.repository.RoleRepository;
import org.modelmapper.ModelMapper;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
public class SpringbootBlogRestApiApplication implements CommandLineRunner {

    @Autowired
    private RoleRepository roleRepository;

    @Bean
    public ModelMapper modelMapper() {
        return new ModelMapper();
    }

    public static void main(String[] args) {
        SpringApplication.run(SpringbootBlogRestApiApplication.class, args);
    }

    @Override
    public void run(String... args) throws Exception {
        Role adminRole = new Role();
        adminRole.setName("ROLE_ADMIN");
        roleRepository.save(adminRole);

        Role userRole = new Role();
        userRole.setName("ROLE_USER");
        roleRepository.save(userRole);
    }
}

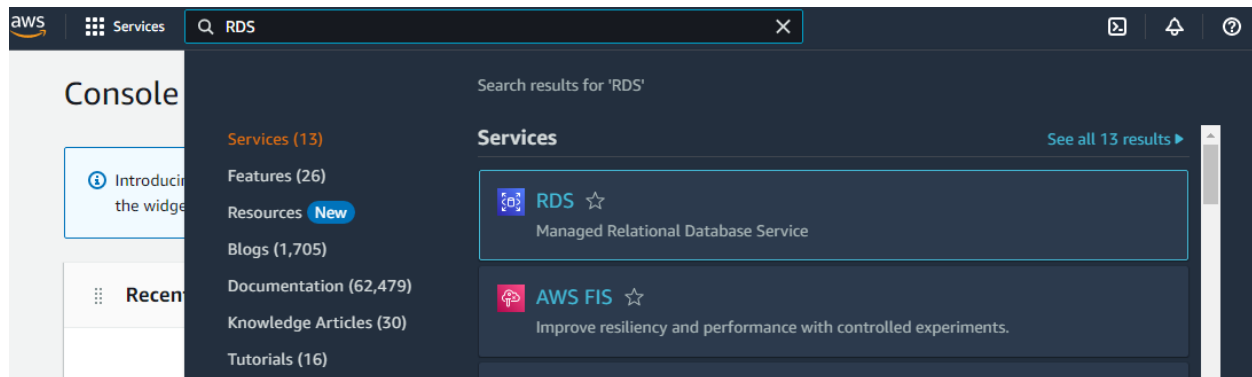
```

Step 3: Create Amazon AWS Account

Link: https://portal.aws.amazon.com/billing/signup?refid=14a4002d-4936-4343-8211-b5a150ca592b&redirect_url=https%3A%2F%2Faws.amazon.com%2Fregistration-confirmation#/start/email

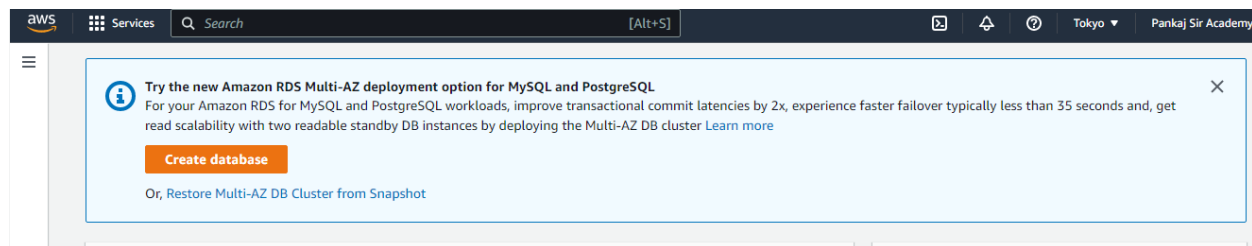
Creating Environment and setting up database in AWS

Step 1: Search for RDS Service:



And Click on Dashboard

Step 2: Click on create database



Step 3: Select MySQL Database:

aws

Services

Search


[Alt+S]


Choose a database creation method [Info](#)


☒ **Standard create**
You set all of the configuration options, including ones for availability, security, backups, and maintenance.

☐ **Easy create**
Use recommended best-practice configurations. Some configuration options can be changed after the database is created.

Engine options

☐ **Amazon Aurora**


☒ **MySQL**


☐ **MariaDB**


Step 4: Select Version and Free Tier

☒ **Show versions that support the Multi-AZ DB cluster** [Info](#)
Create a Multi-AZ DB cluster with one primary DB instance and two readable standby DB instances. Multi-AZ DB clusters provide up to 2x faster transaction commit latency and automatic failover in typically under 35 seconds.

☒ **Show versions that support the Amazon RDS Optimized Writes** [Info](#)
Amazon RDS Optimized Writes improves write throughput by up to 2x at no additional cost.

Engine Version

MySQL 8.0.28

Templates

Choose a sample template to meet your use case.

☐ **Production**
Use defaults for high availability and fast, consistent performance.

☐ **Dev/Test**
This instance is intended for development use outside of a production environment.

☒ **Free tier**
Use RDS Free Tier to develop new applications, test existing applications, or gain hands-on experience with Amazon RDS. [Info](#)

Step 5: Give Database Instance Name, Username(root) & Password(Mysql123\$):

DB instance identifier [Info](#)

Type a name for your DB instance. The name must be unique across all DB instances owned by your AWS account in the current AWS Region.

The DB instance identifier is case-insensitive, but is stored as all lowercase (as in "mydbinstance"). Constraints: 1 to 60 alphanumeric characters or hyphens. First character must be a letter. Can't contain two consecutive hyphens. Can't end with a hyphen.

▼ Credentials Settings

Master username [Info](#)

Type a login ID for the master user of your DB instance.

1 to 16 alphanumeric characters. First character must be a letter.

- ☐ **Manage master credentials in AWS Secrets Manager - new**
Manage master user credentials in Secrets Manager. RDS can generate a password for you and manage it throughout its lifecycle.
- ☐ **Auto generate a password**
Amazon RDS can generate a password for you, or you can specify your own password.

Master password [Info](#)

Constraints: At least 8 printable ASCII characters. Can't contain any of the following: / (slash), '(single quote), "(double quote) and @ (at sign).

Confirm master password [Info](#)

Instance configuration

The DB instance configuration options below are limited to those supported by the engine that you selected above.

DB instance class [Info](#)

- ☐ Standard classes (includes m classes)
- ☐ Memory optimized classes (includes r and x classes)
- ☒ Burstable classes (includes t classes)

db.t3.micro
2 vCPUs 1 GiB RAM Network: 2,085 Mbps

☐ Include previous generation classes

Storage

Storage type [Info](#)

General Purpose SSD (gp2)
Baseline performance determined by volume size

Allocated storage [Info](#)

Storage

Storage type [Info](#)

General Purpose SSD (gp2)
Baseline performance determined by volume size

Allocated storage [Info](#)

20

GiB

The minimum value is 20 GiB and the maximum value is 6,144 GiB

Storage autoscaling [Info](#)

Provides dynamic scaling support for your database's storage based on your application's needs.

☒ Enable storage autoscaling

Enabling this feature will allow the storage to increase after the specified threshold is exceeded.

Maximum storage threshold [Info](#)

Charges will apply when your database autoscales to the specified threshold

1000

GiB

The minimum value is 22 GiB and the maximum value is 6,144 GiB

Connectivity [Info](#)



Compute resource

Choose whether to set up a connection to a compute resource for this database. Setting up a connection will automatically change connectivity settings so that the compute resource can connect to this database.

☒ **Don't connect to an EC2 compute resource**
Don't set up a connection to a compute resource for this database. You can manually set up a connection to a compute resource later.

☐ **Connect to an EC2 compute resource**
Set up a connection to an EC2 compute resource for this database.

Network type [Info](#)

To use dual-stack mode, make sure that you associate an IPv6 CIDR block with a subnet in the VPC you specify.

☒ **IPv4**
Your resources can communicate only over the IPv4 addressing protocol.

☐ **Dual-stack mode**
Your resources can communicate over IPv4, IPv6, or both.

Virtual private cloud (VPC) [Info](#)

Choose the VPC. The VPC defines the virtual networking environment for this DB instance.

Default VPC (vpc-04d4e878bb0953735) ▼

Only VPCs with a corresponding DB subnet group are listed.

DB subnet group [Info](#)

Choose the DB subnet group. The DB subnet group defines which subnets and IP ranges the DB instance can use in the VPC that you selected.

default ▼

Public access [Info](#)

☒ **Yes**
RDS assigns a public IP address to the database. Amazon EC2 instances and other resources outside of the VPC can connect to your database. Resources inside the VPC can also connect to the database. Choose one or more VPC security groups that specify which resources can connect to the database.

☐ **No**
RDS doesn't assign a public IP address to the database. Only Amazon EC2 instances and other resources inside the VPC can connect to your database. Choose one or more VPC security groups that specify which resources can connect to the database.

VPC security group (firewall) [Info](#)

Choose one or more VPC security groups to allow access to your database. Make sure that the security group rules allow the appropriate incoming traffic.

☒ **Choose existing**
Choose existing VPC security groups

☐ **Create new**
Create new VPC security group

Existing VPC security groups

Choose one or more options ▼

default ✕

Keep all further things default.... Click on create database


Estimated monthly costs

The Amazon RDS Free Tier is available to you for 12 months. Each calendar month, the free tier will allow you to use the Amazon RDS resources listed below for free:

- 750 hrs of Amazon RDS in a Single-AZ db.t2.micro, db.t3.micro or db.t4g.micro Instance.
- 20 GB of General Purpose Storage (SSD).
- 20 GB for automated backup storage and any user-initiated DB Snapshots.

[Learn more about AWS Free Tier.](#)

When your free usage expires or if your application use exceeds the free usage tiers, you simply pay standard, pay-as-you-go service rates as described in the [Amazon RDS Pricing page](#).

 You are responsible for ensuring that you have all of the necessary rights for any third-party products or services that you use with AWS services.

Cancel

Create database

How was your experience creating an Amazon RDS database? [Provide feedback](#)

RDS > Databases



Consider creating a Blue/Green Deployment to minimize downtime during upgrades

You may want to consider using Amazon RDS Blue/Green Deployments and minimize your downtime during upgrades. A Blue/Green Deployment provides a staging environment for changes to production databases. [RDS User Guide](#) [Aurora User Guide](#)



Databases



Group resources



Modify

Actions ▾

Restore from S3

Create database

Filter by databases

< 1 > ⚙

DB identifier	Role	Engine	Region & AZ	Size	Status
pankajsiracademy	Instance	MySQL Community	-	db.t3.micro	Creating

✓ Successfully created database [pankajsiracademy](#)

[View connection details](#)

How was your experience creating an Amazon RDS database? [Provide feedback](#)

RDS > Databases



Consider creating a Blue/Green Deployment to minimize downtime during upgrades

You may want to consider using Amazon RDS Blue/Green Deployments and minimize your downtime during upgrades. A Blue/Green Deployment provides a staging environment for changes to production databases. [RDS User Guide](#) [Aurora User Guide](#)

Databases

☒ Group resources



[Modify](#)

[Actions](#) ▼

[Restore from S3](#)

[Create database](#)

< 1 >

DB identifier	Role	Engine	Region & AZ	Size
pankajsiracademy	Instance	MySQL Community	ap-northeast-1d	db.t3.micro

RDS > Databases



Consider creating a Blue/Green Deployment to minimize downtime during upgrades

You may want to consider using Amazon RDS Blue/Green Deployments and minimize your downtime during upgrades. A Blue/Green Deployment provides a staging environment for changes to production databases. [RDS User Guide](#) [Aurora User Guide](#)

Databases

☒ Group resources



[Modify](#)

[Actions](#) ▼

[Restore from S3](#)

[Create database](#)

< 1 >

DB identifier	Role	Engine	Region & AZ	Size
pankajsiracademy	Instance	MySQL Community	ap-northeast-1d	db.t3.micro

Connectivity & security

Use this end point to connect to the database

Endpoint & port

Endpoint

pankajsiracademy.cmua4mt8llo
b.ap-northeast-
1.rds.amazonaws.com

Port

3306

Networking

Availability Zone
ap-northeast-1d

VPC
vpc-04d4e878bb0953735

Subnet group
default-vpc-
04d4e878bb0953735

Subnets
subnet-0aff303f3c0fafdd1
subnet-09b0507ef4d5bef74

Security

VPC security groups
default (sg-
07faaa7751a4072c1)
✓ Active

Publicly accessible
Yes

Certificate authority [Info](#)
rds-ca-2019

Certificate authority date
August 22, 2024, 10:08 (UTC-
07:00)

Connectivity & security

Monitoring

Logs & events

Configuration

Maintenance & backups

Tags

Connectivity & security

Endpoint & port

Endpoint
pankajsiracademy.cmua4mt8llo
b.ap-northeast-
1.rds.amazonaws.com

Port
3306

Networking

Availability Zone
ap-northeast-1d

VPC
vpc-04d4e878bb0953735

Subnet group
default-vpc-
04d4e878bb0953735

Subnets
subnet-0aff303f3c0fafdd1
subnet-09b0507ef4d5bef74

Security

VPC security groups
default (sg-
07faaa7751a4072c1)
Active

Publicly accessible
Yes

Certificate authority Info
rds-ca-2019

Certificate authority date
August 22, 2024, 10:08 (UTC-

Security Groups (1/1) Info

Actions

Export security groups to CSV

Create security group

Filter security groups

search: sg-07faaa7751a4072c1

Clear filters

	Name	Security group ID	Security group name	VPC ID	Description
<input checked="" type="checkbox"/>	-	sg-07faaa7751a4072c1	default	vpc-04d4e878bb0953735 ...	default VPC security gr...

sg-07faaa7751a4072c1 - default

Details

Inbound rules

Outbound rules

Tags

Owner
658080380044

Inbound rules count
1 Permission entry

Outbound rules count
1 Permission entry

Inbound rules

Outbound rules

Tags

You can now check network connectivity with Reachability Analyzer

Run Reachability Analyzer

Inbound rules (1/1)

Manage tags

Edit inbound rules

Filter security group rules

1

	Name	Security group rule...	IP version	Type	Protocol
<input checked="" type="checkbox"/>	-	sgr-0fc36f58b81a7942a	-	All traffic	All

Inbound rules control the incoming traffic that's allowed to reach the instance.

Inbound rules [Info](#)

Security group rule ID	Type Info	Protocol Info	Port range Info	Source Info	Description - optional Info	
sgr-0fc36f58b81a7942a	All traffic ▼	All	All	Anywhere-IPv4		Delete
-	MySQL/Aurora ▼	TCP	3306	Anywhere-IPv6 My IP Anywh... ▲	sg-07faaa7751a4072c1 0.0.0.0/0	Delete

Step 1: Add rule

Step 2: Select MySQL/Aurora

Step 3: Select Anywhere-IPv4

Step 4: Save rules

Cancel Preview changes **Save rules**

Inbound rules | Outbound rules | Tags

[You can now check network connectivity with Reachability Analyzer](#) [Run Reachability Analyzer](#)

Inbound rules (2) [Manage tags](#) [Edit inbound rules](#)

<input type="checkbox"/>	Name ▼	Security group rule... ▼	IP version ▼	Type ▼	Protocol ▼
<input type="checkbox"/>	-	sgr-0fc36f58b81a7942a	-	All traffic	All
<input type="checkbox"/>	-	sgr-0f9f1816b69938ebc	IPv4	MySQL/Aurora	TCP

Go to DB Instance Now...

RDS > Databases > pankajsiracademy

pankajsiracademy

Modify Actions ▼

Summary

DB identifier pankajsiracademy	CPU <div><div></div></div> 3.38%	Status ✔ Available	Class db.t3.micro
Role Instance	Current activity <div><div></div></div> 0 Connections	Engine MySQL Community	Region & AZ ap-northeast-1d

Connectivity & security | Monitoring | Logs & events | Configuration | Maintenance & backups | Tags

Connectivity & security

Endpoint & port	Networking	Security
-----------------	------------	----------

Connecting AWS MySQL Database to MySQL Workbench

Step 1:

Database Tools Scripting Help

Welcome

MySQL Workbench
create and browse databases
design and run SQL queries

[Browse Documentation >](#)

MySQL Connections

local instance MySQL80
root
localhost:3306

Setup New Connection

Connection Name: Type a name for the connection

Connection Method: Method to use to connect to the RDBMS

Parameters SSL Advanced

Hostname: Port: Name or IP address of the server host - and TCP/IP port.

Username: Name of the user to connect with.

Password: The user's password. Will be requested later if it's not set.

Default Schema: The schema to use as default schema. Leave blank to select it later.

Step 2: Go to AWS:

Amazon RDS

Dashboard

- Databases
- Query Editor
- Performance insights
- Snapshots
- Exports in Amazon S3
- Automated backups
- Reserved instances
- Proxies

Subnet groups

Parameter groups

Option groups

Custom engine versions

Connectivity & security

Endpoint & port

Endpoint

pankajsiracademy.cmua4mt8llob.ap-northeast-1.rds.amazonaws.com

Port

3306

Networking

Availability Zone

ap-northeast-1d

VPC

vpc-04d4e878bb0953735

Subnet group

default-vpc-04d4e878bb0953735

Subnets

subnet-0aff303f3c0fafdd1
subnet-09b0507ef4d5bef74

Security

VPC security groups

default (sg-07faaa7751a4072c1)

Active

Publicly accessible

Yes

Certificate authority

rds-ca-2019

Certificate authority date

August 22, 2024, 10:08 (UTC-)

Step 3: Update Localhostname

MySQL Workbench

File Edit View Database Tools Scripting Help

Setup New Connection

Connection Name: aws-mysql-blog

Connection Method: Standard (TCP/IP)

Parameters

Hostname: blob.ap-northeast-1.rds.amazonaws.com

Port: 3306

Username: root

Password: Store in Vault ...

Default Schema:

Test Connection

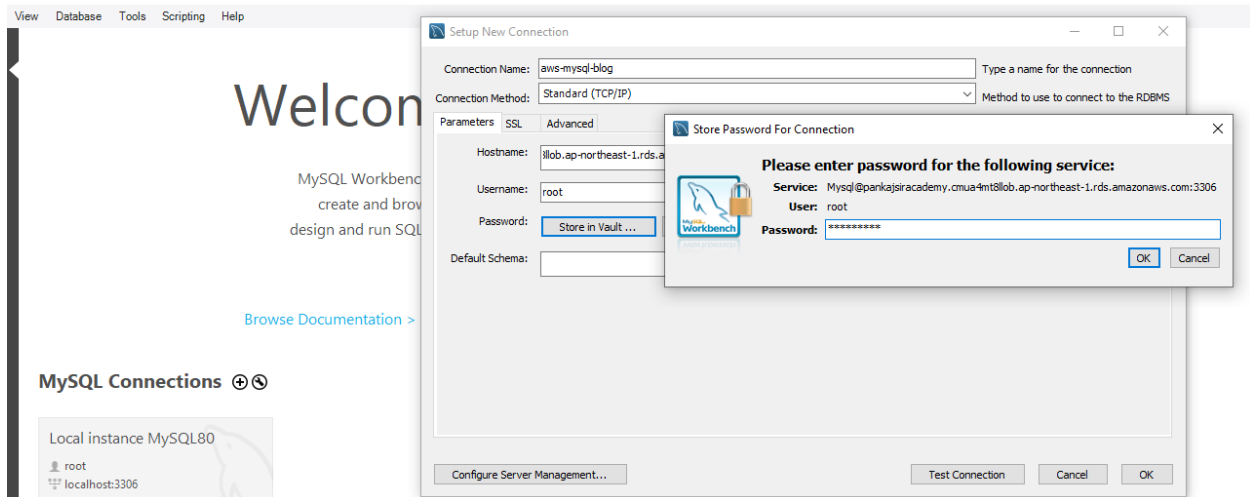
Cancel

OK

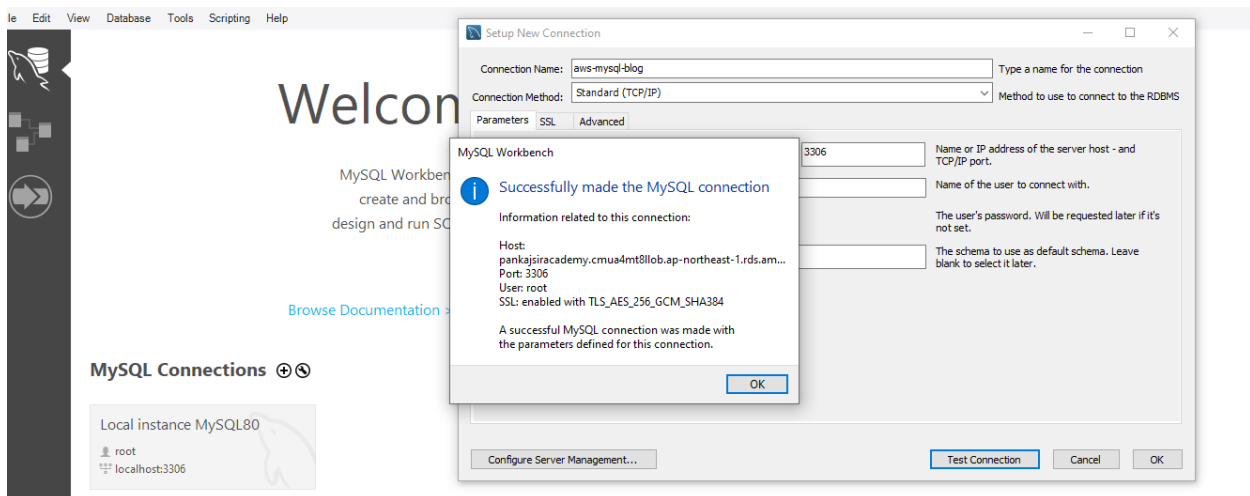
Activate Windows

Go to Settings to activate Windows.

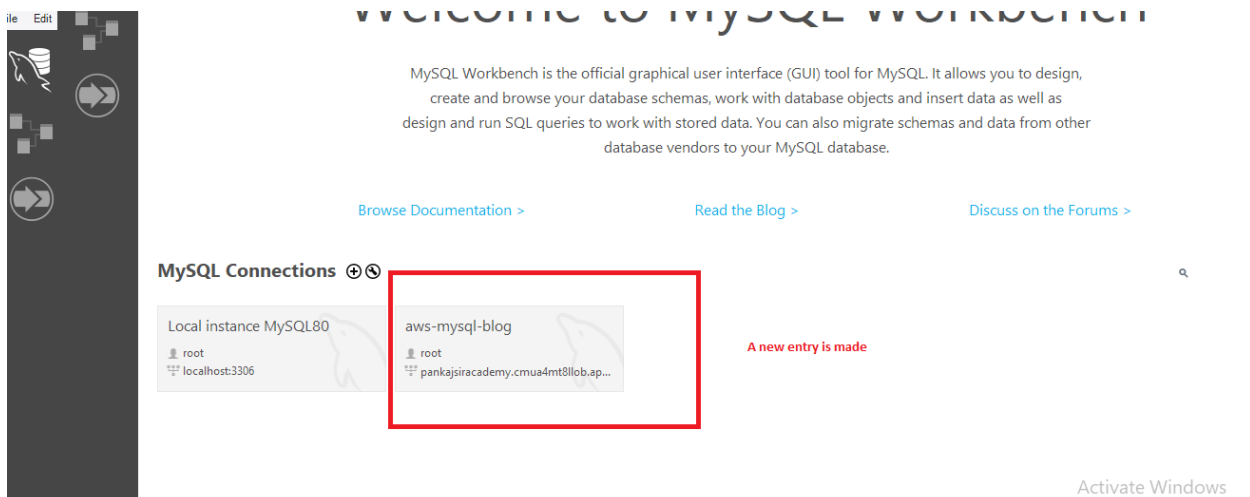
Step 4: Give aws password by clicking on store in vault



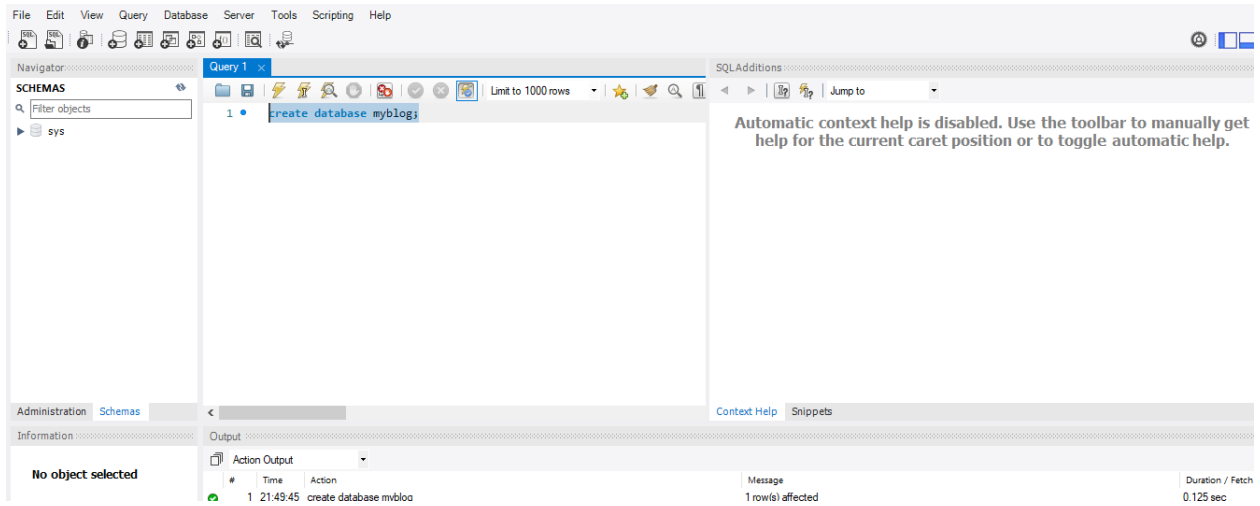
Step 5: Click okay and test the connection



Step 6: Click on ok

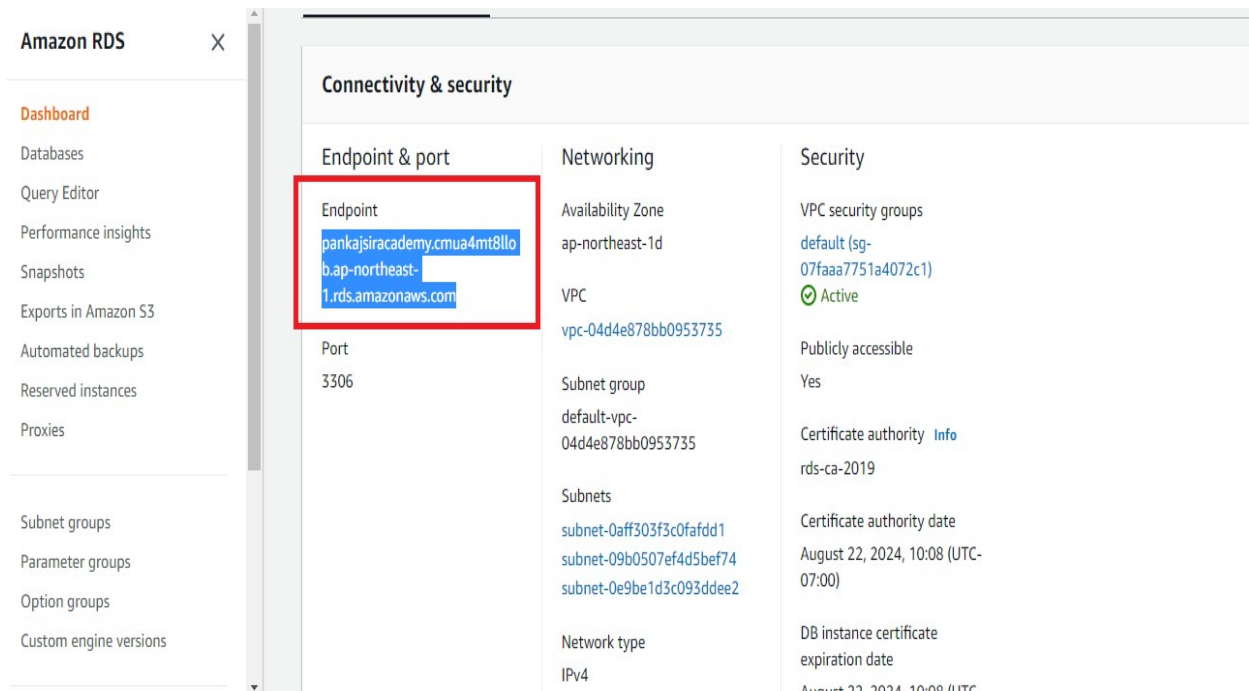


Step 7: Create Database in aws throughmysql workbench



Package Spring Boot App as jar file

Step 1: Copy endpoint from AWS

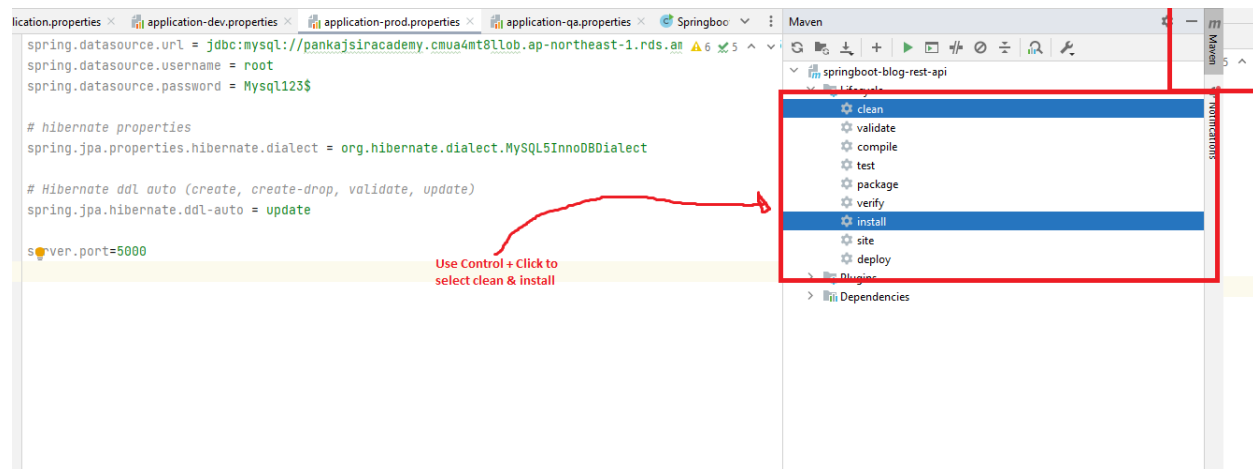


Step 2: Update application-prod.properties file:



```
1 spring.datasource.url = jdbc:mysql://pankajsiracademy.cmua4mt8llob.ap-northeast-1.rds.amazonaws.com:3306/myblog
2 spring.datasource.username = root
3 spring.datasource.password = MySQL123$
4
5 # hibernate properties
6 spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5InnoDBDialect
7
8 # Hibernate ddl auto (create, create-drop, validate, update)
9 spring.jpa.hibernate.ddl-auto = update
10
11 server.port=5000
12
13
14
```

Step 3: Perform maven clean & Install



```
application.properties x application-dev.properties x application-prod.properties x application-qa.properties x SpringbootBlogRestApiApplication.java x
```

```
spring.datasource.url = jdbc:mysql://pankajsiracademy.cmua4mt8llob.ap-northeast-1.rds.amazonaws.com:3306/myblog
spring.datasource.username = root
spring.datasource.password = MySQL123$

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

# Hibernate ddl auto (create, create-drop, validate, update)
spring.jpa.hibernate.ddl-auto = update

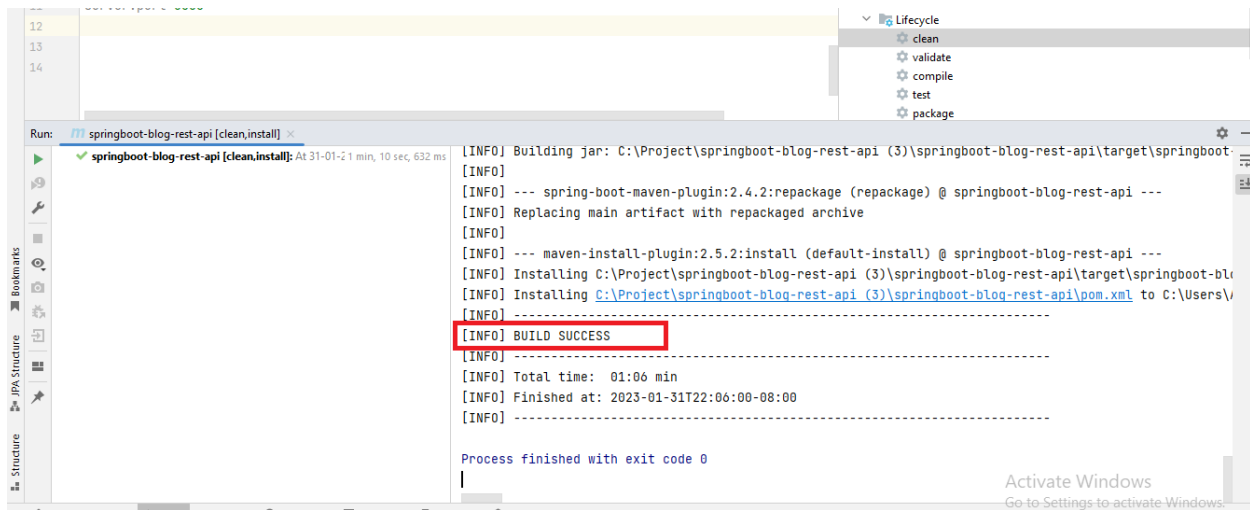
server.port=5000
```

Use Control + Click to select clean & install

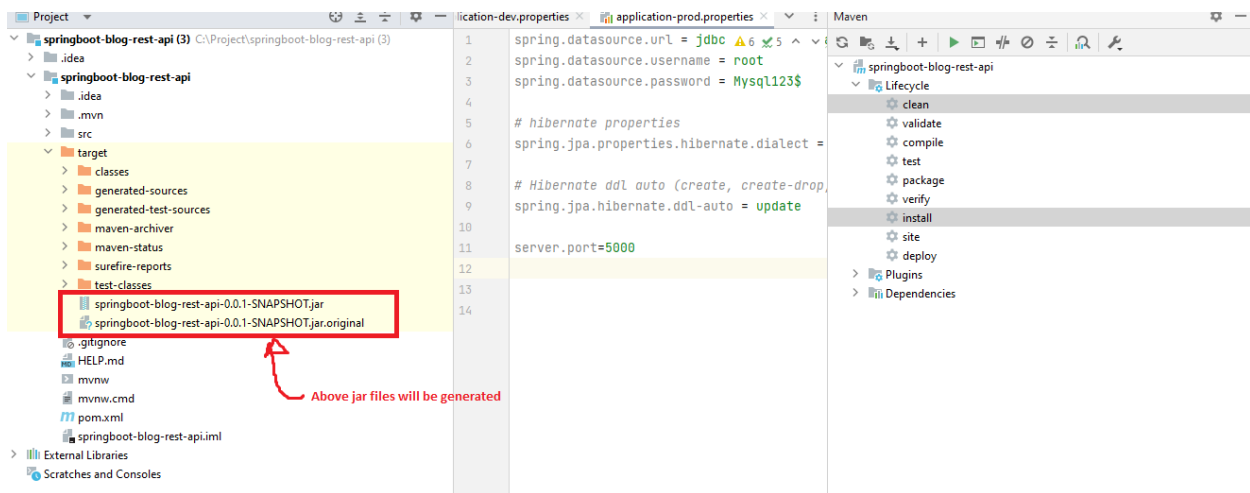
- clean
- validate
- compile
- test
- package
- verify
- install
- site
- deploy

Dependencies

Step 4: In run you should see the following message:

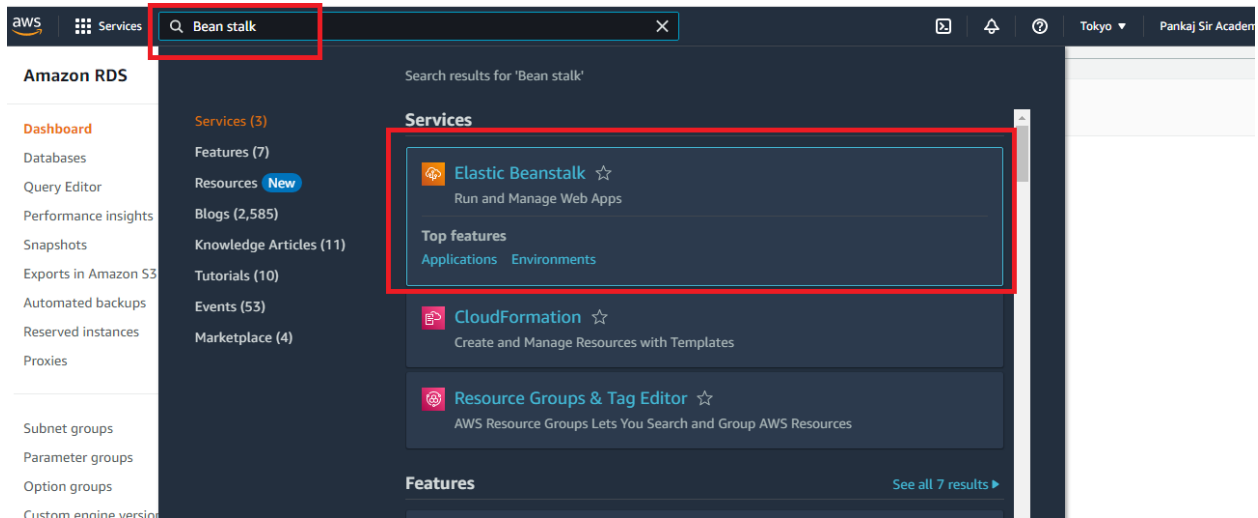


Step 5: See the jar files in IntelliJ generated below:

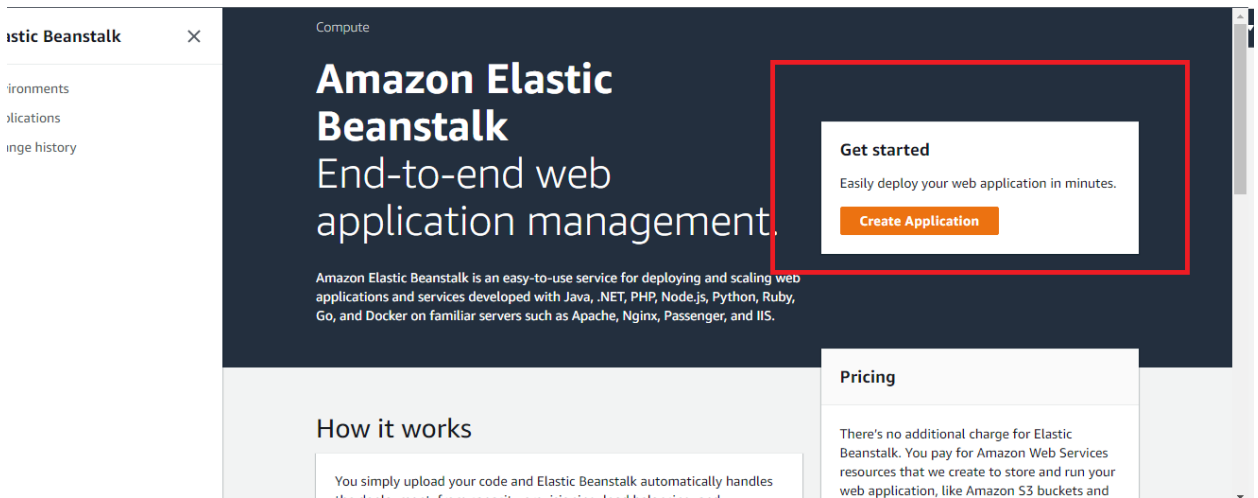


Deploy Spring Boot JAR file on AWS Cloud using Elastic BeanStalk


Step 1: Go Elastic BeanStalk in AWS Console:



Step 2: Click on create Application:






Step 3:

 Services

Search

[Alt+S]

   Tokyo

Elastic Beanstalk

Environments

Applications

Change history

Application information

Application name

MyBlog-Application

Up to 100 Unicode characters, not including forward slash (/).

Application tags

Apply up to 50 tags. You can use tags to group and filter your resources. A tag is a key-value pair. The key must be unique within the resource and is case-sensitive. [Learn more](#)

Key


Value

Remove tag

Add tag




50 remaining

Step 4:

 Services

Search

[Alt+S]

   Tokyo

Elastic Beanstalk

Environments

Applications

Change history

Add tag

50 remaining

Platform

Platform

Java

Platform branch

Corretto 17 running on 64bit Amazon Linux 2

Platform version

3.4.3 (Recommended)

Application code

☒ Sample application

Step 5:

Elastic Beanstalk

×

Environments

Applications

Change history

Application code

☐ Sample application

Get started right away with sample code.

☒ Upload your code

Upload a source bundle from your computer or copy one from Amazon S3.

Source code origin

Version label

Unique name for this version of your application code.

myblog-application-source

Source code origin

Maximum size 512 MB

☒ Local file

☐ Public S3 URL

Choose file

☐ No file uploaded

Step 6:

aws

Services

Search

[Alt+S]

Tokyo

Pankaj Sir Ac

Elastic Beanstalk

×

Environments

Applications

Change history

Version label

Unique name for this version of your application code.

myblog-application-source

Source code origin

Maximum size 512 MB

☒ Local file

☐ Public S3 URL

Choose file

File name : springboot-blog-rest-api-0.0.1-SNAPSHOT.jar

File successfully uploaded

Click Here

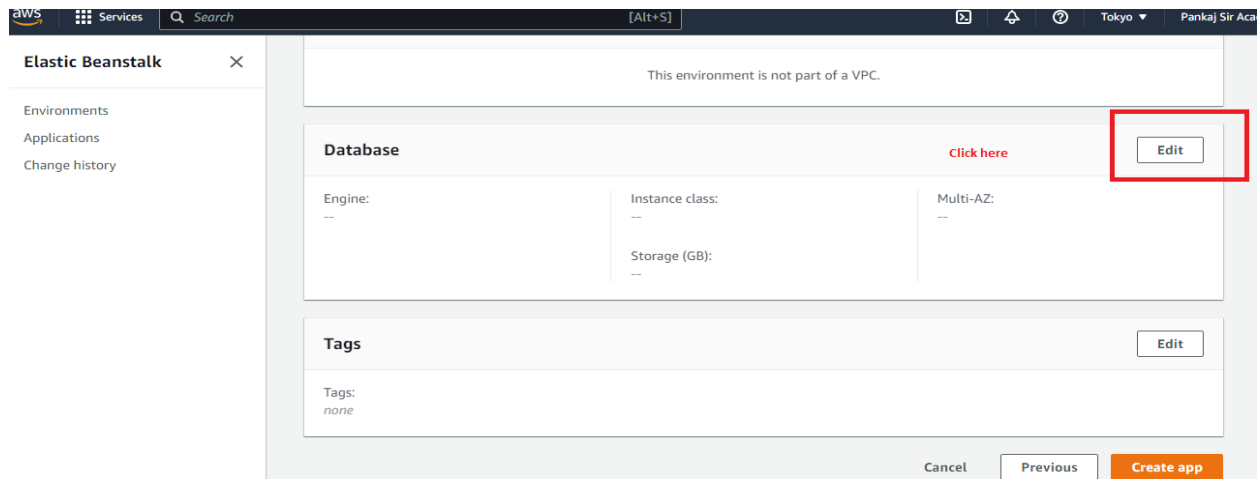
► Application code tags

Cancel

Configure more options

Create application

Step 7:



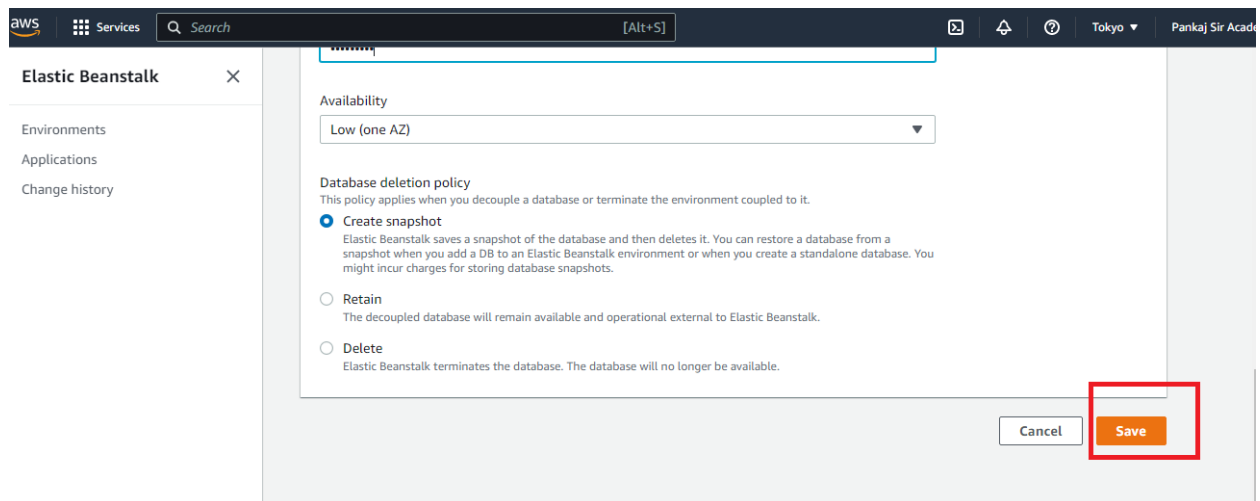
The screenshot shows the AWS Elastic Beanstalk console. On the left, there's a sidebar with 'Elastic Beanstalk' selected, and sub-options: 'Environments', 'Applications', and 'Change history'. The main area displays the 'Database' configuration step. At the top, a message states 'This environment is not part of a VPC.' Below this, the 'Database' section has a 'Click here' link and an 'Edit' button, which is highlighted with a red box. The configuration fields are: 'Engine:' (value: --), 'Instance class:' (value: --), 'Multi-AZ:' (value: --), and 'Storage (GB):' (value: --). Below the 'Database' section is the 'Tags' section with a value of 'none' and an 'Edit' button. At the bottom right, there are three buttons: 'Cancel', 'Previous', and 'Create app'.

Step 8:

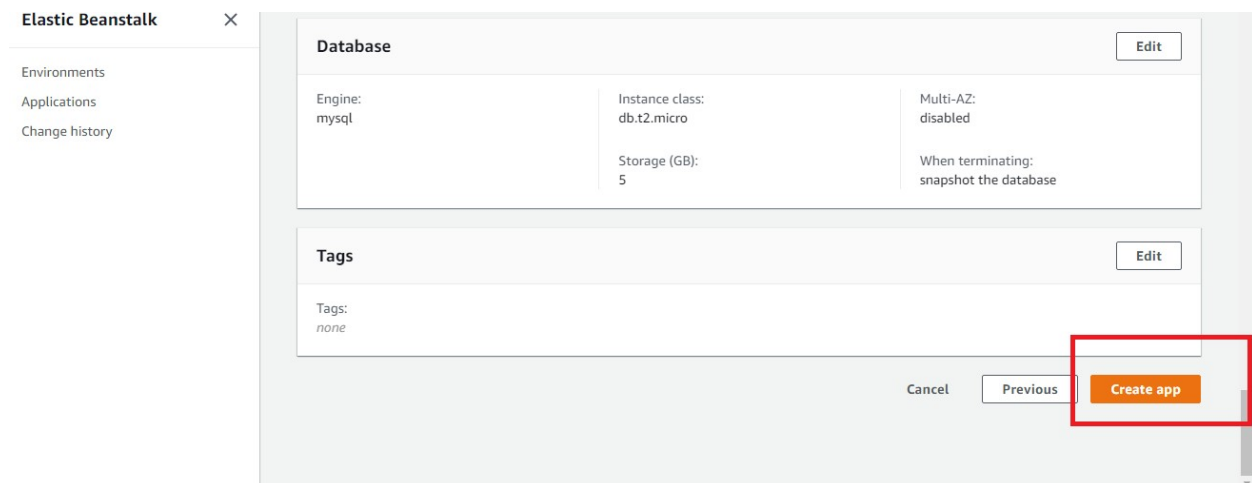


The screenshot shows the AWS Elastic Beanstalk console. On the left, there's a sidebar with 'Elastic Beanstalk' selected, and sub-options: 'Environments', 'Applications', and 'Change history'. The main area displays the 'Instance class' and 'Storage' configuration steps. The 'Instance class' dropdown is set to 'db.t2.micro'. The 'Storage' section has a note 'Choose a number between 5 GB and 1024 GB.' and a value of '5'. Below this, the 'Username' and 'Password' fields are highlighted with a red box. The 'Username' field has the value 'root'. The 'Password' field is empty. Below the 'Username' and 'Password' fields is the 'Availability' dropdown set to 'Low (one AZ)'. At the bottom, there's a 'Database deletion policy' section.

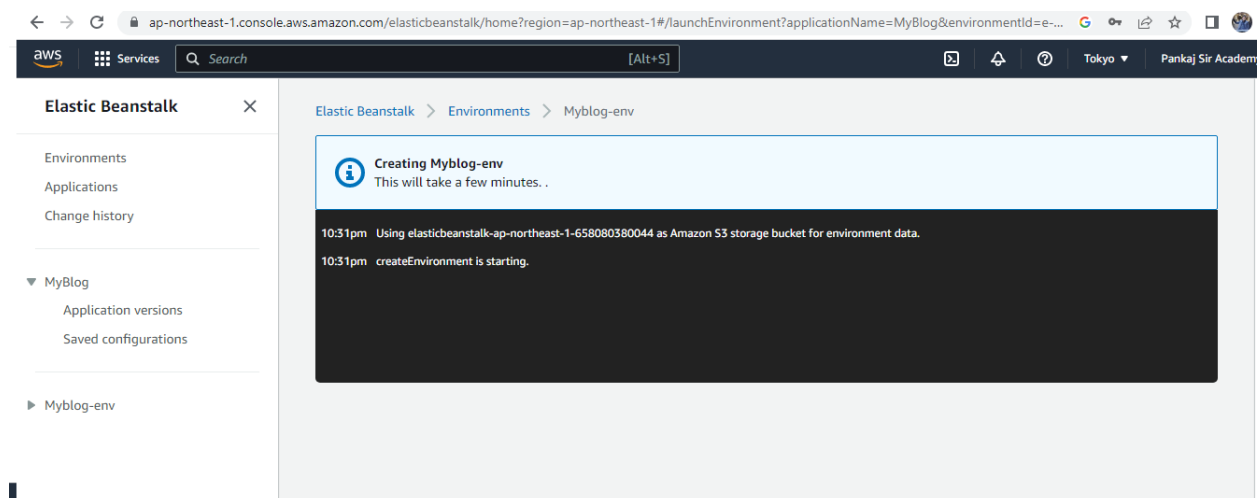
Step 9: Click on save



Step 10:



Step 11:



Step 12:

The screenshot displays the AWS Elastic Beanstalk console interface. On the left, a navigation pane shows the 'Elastic Beanstalk' service with options for 'Environments', 'Applications', and 'Change history'. Below this, a section for 'MyBlog' includes 'Application versions' and 'Saved configurations'. The 'Myblog-env' environment is selected, showing options to 'Go to environment', 'Configuration', 'Logs', 'Health', and 'Monitoring'. The main content area shows the 'Myblog-env' environment details, including its ARN, application name, health status (Ok), running version (myblog-source), and platform (Corretto 17 running on 64bit Amazon Linux 2/3.4.3). A 'Recent events' section is visible at the bottom.

Elastic Beanstalk ×

Environments
Applications
Change history

▼ MyBlog
Application versions
Saved configurations

▼ **Myblog-env**
Go to environment
Configuration
Logs
Health
Monitoring

Elastic Beanstalk > Environments > Myblog-env

Myblog-env
[Myblog-env.eba-mg4mipfw.ap-northeast-1.elasticbeanstalk.com](#) (e-yyba2euymd)
Application name: **MyBlog**

Refresh Actions ▼

Health

Ok
Causes

Running version
myblog-source
Upload and deploy

Platform

Corretto 17 running on 64bit
Amazon Linux 2/3.4.3
Change

Recent events Show all
< 1 >