Perfect 👍 that's the right structure for a clean Spring Boot backend. Since you already have IntelliJ with Spring Boot, I'll give you the step-by-step flow starting from scratch, but this time all table creation + DB schema handled by Spring Boot itself (so you don't need to manually run SQL in pgAdmin).

We'll keep files under these folders:

controller → REST controllers

dto → Request/Response payloads

entity → JPA entities (tables)

repo → Spring Data JPA repositories

services → Business logic

security → Optional (not needed right now unless auth is required)

Step 1: Configure PostgreSQL connection

src/main/resources/application.yml

```
server:
  port: 8080

spring:
  datasource:
    url: jdbc:postgresql://localhost:5432/RWTOOL
    username: postgres     # or your user
    password: 12345        # your password
  jpa:
    hibernate:
      ddl-auto: update     # auto create/update table
    properties:
      hibernate:
        dialect: org.hibernate.dialect.PostgreSQLDialect
```

☞ With ddl-auto: update, Spring Boot will automatically create the table from your @Entity.

Step 2: Create the Entity

entity/Report.java

```
package com.example.rwtool.entity;
```

```
import jakarta.persistence.*;
import java.time.LocalDateTime;

@Entity
@Table(name = "reports")
public class Report {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, unique = true, length = 255)
    private String name;

    @Column(nullable = false)
    private String sourcePath;

    private LocalDateTime createdAt = LocalDateTime.now();
    private LocalDateTime updatedAt = LocalDateTime.now();

    @PreUpdate
    public void onUpdate() {
        this.updatedAt = LocalDateTime.now();
    }

    // Getters and setters
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public String getSourcePath() { return sourcePath; }
    public void setSourcePath(String sourcePath) { this.sourcePath =
sourcePath; }

    public LocalDateTime getCreatedAt() { return createdAt; }
    public void setCreatedAt(LocalDateTime createdAt) { this.createdAt =
createdAt; }

    public LocalDateTime getUpdatedAt() { return updatedAt; }
    public void setUpdatedAt(LocalDateTime updatedAt) { this.updatedAt =
updatedAt; }
}
```

Step 3: Repository

repo/ReportRepository.java

```
package com.example.rwtool.repo;
```

```
import com.example.rwtool.entity.Report;
import org.springframework.data.jpa.repository.JpaRepository;

public interface ReportRepository extends JpaRepository<Report, Long> {
}
```

Step 4: DTOs

dto/ReportRequest.java

```
package com.example.rwtool.dto;

import jakarta.validation.constraints.NotBlank;

public class ReportRequest {

    @NotBlank
    private String name;

    @NotBlank
    private String sourcePath;

    // getters and setters
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    public String getSourcePath() { return sourcePath; }
    public void setSourcePath(String sourcePath) { this.sourcePath =
sourcePath; }
}
```

dto/ReportResponse.java

```
package com.example.rwtool.dto;

public class ReportResponse {
    private Long id;
    private String name;
    private String sourcePath;

    public ReportResponse(Long id, String name, String sourcePath) {
        this.id = id;
        this.name = name;
        this.sourcePath = sourcePath;
    }

    // getters
    public Long getId() { return id; }
    public String getName() { return name; }
    public String getSourcePath() { return sourcePath; }
```

```
}
```

Step 5: Service

services/ReportService.java

```
package com.example.rwtool.services;

import com.example.rwtool.dto.ReportRequest;
import com.example.rwtool.entity.Report;
import com.example.rwtool.repo.ReportRepository;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class ReportService {

    private final ReportRepository repo;

    public ReportService(ReportRepository repo) {
        this.repo = repo;
    }

    public List<Report> getAllReports() {
        return repo.findAll();
    }

    public Report addReport(ReportRequest request) {
        Report report = new Report();
        report.setName(request.getName());
        report.setSourcePath(request.getSourcePath());
        return repo.save(report);
    }

    public Report updateReport(Long id, ReportRequest request) {
        Report report = repo.findById(id)
                .orElseThrow(() -> new RuntimeException("Report not
found"));
        report.setName(request.getName());
        report.setSourcePath(request.getSourcePath());
        return repo.save(report);
    }

    public void deleteReport(Long id) {
        repo.deleteById(id);
    }
}
```

Step 6: Controller

controller/ReportController.java

```
package com.example.rwtool.controller;

import com.example.rwtool.dto.ReportRequest;
import com.example.rwtool.dto.ReportResponse;
import com.example.rwtool.entity.Report;
import com.example.rwtool.services.ReportService;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api")
@CrossOrigin(origins = "*")
public class ReportController {

    private final ReportService service;

    public ReportController(ReportService service) {
        this.service = service;
    }

    @GetMapping("/fetch-all-report")
    public List<ReportResponse> getAllReports() {
        return service.getAllReports().stream()
                .map(r -> new ReportResponse(r.getId(), r.getName(),
r.getSourcePath()))
                .toList();
    }

    @PostMapping("/newreport")
    public ReportResponse createReport(@RequestBody ReportRequest req) {
        Report saved = service.addReport(req);
        return new ReportResponse(saved.getId(), saved.getName(),
saved.getSourcePath());
    }

    @PutMapping("/reports/{id}")
    public ReportResponse updateReport(@PathVariable Long id,
@RequestBody ReportRequest req) {
        Report updated = service.updateReport(id, req);
        return new ReportResponse(updated.getId(), updated.getName(),
updated.getSourcePath());
    }

    @DeleteMapping("/reports/{id}")
    public ResponseEntity<Void> deleteReport(@PathVariable Long id) {
        service.deleteReport(id);
        return ResponseEntity.noContent().build();
```

```
    }
}
```

Step 7: Run & Test

1. Run Spring Boot app (RwtoolApplication main class).

2. Open Postman and test:

GET http://localhost:8080/api/fetch-all-report

POST http://localhost:8080/api/newreport → Body (JSON):

{ "name": "KYC Reports", "sourcePath": "/system/reports/in/kyc/" }

PUT http://localhost:8080/api/reports/1

DELETE http://localhost:8080/api/reports/1

Spring Boot will auto-create the reports table in RWTOOL DB, so you don’t need to manually write SQL.

☞ Do you want me to also prepare a ready-to-import Postman collection so you can test all endpoints quickly?