**Project Documentation for "ReportsProject" Application**

# 1. Introduction

The ReportsProject application is a Spring Boot-based web application that generates and serves dynamic reports. The application allows for the insertion of sample data into a database, retrieves the data, and generates PDF reports with the data, including visual charts. It leverages technologies such as Spring Boot, JPA (Java Persistence API), iText (for PDF generation), and JFreeChart (for charting).

# 2. Technologies Used

* **Java** (version 8 or above)
* **Spring Boot**: For building the RESTful API.
* **Spring Data JPA**: For interacting with the database.
* **iText PDF**: For PDF generation.
* **JFreeChart**: For generating charts.
* **H2 Database**: For storing sample data.
* **Maven**: For managing project dependencies.

# 3. Project Structure

Here is an overview of the project structure:

css

Copy code

src/

├── main/

│ ├── java/

│ │ ├── com/

│ │ │ ├── Reports/

│ │ │ │ ├── ReportsProject/

│ │ │ │ │ ├── Controllers/

│ │ │ │ │ ├── Entities/

│ │ │ │ │ ├── Repositories/

│ │ │ │ │ ├── Services/

│ │ │ │ ├── ReportsProjectApplication.java

├── resources/

│ ├── application.properties

# 4. Overview of Key Components

**4.1. ReportsProjectApplication.java**

This is the main entry point of the Spring Boot application. It contains the main method to run the application.

**Application.java**

@SpringBootApplication

public class ReportsProjectApplication {

public static void main(String[] args) {

SpringApplication.run(ReportsProjectApplication.class, args);

System.out.println("Executed Successfully");

}

}

**4.2. Controllers**

PdfController.java: Handles the generation and download of PDFs.

GET /api/generatepdf: This endpoint triggers the PDF generation process, which includes fetching sample data, creating a table, generating a gender distribution chart, and embedding the chart in a PDF file.

SampleDataController.java: Manages sample data through API calls.

POST /api/insert: Inserts sample data (name, email, and gender) into the database.

GET /api/retrive: Retrieves all the sample data from the database.

**4.3. Entities**

**SampleData.java:** Defines the entity class for sample data. It contains fields for id, name, email, and gender, with appropriate annotations for JPA.

Java

**SampleData.java**

@Entity

@Table(name = "Sampledata")

public class SampleData {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

private String gender;

}

**4.4. Repositories**

**SampleDataRepository.java:** A Spring Data JPA repository that provides CRUD operations for the SampleData entity.

**SampleDataRepository.java**

public interface SampleDataRepository extends JpaRepository<SampleData, Long> {

}

**4.5. Services**

**SampleDataService.java:** Provides methods to save and retrieve sample data from the database.

**SampleDataServices.java**

@Service

public class SampleDataService {

@Autowired

private SampleDataRepository sampleDataRepository;

public SampleData saveSampleData(String name, String email, String gender) {

SampleData sampleData = new SampleData(name, email, gender);

return sampleDataRepository.save(sampleData);

}

public List<SampleData> getAllSampleData() {

return sampleDataRepository.findAll();

}

}

**ChartGenerationService.java:** Creates a gender distribution chart (bar chart) based on the sample data using JFreeChart.

**ChartGenerationService.java**

public class ChartGenerationService {

public ByteArrayOutputStream generateGenderChart(List<SampleData> sampleDataList) {

// Logic to generate chart based on gender data

}

}

**PDFGenerationService.java:** Handles the generation of the PDF document, including the title, sample data table, and chart image.

**PDFGenerationService.java**

@Service

public class PDFGenerationService {

@Autowired

private SampleDataService sampleDataService;

@Autowired

private ChartGenerationService chartGenerationService;

public ByteArrayOutputStream generatePDF() throws DocumentException, IOException {

// Logic to generate PDF document with data table and chart

}

}

**TableGenerationService.java:** Generates a PDF table containing the sample data.

**TableGenerationService.java**

@Service

public class TableGenerationService {

public PdfPTable createDataTable(List<SampleData> sampleDataList) {

// Logic to create a PDF table with sample data

}

}

# 5. Endpoints

* **POST /api/insert**: Inserts sample data into the database.
* **Request Body**: JSON with name, email, and gender.
* **Response**: The saved SampleData entity.
* **GET /api/retrive**: Retrieves all sample data from the database.
* **Response**: A list of SampleData entities.
* **GET /api/generatepdf**: Generates a PDF document with a table of sample data and a gender distribution chart.
* **Response**: The generated PDF as an attachment with the filename SampleDataRajesh.pdf.

# 6. PDF Report Structure

The generated PDF contains the following sections:

* **Title**: "SAMPLE DATA FROM LOCAL DB"
* **Data Table**: A table containing the name, email, and gender of all sample data entries.
* **Gender Distribution Chart**: A bar chart showing the count of males and females in the sample data.

# 7. Database

* **Database**: H2 (in-memory database for development and testing)
* **Table:** Sampledata
* **Columns:** id, name, email, gender

# 8. Configuration

In the application.properties, you can configure the database and other settings.

**application.properties:**

spring.application.name=ReportsProject  
spring.datasource.url=jdbc:postgresql://localhost:5432/rajesh  
spring.datasource.username=postgres  
spring.datasource.password=root  
spring.datasource.driver-class-name=org.postgresql.Driver  
spring.jpa.hibernate.ddl-auto=update  
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect  
spring.jpa.show-sql=true  
spring.jpa.properties.hibernate.format\_sql=true

# 9. How to Run the Application

1. **Build and run the application**: Using Maven:

mvn spring-boot:run

1. **Test the API**:
   * You can test the endpoints using **Postman**.
   * Example POST request to /api/insert:

json

Copy code

{

"name": "Rajesh",

"email": "rajesh@example.com",

"gender": "Male"

}

* + Example GET request to /api/generatepdf: This will download the generated PDF.

# 10. Conclusion

The **ReportsProject** application demonstrates how to create and serve dynamic PDF reports with embedded charts using data stored in a database. It also showcases basic RESTful API concepts using Spring Boot, and integrates third-party libraries such as iText and JFreeChart for PDF and chart generation.