Human-Readable Data to JSON Converter

## Table of Contents

* Introduction
* Project Overview
* Why Use Java
* Prerequisites
* Project Setup
* Procedure
* Implementation Details
* Conclusion

## Introduction

The Human-Readable Data to JSON Converter is a web-based tool designed to streamline the process of converting human-readable data, captured through a user-friendly form, into structured JSON format. This documentation provides an in-depth understanding of the project's purpose, functionality, setup, and usage.

## Project Overview

The project consists of a user interface built using HTML, Bootstrap, and JavaScript to capture user input. The backend is developed using Java with Spring Boot, facilitating data processing and JSON conversion. The converted JSON data is saved to a text file for later access.

## Why Use Java

Java was chosen as the programming language for several reasons:

* Platform Independence: Java is platform-independent due to its "write once, run anywhere" nature, making it suitable for web applications.
* Robust Ecosystem: The Spring Boot framework simplifies development, offering tools for building robust and scalable applications.
* Security: Java's strong security features ensure data protection and secure communication.
* Community Support: The extensive Java community provides resources, libraries, and solutions to various challenges.

## Prerequisites

To run this project, you need:

* Java Development Kit (JDK 17)
* Integrated Development Environment (IDE) (IntelliJ IDEA)
* Spring Boot

## Project Setup

* Clone the project repository.
* Open the project in your preferred IDE.
* Set up the Spring Boot application.

## Procedure

* Start the Spring Boot application to run the backend server.
* Access the application through a web browser.
* Fill in the form fields with the required data.
* Dynamically add topics, content fields, and alt questions as needed.
* Submit the form to convert the input data into JSON format.
* View the converted JSON data on the result page or access it from the saved text file.

## Implementation Details

1. Creating the Spring Boot Project:

- Use Spring Initializer or your IDE to create a new Spring Boot project.

- Set up project metadata, dependencies, and package structure.

2. Frontend Files and Directory:

- Create an `index.html` file for the form interface.

- Include Bootstrap and Font Awesome libraries for styling and icons.

- Set up an `app.js` file to handle dynamic form behavior.

3: Frontend Development

Creating the Form Interface:

* Design the form layout using HTML and Bootstrap classes.
* Include fields for unit number, unit name, topics, content fields, and alt questions.
* Implement buttons to add topics, content fields, and alt questions dynamically.

JavaScript Logic (app.js):

* Set up event listeners for form submission and dynamic interactions.
* Define functions to handle topic and content field additions.
* Implement functions to capture user input and create JSON data for submission.
* Handle form submission using Fetch API to send data to the backend.

4: Backend Development

Data Classes (ContentData, FormData, TopicData):

- Create Java classes to represent the data structure of content, topics, and form data.

- Implement constructors, getters, setters, and a `toString` method to generate JSON strings.

\*\*Spring Boot Controller (Home.java):

@Controller

public class Home {

@GetMapping("/")

public String home(Model model) {

model.addAttribute("formData", new FormData());

return "index";

}

}

* @GetMapping("/"): This annotation maps the root URL "/" to the `home` method.
* home(Model model): This method handles the initial request to the root URL and returns the name of the HTML template ("index") to render.
* model.addAttribute("formData", new FormData());: This adds a new instance of the `FormData` class to the model, which is used to bind form data in the HTML template.

@PostMapping("/result")

public String submitForm(@RequestBody FormData formData) {

System.out.println(formData);

saveJsonToTextFile(formData);

return "redirect:/view";

}

* @PostMapping("/result"): This annotation maps the URL "/result" to the `submitForm` method, which handles form submission using the POST method.
* submitForm(@RequestBody FormData formData): This method receives the form data from the frontend as a `FormData` object in the request body.
* System.out.println(formData);: This line prints the received form data to the console for debugging purposes.
* saveJsonToTextFile(formData);: This method call saves the received form data as JSON to a text file using the `saveJsonToTextFile` method.
* return "redirect:/view";: This line redirects the user to the "/view" URL after form submission.

@GetMapping("/view")

public String view(Model model) {

String jsonFromFile = readJsonFromTextFile();

model.addAttribute("formData", jsonFromFile);

return "result";

}

* @GetMapping("/view"): This annotation maps the URL "/view" to the `view` method, which handles displaying the JSON result page.
* view(Model model): This method prepares the JSON data read from the text file and adds it to the model.
* String jsonFromFile = readJsonFromTextFile();: This line reads the JSON data from the text file using the `readJsonFromTextFile` method.
* model.addAttribute("formData", jsonFromFile);: This adds the JSON data as an attribute named "formData" to the model.
* return "result";: This line returns the name of the HTML template ("result") to render.

public void saveJsonToTextFile(FormData data) {

String json = data.toString();

String fileName = "data.txt";

Path filePath = Path.of(fileName);

try {

Files.writeString(filePath, json, StandardOpenOption.CREATE, StandardOpenOption.WRITE, StandardOpenOption.TRUNCATE\_EXISTING);

System.out.println("JSON data saved to file: " + filePath.toAbsolutePath());

} catch (IOException e) {

e.printStackTrace();

}

}

* saveJsonToTextFile(FormData data): This method takes a `FormData` object as input and saves it as a JSON string to a text file.
* String json = data.toString();: This converts the `FormData` object to a JSON string using its `toString` method.
* Path filePath = Path.of(fileName);: This creates a `Path` object representing the file path.
* Files.writeString(...): This writes the JSON string to the text file, creating the file if it doesn't exist and truncating it if it does.
* System.out.println("JSON data saved to file: " + filePath.toAbsolutePath());: This line prints the file path where the JSON data is saved.

public String readJsonFromTextFile() {

String fileName = "data.txt";

Path filePath = Path.of(fileName);

try {

return Files.readString(filePath);

} catch (IOException e) {

e.printStackTrace();

return "Error reading JSON data from file.";

}

}

}

* readJsonFromTextFile(): This method reads JSON data from the text file and returns it as a string.
* Path filePath = Path.of(fileName);: This creates a `Path` object representing the file path.
* return Files.readString(filePath);: This reads the contents of the text file and returns it as a string.
* catch (IOException e) { ... }: If an exception occurs while reading the file, an error message is printed, and the default error message is returned.

Overall, the `Home.java` controller handles the form submission, saving the JSON data to a text file, and displaying the JSON result page. It uses helper methods to manage the file operations and communicates with the frontend using model attributes and redirections.

5: Styling

CSS Styling (main.css, result.css):

* Customize the appearance of the form, headers, buttons, and result display.
* Use responsive design to ensure a user-friendly experience on different devices.

## Conclusion

The Human-Readable Data to JSON Converter project demonstrates the seamless integration of frontend and backend technologies using Java and Spring Boot. This tool simplifies the conversion of user-provided data into JSON format, making data interchange efficient and user-friendly.