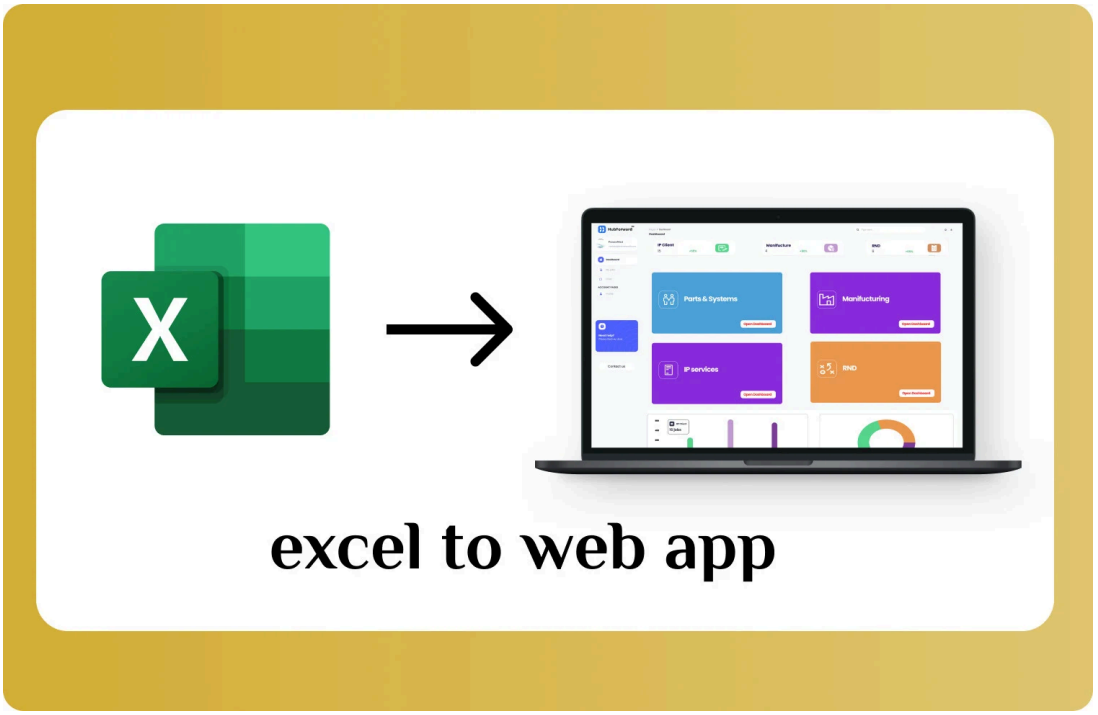
Technical Documentation

BTP PROJECT - EXCEL TO WEB APP



Student Name:
Anurag Kumar Bharti
(B21CS012)

Mentor Name:
Dr. Sumit Kalra

1. Introduction

- **Overview**

  - The Excel to web app project aims to do real-time changes in the web component. When we change data in the excel, it will also change in the web page.This documentation will cover the technologies like html, css, javascript, and in backend Java Spring Boot

- **Scope:** This documentation outlines the steps required to set up your environment, install the necessary dependencies, and configure the project. It also provides a comprehensive guide to understanding the functionality of the project, including its core features and usage. Key areas covered include:
  - **Device Setup**
    Instructions to prepare your device for the project, including required system specifications and pre-installation configurations.
  - **Installations**
    Step-by-step guidelines to install dependencies, libraries, and frameworks used in the project, ensuring a seamless setup process.
  - **Project Setup**
    Detailed procedures for setting up the project, including configuring the backend, frontend, and establishing WebSocket connections.
  - **Functionality Overview**
    Explanation of the project's features, including real-time WebSocket-based communication, dynamic content rendering, and Excel file monitoring for data updates.
  - **Technology Stack**
    Overview of the technologies used, such as Spring Boot, Apache POI, SockJS, STOMP.js, and HTML/CSS for frontend rendering.
  - **Demonstration**
    A walkthrough of the project in action, showcasing its real-time data updates, WebSocket messaging, and template creation features. This section includes screenshots or screen recordings for better understanding.

- **Intended Audience:** (e.g., Developers, End-Users, Admins)
  - **Developers**:
    Individuals who wish to contribute to the project or modify its existing features. This includes frontend and backend developers with knowledge of the relevant tech stack.
  - **End-Users**:
    Users interacting with the project for its intended purpose, such as monitoring real-time updates or managing data dynamically.

- **System Requirements:**

  Hardware Requirements

  - A system with at least the following specifications:
    - Processor: Dual-Core or higher (e.g., Intel i3 or equivalent)
    - RAM: Minimum 4 GB (8 GB recommended)
    - Storage: 10 GB free disk space

  Software Requirements

  - **Operating System**:
    - Windows 10 or higher
  - **Backend Framework**:
    - Java Development Kit (JDK) 17 or higher
    - Spring Boot 2.7 or higher
  - **Frontend Environment**:
    - Web browser (latest versions of Chrome, Firefox, or Edge)
  - **Other Tools**:
    - Node.js v18+
    - npm or Yarn for dependency management

  Network Requirements

  - Internet connectivity for fetching dependencies and real-time WebSocket communication.
  - Port requirements:
    - Port **8088** for backend services.
    - Port **5500** (http://localhost:5500/code.html)

2. Getting Started

Installation Instructions

Here's a step-by-step guide to set up the environment with **Spring Boot**, **Java**, and **Microsoft Excel** integration:

Installing Java

1. **Download and Install Java Development Kit (JDK):**
   - Visit the Oracle JDK or OpenJDK website.
   - Choose the appropriate JDK version for your operating system.
   - Download and run the installer.
2. **Set JAVA_HOME Environment Variable:**
   - For **Windows**:
     - Open System Properties > Advanced > Environment Variables.
     - Add a new system variable:
       - **Variable Name**: JAVA_HOME
       - **Variable Value**: Path to the JDK installation directory (e.g., C:\Program Files\Java\jdk-x.x.x)
     - Verify installation
       .java -version

Installing Spring Boot

1. **Install Spring Boot CLI (Optional):**
   - Download from the Spring Boot CLI website.
   - Add spring-boot-cli/bin to your system's PATH.
   - Verify:

```
spring
--version
```

2. **Using Maven or Gradle:**
- Ensure you have Maven or Gradle installed.
- Create a new Spring Boot project:
    - Use Spring Initializr to generate the project structure.
    - Download and unzip the project.

Installing Microsoft Excel

1. **Office Suite Installation:**
    - Visit the Microsoft Office official website.
    - Download the installer for **Microsoft Excel** as part of the Office suite.
    - Run the installer and follow on-screen instructions.
2. **Ensure Compatibility for Automation:**
    - Install necessary tools for Excel file handling in your Spring Boot project, such as:
        - **Apache POI**:

```
<dependency>
    <groupId>org.apache.poi</groupId>
    <artifactId>poi-ooxml</artifactId>
    <version>5.x.x</version>
</dependency>
```

Prerequisites

- **Java 17+** (or the version required by your Spring Boot application).
- **Spring Boot** compatible IDE (e.g., IntelliJ IDEA, Eclipse, or Visual Studio Code with Java Extensions).
- **Maven** or **Gradle** installed for building the project.
- **Excel** for viewing, creating, and managing `.xlsx` files.

Quick Start Guide

1. **Verify Installations:**
    - Check Java: `java -version`.
    - Check Spring Boot: `spring --version`.
    - Confirm Excel is operational.
2. **Create a Sample Spring Boot Application:**
    - Use Spring Initializr to generate a Maven project with dependencies: `Spring Web` and `Apache POI`.
3. **Write Code to Handle Excel Files:**
    - Example for reading Excel data:

```
import org.apache.poi.ss.usermodel.*;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
```

```java
import java.io.File;
import java.io.FileInputStream;

public class ExcelReader {
    public static void main(String[] args) throws Exception {
        FileInputStream file = new FileInputStream(new
File("sample.xlsx"));
        Workbook workbook = new XSSFWorkbook(file);
        Sheet sheet = workbook.getSheetAt(0);

        for (Row row : sheet) {
            for (Cell cell : row) {
                System.out.print(cell.toString() + "\t");
            }
            System.out.println();
        }
        workbook.close();
    }
}
```

Run the Application:

- Compile and run:

```
mvn clean package
java -jar target/your-app-name.jar
```

Test Integration with Excel:

- Create a sample `.xlsx` file and place it in the project directory.
- Run the above code to ensure data is processed correctly.

3.Core Concepts and Terminology

Spring Boot:

- A framework for building Java-based applications with a focus on simplicity and ease of use.
- It provides pre-configured settings and a variety of built-in features, making it easier to set up standalone, production-ready applications without extensive configuration.
- Spring Boot simplifies dependency management, auto-configuration, and includes an embedded server, so applications can run independently.

Java:

- A high-level, class-based, object-oriented programming language designed for portability across platforms.
- Known for its robustness, security, and platform independence (via the Java Virtual Machine or JVM), Java is widely used for building server-side applications, desktop applications, and more.

Maven Project:

- **Maven** is a build automation and project management tool for Java-based projects.
- It uses an XML file (`pom.xml`) to manage project dependencies, configuration, and build tasks, making it easier to manage dependencies and execute standardized builds.
- A **Maven project** follows a standard directory layout and enables developers to manage complex Java projects with ease.

Excel File:

- A spreadsheet file created by Microsoft Excel, usually in `.xls` or `.xlsx` format.
- Excel files are widely used for storing and analyzing tabular data and can contain various elements like formulas, charts, and formatted cells.

Apache POI:

- A Java library for reading, writing, and manipulating Microsoft Office file formats, including **Excel** files.
- Apache POI supports both `.xls` and `.xlsx` formats, allowing developers to programmatically read and write data to Excel files in Java applications.

SimpMessagingTemplate:

- A Spring framework class for sending messages to WebSocket clients.
- It provides methods to broadcast messages to specific destinations or topics, enabling real-time communication between the server and clients.
- Often used in Spring Boot applications with WebSockets to push data updates or notifications to subscribed clients.

Threads:

- A unit of execution within a process that allows for concurrent execution.
- **Threads** enable a program to perform multiple tasks simultaneously, improving efficiency, especially in I/O operations.
- In Java, threads can be managed using the `Thread` class or the `ExecutorService` interface.

STOMP Client:

- **STOMP** (Simple Text Oriented Messaging Protocol) is a lightweight messaging protocol used to communicate with message brokers or servers.
- In WebSocket applications, a **STOMP client** is used to send and receive messages from specific topics or endpoints, enabling real-time, bidirectional communication.
- It's commonly used in Spring WebSocket implementations, allowing clients to subscribe to topics and receive updates when messages are broadcasted.

SockJS:

- A JavaScript library that provides a WebSocket-like API for browsers that don't support WebSocket connections.
- **SockJS** acts as a fallback mechanism, allowing applications to maintain real-time connections with clients by using alternative protocols when WebSockets are unavailable.

It is often used with STOMP in Spring Boot applications to ensure reliable WebSocket connections across various client environments.

4.Usage documentation and Architecture

a)write data in the excel

- Step 1: Create a new Excel workbook and sheet.
  - Use Apache POI to initialize a new Workbook and create a sheet named "iitjhtml".
- Step 2: Populate the sheet with data.
  - Add a header row(Section, content type, id, link, image link, content) and fill subsequent rows with data (links, text, id).
- Step 3: Save the file.
  - Write the workbook to a file (iitjhtml.xlsx) using a FileOutputStream.
- Outcome: A new Excel file is created with the given data.

b) File watcher:

- Step 1: Asynchronous File Monitoring:

    - The method uses CompletableFuture.runAsync to run the file watcher in a separate thread, ensuring non-blocking execution.
    - This prevents the file monitoring process from interfering with the main application thread.
- Step 2: File Watcher Setup.
    - A WatchService is created to monitor the directory containing the specified file.
    - The directory is registered with the WatchService to listen for ENTRY_MODIFY events, which are triggered when the file is modified.
- Step 3: Trigger an action on modification.
    - When the file is modified, the watcher prints the file name and asynchronously triggers a method (readExcelDataAsync) to re-read the file.
- Outcome: When the file is modified, the watcher prints the file name and asynchronously triggers a method (readExcelDataAsync) to re-read the file.

c) Read excel:

- Step 1: Open the Excel file.
    - Use Apache POI to read the data.xlsx file with a FileInputStream.
- Step 2: Access the desired sheet.
    - Get the first sheet (Sheet) in the workbook.
- Step 3: Iterate through rows and cells.
    - Use a loop to read and print each cell's value from the sheet.
- Outcome: The system reads data from an Excel file and outputs it to the console.and data is ready for sending messages to stream.

d) Web socket connection:

- Step 1: Configure WebSocket endpoints.
    - Create a WebSocket configuration class that enables a messaging broker (`/topic/data-updates`) and sets up an endpoint (`http://localhost:8088/ws`) with SockJS support.
- Step 2: Start the WebSocket server.
    - Deploy the server as part of the Spring Boot application. Server is running on port 8088(`http://localhost:8088/ws`)
- Outcome: The server is ready for WebSocket connections, enabling real-time messaging.

e) Simp Messaging Template

- **Step 1**: Define a REST API endpoint.
    - Use Spring Boot's `@RestController` to expose an endpoint that accepts messages.

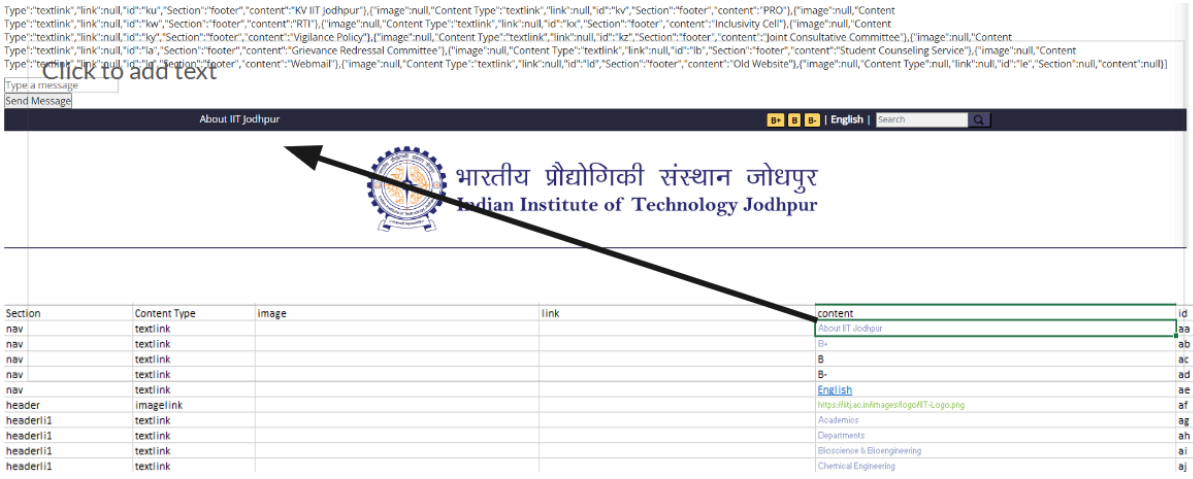
- **Step 2**: Use `SimpMessagingTemplate` to broadcast messages.
    - Convert and send messages to a topic (e.g., `/topic/messages`) using the `convertAndSend` method.
- **Outcome**: Messages are broadcast to all clients subscribed to the specified topic.

f) Subscribe data using stomp client

- Step 1: Initialize a WebSocket connection.
    - Use the SockJS library to connect to the /websocket endpoint.
- Step 2: Create a STOMP client.
    - Use the Stomp.over() function to initialize the STOMP protocol over the WebSocket connection.
- Step 3: Subscribe to a topic.
    - Subscribe to `/topic/data-updates` and handle incoming messages in a callback function.
- Outcome: The client receives real-time updates whenever a new message is sent to the topic.

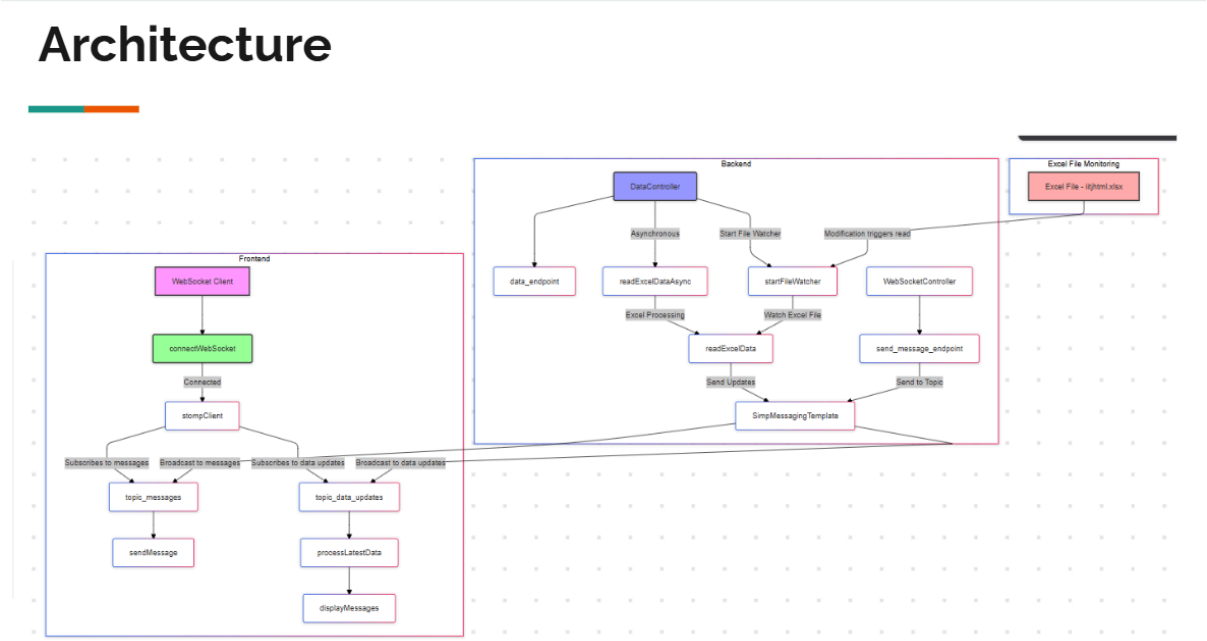
g)display and update messages using html:

- Step 1: Set up an HTML page.
    - Used iitj website source code as a frontend to display messages mapped by id.
- Step 2: Use JavaScript to connect to the WebSocket server.
    - Initialize a WebSocket connection and a STOMP client within a <script> tag.
- Step 3: Update the DOM in real-time.[code optimization for news section in webpage]
    - Append new list items (<li>) to the unordered list whenever a new message is received.
    - Otherwise change content by id
- Outcome: Messages are displayed dynamically(real time) on the web page as they arrive.



Section	Content Type	image	link	content	id
nav	textlink			About IIT Jodhpur	aa
nav	textlink			B+	ab
nav	textlink			B	ac
nav	textlink			B-	ad
nav	textlink			English	ae
header	imagelink			https://iitj.ac.in/images/logo/IIT-Logo.png	af
headerli1	textlink			Academics	ag
headerli1	textlink			Departments	ah
headerli1	textlink			Bioscience & Bioengineering	ai
headerli1	textlink			Chemical Engineering	aj

h) **Architecture:**



i) Assumptions

- When you got live website locally using Go live button in the vs code it will open in http://127.0.0.1:5500/code.html just replace 127.0.0.1 with localhost to overcome cors.
- Use excel file and give the system path for the excel file.

5.References:

1.https://stomp-js.github.io/guide/stompjs/rx-stomp/using-stomp-with-sockjs.html

2.https://mvnrepository.com/artifact/org.apache.poi/poi

3.https://www.iitj.ac.in/