**CCP-Batch Application Documentation**

**1. Introduction**

**CCP-Batch** is an automation tool developed using Spring Boot and Java 8. The application automates the end-to-end (E2E) flow of the main application (2CP portal). It processes input from a CSV file, generates a driver file, uploads the driver file via the MPPM API, creates orders using the data, and allows users to download the resulting PDF files. The application supports bulk order creation through a single REST API call.

**2. Overview**

* **Project Name:** CCP-Batch
* **Environment:** DEV
* **Deployment:** AWS EC2 Instance
* **API Endpoint:** **https://app.ccpdev.shared.banksvcs.net/ccp/api/batch/regressionBulkRun**

**3. Prerequisites**

Before setting up the project, ensure you have the following installed:

* Java 8
* Maven
* Git
* AWS CLI (for deployment to AWS EC2)
* SSH access to the GitLab repository

**4. Cloning the Repository**

To get started with the project, clone the repository using Git:

**git clone git@nikhil.gitlabdedicated.com:natwestgroup/DigitalX/Technology/CoreSolutions/CommunicationCompositionPlatform2CP/2cp/cs-communication-services/ccp-batch.git**

**5. Building the Application**

Navigate to the project directory and build the application using Maven:

bash

Copy code

mvn clean package

This command will compile the project, run the tests, and package the application into a JAR file located in the **target** directory.

**6. Running the Application Locally**

To run the application locally, use the following command:

**mvn clean install**

The application will start and listen on the port specified in the **application.properties** file.

**7. Configuration**

**7.1 Application Properties**

The **application.properties** file contains configuration settings for the Spring Boot application.

properties

Copy code

# Server port server.port=8080 # MPPM API configurations mppm.api.url=https://mppm.api.endpoint mppm.api.key=your-api-key # S3 configurations for PDF storage s3.bucket.name=recp-2cp-dev-pm-files s3.region=eu-west-1 # Logging configuration logging.level.org.springframework=INFO logging.file=application.log

**7.2 Security**

* Use environment variables or a secure vault for sensitive information such as API keys and database credentials.
* Ensure the API endpoint is secured using HTTPS.
* Implement authentication and authorization for API access.

**8. Deployment**

**8.1 Environment Setup**

1. **AWS EC2 Instance:**
   * Launch an EC2 instance with appropriate specifications (e.g., t2.medium).
   * Configure security groups to allow necessary inbound and outbound traffic.
2. **Java Installation:**
   * Install Java 8 on the EC2 instance.
   * Set up environment variables for Java.
3. **Spring Boot Application:**
   * Transfer the packaged JAR file to the EC2 instance.

**8.2 Deployment Steps**

1. **Transfer Files:**
   * Use SCP or a similar tool to transfer the JAR file to the EC2 instance.
2. **Run Application:**
   * SSH into the EC2 instance.
   * Run the application using the command: **java -jar ccp-batch-0.0.1-SNAPSHOT.jar**.
3. **Monitor Application:**
   * Use **screen** or **nohup** to run the application in the background.
   * Monitor logs using **tail -f application.log**.

**8.3 Continuous Integration/Continuous Deployment (CI/CD)**

* Set up a CI/CD pipeline using Jenkins, GitHub Actions, or AWS CodePipeline.
* Automate the build and deployment process to ensure smooth updates.

**9. API Usage**

**9.1 Endpoint**

* **URL:** **https://app.ccpdev.shared.banksvcs.net/ccp/api/batch/regressionBulkRun**
* **Method:** POST
* **Content-Type:** **multipart/form-data**

**9.2 Request Format**

The request should include a multipart/form-data body with the CSV file attached.

**9.3 Response Format**

The response is in JSON format, containing the status of each order and URLs for the generated PDF files.

**Sample Response:**

json

Copy code

{ "30MAY00008": { "Header": "", "Body": { "CreateConnectedOrderResponse": { "SuccessResponse": "Successfully created order with guid = 30MAY00008; proof was triggered for the order; the order was activated" } }, "URL": "https://recp-2cp-dev-pm-files.s3.eu-west-1.amazonaws.com/2CPLTRS\_30MAY00008.pdf" }, "30MAY00009": { "Header": "", "Body": { "CreateConnectedOrderResponse": { "SuccessResponse": "Successfully created order with guid = 30MAY00009; proof was triggered for the order; the order was activated" } }, "URL": "https://recp-2cp-dev-pm-files.s3.eu-west-1.amazonaws.com/2CPLTRS\_30MAY00009.pdf" }, "30MAY00010": { "Header": "", "Body": { "CreateConnectedOrderResponse": { "SuccessResponse": "Successfully created order with guid = 30MAY00010; proof was triggered for the order; the order was activated" } }, "URL": "https://recp-2cp-dev-pm-files.s3.eu-west-1.amazonaws.com/2CPLTRS\_30MAY00010.pdf" }, "30MAY00011": { "Header": "", "Body": { "CreateConnectedOrderResponse": { "SuccessResponse": "Successfully created order with guid = 30MAY00011; proof was triggered for the order; the order was activated" } }, "URL": "https://recp-2cp-dev-pm-files.s3.eu-west-1.amazonaws.com/2CPLTRS\_30MAY00011.pdf" }, "30MAY00012": { "Header": "", "Body": { "CreateConnectedOrderResponse": { "FailureResponse": "The input connector = Visa/MasterCard debit card PAN is unknown" } } }, "30MAY00007": { "Header": "", "Body": { "CreateConnectedOrderResponse": { "SuccessResponse": "Successfully created order with guid = 30MAY00007; proof was triggered for the order; the order was activated" } }, "URL": "https://recp-2cp-dev-pm-files.s3.eu-west-1.amazonaws.com/2CPLTRS\_30MAY00007.pdf" } }

**9.4 Error Handling**

* **SuccessResponse:** Indicates successful order creation and includes the URL for the PDF.
* **FailureResponse:** Provides details of any errors encountered during processing.

**Example Error Response:**

json

Copy code

{ "30MAY00012": { "Header": "", "Body": { "CreateConnectedOrderResponse": { "FailureResponse": "The input connector = Visa/MasterCard debit card PAN is unknown" } } } }

**10. Testing**

**10.1 Unit Testing**

* Use JUnit and Mockito for unit testing the application components.
* Write tests for the CSV Processor, Driver File Generator, MPPM API Client, Order Creator, PDF Generator, and Response Handler.

**Sample Unit Test**

java

Copy code

@RunWith(SpringRunner.class) @SpringBootTest public class CsvProcessorTest { @Autowired private CsvProcessor csvProcessor; @Test public void testProcessCsv() { // Arrange String csvContent = "template,team,centre,customer\_statement\_address,interview\_question\n" + "Template1,Team1,Centre1,Address1,Question1\n"; InputStream inputStream = new ByteArrayInputStream(csvContent.getBytes()); // Act List<OrderData> orderDataList = csvProcessor.processCsv(inputStream); // Assert assertNotNull(orderDataList); assertEquals(1, orderDataList.size()); assertEquals("Template1", orderDataList.get(0).getTemplate()); // Additional assertions... } }

**10.2 Integration Testing**

* Use Spring's **@SpringBootTest** for integration testing.
* Test the interaction between different components of the application.

**Sample Integration Test**

java

Copy code

@RunWith(SpringRunner.class) @SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM\_PORT) public class CcpBatchApplicationTests { @Autowired private TestRestTemplate restTemplate; @Test public void testApiEndpoint() { // Arrange String url = "/ccp/api/batch/regressionBulkRun"; MultiValueMap<String, Object> body = new LinkedMultiValueMap<>(); body.add("file", new ClassPathResource("test-data.csv")); HttpHeaders headers = new HttpHeaders(); headers.setContentType(MediaType.MULTIPART\_FORM\_DATA); HttpEntity<MultiValueMap<String, Object>> requestEntity = new HttpEntity<>(body, headers); // Act ResponseEntity<String> response = restTemplate.postForEntity(url, requestEntity, String.class); // Assert assertEquals(HttpStatus.OK, response.getStatusCode()); // Additional assertions to validate the response... } }

**11. Logging and Monitoring**

**11.1 Logging**

* Configure logging in the **application.properties** file.
* Use SLF4J with Logback for logging.

**11.2 Monitoring**

* Use tools like Prometheus and Grafana for monitoring application metrics.
* Set up alerts for critical metrics and log events.

**12. Maintenance and Support**

**12.1 Troubleshooting**

* Check application logs for error messages and stack traces.
* Use AWS CloudWatch for centralized log management and monitoring.

**12.2 Contact Information**

For support, contact the development team at support@example.com.