Ingestion Module – Complete Line-by-Line Code Explanation

Data Pipeline Service – Ingestion Package
Document Version: 2.0
Purpose: Every line of code explained in detail.

Table of Contents

---

1. UploadResponse.java

Full source code:

```
1 package com.mypolicy.pipeline.ingestion.dto; 2 3 import
com.mypolicy.pipeline.ingestion.model.IngestionStatus; 4 5 /** 6 * Response
after successful file upload. 7 */ 8 public class UploadResponse { 9 private
String jobId; 10 private IngestionStatus status; 11 12 public UploadResponse()
{ 13 } 14 15 public UploadResponse(String jobId, IngestionStatus status) { 16
this.jobId = jobId; 17 this.status = status; 18 } 19 20 public String
getJobId() { return jobId; } 21 public void setJobId(String jobId) {
this.jobId = jobId; } 22 public IngestionStatus getStatus() { return status; }
23 public void setStatus(IngestionStatus status) { this.status = status; } 24
}
```

Line-by-line explanation:

Line	Code	Explanation
1	`package com.mypolicy.pipeline.ingestion.dto;`	Declares that this class belongs to the Data Transfer Object package for the ingestion module.
2	*(blank)*	Blank line for readability.
3	`import com.mypolicy.pipeline.ingestion.model.IngestionStatus;`	Imports the IngestionStatus enum so we can use it as a field type.
4	*(blank)*	Blank line.
5-7	`/** ... */`	Javadoc comment: describes that this class is the response returned after a successful file upload.
8	`public class UploadResponse {`	Starts the class definition. Public means it can be used by other packages.

9	`private String jobId;`	Private field: stores the unique job identifier (e.g. UUID string) returned to the client.
10	`private IngestionStatus status;`	Private field: stores the initial status (UPLOADED) after upload.
11	*(blank)*	Blank line.
12-13	`public UploadResponse() { }`	No-argument constructor. Used by JSON libraries when deserializing; creates an empty object.
14	*(blank)*	Blank line.
15	`public UploadResponse(String jobId, IngestionStatus status) {`	Constructor that accepts both field values.
16	`this.jobId = jobId;`	Assigns the jobId parameter to the instance field. `this.` refers to the object being constructed.
17	`this.status = status;`	Assigns the status parameter to the instance field.

18	`}`	Closes the constructor.
19	*(blank)*	Blank line.
20	`public String getJobId() { return jobId; }`	Getter for jobId. Returns the current value of the jobId field.
21	`public void setJobId(String jobId) { this.jobId = jobId; }`	Setter for jobId. Updates the jobId field with the new value.
22	`public IngestionStatus getStatus() { return status; }`	Getter for status. Returns the current status.
23	`public void setStatus(IngestionStatus status) { this.status = status; }`	Setter for status. Updates the status field.
24	`}`	Closes the class.

---

2. JobStatusResponse.java

Full source code:

```
1 package com.mypolicy.pipeline.ingestion.dto; 2 3 import
com.mypolicy.pipeline.ingestion.model.IngestionStatus; 4 5 import
java.time.LocalDateTime; 6 7 /** 8 * Job status response for BFF/Processing
Service. 9 */ 10 public class JobStatusResponse { 11 private String jobId; 12
private IngestionStatus status; 13 private int processedRecords; 14 private
int totalRecords; 15 private String filePath; 16 private String insurerId; 17
```

```
private String fileType; 18 private LocalDateTime createdAt; 19 private LocalDateTime updatedAt; 20 21 public JobStatusResponse() { 22 } 23 24 public JobStatusResponse(String jobId, IngestionStatus status, int processedRecords, 25 int totalRecords, String filePath, String insurerId, String fileType, 26 LocalDateTime createdAt, LocalDateTime updatedAt) { 27 this.jobId = jobId; 28 this.status = status; 29 this.processedRecords = processedRecords; 30 this.totalRecords = totalRecords; 31 this.filePath = filePath; 32 this.insurerId = insurerId; 33 this.fileType = fileType; 34 this.createdAt = createdAt; 35 this.updatedAt = updatedAt; 36 } 37 38 public String getJobId() { return jobId; } 39 public void setJobId(String jobId) { this.jobId = jobId; } 40 public IngestionStatus getStatus() { return status; } 41 public void setStatus(IngestionStatus status) { this.status = status; } 42 public int getProcessedRecords() { return processedRecords; } 43 public void setProcessedRecords(int processedRecords) { this.processedRecords = processedRecords; } 44 public int getTotalRecords() { return totalRecords; } 45 public void setTotalRecords(int totalRecords) { this.totalRecords = totalRecords; } 46 public String getFilePath() { return filePath; } 47 public void setFilePath(String filePath) { this.filePath = filePath; } 48 public String getInsurerId() { return insurerId; } 49 public void setInsurerId(String insurerId) { this.insurerId = insurerId; } 50 public String getFileType() { return fileType; } 51 public void setFileType(String fileType) { this.fileType = fileType; } 52 public LocalDateTime getCreatedAt() { return createdAt; } 53 public void setCreatedAt(LocalDateTime createdAt) { this.createdAt = createdAt; } 54 public LocalDateTime getUpdatedAt() { return updatedAt; } 55 public void setUpdatedAt(LocalDateTime updatedAt) { this.updatedAt = updatedAt; } 56 }
```

Line-by-line explanation:

Line	Code	Explanation
1	`package com.mypolicy.pipeline.ingestion.dto;`	Package declaration for DTO classes.
3	`import com.mypolicy.pipeline.ingestion.model.IngestionStatus;`	Import for the status enum.
5	`import java.time.LocalDateTime;`	Import for date/time without timezone; used for createdAt and updatedAt.

7-9	`/** Job status response... */`	Javadoc: this DTO is used by BFF and Processing Service to return full job details.
10	`public class JobStatusResponse {`	Class definition.
11	`private String jobId;`	Unique job identifier.
12	`private IngestionStatus status;`	Current lifecycle state (UPLOADED, PROCESSING, COMPLETED, FAILED).
13	`private int processedRecords;`	Number of records processed so far.
14	`private int totalRecords;`	Total records in the file (for progress calculation).
15	`private String filePath;`	Absolute path where the file is stored on disk.
16	`private String insurerId;`	Insurer ID this job belongs to.
17	`private String fileType;`	"normal" or "correction".
18	`private LocalDateTime createdAt;`	When the job was created.

19	`private LocalDateTime updatedAt;`	When the job was last updated.
21-22	`public JobStatusResponse() { }`	No-arg constructor for deserialization.
24-26	`public JobStatusResponse(...)`	Full constructor with all 9 parameters.
27-35	`this.jobId = jobId;` etc.	Assigns each parameter to the corresponding field.
38-55	Getters and setters	Standard JavaBean accessors for all fields; used for JSON serialization and reading/writing.

---

3. ProgressUpdateRequest.java

Full source code:

```
1 package com.mypolicy.pipeline.ingestion.dto; 2 3 /** 4 * Internal request for incrementing processed records. 5 * Idempotent when retried with same delta. 6 */ 7 public class ProgressUpdateRequest { 8 private int processedRecordsDelta; 9 10 public int getProcessedRecordsDelta() { return processedRecordsDelta; } 11 public void setProcessedRecordsDelta(int processedRecordsDelta) { 12 this.processedRecordsDelta = processedRecordsDelta; 13 } 14 }
```

Line-by-line explanation:

Line	Code	Explanation
1	`package com.mypolicy.pipeline.ingestion.dto;`	Package declaration.
3-6	`/** Internal request... */`	Javadoc: used internally when Processing sends progress updates. Idempotent means repeating the same request has the same effect.
7	`public class ProgressUpdateRequest {`	Class definition.
8	`private int processedRecordsDelta;`	Holds how many more records were processed (e.g. 100). Added to the job's processedRecords.
10	`public int getProcessedRecordsDelta() { return processedRecordsDelta; }`	Getter.
11-13	`public void setProcessedRecordsDelta(...)`	Setter. The JSON body `{"processedRecordsDelta": 50}` maps to this.

---

4. StatusUpdateRequest.java

Full source code:

```
1 package com.mypolicy.pipeline.ingestion.dto; 2 3 import
com.mypolicy.pipeline.ingestion.model.IngestionStatus; 4 5 /** 6 * Internal
request for job status transition. 7 * Enforces state machine: UPLOADED →
PROCESSING → COMPLETED | FAILED 8 */ 9 public class StatusUpdateRequest { 10
private IngestionStatus status; 11 private String failureReason; 12 13 public
IngestionStatus getStatus() { return status; } 14 public void
setStatus(IngestionStatus status) { this.status = status; } 15 public String
getFailureReason() { return failureReason; } 16 public void
setFailureReason(String failureReason) { this.failureReason = failureReason; }
17 }
```

Line-by-line explanation:

Line	Code	Explanation
1	`package com.mypolicy.pipeline.ingestion.dto;`	Package declaration.
3	`import com.mypolicy.pipeline.ingestion.model.IngestionStatus;`	Import for status enum.
5-8	Javadoc	Describes that this request changes job status according to the state machine.
9	`public class StatusUpdateRequest {`	Class definition.
10	`private IngestionStatus status;`	Target status (PROCESSING, COMPLETED, or FAILED).
11	`private String failureReason;`	Error message when status is FAILED; null otherwise.

13-16	Getters and setters	Accessors for both fields.

---

5. IngestionStatus.java

Full source code:

```
1 package com.mypolicy.pipeline.ingestion.model; 2 3 /** 4 * Ingestion job lifecycle states. 5 * State machine: UPLOADED → PROCESSING → COMPLETED | FAILED 6 */ 7 public enum IngestionStatus { 8 UPLOADED, PROCESSING, COMPLETED, FAILED 9 }
```

Line-by-line explanation:

Line	Code	Explanation
1	`package com.mypolicy.pipeline.ingestion.model;`	Package for domain models.
3-6	Javadoc	Describes the four allowed states and the flow.
7	`public enum IngestionStatus {`	Enum definition. Only these four values are valid.
8	`UPLOADED, PROCESSING, COMPLETED, FAILED`	UPLOADED = file stored; PROCESSING = being processed; COMPLETED = success; FAILED = error.

---

6. IngestionJob.java

Full source code:

```
1 package com.mypolicy.pipeline.ingestion.model; 2 3 import
org.springframework.data.annotation.Id; 4 import
org.springframework.data.mongodb.core.index.Indexed; 5 import
org.springframework.data.mongodb.core.mapping.Document; 6 7 import
java.time.LocalDateTime; 8 9 /** 10 * Ingestion job document - tracks file
upload and processing lifecycle. 11 * Stored in MongoDB for flexible schema
and high write throughput. 12 * 13 * Consolidated Service: Part of
data-pipeline-service. 14 */ 15 @Document(collection = "ingestion_jobs") 16
public class IngestionJob { 17 18 @Id 19 private String jobId; 20 21 private
String insurerId; 21 private String filePath; 22 23 @Indexed 24 private
IngestionStatus status; 25 private int totalRecords; 26 private int
processedRecords; 27 private String uploadedBy; 28 private String
failureReason; 29 30 @Indexed 31 private LocalDateTime createdAt; 32 private
LocalDateTime updatedAt; 33 34 public IngestionJob() { 35 } 36 37 public
IngestionJob(String jobId, String insurerId, String filePath, IngestionStatus
status, 38 int totalRecords, int processedRecords, String uploadedBy, String
failureReason, 39 LocalDateTime createdAt, LocalDateTime updatedAt) { 40
this.jobId = jobId; 41 this.insurerId = insurerId; 42 this.filePath =
filePath; 43 this.status = status; 44 this.totalRecords = totalRecords; 45
this.processedRecords = processedRecords; 46 this.uploadedBy = uploadedBy; 47
this.failureReason = failureReason; 48 this.createdAt = createdAt; 49
this.updatedAt = updatedAt; 50 } 51 52 public String getJobId() { return
jobId; } 53 public void setJobId(String jobId) { this.jobId = jobId; } 54
public String getInsurerId() { return insurerId; } 55 public void
setInsurerId(String insurerId) { this.insurerId = insurerId; } 56 public
String getFilePath() { return filePath; } 57 public void setFilePath(String
filePath) { this.filePath = filePath; } 58 public IngestionStatus getStatus()
{ return status; } 59 public void setStatus(IngestionStatus status) {
this.status = status; } 60 public int getTotalRecords() { return totalRecords;
} 61 public void setTotalRecords(int totalRecords) { this.totalRecords =
totalRecords; } 62 public int getProcessedRecords() { return processedRecords;
} 63 public void setProcessedRecords(int processedRecords) {
this.processedRecords = processedRecords; } 64 public String getUploadedBy() {
return uploadedBy; } 65 public void setUploadedBy(String uploadedBy) {
this.uploadedBy = uploadedBy; } 66 public String getFailureReason() { return
failureReason; } 67 public void setFailureReason(String failureReason) {
this.failureReason = failureReason; } 68 public LocalDateTime getCreatedAt() {
return createdAt; } 69 public void setCreatedAt(LocalDateTime createdAt) {
this.createdAt = createdAt; } 70 public LocalDateTime getUpdatedAt() { return
updatedAt; } 71 public void setUpdatedAt(LocalDateTime updatedAt) {
this.updatedAt = updatedAt; } 72 }
```

Line-by-line explanation:

Line	Code	Explanation

1	`package com.mypolicy.pipeline.ingestion.model;`	Model package.
3	`import org.springframework.data.annotation.Id;`	@Id marks the MongoDB document ID.
4	`import org.springframework.data.mongodb.core.index.Indexed;`	@Indexed creates an index for faster queries.
5	`import org.springframework.data.mongodb.core.mapping.Document;`	@Document maps this class to a MongoDB collection.
7	`import java.time.LocalDateTime;`	For timestamps.
15	`@Document(collection = "ingestion_jobs")`	MongoDB collection name.
18-19	`@Id private String jobId;`	jobId is the document's primary key.
20-21	`private String insurerId; private String filePath;`	Insurer ID and file storage path.
23-28	`@Indexed status` and other fields	status is indexed for findByStatus; totalRecords, processedRecords, uploadedBy, failureReason.

30-32	`@Indexed createdAt updatedAt`	Timestamps; indexed for time-based queries.
34-35	`public IngestionJob() { }`	No-arg constructor for MongoDB deserialization.
37-50	Full constructor	Assigns all 10 fields.
52-71	Getters and setters	Standard accessors for all fields.

---

7. IngestionJobRepository.java

Full source code:

```
1 package com.mypolicy.pipeline.ingestion.repository; 2 3 import
com.mypolicy.pipeline.ingestion.model.IngestionJob; 4 import
com.mypolicy.pipeline.ingestion.model.IngestionStatus; 5 import
org.springframework.data.mongodb.repository.MongoRepository; 6 import
org.springframework.stereotype.Repository; 7 8 import java.util.List; 9 10 /**
11 * MongoDB repository for ingestion job tracking. 12 * 13 * Consolidated
Service: Part of data-pipeline-service. 14 */ 15 @Repository 16 public
interface IngestionJobRepository extends MongoRepository<IngestionJob, String>
{ 17 List<IngestionJob> findByStatus(IngestionStatus status); 18
List<IngestionJob> findByInsurerId(String insurerId); 19 }
```

Line-by-line explanation:

Line	Code	Explanation
1	`package com.mypolicy.pipeline.ingestion.repository;`	Repository package.

3-4	Imports	IngestionJob entity and IngestionStatus enum.
5	`import org.springframework.data.mongodb.repository.MongoRepository;`	Base interface for MongoDB CRUD.
6	`import org.springframework.stereotype.Repository;`	Marks this as a Spring repository bean.
8	`import java.util.List;`	For list return types.
15	`@Repository`	Spring creates a bean implementing this interface.
16	`extends MongoRepository<IngestionJob, String>`	Entity type = IngestionJob, ID type = String. Provides save, findById, delete, etc.
17	`List<IngestionJob> findByStatus(IngestionStatus status);`	Spring Data generates the query. Finds all jobs with the given status.
18	`List<IngestionJob> findByInsurerId(String insurerId);`	Finds all jobs for a given insurer.

---

8. IngestionController.java

Full source code:

```
1 package com.mypolicy.pipeline.ingestion.controller; 2 3 import
com.mypolicy.pipeline.ingestion.dto.JobStatusResponse; 4 import
com.mypolicy.pipeline.ingestion.dto.ProgressUpdateRequest; 5 import
com.mypolicy.pipeline.ingestion.dto.StatusUpdateRequest; 6 import
com.mypolicy.pipeline.ingestion.dto.UploadResponse; 7 import
com.mypolicy.pipeline.ingestion.service.IngestionService; 8 import
jakarta.validation.Valid; 9 import org.slf4j.Logger; 10 import
org.slf4j.LoggerFactory; 11 import org.springframework.http.HttpStatus; 12
import org.springframework.http.ResponseEntity; 13 import
org.springframework.web.bind.annotation.*; 14 import
org.springframework.web.multipart.MultipartFile; 15 16 import
java.io.IOException; 17 18 /** 19 * Ingestion API: file upload, status
retrieval, progress/status updates. 20 * JWT validation is done at BFF level
for upload. 21 * 22 * Consolidated Service: Part of data-pipeline-service on
port 8082. 23 */ 24 @RestController 25 @RequestMapping("/api/v1/ingestion") 26
@RequiredArgsConstructor 27 public class IngestionController { 27 28 private
static final Logger log = LoggerFactory.getLogger(IngestionController.class);
29 private final IngestionService ingestionService; 30 31
@PostMapping("/upload") 32 public ResponseEntity<UploadResponse> uploadFile(
32 @RequestParam("file") MultipartFile file, 33 @RequestParam("insurerId")
String insurerId, 34 @RequestParam("uploadedBy") String uploadedBy, 35
@RequestParam(value = "fileType", required = false) String fileType) { 36 37
log.info("[Ingestion API] POST /upload - insurerId={}, uploadedBy={}",
insurerId, uploadedBy); 38 39 try { 40 UploadResponse response =
ingestionService.uploadFile(file, insurerId, uploadedBy, fileType); 41
log.info("[Ingestion API] Upload successful: jobId={}", response.getJobId());
42 return ResponseEntity.status(HttpStatus.CREATED).body(response); 43 } catch
(IllegalArgumentException e) { 45 log.warn("[Ingestion API] Upload validation
failed: {}", e.getMessage()); 46 throw e; 47 } catch (IOException e) { 48
log.error("[Ingestion API] File storage failed", e); 49 throw new
RuntimeException("Error storing file", e); 50 } 51 } 52 53
@GetMapping("/status/{jobId}") 54 public ResponseEntity<JobStatusResponse>
getJobStatus(@PathVariable String jobId) { 55 log.debug("[Ingestion API] GET
/status/{}", jobId); 56 JobStatusResponse response =
ingestionService.getJobStatus(jobId); 57 return ResponseEntity.ok(response);
58 } 59 60 @PatchMapping("/{jobId}/progress") 61 public ResponseEntity<Void>
updateProgress( 61 @PathVariable String jobId, 62 @Valid @RequestBody
ProgressUpdateRequest request) { 63 64 log.debug("[Ingestion API] PATCH
/{}/progress - delta={}", jobId, request.getProcessedRecordsDelta()); 65
ingestionService.updateProgress(jobId, request); 66 return
ResponseEntity.noContent().build(); 67 } 68 69
@PatchMapping("/{jobId}/status") 70 public ResponseEntity<Void> updateStatus(
71 @PathVariable String jobId, 72 @Valid @RequestBody StatusUpdateRequest
request) { 73 74 log.info("[Ingestion API] PATCH /{}/status - newStatus={}",
jobId, request.getStatus()); 75 ingestionService.updateStatus(jobId, request);
76 return ResponseEntity.noContent().build(); 77 } 78 79
@GetMapping("/health") 80 public ResponseEntity<String> health() { 81 return
ResponseEntity.ok("Ingestion module healthy"); 82 } 83 }
```

Line-by-line explanation:

Line	Code	Explanation
1	`package com.mypolicy.pipeline.ingestion.controller;`	Controller package.
3-6	DTO imports	UploadResponse, JobStatusResponse, ProgressUpdateRequest, StatusUpdateRequest.
7	`import com.mypolicy.pipeline.ingestion.service.IngestionService;`	Service to delegate business logic.
8	`import jakarta.validation.Valid;`	@Valid triggers validation on request bodies.
9-10	Logger imports	For logging.
11-12	HttpStatus, ResponseEntity	For HTTP responses.
13	`import org.springframework.web.bind.annotation.*;`	@RestController, @PostMapping, @GetMapping, @PatchMapping, @RequestParam, @PathVariable, @RequestBody.
14	`import org.springframework.web.multipart.MultipartFile;`	Represents uploaded file.
16	`import java.io.IOException;`	For file I/O exceptions.

24	`@RestController`	Marks this as a REST controller; methods return data (not view names).
25	`@RequestMapping("/api/v1/ingestion")`	All endpoints start with /api/v1/ingestion.
26	`@RequiredArgsConstructor`	Lombok generates constructor for final fields (ingestionService).
28	`private static final Logger log = ...`	Logger instance for this class.
29	`private final IngestionService ingestionService;`	Injected service; final = set once in constructor.
31	`@PostMapping("/upload")`	Maps POST /api/v1/ingestion/upload to this method.
32-35	Method parameters	file (required), insurerId (required), uploadedBy (required), fileType (optional).
37	`log.info(...)`	Logs upload attempt.

39-50	try-catch	Calls service; returns 201; catches validation (rethrows) and IOException (wraps in RuntimeException).
53	`@GetMapping("/status/{jobId}")`	Maps GET /api/v1/ingestion/status/{jobId}.
54	`@PathVariable String jobId`	Reads jobId from URL path.
55-57	Log, service call, return	Returns 200 with JobStatusResponse.
60-67	updateProgress	PATCH /{jobId}/progress; validates body; returns 204.
69-77	updateStatus	PATCH /{jobId}/status; validates body; returns 204.
79-82	health	GET /health; returns 200 with "Ingestion module healthy".

---

9. PublicIngestionController.java

Full source code:

```
1 package com.mypolicy.ingestion.controller; 2 3 import
com.mypolicy.ingestion.dto.JobStatusResponse; 4 import
```

```
com.mypolicy.ingestion.dto.UploadResponse; 5 import
com.mypolicy.ingestion.service.IngestionService; 6 import org.slf4j.Logger; 7
import org.slf4j.LoggerFactory; 8 import org.springframework.http.HttpStatus;
9 import org.springframework.http.ResponseEntity; 10 import
org.springframework.web.bind.annotation.*; 11 import
org.springframework.web.multipart.MultipartFile; 12 13 import
java.io.IOException; 14 15 /** 16 * Exposed customer-facing ingestion API. 17
* Requires X-API-Key header (validated by ApiKeyAuthFilter). 18 * Delegates to
IngestionService - no business logic here. 19 */ 20 @RestController 21
@RequestMapping("/api/public/v1/ingestion") 22 public class
PublicIngestionController { 23 24 private static final Logger log =
LoggerFactory.getLogger(PublicIngestionController.class); 25 private final
IngestionService ingestionService; 26 27 public
PublicIngestionController(IngestionService ingestionService) { 28
this.ingestionService = ingestionService; 28 } 29 30 @PostMapping("/upload")
31 public ResponseEntity<UploadResponse> uploadFile( 32 @RequestParam("file")
MultipartFile file, 33 @RequestParam("insurerId") String insurerId, 34
@RequestParam("uploadedBy") String uploadedBy, 35 @RequestParam(value =
"fileType", required = false) String fileType) { 36 37 try { 38 UploadResponse
response = ingestionService.uploadFile(file, insurerId, uploadedBy, fileType);
39 return ResponseEntity.status(HttpStatus.CREATED).body(response); 40 } catch
(IllegalArgumentException e) { 41 log.warn("Upload validation failed: {}",
e.getMessage()); 42 throw e; 43 } catch (IOException e) { 44 log.error("File
storage failed", e); 45 throw new RuntimeException("Error storing file", e);
46 } 47 } 48 49 @GetMapping("/status/{jobId}") 50 public
ResponseEntity<JobStatusResponse> getJobStatus(@PathVariable String jobId) {
51 JobStatusResponse response = ingestionService.getJobStatus(jobId); 52
return ResponseEntity.ok(response); 53 } 54 }
```

Line-by-line explanation:

Line	Code	Explanation
1	`package com.mypolicy.ingestion.controller;`	Package (note: differs from pipeline.ingestion; may need alignment).
3-5	Imports	JobStatusResponse, UploadResponse, IngestionService.
20-21	`@RestController @RequestMapping("/api/public/v1/ingestion")`	Base path for public API.

27-28	Constructor	Injects IngestionService.
30-47	uploadFile	Same logic as IngestionController; expects X-API-Key in header (enforced by filter).
49-53	getJobStatus	Returns job status; also requires API key.

---

10. IngestionService.java

Full source code:

```
1 package com.mypolicy.pipeline.ingestion.service; 2 3 import
com.mypolicy.pipeline.ingestion.dto.JobStatusResponse; 4 import
com.mypolicy.pipeline.ingestion.dto.ProgressUpdateRequest; 5 import
com.mypolicy.pipeline.ingestion.dto.StatusUpdateRequest; 6 import
com.mypolicy.pipeline.ingestion.dto.UploadResponse; 7 import
com.mypolicy.pipeline.ingestion.model.IngestionJob; 8 import
com.mypolicy.pipeline.ingestion.model.IngestionStatus; 9 import
com.mypolicy.pipeline.ingestion.repository.IngestionJobRepository; 10 import
com.mypolicy.ingestion.validation.InsurerSchemaValidator; 11 import
com.mypolicy.ingestion.validation.SchemaValidationResult; 12 import
lombok.RequiredArgsConstructor; 13 import org.slf4j.Logger; 14 import
org.slf4j.LoggerFactory; 15 import
org.springframework.beans.factory.annotation.Value; 16 import
org.springframework.stereotype.Service; 17 import
org.springframework.web.multipart.MultipartFile; 18 19 import
java.io.IOException; 20 import java.io.InputStream; 21 import
java.nio.file.Files; 22 import java.nio.file.Path; 23 import
java.nio.file.Paths; 24 import java.time.LocalDateTime; 25 import
java.util.Arrays; 26 import java.util.List; 27 import java.util.UUID; 28 29
@Service 30 @RequiredArgsConstructor 31 public class IngestionService { 32 33
private static final Logger log =
LoggerFactory.getLogger(IngestionService.class); 34 private static final
List<String> ALLOWED_EXTENSIONS = Arrays.asList(".xls", ".xlsx", ".csv"); 35
private static final long MAX_FILE_SIZE_BYTES = 50 * 1024 * 1024; // 50MB 36
37 private final IngestionJobRepository jobRepository; 37 private final
InsurerSchemaValidator schemaValidator; 38 39
@Value("${ingestion.storage.path:storage/ingestion}") 40 private String
```

```
storageBasePath; 41 42 @Value("${ingestion.schema.validate:true}") 42 private
boolean schemaValidationEnabled; 43 44 public UploadResponse
uploadFile(MultipartFile file, String insurerId, String uploadedBy, 45 String
fileType) throws IOException { 46 47 log.info("[Ingestion] Starting file
upload: insurerId={}, filename={}", insurerId, file.getOriginalFilename()); 48
49 validateFile(file); 50 51 String resolvedFileType =
resolveFileType(fileType, file.getOriginalFilename()); 52 53 if
(schemaValidationEnabled && "normal".equals(resolvedFileType)) { 54
SchemaValidationResult schemaResult = schemaValidator.validate(file,
insurerId); 55 if (!schemaResult.isValid()) { 56 throw new
IllegalArgumentException(schemaResult.getErrorSummary()); 57 } 58 } 59 60
String jobId = UUID.randomUUID().toString(); 61 String extension =
getFileExtension(file.getOriginalFilename()); 62 Path storagePath =
Paths.get(storageBasePath); 63 if (!Files.exists(storagePath)) { 64
Files.createDirectories(storagePath); 65 } 66 67 Path filePath =
storagePath.resolve(jobId + extension); 68 try (InputStream inputStream =
file.getInputStream()) { 69 Files.copy(inputStream, filePath); 70 } 71 73
log.info("[Ingestion] File uploaded: jobId={}, insurerId={}, fileType={},
path={}", jobId, insurerId, 74 resolvedFileType, filePath.toAbsolutePath());
75 76 IngestionJob job = new IngestionJob(jobId, insurerId,
filePath.toAbsolutePath().toString(), 77 resolvedFileType,
IngestionStatus.UPLOADED, 0, 0, uploadedBy, null, 78 LocalDateTime.now(),
LocalDateTime.now()); 79 80 jobRepository.save(job); 81 82
log.info("[Ingestion] Job created: jobId={}, status=UPLOADED", jobId); 83 84
return new UploadResponse(jobId, IngestionStatus.UPLOADED); 85 } 86 87 public
JobStatusResponse getJobStatus(String jobId) { 88 log.debug("[Ingestion]
Fetching job status: jobId={}", jobId); 89 90 IngestionJob job =
jobRepository.findById(jobId) 91 .orElseThrow(() -> new
IllegalArgumentException("Job not found: " + jobId)); 92 93 return new
JobStatusResponse(job.getJobId(), job.getStatus(), job.getProcessedRecords(),
94 job.getTotalRecords(), job.getFilePath(), job.getInsurerId(),
job.getFileType(), 95 job.getCreatedAt(), job.getUpdatedAt()); 96 } 97 98
public IngestionJob getJob(String jobId) { 99 return
jobRepository.findById(jobId) 100 .orElseThrow(() -> new
IllegalArgumentException("Job not found: " + jobId)); 101 } 102 104 public
void updateProgress(String jobId, ProgressUpdateRequest request) { 105 if
(request.getProcessedRecordsDelta() <= 0) { 106 throw new
IllegalArgumentException("processedRecordsDelta must be positive"); 107 } 108
109 IngestionJob job = jobRepository.findById(jobId) 110 .orElseThrow(() ->
new IllegalArgumentException("Job not found: " + jobId)); 111 112 if
(job.getStatus() != IngestionStatus.PROCESSING) { 113 throw new
IllegalStateException( 114 "Cannot update progress: job must be in PROCESSING
state, current=" + job.getStatus()); 115 } 116 117
job.setProcessedRecords(job.getProcessedRecords() +
request.getProcessedRecordsDelta()); 118
job.setUpdatedAt(LocalDateTime.now()); 119 jobRepository.save(job); 120 121
log.debug("[Ingestion] Progress updated: jobId={}, processed={}/{}", jobId,
job.getProcessedRecords(), 122 job.getTotalRecords()); 123 } 124 125 public
void updateStatus(String jobId, StatusUpdateRequest request) { 126
IngestionJob job = jobRepository.findById(jobId) 127 .orElseThrow(() -> new
IllegalArgumentException("Job not found: " + jobId)); 128 129 IngestionStatus
newStatus = request.getStatus(); 130 if (newStatus == null) { 131 throw new
```

```
IllegalArgumentException("status is required"); 131 } 132 133 IngestionStatus
oldStatus = job.getStatus(); 134 validateStateTransition(oldStatus,
newStatus); 135 136 job.setStatus(newStatus); 136
job.setFailureReason(request.getFailureReason()); 137
job.setUpdatedAt(LocalDateTime.now()); 138 jobRepository.save(job); 139 140
log.info("[Ingestion] Status transition: jobId={}, {} -> {}", jobId,
oldStatus, newStatus); 141 } 142 143 public void setTotalRecords(String jobId,
int totalRecords) { 144 IngestionJob job = jobRepository.findById(jobId) 145
.orElseThrow(() -> new IllegalArgumentException("Job not found: " + jobId));
146 147 if (job.getStatus() != IngestionStatus.UPLOADED && job.getStatus() !=
IngestionStatus.PROCESSING) { 148 throw new IllegalStateException("Cannot set
totalRecords in state: " + job.getStatus()); 149 } 150 151
job.setTotalRecords(totalRecords); 152 job.setUpdatedAt(LocalDateTime.now());
153 jobRepository.save(job); 154 155 log.debug("[Ingestion] Total records set:
jobId={}, total={}", jobId, totalRecords); 156 } 157 158 private void
validateFile(MultipartFile file) { 159 if (file == null || file.isEmpty()) {
160 throw new IllegalArgumentException("File is empty or missing"); 161 } 162
163 if (file.getSize() > MAX_FILE_SIZE_BYTES) { 164 throw new
IllegalArgumentException( 165 "File size exceeds maximum allowed: " +
(MAX_FILE_SIZE_BYTES / 1024 / 1024) + "MB"); 166 } 167 168 String originalName
= file.getOriginalFilename(); 169 if (originalName == null ||
originalName.isBlank()) { 170 throw new IllegalArgumentException("File name is
missing"); 171 } 172 173 String ext = getFileExtension(originalName); 174 if
(ext == null || !ALLOWED_EXTENSIONS.contains(ext.toLowerCase())) { 175 throw
new IllegalArgumentException( 176 "Invalid file type. Allowed: " +
String.join(", ", ALLOWED_EXTENSIONS)); 177 } 178 } 179 180 private String
getFileExtension(String filename) { 181 if (filename == null) return null; 182
int lastDot = filename.lastIndexOf('.'); 183 return lastDot > 0 ?
filename.substring(lastDot) : null; 184 } 185 186 private String
resolveFileType(String fileTypeParam, String filename) { 187 if (fileTypeParam
!= null && "correction".equalsIgnoreCase(fileTypeParam.trim())) 188 return
"correction"; 189 if (filename != null &&
filename.toLowerCase().contains("_correction")) 190 return "correction"; 191
return "normal"; 192 } 193 194 private void
validateStateTransition(IngestionStatus current, IngestionStatus next) { 195
switch (current) { 196 case UPLOADED: 197 if (next !=
IngestionStatus.PROCESSING) { 198 throw new IllegalStateException("Invalid
transition: UPLOADED -> " + next); 199 } 200 break; 201 case PROCESSING: 202
if (next != IngestionStatus.COMPLETED && next != IngestionStatus.FAILED) { 203
throw new IllegalStateException("Invalid transition: PROCESSING -> " + next);
204 } 205 break; 206 case COMPLETED: 207 case FAILED: 208 throw new
IllegalStateException("No transitions allowed from terminal state: " +
current); 209 default: 210 throw new IllegalStateException("Unknown status: "
+ current); 211 } 212 } 213 }
```

Line-by-line explanation (IngestionService):

Line	Code	Explanation

1	`package com.mypolicy.pipeline.ingestion.service;`	Service package.
3-6	DTO imports	JobStatusResponse, ProgressUpdateRequest, StatusUpdateRequest, UploadResponse.
7-9	Model/repo imports	IngestionJob, IngestionStatus, IngestionJobRepository.
10-11	InsurerSchemaValidator, SchemaValidationResult	Schema validation from ingestion-service module.
12	`import lombok.RequiredArgsConstructor;`	Lombok: generates constructor for final fields.
19-27	java.io, java.nio.file, java.time, java.util	I/O, paths, dates, Arrays, List, UUID.
29	`@Service`	Spring service bean.
30	`@RequiredArgsConstructor`	Constructor injection for jobRepository and schemaValidator.
33	`private static final Logger log = ...`	Logger.

34	`ALLOWED_EXTENSIONS = Arrays.asList(".xls", ".xlsx", ".csv")`	Allowed file extensions.
35	`MAX_FILE_SIZE_BYTES = 50 * 1024 * 1024`	50MB limit.
36-37	`jobRepository`, `schemaValidator`	Injected dependencies.
39-40	`@Value("${ingestion.storage.path:storage/ingestion}")`	Config property with default.
41-42	`@Value("${ingestion.schema.validate:true}")`	Enable/disable schema validation.
47	`log.info(...)`	Log upload start.
49	`validateFile(file)`	Step 1: validate file.
51	`resolveFileType(...)`	Step 2: determine normal vs correction.
53-58	Schema validation block	If enabled and normal: validate schema; throw if invalid.
60	`UUID.randomUUID().toString()`	Generate unique jobId.
61	`getFileExtension(...)`	Get .csv, .xls, or .xlsx.

62-65	Create storage dir	Paths.get, Files.exists, Files.createDirectories.
67	`storagePath.resolve(jobId + extension)`	Full path for saved file.
68-70	try-with-resources	Copy file input stream to disk; auto-close stream.
76-78	`new IngestionJob(...)`	Create job entity. Note: IngestionJob constructor may need fileType parameter.
80	`jobRepository.save(job)`	Persist to MongoDB.
84	`return new UploadResponse(jobId, IngestionStatus.UPLOADED)`	Return response.
90-91	`findById(jobId).orElseThrow(...)`	Load job or throw.
93-95	`new JobStatusResponse(...)`	Build response from job fields. job.getFileType() requires fileType on IngestionJob.
98-101	`getJob`	Returns raw IngestionJob for Processing.
105-106	Delta validation	processedRecordsDelta must be > 0.

109-110	Load job	findById or throw.
112-115	Status check	Must be PROCESSING to update progress.
117-119	Update and save	Add delta to processedRecords, set updatedAt, save.
125-141	updateStatus	Load job, validate transition, set status and failureReason, save.
143-156	setTotalRecords	Only when UPLOADED or PROCESSING.
159-161	validateFile: null/empty	Reject if file missing or empty.
163-166	validateFile: size	Reject if > 50MB.
168-171	validateFile: filename	Reject if no filename.
173-177	validateFile: extension	Reject if extension not in allowed list.
181-184	getFileExtension	lastIndexOf('.'); return substring or null.

186-192	resolveFileType	Param "correction" or filename "_correction" → correction; else normal.
194-212	validateStateTransition	UPLOADED→PROCESSING only; PROCESSING→COMPLETED/FAILED only; COMPLETED/FAILED→none.

End of Document