Below is a comprehensive, categorized list of the most common generic errors and mistakesencountered during Java REST API and Microservices development. This content can be used to create a PDF for internal documentation or onboarding.

Generic Errors in Java REST APIs

Common HTTP Status Code Errors

- 400 Bad Request: The request cannot be fulfilled due to bad syntax or invalid parameters.
- 401 Unauthorized: Authentication is required and has failed or has not been provided.
- 403 Forbidden: The user does not have the necessary permissions for the resource.
- 404 Not Found: The requested resource could not be found on the server.
- 405 Method Not Allowed: The HTTP method used is not supported for this resource.
- 409 Conflict: The request could not be processed because of a conflict in the current state of the resource.
- 415 Unsupported Media Type: The server does not support the media type of the request.
- 422 Unprocessable Entity: The request was well-formed but was unable to be followed due to semantic errors.
- 500 Internal Server Error: A generic server error; something has gone wrong on the server.
- 503 Service Unavailable: The server is not ready to handle the request, often due to overload or maintenance.

Spring Boot/Java-Specific Exception Examples

- HttpMessageNotReadableException: Input JSON cannot be parsed; data sent is malformed.
- MissingServletRequestParameterException: Required request parameter is missing.
- ConstraintViolationException: Input validation error—for example, string length or required field violation.
- TypeMismatchException: The type of a request parameter does not match the expected type.
- ResourceNotFoundException: Entity/resource not found in the database.

- MethodArgumentNotValidException: Input validation errors in request body.
- DataIntegrityViolationException: Violation of database integrity, such as duplicate values.

Generic Errors and Pitfalls in Microservices Development

Architecture & Communication

- Improper Service Boundaries: Monolithic “microservices”; overlapping or unclear responsibilities.
- Service Communication Failures: Network errors, timeouts, and serialization/deserialization issues.
- API Versioning Errors: Failing to maintain backward compatibility; breaking clients when updating service contracts.
- Dependency Conflicts: Version mismatches between services cause runtime errors and integration failures.

Operational & Performance

- Improper Configuration Management: Misconfigured services or environment variables lead to failure to start or connect.
- Insufficient Observability: Lack of centralized logging and tracing hampers debugging and root cause analysis.
- Resource Bottlenecks: Inefficient scaling, lack of proper resource allocation, or server overloads cause latency/availability issues.
- Deployment Mistakes: Manual deployments without CI/CD increase risk of human error and inconsistencies.
- Overlooking Security: Insecure defaults, improper authentication/authorization, or sensitive data leaks.

Example Java Error Response (Spring Boot)

```
json
{
 "timestamp": "2025-08-03T12:30:45.123+0000",
 "status": 400,
 "error": "Bad Request",
 "message": "JSON parse error: Unexpected character ('a' (code 97)): was expecting double-quote to start field name",
```

```
  "path": "/api/resource"
}
```

How to Mitigate Generic Errors

- Implement centralized exception handling and use meaningful, user-friendly error messages.
- Return consistent, well-structured error payloads for all failures.
- Apply validation at each layer (request, business logic, persistence).
- Use circuit breakers, retries, and timeouts for communication between microservices.
- Introduce robust configuration management and environment segregation.
- Automate testing and deployments to reduce manual errors.
- Plan for observability, monitoring, and logging from the start.