# **Design Document for Power BI Metadata REST API**

## **Table of Contents**

1. Introduction
2. Requirements
3. Architecture
4. API Endpoints
5. Data Model
6. Technology Stack
7. Implementation Details
8. Security
9. Documentation
10. Testing
11. Deployment

## **Introduction**

The Power BI Metadata REST API provides information about the objects that form a Power BI model, including tables, columns, measures, and hierarchies. This document outlines the design and implementation details of the API.

## **Requirements**

### **Functional Requirements**

* List all tables in the Power BI model.
* Retrieve details of a specific table.
* List all columns in a specific table.
* Retrieve details of a specific column.
* List all measures in the Power BI model.
* Retrieve details of a specific measure.
* List all hierarchies in the Power BI model.
* Retrieve details of a specific hierarchy.

### **Non-Functional Requirements**

* The API should be secure, implementing authentication and authorization.
* The API should be scalable and performant.
* The API should be documented for ease of use.
* The API should be testable, with unit and integration tests.

## **Architecture**

The architecture of the API will follow a standard RESTful design. It will consist of the following components:

* **Controller Layer**: Handles HTTP requests and responses.
* **Service Layer**: Contains business logic.
* **Repository Layer**: Interacts with the database.
* **Model Layer**: Represents the data structures.

The API will be implemented using a microservices architecture, deployed on a cloud platform (e.g., Azure, AWS, or GCP) to ensure scalability and high availability.

## **API Endpoints**

### **Tables**

* GET /api/tables: List all tables in the Power BI model.
* GET /api/tables/{tableId}: Get details of a specific table.

### **Columns**

* GET /api/tables/{tableId}/columns: List all columns in a specific table.
* GET /api/columns/{columnId}: Get details of a specific column.

### **Measures**

* GET /api/measures: List all measures in the Power BI model.
* GET /api/measures/{measureId}: Get details of a specific measure.

### **Hierarchies**

* GET /api/hierarchies: List all hierarchies in the Power BI model.
* GET /api/hierarchies/{hierarchyId}: Get details of a specific hierarchy.

## **Data Model**

### **Table**

json

{  
 "id": "string",  
 "name": "string",  
 "columns": [  
 {  
 "id": "string",  
 "name": "string",  
 "dataType": "string",  
 "displayFolder": "string"  
 }  
 ],  
 "measures": [  
 {  
 "id": "string",  
 "name": "string",  
 "expression": "string",  
 "displayFolder": "string"  
 }  
 ],  
 "hierarchies": [  
 {  
 "id": "string",  
 "name": "string",  
 "displayFolder": "string",  
 "levels": [  
 {  
 "id": "string",  
 "name": "string"  
 }  
 ]  
 }  
 ]  
}

### **Column**

json

{  
 "id": "string",  
 "tableId": "string",  
 "name": "string",  
 "dataType": "string",  
 "displayFolder": "string"  
}

### **Measure**

json

{  
 "id": "string",  
 "name": "string",  
 "expression": "string",  
 "displayFolder": "string"  
}

### **Hierarchy**

json

{  
 "id": "string",  
 "name": "string",  
 "displayFolder": "string",  
 "levels": [  
 {  
 "id": "string",  
 "name": "string"  
 }  
 ]  
}

## **Technology Stack**

* **Backend**: Spring Boot (Java)
* **Database**: PostgreSQL
* **Security**: Spring Security with JWT
* **Documentation**: Swagger/OpenAPI
* **Testing**: JUnit, Mockito
* **Deployment**: Docker, Kubernetes, Azure/AWS/GCP

## **Implementation Details**

### **Controller Layer**

The controllers will handle incoming HTTP requests and map them to service methods.

java

@RestController  
@RequestMapping("/api")  
public class MetadataController {  
  
 @Autowired  
 private TableService tableService;  
  
 @GetMapping("/tables")  
 public List<Table> getAllTables() {  
 return tableService.getAllTables();  
 }  
  
 @GetMapping("/tables/{tableId}")  
 public Table getTable(@PathVariable String tableId) {  
 return tableService.getTable(tableId);  
 }  
  
 @GetMapping("/tables/{tableId}/columns")  
 public List<Column> getColumnsByTable(@PathVariable String tableId) {  
 return tableService.getColumnsByTable(tableId);  
 }  
  
 @GetMapping("/measures")  
 public List<Measure> getAllMeasures() {  
 return tableService.getAllMeasures();  
 }  
  
 @GetMapping("/hierarchies")  
 public List<Hierarchy> getAllHierarchies() {  
 return tableService.getAllHierarchies();  
 }  
}

### **Service Layer**

The services will contain the business logic and interact with the repositories.

java

@Service  
public class TableService {  
  
 @Autowired  
 private TableRepository tableRepository;  
  
 public List<Table> getAllTables() {  
 return tableRepository.findAll();  
 }  
  
 public Table getTable(String tableId) {  
 return tableRepository.findById(tableId).orElse(null);  
 }  
  
 public List<Column> getColumnsByTable(String tableId) {  
 return columnRepository.findByTableId(tableId);  
 }  
  
 public List<Measure> getAllMeasures() {  
 return measureRepository.findAll();  
 }  
  
 public List<Hierarchy> getAllHierarchies() {  
 return hierarchyRepository.findAll();  
 }  
}

### **Repository Layer**

The repositories will handle database interactions.

java

@Repository  
public interface TableRepository extends JpaRepository<Table, String> {}  
  
@Repository  
public interface ColumnRepository extends JpaRepository<Column, String> {  
 List<Column> findByTableId(String tableId);  
}  
  
@Repository  
public interface MeasureRepository extends JpaRepository<Measure, String> {}  
  
@Repository  
public interface HierarchyRepository extends JpaRepository<Hierarchy, String> {}

## **Security**

* Implement authentication using JWT.
* Secure endpoints with role-based access control.
* Use Spring Security to handle authentication and authorization.

## **Documentation**

* Use Swagger/OpenAPI to document the API endpoints.
* Generate interactive API documentation.

## **Testing**

* Write unit tests for service and repository layers using JUnit and Mockito.
* Write integration tests to ensure end-to-end functionality.
* Use Postman for manual API testing.

## **Deployment**

* Containerize the application using Docker.
* Use Kubernetes for orchestration.
* Deploy to a cloud platform (Azure/AWS/GCP) for scalability and high availability.