**DESIGN DOCUMENT FOR COSMOS DB**

**TOOLING CHANGES**

1. **In the technlogy selection page Cosmos DB option should be added.**
2. **Required imports should be added in the repository.**
3. **Application.yaml changes should be in the resources folder.**
4. **POM changes need to be done in SetupDevelopmentEnvironment.**
5. **Changes need to be done in CreateSpringBootComponent page.**
6. **Metamodel changes need to be done for Cosmos DB.**
7. **Modelling persistent classes of Cosmos DB will be similar to the Mongo DB modelling.( Refer to the persistant classes examples below attached).**

**CODE GENERATION CHANGES**

**APPLICATION.yml changes:**

azure.cosmosdb.uri=${COSMOSDB\_URI}

azure.cosmosdb.key=${COSMOSDB\_KEY}

azure.cosmosdb.database=${COSMOSDB\_DBNAME}

azure.cosmosdb.populateQueryMetrics=true (Optional, default it will be set to false)

**In the repository folder, changes are:**

1. It should be extended as CosmosRepository<Mainclassname, String>
2. Required Imports are:

import com.microsoft.azure.spring.data.cosmosdb.repository.CosmosRepository;

import com.microsoft.azure.spring.data.cosmosdb.repository.ReactiveCosmosRepository;

import org.springframework.stereotype.Repository;

import org.springframework.data.rest.core.annotation.RepositoryRestResource;

1. @RepositoryRestResource(collectionResourceRel = "user", path = "user") This annotation can be used to customize the REST Endpoint.
2. custom queries can be written as:

Eg., List<User> findByEmailAndAddress(String email, Address address);

**In the persistant class changes are:**

**Required imports**:

import org.springframework.data.annotation.Id;

import com.microsoft.azure.spring.data.cosmosdb.core.mapping.Document;

import com.microsoft.azure.spring.data.cosmosdb.core.mapping.PartitionKey;

**Required Annotations:**

* Spring Data @**Id** annotation. There're 2 ways to map a field in domain class to id field of Azure Cosmos DB document.
  + annotate a field in domain class with @**Id**, this field will be mapped to document id in Cosmos DB.
  + set name of this field to id, this field will be mapped to document id in Azure Cosmos DB.
* **Custom collection Name**. By default, collection name will be class name of user domain class. To customize it, add the @**Document**(collection="myCustomCollectionName") annotation to the domain class. The collection field also supports SpEL expressions (eg. collection = "${dynamic.collection.name}" or collection = "#{@someBean.getContainerName()}") in order to provide collection names programmatically/via configuration properties.
* **Supports Optimistic Locking** for specific collections, which means upserts/deletes by document will fail with an exception in case the document was modified by another process in the meanwhile. To enable Optimistic Locking for a collection, just create a string \_etag field and mark it with the @**Version** annotation.

@Document(collection = "myCollection")

class MyDocument {

String id;

String data;

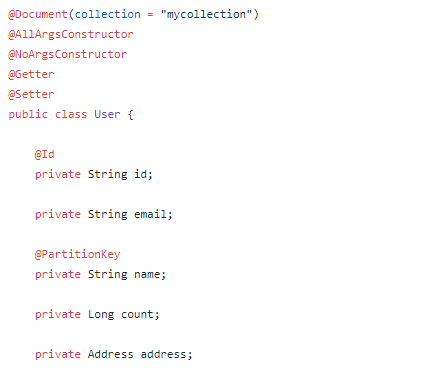
@Version

String \_etag;

}

* Supports List and nested type in domain class.
* Supports **Azure cosmos DB partition** To specify a field of domain class to be partition key field, just annotate it with **@PartitionKey**.

**Refer to the below Eg.,**



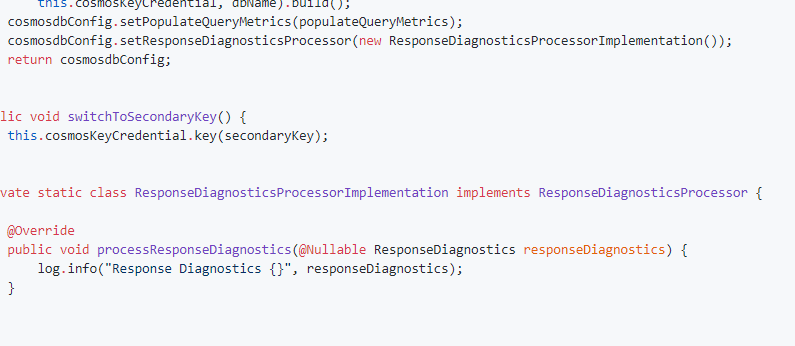
**In the config folder an extra file called RepositoryConfig.java should be added**

Required Imports:

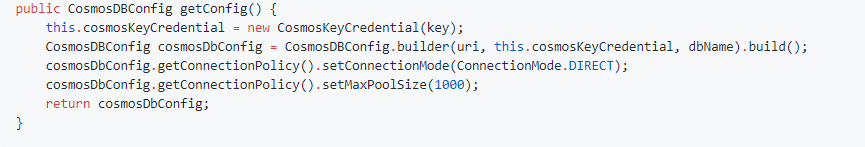
* import com.azure.data.cosmos.CosmosKeyCredential;
* import com.microsoft.azure.spring.data.cosmosdb.config.AbstractCosmosConfiguration;
* import com.microsoft.azure.spring.data.cosmosdb.config.CosmosDBConfig;
* import com.microsoft.azure.spring.data.cosmosdb.core.ResponseDiagnostics;
* import com.microsoft.azure.spring.data.cosmosdb.core.ResponseDiagnosticsProcessor
* import com.microsoft.azure.spring.data.cosmosdb.repository.config.EnableReactiveCosmosRepositories;
* **CosmosKeyCredential** feature provides capability to rotate keys on the fly. You can switch keys using **switchToSecondaryKey().**
* Use **@EnableCosmosRepositories** to enable sync repository support.
* For reactive repository support, use @EnableReactiveCosmosRepositories. These are applied above 2.2.x version.
* 2.2.x supports **Response Diagnostics** String and Query Metrics. Set **populateQueryMetrics**flag to true in application.properties to enable query metrics.
* Implement ResponseDiagnosticsProcessor to log diagnostics information.

**File is:**



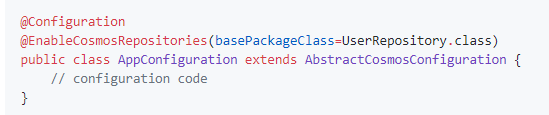


If you want to customize config,



The file is attached

By default, **@EnableCosmosRepositories** will scan the current package for any interfaces that extend one of Spring Data's repository interfaces. We can use it to annotate your Configuration class to scan a different root package by type if your project layout has multiple projects and it's not finding your repositories.



**Changes in the pom.xml are**:

Following dependency need to be added:

<dependency>

<groupId>com.microsoft.azure</groupId>

<artifactId>spring-data-cosmosdb</artifactId>

<version>2.2.4</version>

</dependency>

**NOTE**: Running Custom queries and its related changes in the repository need to be identified.