Message Service Architecture and Design

# **Architecture**

The message service is created based on n-tier client-server architecture using REST web service since it is lightweight web service and is more flexible compared to SOAP and it has the following advantages:

* No expensive tools require to interact with the Web service
* Efficient (SOAP uses XML for all messages, REST can use smaller message formats)
* Fast (no extensive processing required)
* Closer to other Web technologies in design philosophy

# **Service design**

### Design Decisions

The following table represents the design decisions of message service.

|  |  |  |
| --- | --- | --- |
| **Problem** | **Solution** | **Reason** |
| Architecture | REST | lightweight web service and is more flexible compared to SOAP. Please refer to architecture section. |
| Multi-Layer or 2 layer application | Multi-layer application | Split the application into multi-layers to let each layer deal with its concern for example controller layer is responsible to accept request from user and send response and service layer responsible to prepare data to be sent to the repository layer which responsible to communicate with data base. This design allow us to maintain and add future updates to the system in easy way and its increase the security of the application. |
| program to interface or concrete implementation | program to interface | programming to interface rather than concrete implementation confirm the concept of hiding implementation from others and in case we want to change the implementation we just update the concrete class without altering the calling classes. |

The message service has the following layers.

1. Controller layer: a REST web service that represents the access layer for the service.
2. Service layer: a middle layer between the access and repository layers it prevents the access layer from communicating direct with the repository and it prepares the domain entity to be sent to repository layer and eventually stored in the database.
3. Domain Layer: contains entities to be used by repository layer and stored into the database.
4. Repository layer: it is responsible for performing Create, Update, Delete, and Read (CRUD) operations on entities in the domain layer and communicate with the database.

### Controller layer

Controller layer has the following classes:

* MessageController: called by MessageServiceApplication class and has the following operations:
  1. getMessageIdStatus
  2. sendMessage
  3. sendBulkMessage

The MessageController interact with service layer using service layer interface as an instance variable.

### Service Layer

Service layer has the following classes and interfaces:

* MessageServiceInterface: it has the following services:
  1. getMsgIdStatus
  2. sendMsg
  3. sendBulkMsgs
* MessageServiceImpl: represents a concrete implementation of MessageServiceInterface
* MessageStatus: it is a value object that filled by service layer returned to the controller layer when getMsgIdStatus service is called.

### Model Layer

Model layer has the following classes and interfaces:

* Message: it is an entity object that contains all message information.
* ModelFactoryInterface: a Factory interface to create model objects.
* ModelFactoryImple: a concrete implementation of ModelFactoryInterface.
* Status: An Enum class contains message status values.
* Utilities: contains utilities operations used by domain classes.

### Repository Layer

Repository layer has the following interfaces:

* MessageRepositoryInterface: this interface extends its operations from JpaRepository interface which has all necessary CRUD operations.

# **Technology Details**

The following table shows the technologies that has been used to implement message service.

|  |  |  |
| --- | --- | --- |
| **Technology** | **Version** | **Notes** |
| Tomcat Server | 8.5.31 |  |
| Java | 1.8.0\_171 |  |
| Spring boot | 2.0.3 | hibernate for DAO and hazelcast for caching |
| MySQL Database | 8.0 |  |