# DESIGN

The BridgeApp design consists of 3 main packages:

1. **Controllers package –** holds controllers responsible for handling HTTP requests from WebService and calling appropriate methods on LoraService.
2. **Application package –** presents two main functionalities of BridgeApp, such as:
   1. **Handling data from LoraWAN –** receiving data from LoraWAN, translating received data and sending translated data to the corresponding MongoDB collection.
   2. **Handling requests from WebService –** reacting properly to the HTTP requests, by sending appropriate data to specific device through WebSockets.

The translation of data from bytes to objects and vice versa is handled strictly by LoraTranslator static class.

The communication with Lora is accomplished by WebSockets. LoraClient class is responsible for opening the WebSocket connection, listening to uplink messages from the device and sending downlink messages to the device.

In order to simplify choosing proper Repository interface and make it easier to extend if needed, the visitor pattern is used in the following way:

1. Each of the objects that represents the data from sensor (and extends Data class) is implementing Element interface and implement its own version of acceptVisitor method.
2. When in MongoRepository, a save method is called with a List of Elements as argument, a method acceptVisitor will be called on each Element object.
3. Depending on Elements’ type, it is calling a method from Visitor interface (that MongoRepository is implementing) which corresponds to the type of the object data.
4. Methods implemented in MongoRepository from Visitor are calling correct Repositories Interfaces which are inserting data into MongoDB

Thanks to this approach, we make use of polymorphism to control flow of the program and the code is simpler. It also allows adding new data type easily.

1. **Persistence package –** holds specific interfaces for all necessary collections in MongoDB. Thanks to using Spring Data Framework, all operations on database are abstracted away from us. All that needs to be done in order to save object to database is to create an Interface that extends a *MongoRepository* interface from Spring framework which is writing objects to specific collection in database and provides methods that perform CRUD operations. Later, to get access to repository object, dependency injection is used.

**Communication**

The communication between device and BridgeApp is accomplished by LoraWAN IoT communication. The device is sending uplink messages and receiving downlink messages. The structure of message is a JSON formatted string. That format is required by LoraWAN and contains information like timestamp, port number or message command. However the important part is data. This part of message is specified by us and it contains compressed data from sensors. The format of data is a hexadecimal string, where each byte represents different part of data. The data has 6 bytes length and the meaning of each byte is following:

|  |  |
| --- | --- |
| Byte no. | Meaning |
| 1 | CO2 low byte |
| 2 | CO2 high byte |
| 3 | Temperature |
| 4 | Humidity |
| 5 | Light |
| 6 | Movement |

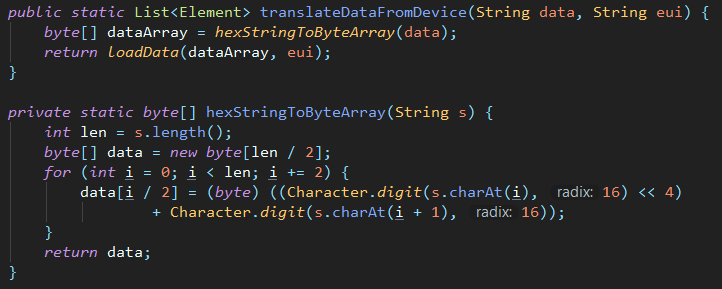
The CO2 sensor data requires two bytes of space to store as the reading from sensor can be more than 255.

# IMPLEMENTATION

LoraTranslator is a static class responsible for conversion of:

1. Received data from device uplink message in form of hexadecimal string into corresponding object
2. Requested operation code into a LoraDownlinkMessage object, which is sent to the specific device

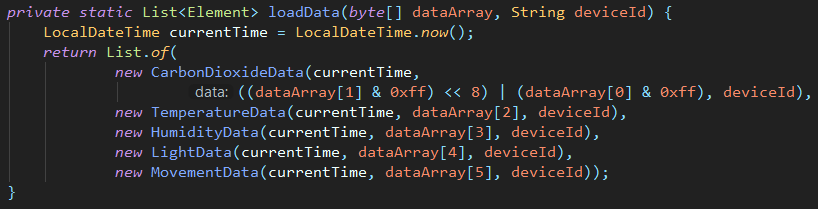
The hexadecimal string passed as parameter in translateDataFromDevice static method is firstly converted into array of bytes in the hexStringToByteArray static method, where the array size is half of the length of the string passed as parameter. Then each two characters from string are converted into byte value and the array of bytes is returned. (Figure 1)



Figure

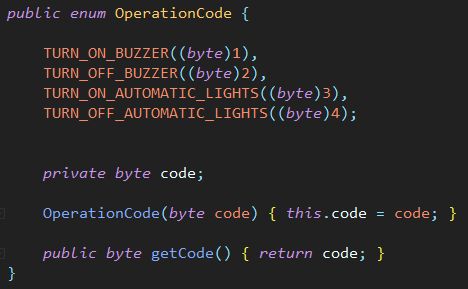
Once the data is translated into array of bytes, the loadData static method is used to create corresponding data objects which are later sent to the appropriate repository. (Figure 2)

The description of the structure of the data received from the device is discussed in the Communication section.

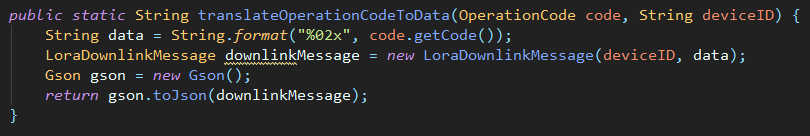


Figure

In the case of sending messages to the device, each of the operationCode enum constants is represented by a byte value which is converted into hexadecimal string and later is used with deviceID to create LoraDownlinkMessage object. This object is finally returned in JSON format and sent to the device. For the JSON conversion the Gson library is used. (Figures 3 and 4)



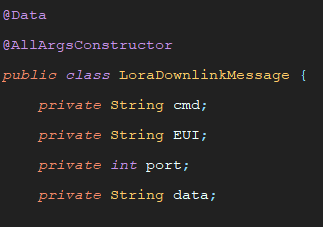
Figure



Figure

## LOMBOK

Lombok library has been used to generate methods like constructor, getters, setters, toString() and equals() for data classes. As those classes purpose is only to represent the data and they don’t have any business logic, than solution has been chosen to maintain those methods as the classes were changing during implementation process. On the illustioantion below you can see a usage of @Data annotation (which generates getters, setters, toString() and equals()) and @AllArgsConstructor (which provides constructor that takes all fields as parameters).



# TEST CASES

|  |  |  |  |
| --- | --- | --- | --- |
| Test no. | Test description | Expected outcome | Does actual outcome match expected |
| 1. | A message is sent from a device and is received from LoraWAN websocket by BridgeApp | The message is received and correctly translated from hexa string to objects. | YES |
| 2. | Received by BridgeApp data object are stored in the MongoDB | Object are translated and stored into MongoDB to corresponding collections. | YES |
| 3. | A web request is made to the BridgeApp and a message is send to LoraWAN to specific device with appropriate command code | Depending on request type, a correct message object is created with corresponding command code. The message is translated to JSON and send through web socket. | YES |