# Template code review

## Overview

This code is main class for Library bot system. It contains core functions for interactions with Telegram API and database controller.

## Goals

Implementation all interactions with Telegram API

## Usage Scenarios

Object of this class is used by Controller for connection with database and website API.

## Class Diagrams

## Code

## Categories for review

🟥 - Error 🟧 - Warning 🟩 - Well done!

## 1. Design decisions

1. Multiple import may be replaced by single one, e.g. from telegram import (**Reviewer1**)
   1.1. Ok, I will change design [**TO_BE_FIXED**]

## 2. API design

1. API is provided by library (relying on trusted code)
(Reviewer4)
2. Libraries imported separately, no odd “all in” imports. (**Reviewer4**)

## 3. Architecture (including inheritance hierarchy, if any)

1. Class is containing too much of business logic => should be divided by smaller modules (**Reviewer1**)
   1.1. (**Owner**) This code is already a separate module. Specify what part of code should be divided?
   1.2. (**Reviewer**) Confusing methods which consist a lot of if/else statements. Would be better if you would separate these statements into another method which makes all these checks. (lines 127-204 && lines 206-248)
   1.3. (**Owner**) These four methods (**get_data, get_message, online_button_checker, online_init**) are used for unification of online keyboard handling. I had to create these methods because initially information from online keyboards are catching by only one function (**online_button_checker**). After that this information in pipeline is directed to the appropriate method defined in other modules. Separation of these methods will make code more complicated.
   1.4. (**Reviewer**) In this case I understand the way Owner designed methods. [**CLOSED**]

## 4. Implementation techniques, in particular choice of data structures and algorithms

1. Good usage of built in data structures. (**Reviewer1**)

## 5. Exceptions handling - Contracts

1. Error handling included. (253 - 255, **Reviewer1**)

## 6. Programming style, names

1. Code blocks should be divided by blank lines to increase readability (everywhere, **Reviewer2**)
   1.1. (**Owner**) Ok, I will change style [**TO_BE_FIXED**]
2. It is unnecessary to use multiple lines string constants as single line. (everywhere, **Reviewer2**)
   2.1. (**Owner**) Ok, I will change the style [**TO_BE_FIXED**]
3. An expression for splitting a list into chunks is too complex. It can be replaced with [data_list[i: i + n] for i in range(0, len(data_list), n)] (line 123, **Reviewer2**)
   3.1. (**Owner**) I agree with your suggestion. It will be changed [**TO_BE_FIXED**]
4. The "evil" function eval must be removed. (line 167, **Reviewer2**)
   4.1. (**Owner**) I agree with your suggestion. It will be changed [**TO_BE_FIXED**]
5. A generator expression can be replaced with filter function. (line 179, **Reviewer2**)
   5.1. (**Owner**) I agree with your suggestion. It will be changed [**TO_BE_FIXED**]
6. In get_data method, you always check for an empty list (lines 101-121). DRY(don't repeat yourself) and just assign different text for different cases, then after nested if statement make a single check for an empty list. It will make your code 4 (or 5) lines shorter and will make it more understandable. (**Reviewer3**)
   6.1. (**Owner)** This is a good suggestion. I will fixed it. [**TO_BE_FIXED**]
7. I would suggest to rename "chat" to "chat_id" everywhere, so to everyone (especially to callers) it is clear that it is actually "chat_id", like with "message_id" or "user_id". (**Reviewer3**)
   7.1. (**Owner**) Ok, I will change the style [**TO_BE_FIXED**]

## 7. Comments and documentation

1. DocStrings should be used instead of a regular comments (**Reviewer1**)
   1.1. (**Owner**) Ok, I will add documentation accordingly Python Docstring Conventions [**TO_BE_FIXED**]