Doon Architecture

main.py

File that runs everything

```
from check_notifs.check_notifs_loop import check_notifs_loop, notify_queue

def main():

    #run the bot
    create_bot()

    #run check_notifs_loop which runs all notification queries
    #check_notifs_loop adds a notification to notify_queue if its condition has been satisfied
    query_task = asyncio.create_task(check_notifs_loop())

    #while the query loop is running
    while True:

        check the notify_queue to see if there are any notifications that need to be sent

        #then notify the user!
        await notify_user(bot, user_id, first_name, notif_name, monitored_statistic_name, threshold,
resulting_statistic)

if __name__ == '__main__':
    asyncio.run(main())
```

bot.py

Defines user commands.

```
import store_in_db.py

def create_bot():

    #initialize bot
    bot = telebot.TeleBot(TELEGRAM_API_KEY)

    # all bot message_handlers including notification creator and other commands
    @bot.message_handler
    def example_message(message):
        #when this message is sent to the bot, create a notification
        #for the user by storing it in the database!
        handle_notification_creation(user_id, message)
```

handle_notification_creation.py

central point of calling helper functions to store a user notification in the database upon creation

```
def handle_notification_creation(bot, message, query_id, condition_text)
  # Extract information from message
  notification_info, user_info= extract_all_info_from_message(message, query_id)

   #Connect to database
  cnx = connect_to_db()

   #Validate notification
  response_code, current_value = check_if_notif_is_valid(cnx, ....)

  #Define actions based on response code
  Switch (response_code)
    Case 0:
      #if the notification is valid

      #STEP 1: check if user is need/needs to be stored in the database and act accordingly
      check_if_user_and_store_if_new(cnx, user_info)

      #STEP 2: store the notification
      store_notification(cnx, notification_info)

      Reply to user with success message including current value and threshold

    Case 1 - 3:
      #Notification has some issue
      Send a issue message to the user!
  EndSwitch
```

check_notifs_loop.py

Constantly checks each notification on a periodic loop and asynchronously loads notifications that have triggered into a notification queue

```
INTERVAL = 30 seconds (for example)

#notification queue
notify_queue = asyncio.Queue()

# check_notifs_loop function for continuous notification checking
def async check_notifs_loop():

  #Infinite loop to continuously check notifications
  Loop Forever:

    #STEP 1: Retrieve all notification IDs from the database
    notifs = get_notifs(connection)

    #STEP 2: Concurrently check each notification to see if it needs to notify
    For each notif_id in notifs
      #check_notif will also add the notification to the queue if it is activated!
      Create task to run check_notif(notif_id, notify_queue)
    EndFor
```

check_notif.py

Function that checks the notification that it is fed to see if it should notify its user

```
def check_notif(pool, notif_id, notify_queue):

  # STEP 1: Execute the notification's query to retrieve the current statistic
  resulting_statistic = run the query and get the number

  # STEP 2: Determine if the notification condition is met
  #check_condition() checks if the notification's condition has been satisfied
  need_to_notify, monitored_statistic_name, comparator, threshold = check_condition()

  # STEP 3: If notification condition is met, prepare and queue the notification
  If need_to_notify:
    // Add notification details to notify_queue for processing
    await notify_queue.put(notification info)
  Else:
     do nothing
```