Notification system

# OVERVIEW

Our notification system only have one generic endpoint. It can notification to different types of services (email - sms - push notification) and to single or many people.

# Endpoint

<http://localhost:3000/api/v1/notification>

# Method

POST

# Body

## type

This field accepts string, and it should be one of these types (sms - email - push notification)

## users

This field accepts an array of users IDs

## message

This field accepts string, and it will contain the message that will be sent to user

# Request example

|  |
| --- |
| {  "type": "sms",  "message": "This is your first notification",  "users": ["606377b590c41234e6e4b5da", "606377db90c41234e6e4b5db", "606377db90c41234e6e4b5db", "606377db90c41234e6e4b5db"] } |

# Response

## Success response

|  |
| --- |
| {  "message": "Your notifcations enqueued successfully" } |

## Fail response on missing any of request body

|  |
| --- |
| {  "message": "One or more required fields not provided" } |

## Error response

|  |
| --- |
| {  "message": "Something went wrong" } |

After request submitted successfully the message sent to it service queue for example if the message type was SMS then it sent to “sms-queue” and wait for it’s turn to be executed.

# Workers

For executing the queues we have worker that run on separated process to be triggered to work it should be run through this command:

|  |
| --- |
| " node workers/notificationWorker " |

Worker start with making invoking the connect method with rabbit MQ channel to consume data then open 3 consumers. Each consumer work with service from our notification services (SMS - Email - Push Notification).

Consumer will the set the message number in Redis to save it and check if it hit the maximum limit.

When one of the consumer hit the max limit for requests per minute the consumer setTimeout for the acknowledgment for the last executed message till the end of this minute and once it sends the acknowledgment it will consume the next message for this service

|  |
| --- |
| const runConsumer = (channel, queue) => {  return channel.consume(  queue.key,  async function (msg) {  try {  let time = await utils.getTime(queue.type);  await redis.setAsync(  `${queue.type}-currentPerMint`,  ++time.currentMessagesPerMint,  "EX",  utils.leftSeconds()  );  setTimeout(() => channel.ack(msg), time.value);  } catch (error) {  console.log(error);  }  },  {  noAck: false,  consumerTag: queue.key,  }  ); }; |