1
classe DeleteMessages.java

método Create response linha 87

bad smell detectado : Complex Conditional [in method createResponse]: The conditional expression input.body() == null || input.body().getStatus() == null || input.body().getStatus() != SERVER_RESPONSE_SUCCESS is complex.

antes da refatoração:

```
	protected PNDeleteMessagesResult createResponse(Response<DeleteMessagesEnvelope> input) throws PubNubException {
	if (input.body() == null || input.body().getStatus() == null || input.body().getStatus() != SERVER_RESPONSE_SUCCESS) {
	String errorMsg = null;

	if (input.body() != null && input.body().getErrorMessage() != null) {
		errorMsg = input.body().getErrorMessage();
	} else {
		errorMsg = "n/a";
	}

	throw PubNubException.builder()
		.pubnubError(PubNubErrorBuilder.PNERROBJ_PARSING_ERROR)
		.errormsg(errorMsg)
		.build();
	}

	return PNDeleteMessagesResult.builder().build();
	}
```

após refatoração:

```
	void doIt(Response<DeleteMessagesEnvelope> input) throws PubNubException{
	String errorMsg = null;

	if (input.body() != null && input.body().getErrorMessage() != null) {
	errorMsg = input.body().getErrorMessage();
	} else {
	errorMsg = "n/a";
	}

	throw PubNubException.builder()
```

```
                .pubnubError(PubNubErrorBuilder.PNERROBJ_PARSING_ERROR)
                .errormsg(errorMsg)
                .build();
        }
        @Override
        protected PNDeleteMessagesResult
createResponse(Response<DeleteMessagesEnvelope> input) throws PubNubException {

        if(input.body() == null != SERVER_RESPONSE_SUCCESS){
        doIt();
        }
        if(input.body().getStatus() == null != SERVER_RESPONSE_SUCCESS){
        doIt();
        }
        if (  input.body().getStatus() != SERVER_RESPONSE_SUCCESS) {
        doIt();
        }

        return PNDeleteMessagesResult.builder().build();
        }
```

---

2
classe Endpoint.java

método async linha 146

bad smell detectado :Complex Conditional [in method async]: The conditional expression responseBody != null && mapper.isJsonObject(responseBody) && mapper.hasField(responseBody`"payload") is complex.

antes da refatoração:

```
        @Override
        public void onResponse(Call<Input> performedCall, Response<Input> response) {
                Output callbackResponse;

                if (isError(response)) {

                String responseBodyText;
                JsonElement responseBody;
                JsonElement responseBodyPayload = null;
                ArrayList<String> affectedChannels = new ArrayList<>();
                ArrayList<String> affectedChannelGroups = new ArrayList<>();

                try {
```

```
                responseBodyText = response.errorBody().string();
                } catch (IOException e) {
                responseBodyText = "N/A";
                }

                try {
                responseBody = mapper.fromJson(responseBodyText, JsonElement.class);
                } catch (PubNubException e) {
                responseBody = null;
                }
                if(responseBody != null ){
                responseBodyPayload = mapper.getField(responseBody, "payload");
                }
                if(mapper.isJsonObject(responseBody)){
                responseBodyPayload = mapper.getField(responseBody, "payload");
                }
                if ( mapper.isJsonObject(responseBody) && mapper.hasField(responseBody,
                        "payload")) {
                responseBodyPayload = mapper.getField(responseBody, "payload");
                }

                PNStatusCategory pnStatusCategory =
PNStatusCategory.PNUnknownCategory;
                final PubNubException ex = createPubNubException(response,
responseBodyText, responseBody);

                if (response.code() == HttpURLConnection.HTTP_FORBIDDEN) {
                pnStatusCategory = PNStatusCategory.PNAccessDeniedCategory;

                if (responseBodyPayload != null && mapper.hasField(responseBodyPayload,
"channels")) {
                        Iterator<JsonElement> it =
mapper.getArrayIterator(responseBodyPayload, "channels");
                        while (it.hasNext()) {
                        JsonElement objNode = it.next();
                        affectedChannels.add(mapper.elementToString(objNode));
                        }
                }

                if (responseBodyPayload != null && mapper.hasField(responseBodyPayload,
"channel-groups")) {
                        Iterator<JsonElement> it =
mapper.getArrayIterator(responseBodyPayload, "channel-groups");
                        while (it.hasNext()) {
                        JsonElement objNode = it.next();
                        String channelGroupName =
                                mapper.elementToString(objNode).substring(0, 1).equals(":")
                                ? mapper.elementToString(objNode).substring(1)
```

```
                        : mapper.elementToString(objNode);
                affectedChannelGroups.add(channelGroupName);
                }
            }

            }

            if (response.code() == HttpURLConnection.HTTP_BAD_REQUEST) {
            pnStatusCategory = PNStatusCategory.PNBadRequestCategory;
            }

            callback.onResponse(null,
                createStatusResponse(pnStatusCategory, response, ex,
affectedChannels,
                affectedChannelGroups));
            return;
            }
            storeRequestLatency(response, getOperationType());

            try {
            callbackResponse = createResponse(response);
            } catch (PubNubException pubnubException) {
            callback.onResponse(null,

createStatusResponse(PNStatusCategory.PNMalformedResponseCategory, response,
                pubnubException, null, null));
            return;
            }

            callback.onResponse(callbackResponse,
            createStatusResponse(PNStatusCategory.PNAcknowledgmentCategory,
response,
                null, null, null));
        }


após refatoração:
        @Override
        public void onResponse(Call<Input> performedCall, Response<Input> response) {
            Output callbackResponse;

            if (isError(response)) {

            String responseBodyText;
            JsonElement responseBody;
            JsonElement responseBodyPayload = null;
            ArrayList<String> affectedChannels = new ArrayList<>();
            ArrayList<String> affectedChannelGroups = new ArrayList<>();
```

```
                try {
                responseBodyText = response.errorBody().string();
                } catch (IOException e) {
                responseBodyText = "N/A";
                }

                try {
                responseBody = mapper.fromJson(responseBodyText, JsonElement.class);
                } catch (PubNubException e) {
                responseBody = null;
                }

                if ( responseBody != null && mapper.isJsonObject(responseBody) &&
mapper.isJsonObject(responseBody) && mapper.hasField(responseBody,
                        "payload")) {
                responseBodyPayload = mapper.getField(responseBody, "payload");
                }

                PNStatusCategory pnStatusCategory =
PNStatusCategory.PNUnknownCategory;
                final PubNubException ex = createPubNubException(response,
responseBodyText, responseBody);

                if (response.code() == HttpURLConnection.HTTP_FORBIDDEN) {
                pnStatusCategory = PNStatusCategory.PNAccessDeniedCategory;

                if (responseBodyPayload != null && mapper.hasField(responseBodyPayload,
"channels")) {
                        Iterator<JsonElement> it =
mapper.getArrayIterator(responseBodyPayload, "channels");
                        while (it.hasNext()) {
                        JsonElement objNode = it.next();
                        affectedChannels.add(mapper.elementToString(objNode));
                        }
                }

                if (responseBodyPayload != null && mapper.hasField(responseBodyPayload,
"channel-groups")) {
                        Iterator<JsonElement> it =
mapper.getArrayIterator(responseBodyPayload, "channel-groups");
                        while (it.hasNext()) {
                        JsonElement objNode = it.next();
                        String channelGroupName =
                                mapper.elementToString(objNode).substring(0, 1).equals(":")
                                ? mapper.elementToString(objNode).substring(1)
                                : mapper.elementToString(objNode);
                        affectedChannelGroups.add(channelGroupName);
```

```
                    }
                }

                }

                if (response.code() == HttpURLConnection.HTTP_BAD_REQUEST) {
                pnStatusCategory = PNStatusCategory.PNBadRequestCategory;
                }

                callback.onResponse(null,
                        createStatusResponse(pnStatusCategory, response, ex,
affectedChannels,
                        affectedChannelGroups));
                return;
                }
                storeRequestLatency(response, getOperationType());

                try {
                callbackResponse = createResponse(response);
                } catch (PubNubException pubnubException) {
                callback.onResponse(null,

createStatusResponse(PNStatusCategory.PNMalformedResponseCategory, response,
                        pubnubException, null, null));
                return;
                }

                callback.onResponse(callbackResponse,
                createStatusResponse(PNStatusCategory.PNAcknowledgmentCategory,
response,
                        null, null, null));
        }
```

---

3
classe Endpoint.java

método onResponse linhja 147

bad smell detectado : Long Statement [in method onResponse]: The length of the statement "String channelGroupName=mapper.elementToString(objNode).substring(0`1).equals(":") ? mapper.elementToString(objNode).substring(1) : mapper.elementToString(objNode);" is 164.

antes da refatoração:

```
        @Override
        public void onResponse(Call<Input> performedCall, Response<Input> response) {
                Output callbackResponse;
```

```
                if (isError(response)) {

                String responseBodyText;
                JsonElement responseBody;
                JsonElement responseBodyPayload = null;
                ArrayList<String> affectedChannels = new ArrayList<>();
                ArrayList<String> affectedChannelGroups = new ArrayList<>();

                try {
                responseBodyText = response.errorBody().string();
                } catch (IOException e) {
                responseBodyText = "N/A";
                }

                try {
                responseBody = mapper.fromJson(responseBodyText, JsonElement.class);
                } catch (PubNubException e) {
                responseBody = null;
                }
                if(responseBody != null ){
                responseBodyPayload = mapper.getField(responseBody, "payload");
                }
                if(mapper.isJsonObject(responseBody)){
                responseBodyPayload = mapper.getField(responseBody, "payload");
                }
                if ( mapper.isJsonObject(responseBody) && mapper.hasField(responseBody,
                        "payload")) {
                responseBodyPayload = mapper.getField(responseBody, "payload");
                }

                PNStatusCategory pnStatusCategory =
PNStatusCategory.PNUnknownCategory;
                final PubNubException ex = createPubNubException(response,
responseBodyText, responseBody);

                if (response.code() == HttpURLConnection.HTTP_FORBIDDEN) {
                pnStatusCategory = PNStatusCategory.PNAccessDeniedCategory;

                if (responseBodyPayload != null && mapper.hasField(responseBodyPayload,
"channels")) {
                        Iterator<JsonElement> it =
mapper.getArrayIterator(responseBodyPayload, "channels");
                        while (it.hasNext()) {
                        JsonElement objNode = it.next();
                        affectedChannels.add(mapper.elementToString(objNode));
                        }
                }
```

```
                if (responseBodyPayload != null && mapper.hasField(responseBodyPayload,
"channel-groups")) {
                        Iterator<JsonElement> it =
mapper.getArrayIterator(responseBodyPayload, "channel-groups");
                        while (it.hasNext()) {
                        JsonElement objNode = it.next();
                        String channelGroupName =
                                mapper.elementToString(objNode).substring(0, 1).equals(":")
                                ? mapper.elementToString(objNode).substring(1)
                                : mapper.elementToString(objNode);
                        affectedChannelGroups.add(channelGroupName);
                        }
                }

                }

                if (response.code() == HttpURLConnection.HTTP_BAD_REQUEST) {
                pnStatusCategory = PNStatusCategory.PNBadRequestCategory;
                }

                callback.onResponse(null,
                        createStatusResponse(pnStatusCategory, response, ex,
affectedChannels,
                        affectedChannelGroups));
                return;
                }
                storeRequestLatency(response, getOperationType());

                try {
                callbackResponse = createResponse(response);
                } catch (PubNubException pubnubException) {
                callback.onResponse(null,

createStatusResponse(PNStatusCategory.PNMalformedResponseCategory, response,
                        pubnubException, null, null));
                return;
                }

                callback.onResponse(callbackResponse,
                createStatusResponse(PNStatusCategory.PNAcknowledgmentCategory,
response,
                        null, null, null));
        }
```

após refatoração:

```
        public void onResponse(Call<Input> performedCall, Response<Input> response) {
                Output callbackResponse;

                if (isError(response)) {

                String responseBodyText;
                JsonElement responseBody;
                JsonElement responseBodyPayload = null;
                ArrayList<String> affectedChannels = new ArrayList<>();
                ArrayList<String> affectedChannelGroups = new ArrayList<>();

                try {
                responseBodyText = response.errorBody().string();
                } catch (IOException e) {
                responseBodyText = "N/A";
                }

                try {
                responseBody = mapper.fromJson(responseBodyText, JsonElement.class);
                } catch (PubNubException e) {
                responseBody = null;
                }
                if(responseBody != null ){
                responseBodyPayload = mapper.getField(responseBody, "payload");
                }
                if(mapper.isJsonObject(responseBody)){
                responseBodyPayload = mapper.getField(responseBody, "payload");
                }
                if ( mapper.isJsonObject(responseBody) && mapper.hasField(responseBody,
                        "payload")) {
                responseBodyPayload = mapper.getField(responseBody, "payload");
                }

                PNStatusCategory pnStatusCategory =
PNStatusCategory.PNUnknownCategory;
                final PubNubException ex = createPubNubException(response,
responseBodyText, responseBody);

                if (response.code() == HttpURLConnection.HTTP_FORBIDDEN) {
                pnStatusCategory = PNStatusCategory.PNAccessDeniedCategory;

                if (responseBodyPayload != null && mapper.hasField(responseBodyPayload,
"channels")) {
                        Iterator<JsonElement> it =
mapper.getArrayIterator(responseBodyPayload, "channels");
                        while (it.hasNext()) {
                        JsonElement objNode = it.next();
                        affectedChannels.add(mapper.elementToString(objNode));
```

```
                    }
                }

                if (responseBodyPayload != null && mapper.hasField(responseBodyPayload,
"channel-groups")) {
                    Iterator<JsonElement> it =
mapper.getArrayIterator(responseBodyPayload, "channel-groups");
                    while (it.hasNext()) {
                    JsonElement objNode = it.next();
                    String channelGroupName = null;
                    if(mapper.elementToString(objNode).substring(1)){

                    channelGroupName = mapper.elementToString(objNode).substring(0,
1).equals(":");
                    }else {
                    mapper.elementToString(objNode);
                    }


                    affectedChannelGroups.add(channelGroupName);
                    }
                }

                }

                if (response.code() == HttpURLConnection.HTTP_BAD_REQUEST) {
                pnStatusCategory = PNStatusCategory.PNBadRequestCategory;
                }

                callback.onResponse(null,
                    createStatusResponse(pnStatusCategory, response, ex,
affectedChannels,
                    affectedChannelGroups));
                return;
                }
                storeRequestLatency(response, getOperationType());

                try {
                callbackResponse = createResponse(response);
                } catch (PubNubException pubnubException) {
                callback.onResponse(null,

createStatusResponse(PNStatusCategory.PNMalformedResponseCategory, response,
                    pubnubException, null, null));
                return;
                }

                callback.onResponse(callbackResponse,
```

```
                createStatusResponse(PNStatusCategory.PNAcknowledgmentCategory,
response,
                        null, null, null));
        }
```

---

4
classe History.java

método createResponse linha 117

bad smell detectado : Long Statement [in method createResponse]: The length of the statement "throw PubNubException.builder().pubnubError(PubNubErrorBuilder.PNERROBJ_HTTP_ERROR).errormsg("History is disabled").jso(input.body()).build();" is 142.
antes da refatoração:

```
        protected PNHistoryResult createResponse(Response<JsonElement> input) throws
PubNubException {
        PNHistoryResult.PNHistoryResultBuilder historyData = PNHistoryResult.builder();
        List<PNHistoryItemResult> messages = new ArrayList<>();
        MapperManager mapper = getPubnub().getMapper();

        if (input.body() != null) {
        Long startTimeToken =
mapper.elementToLong(mapper.getArrayElement(input.body(), 1));
        Long endTimeToken =
mapper.elementToLong(mapper.getArrayElement(input.body(), 2));

        historyData.startTimetoken(startTimeToken);
        historyData.endTimetoken(endTimeToken);


        if (mapper.getArrayElement(input.body(), 0).isJsonArray()) {
                Iterator<JsonElement> it =
mapper.getArrayIterator(mapper.getArrayElement(input.body(), 0));
                while (it.hasNext()) {
                JsonElement historyEntry = it.next();
                PNHistoryItemResult.PNHistoryItemResultBuilder historyItem =
PNHistoryItemResult.builder();
                JsonElement message;

                if (includeTimetoken || includeMeta) {
                message = processMessage(mapper.getField(historyEntry, "message"));
                if (includeTimetoken) {
                        historyItem.timetoken(mapper.elementToLong(historyEntry,
"timetoken"));
                }
```

```
                if (includeMeta) {
                        historyItem.meta(mapper.getField(historyEntry, "meta"));
                }
                } else {
                message = processMessage(historyEntry);
                }

                historyItem.entry(message);
                messages.add(historyItem.build());
                }
        } else {
                throw
PubNubException.builder(PubNubErrorBuilder.PNERROBJ_HTTP_ERROR)
                .pubnubError()
                .errormsg("History is disabled")
                .jso(input.body())
                .build();
        }


        historyData.messages(messages);
        }

        return historyData.build();
        }



após refatoração:
  @Override
        protected PNHistoryResult createResponse(Response<JsonElement> input) throws
PubNubException {
        PNHistoryResult.PNHistoryResultBuilder historyData = PNHistoryResult.builder();
        List<PNHistoryItemResult> messages = new ArrayList<>();
        MapperManager mapper = getPubnub().getMapper();

        if (input.body() != null) {
        Long startTimeToken =
mapper.elementToLong(mapper.getArrayElement(input.body(), 1));
        Long endTimeToken =
mapper.elementToLong(mapper.getArrayElement(input.body(), 2));

        historyData.startTimetoken(startTimeToken);
        historyData.endTimetoken(endTimeToken);


        if (mapper.getArrayElement(input.body(), 0).isJsonArray()) {
```

```
                Iterator<JsonElement> it =
mapper.getArrayIterator(mapper.getArrayElement(input.body(), 0));
                while (it.hasNext()) {
                JsonElement historyEntry = it.next();
                PNHistoryItemResult.PNHistoryItemResultBuilder historyItem =
PNHistoryItemResult.builder();
                JsonElement message;

                if (includeTimetoken || includeMeta) {
                message = processMessage(mapper.getField(historyEntry, "message"));
                if (includeTimetoken) {
                        historyItem.timetoken(mapper.elementToLong(historyEntry,
"timetoken"));
                }
                if (includeMeta) {
                        historyItem.meta(mapper.getField(historyEntry, "meta"));
                }
                } else {
                message = processMessage(historyEntry);
                }

                historyItem.entry(message);
                messages.add(historyItem.build());
                }
        } else {
                String m = "History is disabled";
                String e = PubNubErrorBuilder.PNERROBJ_HTTP_ERROR;
                throw PubNubException.builder(e)
                .pubnubError()
                .errormsg(m)
                .jso(input.body())
                .build();
        }


        historyData.messages(messages);
        }

        return historyData.build();
        }
```

---

5

classe History.java

método processMessage linha 179

bad smell detectado : Long Statement [in method processMessage]: The length of the statement "Crypto crypto=new Crypto(this.getPubnub().getConfiguration().getCipherKey()`this.getPubnub().getConfiguration().isUseRandomInitializationVector());" is 147.

antes da refatoração:

```
	private JsonElement processMessage(JsonElement message) throws PubNubException {
	// if we do not have a crypto key, there is no way to process the node; let's return.
	if (this.getPubnub().getConfiguration().getCipherKey() == null) {
	return message;
	}

	Crypto crypto = new Crypto(this.getPubnub().getConfiguration().getCipherKey(), this.getPubnub().getConfiguration().isUseRandomInitializationVector());
	MapperManager mapper = getPubnub().getMapper();
	String inputText;
	String outputText;
	JsonElement outputObject;

	if (mapper.isJsonObject(message) && mapper.hasField(message, "pn_other")) {
	inputText = mapper.elementToString(message, "pn_other");
	} else {
	inputText = mapper.elementToString(message);
	}

	outputText = crypto.decrypt(inputText);
	outputObject = this.getPubnub().getMapper().fromJson(outputText, JsonElement.class);

	// inject the decoded response into the payload
	if (mapper.isJsonObject(message) && mapper.hasField(message, "pn_other")) {
	JsonObject objectNode = mapper.getAsObject(message);
	mapper.putOnObject(objectNode, "pn_other", outputObject);
	outputObject = objectNode;
	}

	return outputObject;
	}
```

após refatoração:

```
  private JsonElement processMessage(JsonElement message) throws PubNubException {
	// if we do not have a crypto key, there is no way to process the node; let's return.
	if (this.getPubnub().getConfiguration().getCipherKey() == null) {
```

```
        return message;
        }
        String key = this.getPubnub().getConfiguration().getCipherKey();
                String initi
=this.getPubnub().getConfiguration().isUseRandomInitializationVector();
        Crypto crypto = new Crypto(key,initi );
        MapperManager mapper = getPubnub().getMapper();
        String inputText;
        String outputText;
        JsonElement outputObject;

        if (mapper.isJsonObject(message) && mapper.hasField(message, "pn_other")) {
        inputText = mapper.elementToString(message, "pn_other");
        } else {
        inputText = mapper.elementToString(message);
        }

        outputText = crypto.decrypt(inputText);
        outputObject = this.getPubnub().getMapper().fromJson(outputText,
JsonElement.class);

        // inject the decoded response into the payload
        if (mapper.isJsonObject(message) && mapper.hasField(message, "pn_other")) {
        JsonObject objectNode = mapper.getAsObject(message);
        mapper.putOnObject(objectNode, "pn_other", outputObject);
        outputObject = objectNode;
        }

        return outputObject;
        }
```

---

6

classe Crypto.java

método initCiphers linha 52

bad smell detectado : Magic Number [in method initCiphers]: The method contains a magic number: 32

antes da refatoração:

```
 private void initCiphers() throws PubNubException {
        if (INIT && !dynamicIV)
        return;
        try {
```

```
	keyBytes = new
String(hexEncode(sha256(this.cipherKey.getBytes(ENCODING_UTF_8))),
ENCODING_UTF_8)
			.substring(0, 32)
			.toLowerCase().getBytes(ENCODING_UTF_8);
	if (dynamicIV){
			ivBytes = new byte[16];
			new Random().nextBytes(ivBytes);
	}
	else {
			ivBytes = initializationVector.getBytes(ENCODING_UTF_8);
			INIT = true;
	}
	} catch (UnsupportedEncodingException e) {
	throw PubNubException.builder().pubnubError(newCryptoError(11,
e.toString())).errormsg(e.getMessage()).cause(e).build();
	}
	}
```

após refatoração:

---

7
classe Crypto.java

método initCiphers linha 52

bad smell detectado : Magic Number [in method initCiphers]: The method contains a magic number: 32

antes da refatoração:

```
  private void initCiphers() throws PubNubException {
	if (INIT && !dynamicIV)
	return;
	try {

	keyBytes = new
String(hexEncode(sha256(this.cipherKey.getBytes(ENCODING_UTF_8))),
ENCODING_UTF_8)
			.substring(0, 32)
			.toLowerCase().getBytes(ENCODING_UTF_8);
	if (dynamicIV){
			ivBytes = new byte[16];
			new Random().nextBytes(ivBytes);
```

```
        }
        else {
                ivBytes = initializationVector.getBytes(ENCODING_UTF_8);
                INIT = true;
        }
        } catch (UnsupportedEncodingException e) {
        throw PubNubException.builder().pubnubError(newCryptoError(11,
e.toString())).errormsg(e.getMessage()).cause(e).build();
        }
        }
```

após refatoração:

```
public static final int n1 = 32;
  private void initCiphers() throws PubNubException {
        if (INIT && !dynamicIV)
        return;
        try {

        keyBytes = new
String(hexEncode(sha256(this.cipherKey.getBytes(ENCODING_UTF_8))),
ENCODING_UTF_8)
                .substring(0, n1)
                .toLowerCase().getBytes(ENCODING_UTF_8);
        if (dynamicIV){
                ivBytes = new byte[16];
                new Random().nextBytes(ivBytes);
        }
        else {
                ivBytes = initializationVector.getBytes(ENCODING_UTF_8);
                INIT = true;
        }
        } catch (UnsupportedEncodingException e) {
        throw PubNubException.builder().pubnubError(newCryptoError(11,
e.toString())).errormsg(e.getMessage()).cause(e).build();
        }
        }
```

---

8

classe Crypto.java

método initCiphers linha 52

bad smell detectado : Magic Number [in method initCiphers]: The method contains a magic number: 16

antes da refatoração:

```
 private void initCiphers() throws PubNubException {
        if (INIT && !dynamicIV)
        return;
        try {

        keyBytes = new
String(hexEncode(sha256(this.cipherKey.getBytes(ENCODING_UTF_8))),
ENCODING_UTF_8)
                .substring(0, n1)
                .toLowerCase().getBytes(ENCODING_UTF_8);
        if (dynamicIV){
                ivBytes = new byte[16];
                new Random().nextBytes(ivBytes);
        }
        else {
                ivBytes = initializationVector.getBytes(ENCODING_UTF_8);
                INIT = true;
        }
        } catch (UnsupportedEncodingException e) {
        throw PubNubException.builder().pubnubError(newCryptoError(11,
e.toString())).errormsg(e.getMessage()).cause(e).build();
        }
        }
```

após refatoração:

```
  public static final int n2 = 16;

 private void initCiphers() throws PubNubException {
        if (INIT && !dynamicIV)
        return;
        try {

        keyBytes = new
String(hexEncode(sha256(this.cipherKey.getBytes(ENCODING_UTF_8))),
ENCODING_UTF_8)
                .substring(0, n1)
                .toLowerCase().getBytes(ENCODING_UTF_8);
        if (dynamicIV){
                ivBytes = new byte[n2];
```

```
                new Random().nextBytes(ivBytes);
        }
        else {
                ivBytes = initializationVector.getBytes(ENCODING_UTF_8);
                INIT = true;
        }
        } catch (UnsupportedEncodingException e) {
        throw PubNubException.builder().pubnubError(newCryptoError(11,
e.toString())).errormsg(e.getMessage()).cause(e).build();
        }
        }
```

---

9
classe Crypto.java

método initCiphers linha 52

bad smell detectado : Magic Number [in method initCiphers]: The method contains a magic number: 11

antes da refatoração:

```
  private void initCiphers() throws PubNubException {
        if (INIT && !dynamicIV)
        return;
        try {

        keyBytes = new
String(hexEncode(sha256(this.cipherKey.getBytes(ENCODING_UTF_8))),
ENCODING_UTF_8)
                .substring(0, n1)
                .toLowerCase().getBytes(ENCODING_UTF_8);
        if (dynamicIV){
                ivBytes = new byte[n2];
                new Random().nextBytes(ivBytes);
        }
        else {
                ivBytes = initializationVector.getBytes(ENCODING_UTF_8);
                INIT = true;
        }
        } catch (UnsupportedEncodingException e) {
        throw PubNubException.builder().pubnubError(newCryptoError(11,
e.toString())).errormsg(e.getMessage()).cause(e).build();
        }
        }
```

após refatoração:

```
   public static final int n3 = 11;
     private void initCiphers() throws PubNubException {
          if (INIT && !dynamicIV)
          return;
          try {

          keyBytes = new
String(hexEncode(sha256(this.cipherKey.getBytes(ENCODING_UTF_8))),
ENCODING_UTF_8)
                    .substring(0, n1)
                    .toLowerCase().getBytes(ENCODING_UTF_8);
          if (dynamicIV){
                    ivBytes = new byte[n2];
                    new Random().nextBytes(ivBytes);
          }
          else {
                    ivBytes = initializationVector.getBytes(ENCODING_UTF_8);
                    INIT = true;
          }
          } catch (UnsupportedEncodingException e) {
          throw PubNubException.builder().pubnubError(newCryptoError(n3,
e.toString())).errormsg(e.getMessage()).cause(e).build();
          }
          }
```

---

10
classe Crypto.java

método hexEncode linha 78

bad smell detectado : Magic Number [in method hexEncode]: The method contains a magic number: 16

antes da refatoração:

```
  public static byte[] hexEncode(byte[] input) throws PubNubException {
          StringBuffer result = new StringBuffer();
          for (byte byt : input)
          result.append(Integer.toString((byt & 0xff) + 0x100, 16).substring(1));
          try {
          return result.toString().getBytes(ENCODING_UTF_8);
          } catch (UnsupportedEncodingException e) {
```

```
        throw PubNubException.builder().pubnubError(newCryptoError(12,
e.toString())).errormsg(e.getMessage()).cause(e).build();
        }
        }
```

após refatoração:

```
        public static final int n4 = 16;
    public static byte[] hexEncode(byte[] input) throws PubNubException {
        StringBuffer result = new StringBuffer();
        for (byte byt : input)
        result.append(Integer.toString((byt & 0xff) + 0x100, n4).substring(1));
        try {
        return result.toString().getBytes(ENCODING_UTF_8);
        } catch (UnsupportedEncodingException e) {
        throw PubNubException.builder().pubnubError(newCryptoError(12,
e.toString())).errormsg(e.getMessage()).cause(e).build();
        }
        }
```

---

11

classe Crypto.java

método hexEncode linha 78

bad smell detectado :Magic Number [in method hexEncode]: The method contains a magic number: 12

antes da refatoração:

```
  public static byte[] hexEncode(byte[] input) throws PubNubException {
        StringBuffer result = new StringBuffer();
        for (byte byt : input)
        result.append(Integer.toString((byt & 0xff) + 0x100, 16).substring(1));
        try {
        return result.toString().getBytes(ENCODING_UTF_8);
        } catch (UnsupportedEncodingException e) {
        throw PubNubException.builder().pubnubError(newCryptoError(12,
e.toString())).errormsg(e.getMessage()).cause(e).build();
        }
        }
```

após refatoração:

```
        public static final int n5 = 12;
  public static byte[] hexEncode(byte[] input) throws PubNubException {
        StringBuffer result = new StringBuffer();
        for (byte byt : input)
        result.append(Integer.toString((byt & 0xff) + 0x100, n4).substring(1));
        try {
        return result.toString().getBytes(ENCODING_UTF_8);
        } catch (UnsupportedEncodingException e) {
        throw PubNubException.builder().pubnubError(newCryptoError(n5,
e.toString())).errormsg(e.getMessage()).cause(e).build();
        }
        }
```

---

12

classe Crypto.java

método hexEncode linha 78

bad smell detectado : Magic Number [in method hexEncode]: The method contains a magic number: 0xff

antes da refatoração:

```
   public static byte[] hexEncode(byte[] input) throws PubNubException {
        StringBuffer result = new StringBuffer();
        for (byte byt : input)
        result.append(Integer.toString((byt & 0xff) + 0x100, 16).substring(1));
        try {
        return result.toString().getBytes(ENCODING_UTF_8);
        } catch (UnsupportedEncodingException e) {
        throw PubNubException.builder().pubnubError(newCryptoError(12,
e.toString())).errormsg(e.getMessage()).cause(e).build();
        }
        }
```

após refatoração:

```
        public static final int n6 = 0xff;
  public static byte[] hexEncode(byte[] input) throws PubNubException {
        StringBuffer result = new StringBuffer();
        for (byte byt : input)
        result.append(Integer.toString((byt & n6) + 0x100, n4).substring(1));
        try {
        return result.toString().getBytes(ENCODING_UTF_8);
        } catch (UnsupportedEncodingException e) {
```

```
        throw PubNubException.builder().pubnubError(newCryptoError(n5,
e.toString())).errormsg(e.getMessage()).cause(e).build();
        }
        }
```

---

13
classe Crypto.java

método hexEncode linha 78

bad smell detectado :Magic Number [in method hexEncode]: The method contains a magic number: 0x100

antes da refatoração:

```
  public static byte[] hexEncode(byte[] input) throws PubNubException {
        StringBuffer result = new StringBuffer();
        for (byte byt : input)
        result.append(Integer.toString((byt & 0xff) + 0x100, 16).substring(1));
        try {
        return result.toString().getBytes(ENCODING_UTF_8);
        } catch (UnsupportedEncodingException e) {
        throw PubNubException.builder().pubnubError(newCryptoError(12,
e.toString())).errormsg(e.getMessage()).cause(e).build();
        }
        }
```

após refatoração:

```
        public static final int n7 = 0x100;
  public static byte[] hexEncode(byte[] input) throws PubNubException {
        StringBuffer result = new StringBuffer();
        for (byte byt : input)
        result.append(Integer.toString((byt & n6) + n7, n4).substring(1));
        try {
        return result.toString().getBytes(ENCODING_UTF_8);
        } catch (UnsupportedEncodingException e) {
        throw PubNubException.builder().pubnubError(newCryptoError(n5,
e.toString())).errormsg(e.getMessage()).cause(e).build();
        }
        }
```

---

14classe ReconnectionManager.java

método getBooleanTypeAdapter linha 182

bad smell detectado : Magic Number [in method getNextInterval]: The method contains a magic number: 2
antes da refatoração:

```
int getNextInterval() {
	int timerInterval = LINEAR_INTERVAL;
	failedCalls++;

	if (pnReconnectionPolicy == PNReconnectionPolicy.EXPONENTIAL) {
	exponentialMultiplier++;
	timerInterval = (int) (Math.pow(2, exponentialMultiplier) - 1);
	if (timerInterval > MAX_EXPONENTIAL_BACKOFF) {
		timerInterval = MIN_EXPONENTIAL_BACKOFF;
		exponentialMultiplier = 1;
		log.debug("timerInterval > MAXEXPONENTIALBACKOFF at: " +
Calendar.getInstance().getTime().toString());
	} else if (timerInterval < 1) {
		timerInterval = MIN_EXPONENTIAL_BACKOFF;
	}
	log.debug("timerInterval = " + timerInterval + " at: " +
Calendar.getInstance().getTime().toString());
	}

	if (pnReconnectionPolicy == PNReconnectionPolicy.LINEAR) {
	timerInterval = LINEAR_INTERVAL;
	}

	return timerInterval;
	}
```

após refatoração:

```
 public static final int dois = 2;
	int getNextInterval() {
	int timerInterval = LINEAR_INTERVAL;
	failedCalls++;

	if (pnReconnectionPolicy == PNReconnectionPolicy.EXPONENTIAL) {
	exponentialMultiplier++;
	timerInterval = (int) (Math.pow(dois, exponentialMultiplier) - 1);
	if (timerInterval > MAX_EXPONENTIAL_BACKOFF) {
		timerInterval = MIN_EXPONENTIAL_BACKOFF;
		exponentialMultiplier = 1;
		log.debug("timerInterval > MAXEXPONENTIALBACKOFF at: " +
Calendar.getInstance().getTime().toString());
	} else if (timerInterval < 1) {
```

```
                timerInterval = MIN_EXPONENTIAL_BACKOFF;
        }
        log.debug("timerInterval = " + timerInterval + " at: " +
Calendar.getInstance().getTime().toString());
        }

        if (pnReconnectionPolicy == PNReconnectionPolicy.LINEAR) {
        timerInterval = LINEAR_INTERVAL;
        }

        return timerInterval;
        }
```

---

15
classe StateManager.java

método changeTemporary

bad smell detectado : Long Identifier [in method changeTemporary]: The length of the field temporaryUnavailableChannelGroups is 33.

antes da refatoração:

```
 private final List<TemporaryUnavailableItem> temporaryUnavailableChannelGroups = new
ArrayList<>();
  private void changeTemporary(ChangeTemporaryUnavailableOperation operation) {
        for (String channel : operation.getUnavailableChannels()) {
        temporaryUnavailableChannels.add(new TemporaryUnavailableItem(channel, new
Date()));
        }
        for (String channelGroup : operation.getUnavailableChannelGroups()) {
        temporaryUnavailableChannelGroups.add(new
TemporaryUnavailableItem(channelGroup, new Date()));
        }

        removeTemporaryUnavailableChannels(operation.getAvailableChannels());

removeTemporaryUnavailableChannelGroups(operation.getAvailableChannelGroups());
        }
```

após refatoração:

```
 private final List<TemporaryUnavailableItem> temporaryUnChGr = new ArrayList<>();
  private void changeTemporary(ChangeTemporaryUnavailableOperation operation) {
        for (String channel : operation.getUnavailableChannels()) {
        temporaryUnavailableChannels.add(new TemporaryUnavailableItem(channel, new
Date()));
        }
```

```
        for (String channelGroup : operation.getUnavailableChannelGroups()) {
        temporaryUnChGr.add(new TemporaryUnavailableItem(channelGroup, new Date()));
        }

        removeTemporaryUnavailableChannels(operation.getAvailableChannels());

removeTemporaryUnavailableChannelGroups(operation.getAvailableChannelGroups());
        }
```

---

16
classe StateManager.java

método resetTemporaryUnavailableChannelsAndGroups linha 364

bad smell detectado :Long Identifier [in method resetTemporaryUnavailableChannelsAndGroups]: The length of the field temporaryUnavailableChannelGroups is 33.

antes da refatoração:

```
  private final List<TemporaryUnavailableItem> temporaryUnavailableChannelGroups= new
ArrayList<>();
   private void resetTemporaryUnavailableChannelsAndGroups() {
        temporaryUnavailableChannels.clear();
        temporaryUnavailableChannelGroups.clear();
        }
```

após refatoração:

```
 private final List<TemporaryUnavailableItem> temporaryUnChGr = new ArrayList<>();
   private void resetTemporaryUnavailableChannelsAndGroups() {
        temporaryUnavailableChannels.clear();
        temporaryUnChGr.clear();
        }
```

---

17
classe StateManager.java

método subscribedToOnlyTemporaryUnavailable linha 391

bad smell detectado : Long Identifier [in method subscribedToOnlyTemporaryUnavailable]: The length of the field subscribedToOnlyTemporaryUnavailable is 36.

antes da refatoração:

```
        private boolean subscribedToOnlyTemporaryUnavailable() {
        return effectiveChannels().isEmpty() && effectiveChannelGroups().isEmpty();
```

```
        }
```

após refatoração:

```
        private boolean subscribedToOnTeUn() {
        return effectiveChannels().isEmpty() && effectiveChannelGroups().isEmpty();
        }
```

---

18
classe StateManager.java

método effectiveChannelGroups linha 409

bad smell detectado : Long Identifier [in method effectiveChannelGroups]: The length of the field temporaryUnavailableChannelGroups is 33.

antes da refatoração:

```
        private List<String> effectiveChannelGroups(boolean includePresence) {
        final List<String> effectiveChannelGroupsList = prepareMembershipList(groups,
presenceGroups, includePresence);

effectiveChannelGroupsList.removeAll(channelGroupsToPostponeSubscription(temporaryUn
availableChannelGroups));
        return effectiveChannelGroupsList;
        }
```

após refatoração:

```
        private List<String> effectiveChannelGroups(boolean includePresence) {
        final List<String> effectiveChannelGroupsList = prepareMembershipList(groups,
presenceGroups, includePresence);

effectiveChannelGroupsList.removeAll(channelGroupsToPostponeSubscription(temporaryUn
ChGr));
        return effectiveChannelGroupsList;
        }
```

---

19
classe DuplicationManager.java

método getKey linha 18

bad smell detectado : Long Statement [in method getKey]: The length of the statement "String aux=message.getPublishMetaData().getPublishTimetoken().toString().concat("-").concat(Integer.toString(message.getPayload().hashCode()));" is 143.

antes da refatoração:

```
	private String getKey(SubscribeMessage message) {
	String aux = message.getPublishMetaData().getPublishTimetoken().toString().concat("-").concat(Integer.toString(message.getPayload().hashCode()));
	return aux;
	}
```

após refatoração:

```
  private String getKey(SubscribeMessage message) {
	String um = message.getPublishMetaData().getPublishTimetoken().toString();
		String dois = Integer.toString(message.getPayload().hashCode());

	String aux = um.concat("-").concat(dois);
	return aux;
	}
```

---

20

classe AppEngineFactory.java

método execute linha 35

bad smell detectado : Magic Number [in method execute]: The method contains a magic number: 200

antes da refatoração:

```
 public Response execute() throws IOException {
	request = PubNubUtil.signRequest(request, pubNub.getConfiguration(), pubNub.getTimestamp());

	URL url = request.url().url();
	final HttpURLConnection connection = (HttpURLConnection) url.openConnection();
	connection.setUseCaches(false);
	connection.setDoOutput(true);
	connection.setRequestMethod(request.method());

	Headers headers = request.headers();
	if (headers != null) {
	for (int i = 0; i < headers.size(); i++) {
```

```
                String name = headers.name(i);
                connection.setRequestProperty(name, headers.get(name));
        }
        }

        if (request.body() != null) {
        BufferedSink outbuf;
        outbuf = Okio.buffer(Okio.sink(connection.getOutputStream()));
        request.body().writeTo(outbuf);
        outbuf.close();
        }

        connection.connect();

        final BufferedSource source =
Okio.buffer(Okio.source(connection.getInputStream()));
        if (connection.getResponseCode() != 200) {
        throw new IOException("Fail to call " + " :: " + source.readUtf8());
        }
        Response response = new Response.Builder()
                .code(connection.getResponseCode())
                .message(connection.getResponseMessage())
                .request(request)
                .protocol(Protocol.HTTP_1_1)
                .body(new ResponseBody() {
                @Override
                public MediaType contentType() {
                return MediaType.parse(connection.getContentType());
                }

                @Override
                public long contentLength() {
                String contentLengthField = connection.getHeaderField("content-length");
                long contentLength;
                try {
                        contentLength = Long.parseLong(contentLengthField);
                } catch (NumberFormatException ignored) {
                        contentLength = -1;
                }
                return contentLength;
                }

                @Override
                public BufferedSource source() {
                return source;
                }
                })
                .build();
```

```
        return response;
        }
```

após refatoração:

```
public static final int num = 200;
public Response execute() throws IOException {
        request = PubNubUtil.signRequest(request, pubNub.getConfiguration(),
pubNub.getTimestamp());

        URL url = request.url().url();
        final HttpURLConnection connection = (HttpURLConnection) url.openConnection();
        connection.setUseCaches(false);
        connection.setDoOutput(true);
        connection.setRequestMethod(request.method());

        Headers headers = request.headers();
        if (headers != null) {
        for (int i = 0; i < headers.size(); i++) {
                String name = headers.name(i);
                connection.setRequestProperty(name, headers.get(name));
        }
        }

        if (request.body() != null) {
        BufferedSink outbuf;
        outbuf = Okio.buffer(Okio.sink(connection.getOutputStream()));
        request.body().writeTo(outbuf);
        outbuf.close();
        }

        connection.connect();

        final BufferedSource source =
Okio.buffer(Okio.source(connection.getInputStream()));
        if (connection.getResponseCode() != num){
        throw new IOException("Fail to call " + " :: " + source.readUtf8());
        }
        Response response = new Response.Builder()
                .code(connection.getResponseCode())
                .message(connection.getResponseMessage())
                .request(request)
                .protocol(Protocol.HTTP_1_1)
                .body(new ResponseBody() {
                @Override
                public MediaType contentType() {
                return MediaType.parse(connection.getContentType());
```

```
            }

            @Override
            public long contentLength() {
            String contentLengthField = connection.getHeaderField("content-length");
            long contentLength;
            try {
                    contentLength = Long.parseLong(contentLengthField);
            } catch (NumberFormatException ignored) {
                    contentLength = -1;
            }
            return contentLength;
            }

            @Override
            public BufferedSource source() {
            return source;
            }
            })
            .build();
    return response;
    }
```

---