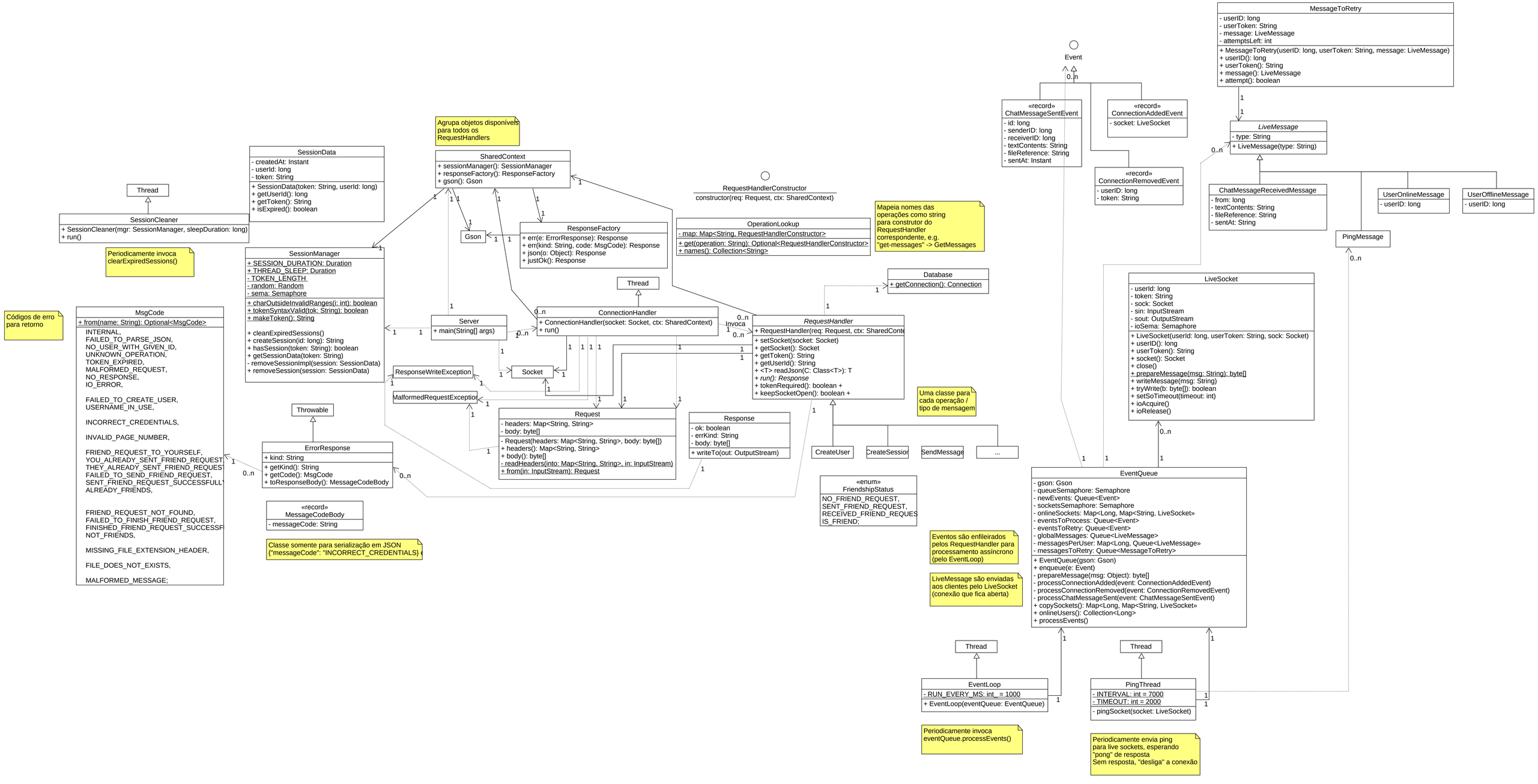
Thread
SessionCleaner
+ SessionCleaner(mgr: SessionManager, sleepDuration: long)
+ run()
Periodicamente invoca clearExpiredSessions()
SessionData
- createdAt: Instant
- userId: long
- token: String
+ SessionData(token: String, userId: long)
+ getUserId(): long
+ getToken(): String
+ isExpired(): boolean
Agrupa objetos disponíveis para todos os RequestHandlers
SharedContext
+ sessionManager(): SessionManager
+ responseFactory(): ResponseFactory
+ gson(): Gson
Gson
ResponseFactory
+ err(e: ErrorResponse): Response
+ err(kind: String, code: MsgCode): Response
+ json(o: Object): Response
+ justOk(): Response
RequestHandlerConstructor
constructor(req: Request, ctx: SharedContext)
OperationLookup
- map: Map<String, RequestHandlerConstructor>
+ get(operation: String): Optional<RequestHandlerConstructor>
+ names(): Collection<String>
Mapeia nomes das operações como string para construtor do RequestHandler correspondente, e.g. "get-messages" -> GetMessages
SessionManager
+ SESSION_DURATION: Duration
+ THREAD_SLEEP: Duration
- TOKEN_LENGTH
- random: Random
- sema: Semaphore
+ charOutsideInvalidRanges(i: int): boolean
+ tokenSyntaxValid(tok: String): boolean
+ makeToken(): String
+ cleanExpiredSessions()
+ createSession(id: long): String
+ hasSession(token: String): boolean
+ getSessionData(token: String)
- removeSessionImpl(session: SessionData)
+ removeSession(session: SessionData)
Thread
Server
+ main(String[] args)
ConnectionHandler
+ ConnectionHandler(socket: Socket, ctx: SharedContext)
+ run()
Database
+ getConnection(): Connection
Códigos de erro para retorno
MsgCode
+ from(name: String): Optional<MsgCode>
INTERNAL,
FAILED_TO_PARSE_JSON,
NO_USER_WITH_GIVEN_ID,
UNKNOWN_OPERATION,
TOKEN_EXPIRED,
MALFORMED_REQUEST,
NO_RESPONSE,
IO_ERROR,
FAILED_TO_CREATE_USER,
USERNAME_IN_USE,
INCORRECT_CREDENTIALS,
INVALID_PAGE_NUMBER,
FRIEND_REQUEST_TO_YOURSELF,
YOU_ALREADY_SENT_FRIEND_REQUEST
THEY_ALREADY_SENT_FRIEND_REQUEST
FAILED_TO_SEND_FRIEND_REQUEST,
SENT_FRIEND_REQUEST_SUCCESSFULLY
ALREADY_FRIENDS,
FRIEND_REQUEST_NOT_FOUND,
FAILED_TO_FINISH_FRIEND_REQUEST,
FINISHED_FRIEND_REQUEST_SUCCESSF
NOT_FRIENDS,
MISSING_FILE_EXTENSION_HEADER,
FILE_DOES_NOT_EXISTS,
MALFORMED_MESSAGE;
Socket
ResponseWriteException
MalformedRequestException
Invoca
RequestHandler
+ RequestHandler(req: Request, ctx: SharedConte
+ setSocket(socket: Socket)
+ getSocket(): Socket
+ getToken(): String
+ getUserId(): String
+ <T> readJson(C: Class<T>): T
+ run(): Response
+ tokenRequired(): boolean +
+ keepSocketOpen(): boolean +
Uma classe para cada operação / tipo de mensagem
CreateUser
CreateSession
SendMessage
...
Throwable
ErrorResponse
+ kind: String
+ getKind(): String
+ getCode(): MsgCode
+ toResponseBody(): MessageCodeBody
«record»
MessageCodeBody
- messageCode: String
Classe somente para serialização em JSON
{"messageCode": "INCORRECT_CREDENTIALS}
Request
- headers: Map<String, String>
- body: byte[]
- Request(headers: Map<String, String>, body: byte[])
+ headers(): Map<String, String>
+ body(): byte[]
- readHeaders(into: Map<String, String>, in: InputStream)
+ from(in: InputStream): Request
Response
- ok: boolean
- errKind: String
- body: byte[]
+ writeTo(out: OutputStream)
«enum»
FriendshipStatus
NO_FRIEND_REQUEST,
SENT_FRIEND_REQUEST,
RECEIVED_FRIEND_REQUES
IS_FRIEND;
Event
«record»
ChatMessageSentEvent
- id: long
- senderID: long
- receiverID: long
- textContents: String
- fileReference: String
- sentAt: Instant
«record»
ConnectionAddedEvent
- socket: LiveSocket
«record»
ConnectionRemovedEvent
- userID: long
- token: String
MessageToRetry
- userID: long
- userToken: String
- message: LiveMessage
- attemptsLeft: int
+ MessageToRetry(userID: long, userToken: String, message: LiveMessage)
+ userID(): long
+ userToken(): String
+ message(): LiveMessage
+ attempt(): boolean
LiveMessage
- type: String
+ LiveMessage(type: String)
ChatMessageReceivedMessage
- from: long
- textContents: String
- fileReference: String
- sentAt: String
UserOnlineMessage
- userID: long
UserOfflineMessage
- userID: long
PingMessage
LiveSocket
- userId: long
- token: String
- sock: Socket
- sin: InputStream
- sout: OutputStream
- ioSema: Semaphore
+ LiveSocket(userId: long, userToken: String, sock: Socket)
+ userID(): long
+ userToken(): String
+ socket(): Socket
+ close()
+ prepareMessage(msg: String): byte[]
+ writeMessage(msg: String)
+ tryWrite(b: byte[]): boolean
+ setSoTimeout(timeout: int)
+ ioAcquire()
+ ioRelease()
EventQueue
- gson: Gson
- queueSemaphore: Semaphore
- newEvents: Queue<Event>
- socketsSemaphore: Semaphore
- onlineSockets: Map<Long, Map<String, LiveSocket»
- eventsToProcess: Queue<Event>
- eventsToRetry: Queue<Event>
- globalMessages: Queue<LiveMessage>
- messagesPerUser: Map<Long, Queue<LiveMessage»
- messagesToRetry: Queue<MessageToRetry>
+ EventQueue(gson: Gson)
+ enqueue(e: Event)
- prepareMessage(msg: Object): byte[]
- processConnectionAdded(event: ConnectionAddedEvent)
- processConnectionRemoved(event: ConnectionRemovedEvent)
- processChatMessageSent(event: ChatMessageSentEvent)
+ copySockets(): Map<Long, Map<String, LiveSocket»
+ onlineUsers(): Collection<Long>
+ processEvents()
Eventos são enfileirados pelos RequestHandler para processamento assíncrono (pelo EventLoop)
LiveMessage são enviadas aos clientes pelo LiveSocket (conexão que fica aberta)
Thread
EventLoop
- RUN_EVERY_MS: int_ = 1000
+ EventLoop(eventQueue: EventQueue)
Periodicamente invoca eventQueue.processEvents()
Thread
PingThread
- INTERVAL: int = 7000
- TIMEOUT: int = 2000
- pingSocket(socket: LiveSocket)
Periodicamente envia ping para live sockets, esperando "pong" de resposta
Sem resposta, "desliga" a conexão