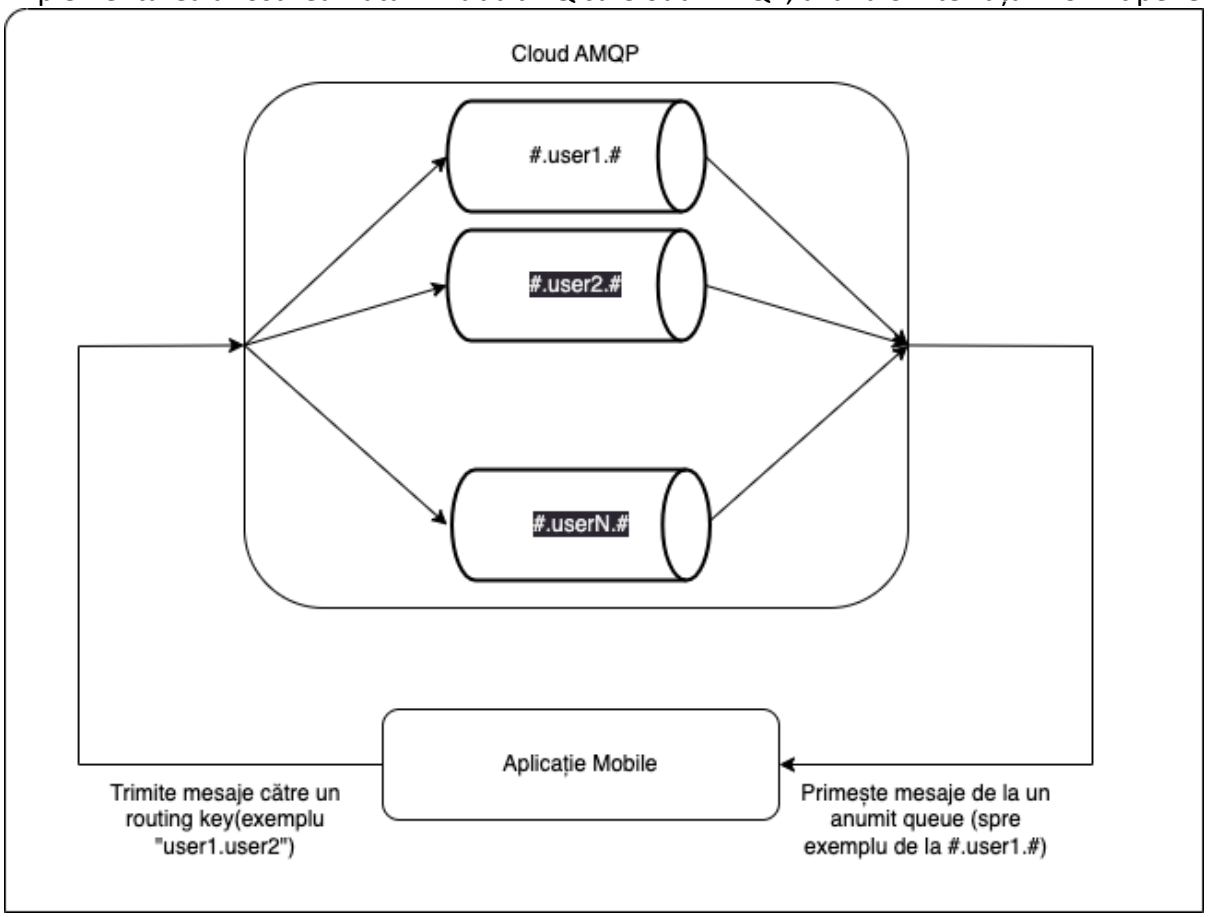
Implementare Aplicație Multichat

Anton Ioana-Carina
Fariseu Adriana-Teodora

Table of Contents

Arhitectură și tehnologie folosită

Implementarea a fost realizată în RabbitMQ cu Cloud AMQP, având o interfață în Swift pe iOS.



Cloud AMQP

Este un serviciu de cloud care permite folosirea de RabbitMQ fără a instala erlang. Acesta oferă posibilitatea de a crea exchange-uri care pot conține cozi ce pot fi legate la un routing key

Pentru implementarea chat-ului s-a folosit un exchange de tip topic care are câte coadă pentru fiecare user cu binding key-uri cu pattern-uri ”#.user.#” .Atunci când se trimite un mesaj, se va trimite cu un routing key în formatul „participant1.participant2.[...].participantN”, mesajele ajungând astfel în fiecare coadă care conține participantul respectiv pe orice poziție a cheii, indiferent dacă se află la început, mijloc sau sfârșit, deoarece # implică zero sau mai multe cuvinte. Spre exemplu, avem conversația cu participanții user1, user3 și user5. Pentru ca mesajul să fie transmis la toți utilizatorii, cheia va trebui să fie ”user1.user3.user5”

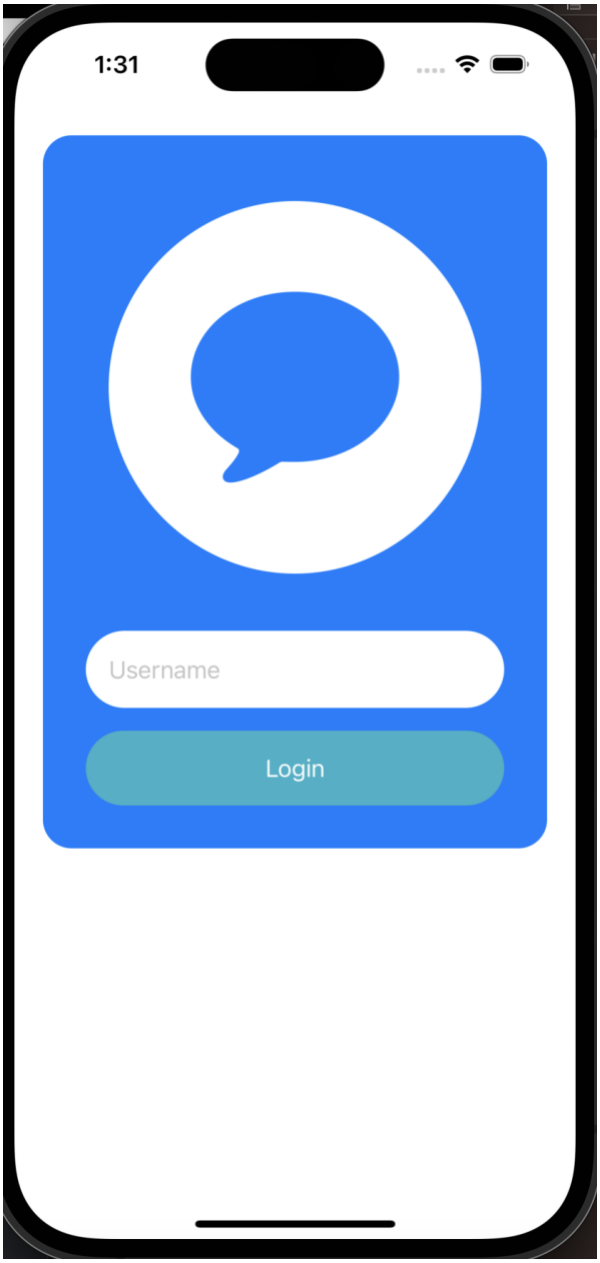
Aplicația de mobile

Este interfața grafică unde utilizatorul poate să trimită mesaje în cozile din RabbitMQ. Cozile unde se trimit mesajele sunt abstractizate sub formă de conversații, unde sunt reținuți utilizatorii cărora li se vor publica mesajele din conversație. La trimiterea unui mesaj, se va face send la un singur mesaj cu routing key-ul în formatul precizat anterior. Aceasta este realizată în SwifUI

Funcționalități

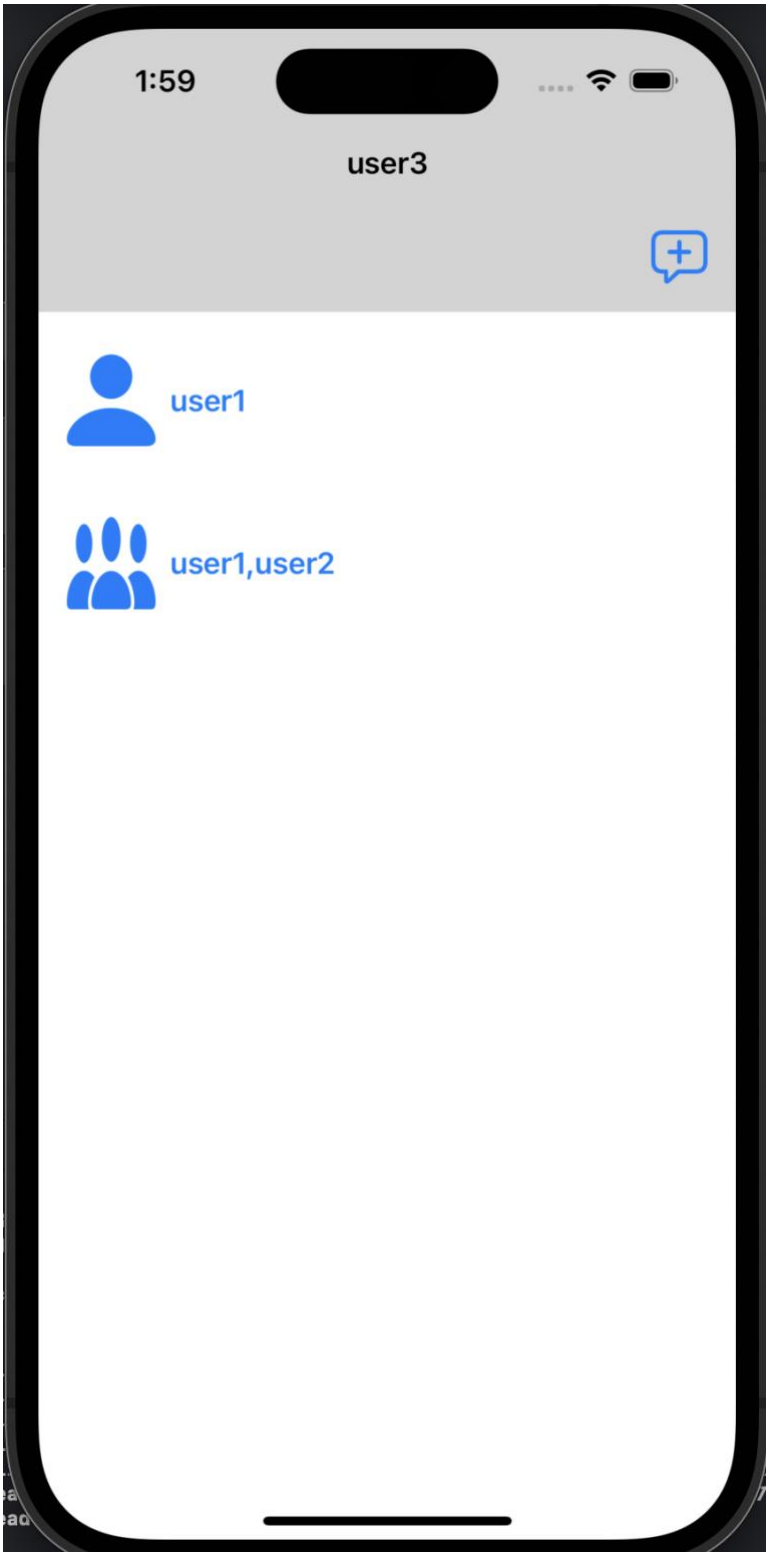
Conectare

Atunci când se lansează aplicația, va apărea un ecran de logare unde se poate introduce utilizatorul. Există 4 utilizatori predefiniți cu care se poate face logarea: user1, user2, user3 și user4. În momentul în care se va face login, va fi stabilită conexiunea la Cloud AMPQ



Conversații

Conversațiile apar pe pagina de logare, având iconițe sugestive în funcție de câte persoane se află. Ele sunt denumite după participanții lor pentru a putea obține cheia pentru publicarea mesajului, înlocuind virgulele cu puncte



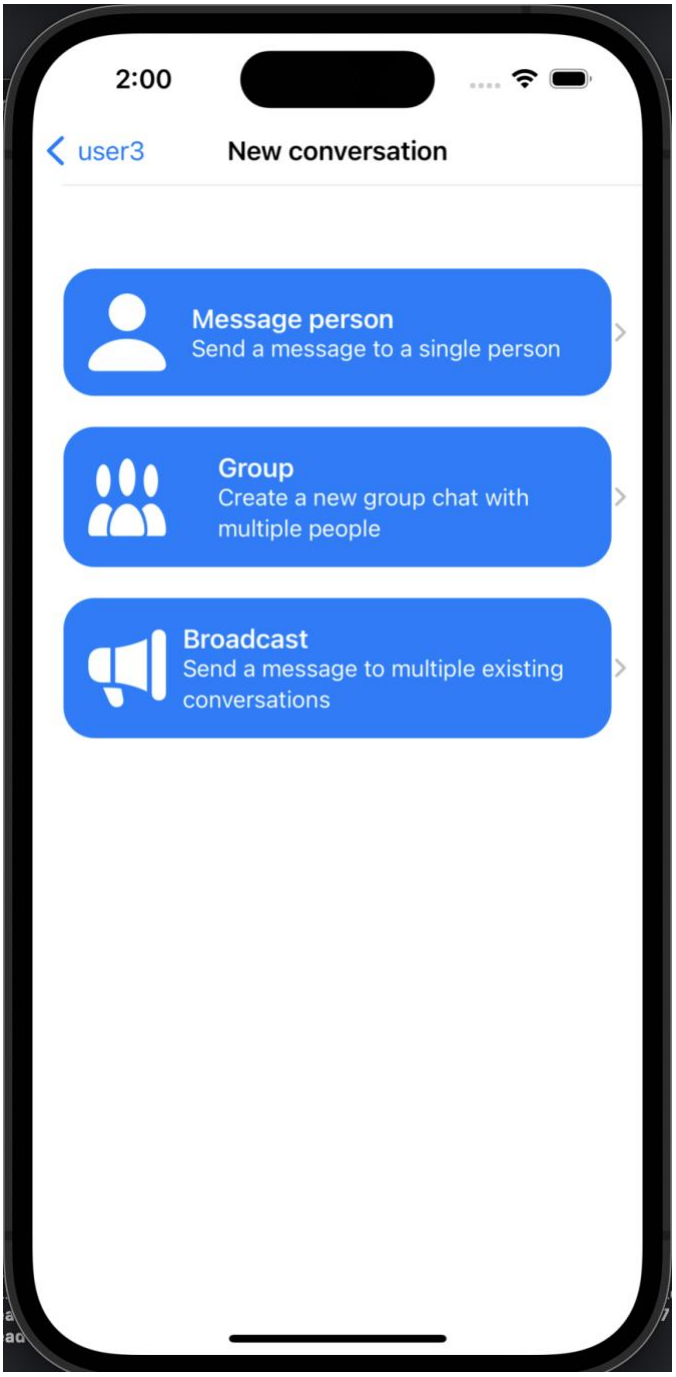
Creare de conversații

Butonul cu plus din colțul de sus al ecranului anterior oferă posibilitatea de a crea o nouă conversație. Conversația poate fi de mai multe tipuri

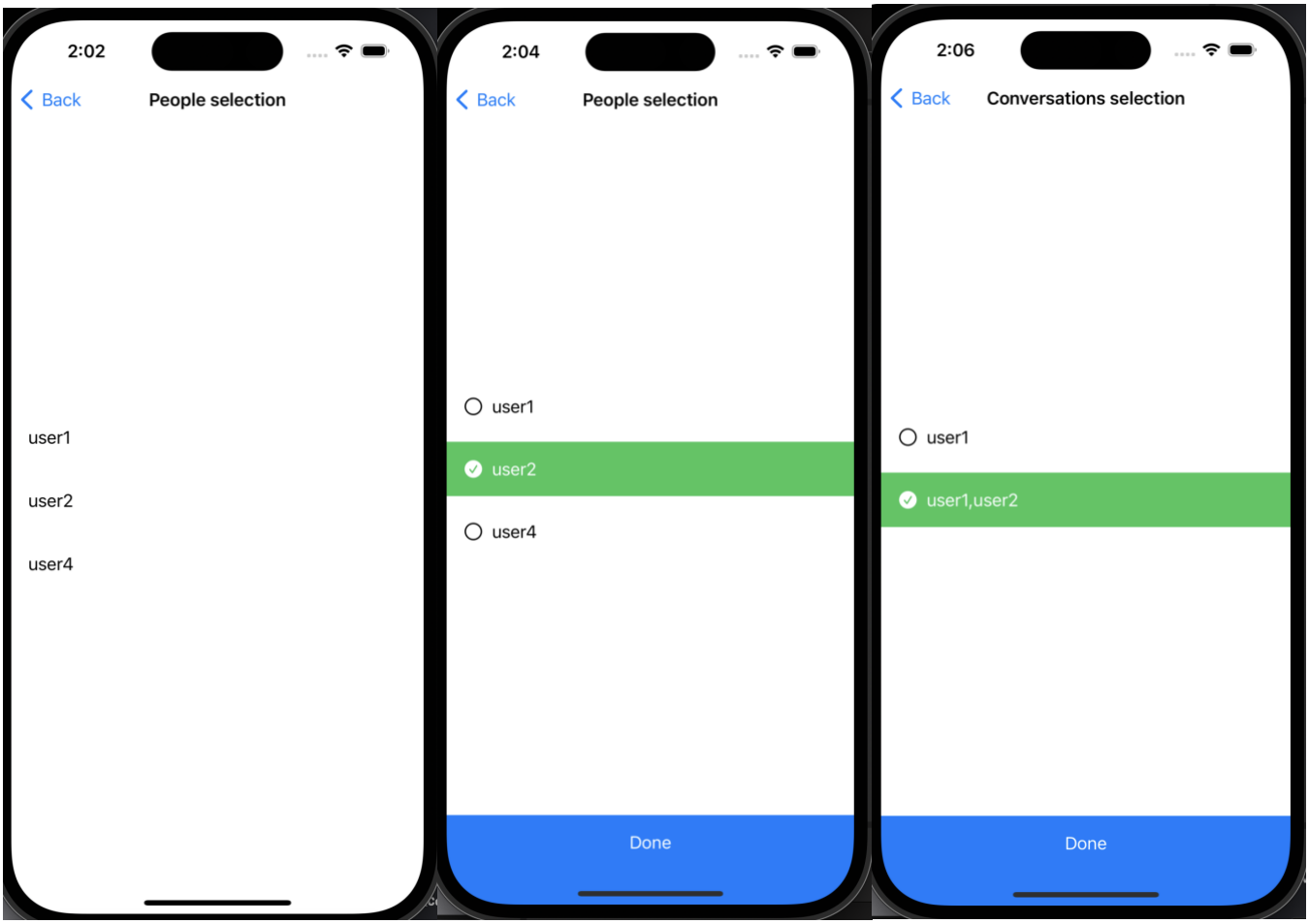
Tipuri de conversații

Există 3 tipuri de conversații disponibile:

- 1 la 1 – conversații private între doar 2 useri
- 1 la n – trimiterea unui mesaj în stil multicast către mai multe conversații deja existente, fără să facă o conversație nouă
- N la M – grupuri în care mai multe persoane pot să comunice simultan

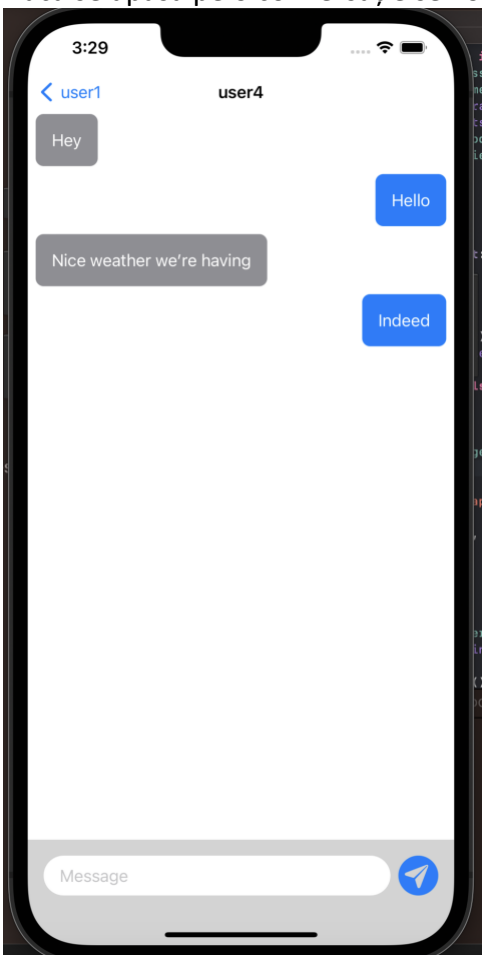


După selectarea tipului, în funcție de ce s-a ales, se vor putea selecta unul sau mai mulți useri sau mai multe conversații. Mai jos se pot vizualiza selecțiile pentru conversație privată, conversație de tip grup și conversație de tip multicast respectiv:



Mesagerie

Dacă se apasă pe o conversație se vor putea vedea mesajele din aceasta



Clase

RabbitService

Se ocupă cu conexiunea aplicației la CloudAMQP. Această conexiune este folosită pentru a primi sau trimite mesaje

```
import Foundation
import RMQClient

class RabbitService {
    static var shared = RabbitService()
    let connection = RMQConnection(uri: "amqps://hmejcqoa:ByNiIG-z6iiY7EQxE7A_lcQfv31HPCTL@shark.rmq.cloudamqp.com/hmejcqoa",
                                   delegate: RMQConnectionDelegateLogger())


    private init() {

    }

    func connect(onSuccess: @escaping ()->Void = {}) {
        connection.start {
            print("Connected")
            onSuccess()
        }
    }
}
```

ConversationConnection

Se conectează la exchange pe coada corespunzătoare utilizatorului logat. Coada aceasta va fi folosită pentru a asculta mesage, în timp ce exchange-ul se folosește pentru a publica mesaje pe diferite routing keys

```
import Foundation
import RMQClient

protocol Connection {
    var exchange: RMQExchange { get set }
    var channel: RMQChannel { get set }
    var queue: RMQQueue { get set }
}

struct ConversationConnection: Connection {
    var exchange: RMQExchange

    var channel: RMQChannel

    var queue: RMQQueue

    init() {
        channel = RabbitService.shared.connection.createChannel()
        exchange = channel.topic("conv")
        let user = SessionHandler.shared.user!.name!
        queue = channel.queue("\(user).conv")
        queue.bind(exchange, routingKey: "#.\(user).#")
    }
}
```

RabbitPublisher

Pentru trimiterea de mesaje la cozile corespunzătoare din RabbitMQ. Metoda send specifică routing key-ul cu care va fi trimis

```
import Foundation
import RMQClient

class RabbitPublisher<T:RabbitCodable> {
    let encoder = JSONEncoder()
    let connection: Connection

    init(connection: Connection) {
        self.connection = connection
    }

    deinit {
        connection.channel.close()
    }

    func send(message: T, on routingKey: String) {

        guard let data = message.rabbitMessage(with: encoder) else { return }
        connection.exchange.publish(data, routingKey: routingKey)
    }
}
```

Exemplu de utilizare send în clasa de ViewModel pentru o conversație:

```
func send(message: String) {
    guard let author = SessionHandler.shared.user?.name else { return }
    let rmsg = Message(content: message, author: author, conversation: conversation.id)
    guard let db = SessionHandler.shared.db,
          let routingKey = routingKey
            else { return }
    //adaugare in baza de date
    do {
        try rmsg.add(to: db)
    } catch (let error) {
        print("Failed to uplod message to firestore: \(error)")
    }
    rabbitProducer?.send(message: rmsg, on: routingKey)
}
```

Compunere routing key:

```
routingKey = name.replacingOccurrences(of: ",", with: ".")
```

Unde name este numele conversației

RabbitListener

Pentru a recepționa mesajele primite pe coada specificată în clasa de ConversationConnectio

```swift
import Foundation
import RMQClient

class RabbitListener<T: RabbitCodable> {
    let decoder = JSONDecoder()
    let connection = RabbitService.shared.connection
    @Published var message: T?

    func listen(to connection: Connection) {
        if self.connection.isOpen() {
            connection.queue.subscribe { [weak self] message in
                guard let self = self,
                      let body = message.body,
                      let message = T.decodeMessage(data: body, with: self.decoder)
                else { return }
                print("Received: \(message)")
                self.message = message
            }
        } else {
            print("Connection isn't open")
        }
    }
}
```

Exemplu de utilizare:

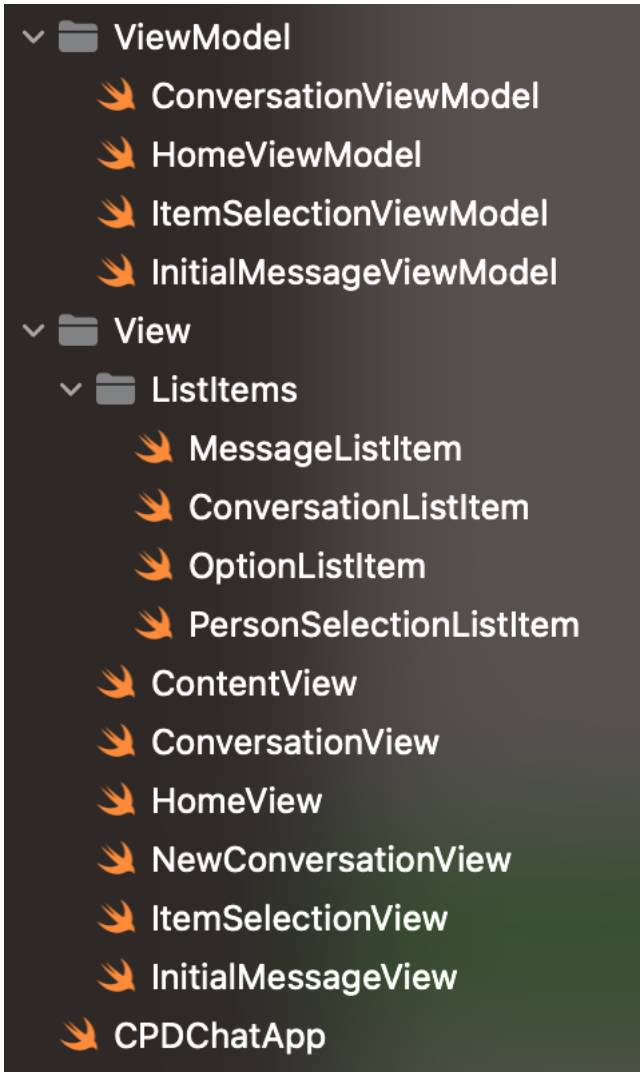
```swift
    init(conversation: Conversation) {
        self.conversation = conversation
        if let name = conversation.name {
            routingKey = name.replacingOccurrences(of: ",", with: ".")
            if let session = SessionHandler.shared.user?.name {
                routingKey = routingKey?.appending(".\(session)")
            }
        }
        if let id = conversation.id {
            connection = ConversationConnection()
            rabbitProducer = RabbitPublisher(connection: connection!)
            rabbitListener = RabbitListener()
            rabbitListener?.listen(to: connection!)
        }

        setupBindings()
    }
    func setupBindings() {
        rabbitListener?.$message.receive(on: DispatchQueue.main).sink { [weak self] message in

            guard let message = message, message.conversation == self?.conversation.id,
            ((self?.messages.contains(message)) == false) else { return }
            self?.messages.append(message)
            print("Added message to conversation")
        }.store(in: &subscribers)
    }
```

Clase de UI

Deoarece este o aplicație de mobile, arhitectura este una de Model View ViewModel. Fără a aglomera documentația cu cod irelevant de UI, acestea sunt clasele pentru ViewModel-uri și View-uri



Model de date

```
struct User: Codable {
    @DocumentID var name: String?
}
```

```
struct Conversation: Codable, Identifiable {
    @DocumentID var id: String? = UUID().uuidString
    var name: String?
    var type: ConversationType = .person
}
```

```
struct Message: Codable, Identifiable, Comparable, Hashable {
    @DocumentID var id: String? = UUID().uuidString
    var content: String
    var author: String
    var date: Date = Date()
    var conversation: String?
```

Pentru a nu pierde conversațiile și mesajele după ce sunt consumate, există stocare în Firestore de la Firebase (s-a încercat inițial cu MongoDB, însă librăria de Swift are niște dependențe care se bat cap în cap), care este o bază de date NoSQL bazată pe documente. Există colecții de useri, care conțin documentele userilor care au subcolecții de ID-uri pentru conversații. De asemenea, există o colecție de conversații cu documente cu id-urile, numele, tipul și o subcolecție care conține mesajele