RescueStation: advanced use prototype: Messaging

Challenge Rating: 4/5

This is part of the RescueStation advanced use prototype. Want a project with a heart? Want some mobile app javascript? How about some multi-platform push notifications? Then this project is for you.

Your project

This project will construct our first prototype app, which will illustrate the processes of connecting people and sending messages between them.

We need to evaluate if a quick, unobtrustive message exchange with a supportive influence can act as a pick-me up.

If it does, we need to start to evaluate its effect over the longer term. Does it dimish with time? Is it different if more than one supportive contact is used?

Moreover, we'd like to know what pressures responding to messages has on the supporter. Are they a good alternative to texting, or phoning? Are they too much of a distraction?

We'll do this evaluation in two ways:

First, we will rely on the anecdotal reports from the users of the system; both rescuers and rescuees.

Second, we will tie this in with data which we'll obtain using our sister project; using on-board sensors to obtain physiologial indicators of stress, inhibition and arousal.

Benefits

To Us:

This is the next step in a long-running development of RescueStation: we hope that the product will help everyone who uses it. But, if we find that it doesn't, or could be unsafe, we will have solid evidence for it, which will contribute to our understanding of the psychological effects of mobile applications.

To You:

You will be developing a cross-platform mobile application, using a tool chain called Ionic, which sits on top of an open source technology called Cordova. It all uses Javascript. We transferred to using Ionic/Cordova from using native kits, because it meant that we could leverage one language across our whole development: from powering our Raspberry Pis, to mobile, to server-side scripting, we can use Javascript.

Because this language underpins so much of today's technology stack, developers are much in demand.

Successfully completing this project, you will have become proficient in:

- Ionic 1 / Angular JS development
- Javascript asynchronous calls
- REST APIs and PaaS
- Multiplatform Push notifications

Development Team

Skillset

- Javascript - Angualar JS
- HTML, CSS
- Ionic / Cordova
- Javascript - Node JS
- Ably - Realtime data service

Difficulty

Challenging: 4/5

Development Stages

Because this app is so important, we will be breaking down development in separate, verifiable stages.

Create Skeleton App

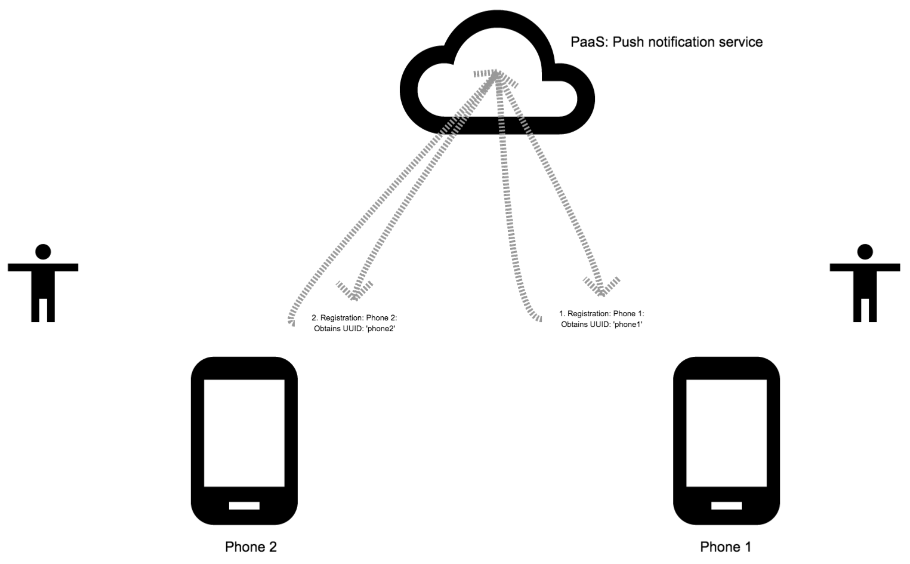
This stage will enable you to get to grips with a cross-platform software development kit.

A skeleton app is available, called 'little list'.

Add your chosen push notification SDK for your provider. This will be a combination of a REST API and a client library, which will recieve the push notification on whatever OS it is installed on. (Android / iOS)

Verify the App Installation ID

Verify the App Installation ID - a UUID which is created when the app is run for the first time. (Comes with most push notification SDKs)


PaaS: Push notification service
2. Registration: Phone 2: Obtains UUID: 'phone2'
1. Registration: Phone 1: Obtains UUID: 'phone1'
Phone 2
Phone 1

Test App 1

This test app ensures the basic connection to a push provider can be made.

In this case the push provider is 'naked' - that is, the mobile client interacts directly with it. Normally, the client would go through a service which wraps the push provider.

The app will demonstrate the use of a UUID to identify itself as a target for push notifications. It will demonstrate a method for specifying another target UUID to which messages can be sent to.

The UI will demonstrate:

1. A display of the app's own UUID.
2. Input for second UUID.
3. Button to send message
4. List of received messages

The test will demostrate:

App installed on two devices.

App 1 sends a request for a push notification to naked push provider.

App 2 on second device receives the push notification.

Test App 2

This test App introduces a definite switch between rescuer and rescuee roles, which will be important later.

We will define the following:

Role IDs:

- 0: Rescuer
- 1: Rescuee

The UI will demostrate:

As Test App 1, plus:

1. A UI switch between roles
2. a different background colour for each role
   1. Orange: Rescuer
   2. Green: Rescuer

Test App 3

This test app introduces the basic messaging protocol.

Push notification services don't guarantee receipt of their notifications: messages are pushed out, and forgotten.

In the future, we need to know that the message has reached its target. Someone will be counting on it.

We fill the tiny payload of the push service with our own metadata, leaving enough space for a payload of our own. The metadata enables us to define extra message types to our protocol, including an ACK and a NACK, which are sent in response to a received message, and let the sender know it got there OK, or if there was some sort of mistake, and the reciving app didn't know what to do with it. NACK messages are sent when the code handling the payload finds an error with its data, and cannot process it further.

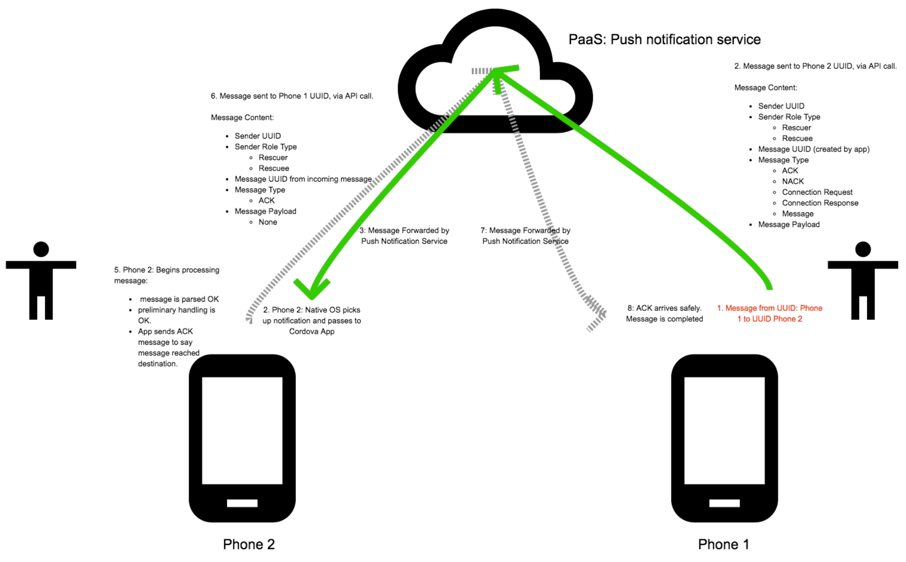
We will define the following:

1. **Message Type ID:**
   - 0: ACK (acknowledgement)
   - 1: NACK (not an ACK)
   - 2: Connection Request
   - 3: Connection Response
   - 3: Message
2. **Message ACK/NACK timeout in seconds:**
   - 10
3. **Connection Request Responses:**
   - 1: Connection Granted
   - 0: Connection Rejected

We will define the following behaviour:

1. **On a message send**
   1. Construct the message. Use the message payload of push provider to carry:
      - UUID of sender,

        - UUID of message (new for each message),
        - Message Type ID,
        - Sender's Role Type ID.
        - Reserve the rest of the payload. It's content will depend on the message type Id.
    2. Send the message, wait for an ACK from the receiver
    3. On a timeout: error, saying the message has got lost
    4. On a NACK: report an error, saying the message was garbled (the message payload was unexpected)
2. On message received:
    - immediately construct a new message to the sender, **with the same Message UUID**, but a message type of ACK.


PaaS: Push notification service
6. Message sent to Phone 1 UUID, via API call.
Message Content:
• Sender UUID
• Sender Role Type
∘ Rescuer
∘ Rescuee
• Message UUID from incoming message
• Message Type
∘ ACK
• Message Payload
∘ None
2. Message sent to Phone 2 UUID, via API call.
Message Content:
• Sender UUID
• Sender Role Type
∘ Rescuer
∘ Rescuee
• Message UUID (created by app)
• Message Type
∘ ACK
∘ NACK
∘ Connection Request
∘ Connection Response
∘ Message
• Message Payload
3: Message Forwarded by Push Notification Service
7: Message Forwarded by Push Notification Service
5. Phone 2: Begins processing message:
• message is parsed OK
• preliminary handling is OK.
• App sends ACK message to say message reached destination.
2. Phone 2: Native OS picks up notification and passes to Cordova App
8: ACK arrives safely. Message is completed
1. Message from UUID: Phone 1 to UUID Phone 2
Phone 2
Phone 1

Test App 4

This test app confirms a working connection protocol. Once this app is completed, we can connect and send messages to multiple recipients.

This description of the app UI structure should be enough to get you going, but the connection protocol is most important: it allows the previously outlined messaging protocol to be used in conjunction with a QR code shown on-screen, to connect two people who are next to one another.

Structure:

- Home Screen
    - role switch button
        - app changes colour and behaviour on role switch
    - send button (enabled for Rescuee Mode Only)
    - scrollable list for received messages

            - received message
                - text: nickname of sender
                - button: 'respond'
    - menu
        - profile
        - connections
- Profile screen:
    - input: nickname
- Connections screen:
    - Scrollable List of Connections
        - Single Select - **enabled only in Rescuee Mode**
            - selected item is the conact to which the 'big button' message will go.
        - List item
            - Text: Nickname
            - Button: Send - **enabled only in Rescuer Mode**
            - Swipe to reveal delete button
    - 'Add' button - activates 'New Connection' screen
- New Connection screen:
    - Wizard functionality to add a new connection, dependent on the current role

Send Behaviour

1. User presses 'send' button on either Home Screen, or connections screen
    - App contacts PaaS to send notification.
        - App listens for PaaS 200 OK response
            - No response: Error
        - App listens for ACK response
            - Timeout: Error
            - NACK: error
2. User presses 'respond' button on message list item
    - 'respond' button disabled

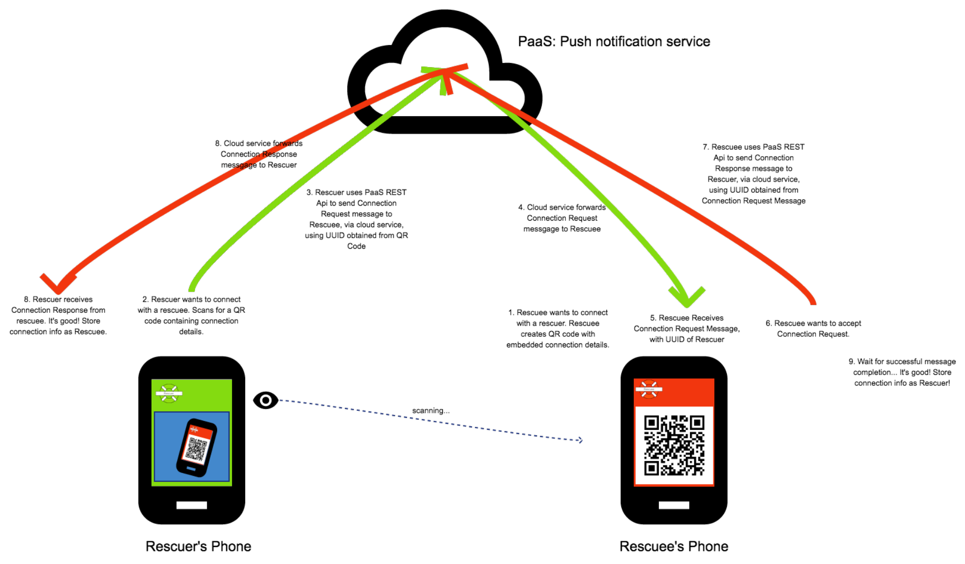
- App contacts PaaS to send notification.
    - App listens for PaaS 200 OK response
        - Other response / HTTP timeout
            - report error
            - respond button: enabled
    - App listens for ACK response
        - Timeout or NACK:

        - report error
        - respond button: enabled
    - ACK:
        - respond button changes to Green indicator

Receive Behaviour

1. Message pops up as a toast
2. Message added to homescreen message list

Behaviour for a new connection



PaaS: Push notification service
1. Rescuee wants to connect with a rescuer. Rescuee creates QR code with embedded connection details.
2. Rescuer wants to connect with a rescuee. Scans for a QR code containing connection details.
3. Rescuer uses PaaS REST Api to send Connection Request message to Rescuee, via cloud service, using UUID obtained from QR Code
4. Cloud service forwards Connection Request messgage to Rescuee
5. Rescuee Receives Connection Request Message, with UUID of Rescuer
6. Rescuee wants to accept Connection Request.
7. Rescuee uses PaaS REST Api to send Connection Response message to Rescuer, via cloud service, using UUID obtained from Connection Request Message
8. Cloud service forwards Connection Response messgage to Rescuer
8. Rescuer receives Connection Response from rescuee. It's good! Store connection info as Rescuee.
9. Wait for successful message completion... It's good! Store connection info as Rescuer!
scanning...
Rescuer's Phone
Rescuee's Phone

The following functionality is implemented on the 'New Connection Screen'

1. If the Role is **Rescuee**:
    1. Construct a QR code which embeds the App Installation UID, and the profile Nickname
    2. Display **Step 1**:
        1. The Large QR Code
        2. Cancel Button
        3. Continue Button (disabled)
    3. Wait to Receive a **Connection Request** Message
        1. On Receipt of a Connection Request Message:
        2. Display the Payload of the Message (the nickname of the rescuer)
        3. Enable the Continue Button
    4. On Continue:
        1. Display **Step 2:**
            1. QR scanner
            2. Cancel Button

         3. Continue Button (disabled)
      2. Scan For a QR Code (it's the rescuer's App Installation UID and the nickname)
   5. On A Successful Scan:
      1. Compare the nickname and App Installation UUID from the Connection Request and the QR code
      2. On equality:
         1. Store the nickname and the App Installation UUID as a RESCUER
         2. Send a message of type **Connection Response**. Payload - Connection Granted
         3. Return to the Connections Screen
      3. Else
         1. Send a message of type **Connection Response**. Payload - Connection Rejected
         2. Return to the Connections Screen
   6. On an unsuccessful Scan:
      1. Send a message of type **Connection Response**. Payload - Connection Rejected
      2. Return to the Connections Screen
   7. On Cancel:
      1. Send a message of type Connection Response. Payload - Connection Rejected
      2. Return to the Connections Screen
2. If the Role is **Rescuer**
   1. Display:
      1. QR Scanner
      2. Cancel Button
   2. On a Successful Scan:
      1. Decode the QR code:
         1. Rescuee's App Installation ID
         2. Rescuee's Nickname
      2. Send a message of type Connection Request:
         1. Payload: Nickname
      3. Wait For a Connection Response Message.
      4. On Connection Response Message: Connection Granted
         1. Store the nickname and the App Installation UUID as a RESCUEE
         2. Return to the Connections Screen
      5. Else:
         1. Return to the Connections Screen
   3. On Cancel:
      1. Return to the Connections Screen

Beyond the Test Apps

The test apps are simply intended to ensure that the protocols are working. It really important to get the UI as clean and as friendly as possible.