SaiN: Staffing Artificial Intelligent Network

Introduction

This document provides an overview of the SaiN application, that will allow the hospital to efficiently manage shifts. The application is designed to be utilized by head staff for the creation, deletion, and overall management of staff shifts. It centralizes shift management tasks, providing a streamlined interface for administrative personnel to efficiently handle shift scheduling.

Application Features

Head staff members use the mobile application interface to perform the following actions:

1. **Create Shift Requests:** Input details for new shifts, including position, date, time, and response deadline, **Figure 1** and **2** demonstrates.
2. **Manage Shift Requests:** View and manage Open and Assigned shift requests with the ability to cancel Open shift requests, which will notify any staff member that applies to it that it has been canceled. **Figure 3** demonstrates the Open Shift requests and **Figure 4** the ability to cancel them. **Figure 5** shows the Assigned Shift requests.
3. **Confirmation and Error Messages:** The app provides confirmations upon successful creation of shift requests and error messages if there is a problem fetching data from the server. **Figure 6** demonstrates.

SMS Functionality with Twilio Integration

The SMS functionality within the Flask application is enabled through integration with Twilio's messaging API, used to both send and receive text messages. Here's a detailed explanation of its operation:

1. **Twilio Integration:** The Flask app utilizes Twilio's API for SMS services, enabling text message interactions with staff.
2. **Webhook Configuration:** A webhook is set up in Twilio, linked to the Flask app's SMS endpoint. This webhook is triggered whenever an SMS is received, prompting Twilio to send a request to the Flask app. For this Flask application it is ‘/sms_webhook’ and it must be added to the purchased Twilio phone number on their website as shown in **Figure 7**. Their website also offers additional monitoring information that can be utilized.
3. **Receiving SMS:** The Flask app processes incoming SMS messages, extracting details like message content and sender information.
4. **Processing SMS:** The Flask app is programmed to understand specific SMS formats, such as 'ACCEPT <RequestID>' for shift acceptance. Instructions on how to reply to SMS are provided to users, ensuring clarity in communication.

5. **Responding to SMS:** Based on the received message, the Flask app sends appropriate responses, including confirmations or error messages.
6. **Database Interaction:** The Flask app updates the database in response to SMS actions, like confirming a shift acceptance. It also retrieves information from the database at the shifts respond by time to send out SMS confirmation to the staff member who received the shift and a rejection to everyone else who bid on it.
7. **Error Handling and Confirmation:** It also manages incorrect SMS formats, guiding users towards the correct response mechanism.

Maintenance using Log Files

The Flask backend of the application generates log files crucial for maintenance and monitoring system activities. These logs record detailed information about operations conducted through the mobile app and SMS interactions, as well as any errors or anomalies.

The technical staff should routinely check the **'app.log'** file for unusual activities or errors and use it to identify and resolve issues. They are currently set up to reset when the log file reaches a size of 1 GB.

Adjusting Application Settings

To change the mobile application's endpoint configuration, the technical team should edit the '**config.json'** file located in the assets folder of the mobile application. After making the necessary changes, the app needs to be rebuilt using Flutter commands for the changes to take effect.

Building and Retrieving the Application for Mobile Platforms

To ensure the application is correctly built and retrieved for both Android and iOS platforms, follow these step-by-step instructions:

Android Deployment

1. **Prepare the Application:**
    - Update the config.json file in the asset folder, if needed.
2. **Build the APK:**
    - Open a terminal window or cmd
    - Navigate to the root directory of the Flutter project.
    - Run the following commands:
        i. flutter clean
        ii. flutter build apk –release
    - This compiles the application into an APK file.

3. **Locate the APK:**
    - Once the build process is complete, find the APK in the following path within the Application folder 'build\app\outputs\flutter-apk\app-release.apk '.
4. **Distribute the APK:**
    - It's recommended to first test the APK on a physical device or an emulator to ensure everything works as expected.
    - The APK file can be directly shared with users.

iOS Deployment

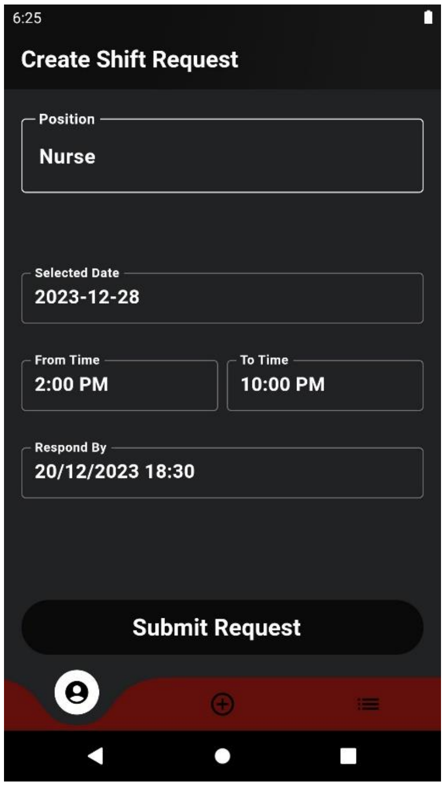
1. **Prepare the Application:**
    - Update the config.json file in the asset folder, if needed.
    - Ensure you have an Apple Developer account, specifically set up for enterprise or internal app distribution.
2. **Build the IOS App:**
    - Open a terminal window or cmd
    - Navigate to the root directory of the Flutter project.
    - Run the following commands:
        i. flutter clean
        ii. flutter build ios
3. **Open the Project in Xcode:**
    - Navigate to the ios folder within your Flutter project.
    - Open the Runner.xcworkspace file in Xcode.
4. **Configure Signing and Team Settings:**
    - In Xcode, select your development team for enterprise distribution.
    - Set up signing configurations tailored for internal distribution.
5. **Prepare for Distribution:**
    - First test the application on an IOS device
    - In Xcode, select 'Product' > 'Archive' to create an archive of the app.
    - After archiving, you can export the application from the Organizer window. Choose the option for Ad Hoc distribution or enterprise distribution depending on your Apple Developer account type.
6. **Distribute the App within the Hospital:**
    - Once you have the exported **'.ipa'** file, you can distribute it to selected hospital staff using one of the following methods:
        i. **Mobile Device Management (MDM):** If the hospital uses an MDM solution, you can distribute the app through it. This allows centralized management of app deployment.
        ii. **Ad Hoc Distribution:** For a limited number of devices (up to 100), you can use Ad Hoc distribution. You will need to register each device's UDID (Unique Device Identifier) in your Apple Developer account.
        iii. **Enterprise Distribution:** If you have an Apple Developer Enterprise Program account, you can distribute the app internally without the need for device registration. You can host the **'.ipa'** file on an internal server or intranet, allowing staff to download and install it directly.

Deploying apps on iOS is more complex than on Android, as iOS doesn't support direct installations like APKs. However, there are alternatives for iOS deployment, such as publishing on the Apple App Store, employing Ad Hoc or Enterprise distribution methods, or utilizing TestFlight for beta testing. Each option has its own set of unique requirements and procedures.

GitHub Project Link: https://github.com/Hisham632/SMS-staffing-application

Contact me at hishamazzi1@gmail.com if you would like to make the repository private after you fork it or add you as a collaborator.

Figures


6:25

Create Shift Request

Position: Nurse

Selected Date: 2023-12-28

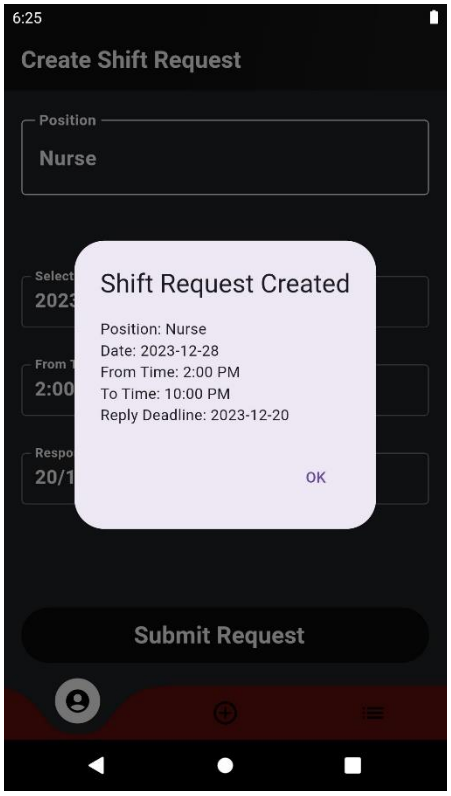
From Time: 2:00 PM

To Time: 10:00 PM

Respond By: 20/12/2023 18:30

Submit Request

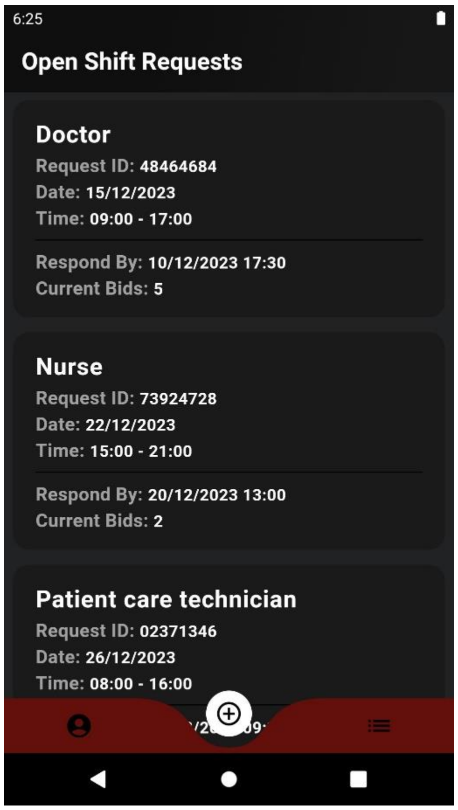
Figure 1


Shift Request Created

Position: Nurse
Date: 2023-12-28
From Time: 2:00 PM
To Time: 10:00 PM
Reply Deadline: 2023-12-20

OK

Figure 2


6:25

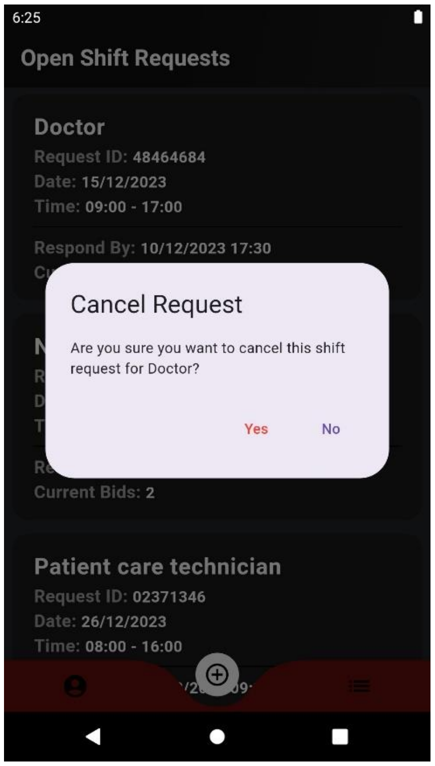
Open Shift Requests

Doctor
Request ID: 48464684
Date: 15/12/2023
Time: 09:00 - 17:00
Respond By: 10/12/2023 17:30
Current Bids: 5

Nurse
Request ID: 73924728
Date: 22/12/2023
Time: 15:00 - 21:00
Respond By: 20/12/2023 13:00
Current Bids: 2

Patient care technician
Request ID: 02371346
Date: 26/12/2023
Time: 08:00 - 16:00

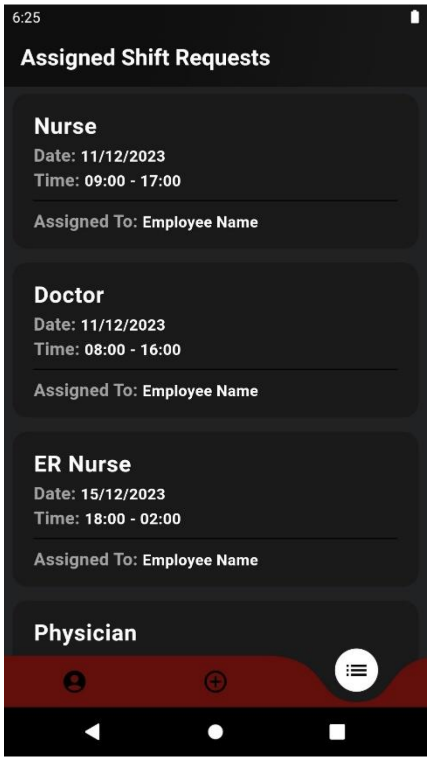
Figure 3


Cancel Request

Are you sure you want to cancel this shift request for Doctor?

Yes No

Figure 4


6:25

Assigned Shift Requests

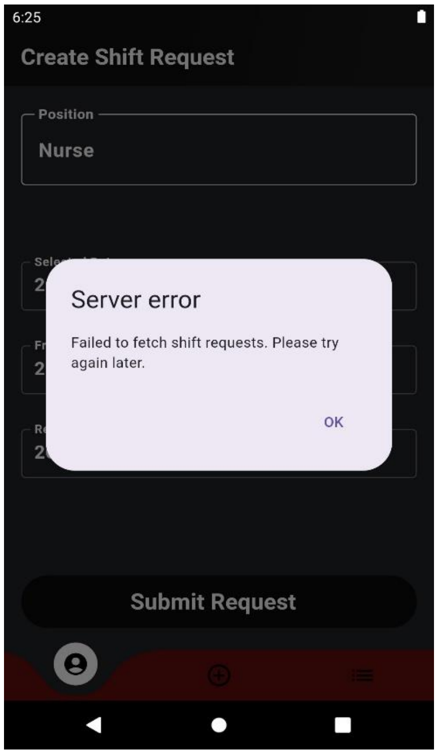
Nurse
Date: 11/12/2023
Time: 09:00 - 17:00
Assigned To: Employee Name

Doctor
Date: 11/12/2023
Time: 08:00 - 16:00
Assigned To: Employee Name

ER Nurse
Date: 15/12/2023
Time: 18:00 - 02:00
Assigned To: Employee Name

Physician

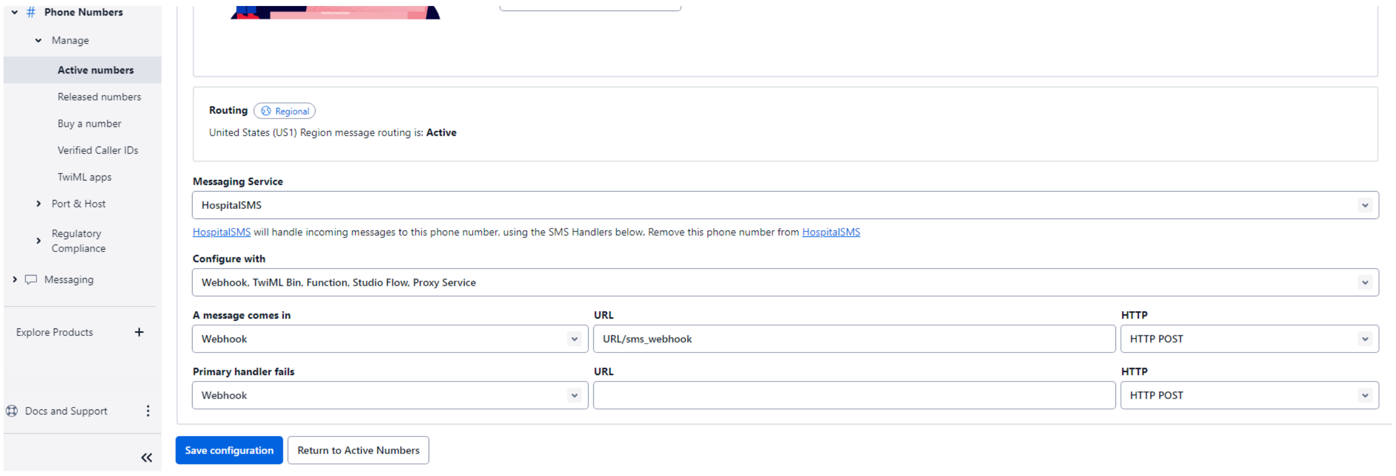
Figure 5


Server error

Failed to fetch shift requests. Please try again later.

OK

Figure 6



Routing Regional

United States (US1) Region message routing is: **Active**

Messaging Service

HospitalSMS

HospitalSMS will handle incoming messages to this phone number, using the SMS Handlers below. Remove this phone number from HospitalSMS

Configure with

Webhook, TwiML Bin, Function, Studio Flow, Proxy Service

A message comes in	URL	HTTP
Webhook	URL/sms_webhook	HTTP POST

Primary handler fails	URL	HTTP
Webhook		HTTP POST

Figure 7