



Housing Solutions

Mobile Application

Technical Specification

May 2021

Prepared by

Kristofor Slattery, Engineering Manager





Contents

1. About this document
 - 1.1. Version history & stakeholders
 - 1.2. Ubiquitous language
 - 1.3. Technology stack
 - 1.4. Target devices and viewports
 - 1.5. Operating system versions
 - 1.6. Testing
 - 1.7. Accessibility
2. App stores & deployment
3. API's
4. Screens & Features
 - 4.1. Core elements
 - 4.2. Login / Forgotten Password
 - 4.3. Sign up / Register
 - 4.4. Splash / Load Screen
 - 4.5. Home Screen (dashboard)
 - 4.6. News & Media feed
 - 4.7. Rent Transactions (Payment history)
 - 4.8. Make a payment
 - 4.9. Repairs Diagnosis
 - 4.9.1. AI image diagnosis
 - 4.9.2. Edison (Speech recognition AI)
 - 4.9.3. Traditional diagnosis method
 - 4.10. Scheduling a repair
 - 4.11. Forms
 - 4.11.1. Feedback
 - 4.12. Web chat Screen
 - 4.13. Message centre Screen
5. SQLite Database
 - 5.1. Table structure
 - 5.1.1. System settings
 - 5.1.2. Tenant
 - 5.1.3. Transactions
 - 5.1.4. Repairs history
 - 5.1.5. Message centre
 - 5.2. Field encryption
6. Wifi / internet connection
7. Push Notifications
 - 7.1. Push notification workflow
 - 7.2. Umbraco push notification tool
 - 7.3. Push notification types
 - 7.4. Member groups & distribution

- 7.5. Event triggers
- 8. Payment gateway
- 9. Message Centre
 - 9.1. Active threads
 - 9.2. Archived threads
- 10. Web Chat

1. About this document

Throughout various sections of this document you will find icons highlighting key dependencies of specific functionality. For example, when we talk about logging into the application the following icons will appear at the top of the section...

Icon	Description
	SQLite database - information is stored in a local SQL database for offline use and long term state management
	API - Connection to a specific API is required in order for the screen to function correctly. There is a direct dependency on data from Umbraco or a third party housing API such as Capita open housing
	Wifi Required - Wifi connection is required to use feature
	Web View - Content on the screen is IFramed in using a web view control. This will be the case for Umbraco forms or web chat

1.1 Version history & stakeholders

Version	Date	Author	Changes
v1.0	30/05/2021	Kris Slattery	First draft of specification

Stakeholders

Name	Title / Company
Rich Harvey	Housing Solutions
Sam Denslow	Housing Solutions
Andy Whyte	Housing Solutions
Amanda Stockhill	Housing Solutions
Laurence Earl	Prodo Digital
Jenny Bradshaw	Prodo Digital
Josh Hughes	Prodo Digital
Kris Slattery	Prodo Digital

1.2 Ubiquitous language

Simulator/Emulator - A software image of a physical device which is run on a PC or Mac which mirrors the device functionality and behaviour without the need for a physical device

SQLite database - local data storage for the application to persist information across shutdowns and thread closes.

API - Application Programming Interface, a web based service which allows our application to read /write data to and from 3rd party systems.

Web View - A react native component that allows us to embed a HTML web browser into an application screen. This shares similarity to web page IFrames.

Target Devices - All of the smartphone models that the application will be compatible with.

Viewport - a frame area on the smartphone screen which has a software pixel height and width.

Operating system - this refers to either IOS or Android operating systems

Encryption - The process of converting information into a format which obfuscates the original content.

Decryption - The process of reverting an encrypted block of information back in to a readable state

1.3 Technology stack

The following table outlines the technology stack (frameworks, languages and development tools) used to build the Housing Solutions mobile application. All technologies, frameworks and languages will be the latest version released at the time of the build unless otherwise stated.

React Native: React Native is an open-source mobile application framework used to develop software for Apple and Android devices.

Expo: Expo a toolchain built around **React Native** to help you quickly start an app. It provides a set of tools that simplify the development and testing of **React Native** apps and arms you with the components of user interface and services that are usually available in third-party **native React Native** components.

SQLite: SQLite is a software library that provides a relational database management system with many similarities syntactically to MSSQL. This will be installed with each separate installation and will be used to persist data between application start ups.

Redux: Redux is an open-source JavaScript library for managing application state during application operation. It will be used to persist data between screens etc. This will be used in conjunction with SQLite.

NodeJS: Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser. React native has a dependency on NodeJS.

Visual studio code: An integrated development environment (IDE) used to develop the application.

Umbraco: Umbraco is an open-source content management system platform for publishing content on the World Wide Web and intranets. Currently Housing Solutions web portal is running version 7.15.4 assembly: 1.0.7381.11453.

Umbraco API: An application programming interface built into Umbraco to expose Umbraco specific services such as portal authentication. This API will be exposed to the installed version of the mobile application so that we can utilise current portal membership.

Testflight - Apples testing environment for new applications

Edison - Prodo's AI voice recognition software

1.4 Target devices & viewports

Due to the enormous amount of mobile devices on the market, we will design the user interface to target a specific set of viewports as opposed to specifying the device models the application is compatible with (**NB** this list below may increase during the build if a particular viewport is found not to be catered for). The application interface will be responsive to the viewports outlined below and will universally cover both IOS and Android devices. The table also highlights a selection of devices that fall into the viewport range, building the app user interface to cover this range will maximise device compatibility.

Height	Width	Devices
926	414	iPhone 12 Pro Max
919	412	OnePlus 9 Pro

896	414	iPhone 11 Pro Max, iPhone XR
883	412	Samsung Galaxy Note20 Ultra
892	412	Samsung Galaxy A52
869	412	Samsung Galaxy Note 10+, Google Pixel 4 XL, Samsung Galaxy Note 10, Galaxy A40
853	480	Samsung Galaxy Note 5,
854	384	Samsung S21 Ultra Samsung S21 Plus , Samsung S20+
851	393	Google Pixel 4a
846	414	iPhone XS Max, iPhone 11
846	412	Samsung Galaxy Note 9, Google Pixel 3 XL
844	390	iPhone 12, iPhone 12 Pro
830	393	Google Pixel 4
823	412	Google Pixel 2 XL
812	375	iPhone XS, iPhone X
808	393	Google Pixel 3a
786	393	Google Pixel 3
780	360	iPhone 12 Mini
740	360	Samsung Galaxy S9+, Samsung Galaxy S9, Samsung Galaxy S8+
736	414	iPhone 8, 7, 6s Plus, Nexus 5X, Google Pixel, One Plus 3
732	412	Google Pixel XL
731	411	Nexus 6P
667	375	IPhone 6, 6s, 7, 8
640	360	LG G5, Samsung Galaxy S7 Edge, Samsung Galaxy S7
598	320	iPhone SE, iPhone 5s, 5c, 5

1.5 Operating system versions

The application will be developed and released for both IOS and Android and will be compatible for the following operating system versions. Please note that this list includes operating systems that are available at the time of development, future versions of these operating systems are not included with this build.

Operating System	Version Support	Current Version (05/21)
IOS	11 or newer	14.6
Android	5.0 (API 21) or newer	11.0

1.6 Testing

There are over 1500 models of Android phones in use today and around 24 different iPhone models so how do we write test plans to cover the majority of these devices? Obviously it would be impossible to physically test on all of these models! In order to achieve a larger compatibility window we will use the data outlined in the previous sections, 1.4 and 1.5.

Through a combination of the large cross section of viewport dimensions (1.4) and the stipulated compatible operating systems (1.5) we will perform user functionality tests on simulators/Emulators and physical devices that fall within the our outlined criteria. Android emulation will be done using Android Studio and IOS simulation will be done using XCode. Although it would be nearly impossible to test the application on every single device type, we aim to get a high compatibility percentage and will ensure that it operates as expected on the most popular current smartphones.

Top 8 device models as of May 2021

- Apple iPhone 12.
- Samsung Galaxy S21 Ultra.
- Samsung Galaxy S21 / S21 Plus.
- Apple iPhone 12 Pro Max.
- Apple iPhone 12 mini.
- Samsung Galaxy Note 20 Ultra.
- OnePlus 9.
- Samsung Galaxy A52

The way applications are user tested differs between Apple and Google and both require us to set up specific testing pipelines that conform to the guidelines of each company.

User testing on IOS

The process outlined by Apple is quite strict and will require us to set up the following items in order to roll the app out to select users for testing.

- A valid IOS developers licence

- An app store connect account which includes TestFlight
- App delivery mechanism - IOS Transporter

Tester user requirements:

- A valid Apple store account
- An iPhone
- TestFlight app (free app found on the app store)

Test users are added to the test group in TestFlight, when the app is marked as 'ready for testing' each test user receives an email with a link inviting them to the test group, after accepting the T&C's they will then be able to install the app via TestFlight. The app will remain installed and fully functional for 90 days after installation. Please note an application in test does not require Apple approval, however this WILL be a requirement to be launched to the live store.

User testing on Android

The pre release user testing process from Google is very similar to the one from Apple with a few minor differences. No developer licence is required and no specific technology is required to deploy the test application to the Google Play Store. The following is required:

- A Google play developer account to be created

Tester user requirements:

- No special software is required

Test users are added to the test group in the developers console, when the app is marked as 'ready for testing' each test user receives an email with a link inviting them to the test group, after accepting the T&C's they will then be able to install the app via the Google Play Store. The app will remain installed and fully functional until it is published to the live store or until the test version is pulled from the developer console. Please note an application in test does not require Google approval, however this WILL be a requirement to be launched to the live store.

1.7 Accessibility

TBC - requires designs to complete

2. App Stores & Deployment

Both app stores (Apple & Google) follow a similar process to get your app initially approved before it is publicly available. As well as the app passing an internal security check and code test we are all required to provide all of the application listing information, screenshots and data. It is not possible to submit an application for approval without providing this information.

After the app is approved we can then release the application to the stores and it will be available publicly for download. After this initial release there will be a requirement for updates, patches and new feature functionality to be deployed. If a new piece of functionality contains a lot of new code it may require approval again from the store so it will need to be taken into account as the approval process can take anything from a few hours to a few days.

Users who already have the application installed on their phone will not be required to re-install it after every new release the app stores take care of this. After a new application version is published to the store it will automatically update on all target devices before the app is used again. Please note that the user will be required to log back into the app after it has updated.

3. APIs

APIs will be required on various screens/sections of the app to request or send data to Housing Solutions internal 3rd party systems. These systems manage tenant information, payments and repairs scheduling.

Housing Solutions already have an existing web based portal was built by Prodo the code base for this project already contains API integrations for the following systems:

- ❖ **Capita Open Housing** - For tenant information and message centre
- ❖ **Capita payment portal** - For rent payments
- ❖ **Civica Servitor** - Handles all requests for raising and managing repair schedules
- ❖ **Documotive** - Document management (**NB not required for this version of the app**)
- ❖ **Microsoft Cognitive Services** - Image and speech recognition
- ❖ **Umbraco Members API** - Registration, authentication and forgotten password
- ❖ **Umbraco Content API** - News & Media feed

We will establish a machine to machine connection between the app and the above APIs, this will be achieved through the use of a bearer token and will happen automatically when a user logs into the app.

The above APIs will then surface tenant, payment and repairs data where required, this will be explored in more detail on a screen by screen basis in section 4 of this document, screens & Features.

The Umbraco members API will also be utilized by the app, this will handle all of the authentication features that the app will require specifically the following processes:

- Logging into the app
- Registry and sign up if they're not already a portal user
- Password reset



Please note that in order to access the APIs via the app the current portal code base will require modification to expose the APIs securely over HTTPS.

4. Screens & Features

The following section will give a functional explanation of the features and operations of each screen. **Please note** that this may change depending on the output of the designs.

4.1 Core elements

4.2 Login / Forgotten password



Dependency Description	
	An internet connection either by WiFi or mobile data is required for authentication. Login will not be possible without it
	The Umbraco member service API is used to authenticate the user

Screen functionality overview

Components

Description

4.3 Sign up / Register

Dependency Description	
	An internet connection either by WiFi or mobile data is required for authentication. Registry will not be possible without it
	The Umbraco member service API is used to authenticate the user

Screen functionality overview

Components

Description



4.4 Splash / Load Screen

Screen functionality overview

Components

Description

4.5 Home Screen (dashboard)



Dependency Description	
	An internet connection either by WiFi or mobile data is required for authentication. TBC
	Although the majority of data on this screen will come from the SQLite database, some data will be powered by API's. This is TBC <ul style="list-style-type: none">• Capita open housing• Umbraco content service

Screen functionality overview

Components

Description

4.6 News & Media feed

Dependency Description	
	An internet connection either by WiFi or mobile data is required for authentication. TBC
	Although the majority of data on this screen will come from the SQLite database, some data will be powered by API's. This is TBC <ul style="list-style-type: none">• Umbraco content service



Screen functionality overview

Components



Description

4.7 Rent Transactions (Payment history)

Dependency Description

	An internet connection either by WiFi or mobile data is required for authentication. TBC
	Historic transactions will be sorted in the local database however it will be necessary to make calls to the below API to look for the latest transactions. This is TBC <ul style="list-style-type: none"> • Capita open housing

4.8 Make a Payment

Dependency Description	
	An internet connection either by WiFi or mobile data is required for authentication. TBC
	<ul style="list-style-type: none"> • TBC

Screen functionality overview

Components

Description

4.9 Repairs Diagnosis

4.9.1 AI Image diagnosis

- TBC

4.9.2 Edison (AI speech recognition)

- TBC

4.9.3 Traditional diagnosis method

4.10 Scheduling a Repair

4.11 Forms

4.11.1 Feedback

The feedback back form is hosted on a portal web page and displayed in a dedicated screen using a webview control. Submitting the form sends the tenant reference in the query string

4.12 Web Chat Screen

4.13 Message Centre Screen

5. SQLite Database

In order to persist data between application shutdowns (app thread closure on the device) we will use a SQLite database to store data long term. The tables in this database can be read from and written to, we'll go through this database structure in the next section of this document. Please note that this data will be purged from the device if the application is uninstalled and certain fields will also be encrypted to prevent database tampering, these fields will be highlighted.

5.1 Table structure

The following lists outline the tables that will be present in the SQLite database, each list will stipulate the fields contained in the database table, the field data type and whether or not the field is encrypted. Please note that we will not need to store all of the tenant data that the web portal contains only the data that the application requires to operate. Reasons for this are:

- So that the application can operate to some degree offline
- To reduce the amount of API calls/repeat API calls (and potential mobile data consumption) that application needs to make
- To store non sensitive application related data

TBC - Data outlined below may be subject to alteration but must be defined and agreed on before build start.

5.1.1 System settings

The settings table contains application specific information such as temporary authentication tokens and accessibility preferences.

Field name	Data type	Is encrypted
id	integer	no
token	text	yes
accessibilitySettings	text (json blob)	no

5.1.2 Tenant

The account table contains application specific information about the tenancy, the tenant and any other household members. A significant portion of the fields in this table are encrypted due to the sensitive nature of the information.

Field name	Data type	Is encrypted
id	integer	no
name	text	yes
dob	text	yes
email	text	yes
addressLine1	text	yes
addressLine2	text	yes
addressLine3	text	yes
postcode	text	yes
tenancyReference	text	yes
agreementStartDate	integer(older than 01/01/1970?)	no
agreementStartDate	integer	no
householdMembers	text	yes
mobileTelno	text	yes
homeTelno	text	yes

workTelno	text	yes
estateInspectionOfficers	text	no
generalContacts	text	yes
dateCreated	integer	no
dateUpdated	integer	no

5.1.3 Transactions

The transaction table contains rent payment records, the data here is not directly linked to a tenant so encryption is not required. This table is used to retrieve and display historic rent payments locally on the device rather than making external API calls each time the data is requested. If the user requests older payment records that are not stored locally on the application then an API request will be required to retrieve this data.

Field name	Data type	Is encrypted
id	integer	no
summary	text	no
amount	integer	no
transactionDate	integer	no
currentBalance	integer	no
dateCreated	integer	no

5.1.4 Repairs history

The repairs history table stores history repairs for the tenant. If the repairs are not found locally then an API call will be made to retrieve the data from Capita. None of the data in this table is GDPR sensitive so field encryption is not required.

Field name	Data type	Is encrypted
id	integer	no
repairId (sor code)	text	no
title	text	no
statusType	integer (enum value)	no

repairType (personal / communal)	integer (enum value)	no
dateRaised	integer	no
targetCompletionDate	integer	no
dateUpdated	integer	no
dateClosed	integer	no

5.1.5 Message centre

The message centre table stores the closed message communications that are powered by the Capita API. These conversations are stored locally for fast historic retrieval. All current active conversations are not stored here until they are closed. Some of the fields that may contain sensitive information will be encrypted for security purposes, specifically the description and notes fields.

Field name	Data type	Is encrypted
id	integer	no
reference	text	no
type	text	no
completionDate	integer	no
dateCreated	integer	no
description	text	yes
notes	Text (json blob)	yes

5.2 Field encryption

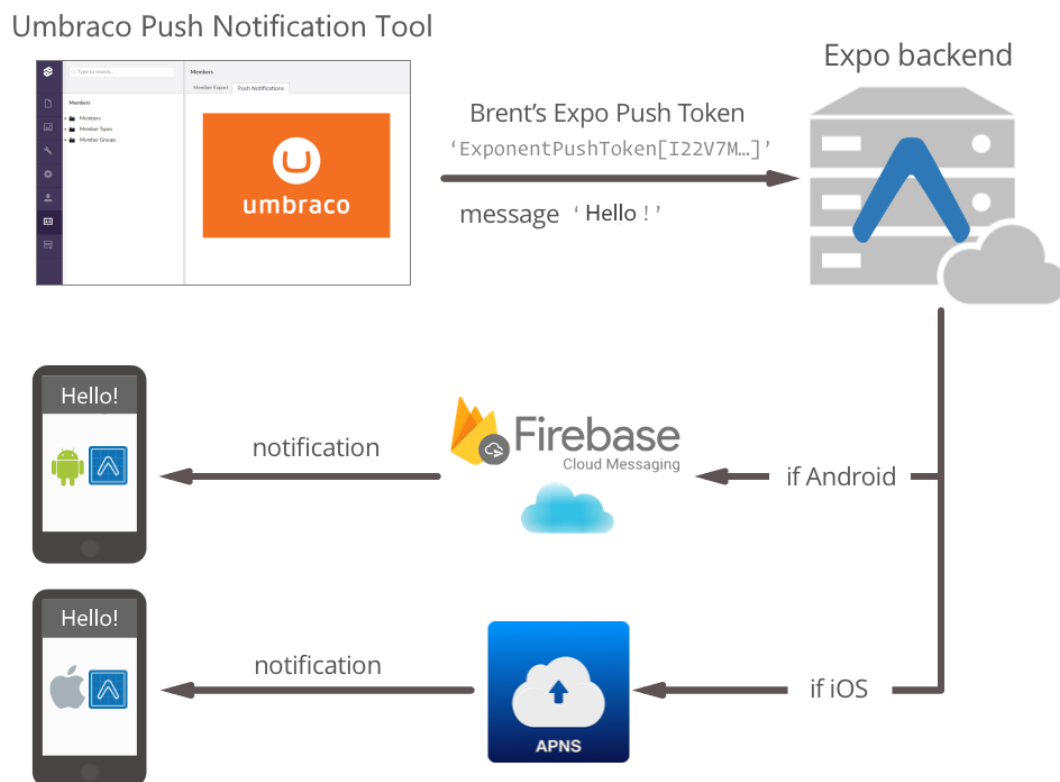
All fields that have been marked as sensitive will be encrypted in the SQLite database to ensure data security. We will use an AES 256 encryption.

6. Wifi / Internet Connection

Regular checks will be done on screens that require an internet connection, if no connection is found a popup will appear forming the user that they must connect to the internet to fully utilise the functionality of that screen.

7. Push Notifications

The following section will give a functional explanation of how we will achieve push notifications for the application. IOS and Android provide their own services to handle push notifications, APNs (Apple Push Notification service) and FCM (Firebase Cloud Messaging). React Native Expo does provide a tool that wraps the integration of both providers into one single API that we can target however we still need to write the code to integrate with the Expo push notification API. We also need to write a custom Umbraco interface which allows a CMS user to target and send push notifications to either groups of tenants or individuals. Below is a diagram which illustrates the push notification workflow.



7.1 Push notification workflow

1. A user with the correct level of permissions logs into the umbraco CMS and navigates to the push notification tab in the members section.
2. The user selects the type of push notification (**NB** types listed in the next section) they wish to send from a drop down list.
3. The user can then select who they wish to send the push notification to, an individual or a group.(NB groups are listed in the next section)
4. The user then adds a title for the push notification, this can be up to 64 characters in length.
5. User then adds the message body text, this can be up to 240 characters in length
6. Finally the user presses the 'Send' button, a confirmation alert pops up to confirm that they're happy with the message, user confirms and the push notifications are sent.
7. If the user declines the confirmation the push notifications are not sent.

Upon clicking send a JSON object containing the data of the push notification campaign will be created and saved to a location (location **TBC**) to act as a send receipt. Please note that custom push notifications can not be stored in Umbraco for later use. If the user navigates away from the push notifications tab in Umbraco before clicking send the message will be lost.


7.2 Umbraco push notification tool

The Umbraco push notification tool is a custom built page in the members section of the CMS. The look and feel of it will remain consistent with Umbraco so no additional design time will be required. The interface essentially is a multi-step form which allow a CMS user to do the following:

- Select the type of push notification, payment alert, event, custom etc
- Select a distribution list, with a group or individual(s)
- Create a title (up to 64 characters)
- Create the message body (up to 240 character)

Below is a wireframe illustrating the layout of the push notification tool

Push Notification Tool

Select push notification type 

Select your recipients...

Enter a title (64 characters max)

Type your message (240 characters max)

Recipients
recipient 1,
recipient 2,
...

Send

7.3 Push notification types

The following list is to be confirmed (TBC) and the push notification type listed below may change.

- **Custom notification** - a fully customisable push notification. The title and content body are set by the CMS user and the distribution list can be anything from a single individual, custom list of individuals or a predefined group. This push notification type is manually triggered.
- **Repair reminder** - A pre-set notification with a fixed message and title and a variable date which notifies the recipient of a scheduled repair engineer visit. This notification is triggered automatically. Automatic notifications are to be triggered from a predefined list that we can poll on a regular basis. The list and trigger frequency are to be confirmed (TBC).
- **Communal repair** - A notification which alerts all recipients that belong to a particular distribution group that a neighbour has already raised a communal repair. This notification is triggered manually and achieved by creating a custom push

notification and sending it to a specific distribution group. This may require further scoping (**TBC**).

- **Operative enroute** - A pre-set notification with a fixed message which notifies the recipient that an engineer is on the way. This notification would be automatically triggered. The mechanism to achieve this notification type requires further discussion (**TBC**).
- **Rent due** - A pre-set notification with a fixed message and title which notifies the recipient that their rent payment is due. This notification is triggered automatically. Automatic notifications are to be triggered from a predefined list that we can poll on a regular basis. The list and trigger frequency are to be confirmed (**TBC**).
- **Payment received** - A pre-set notification with a fixed message and title which notifies the recipient that a payment has been received. This notification is triggered automatically. Automatic notifications are to be triggered from a predefined list that we can poll on a regular basis. The list and trigger frequency are to be confirmed (**TBC**).

7.4 Member groups & distribution lists

In the previous section (7.1) we outlined the push notification workflow, step 3 of this workflow outlines how a CMS user can select an individual or distribution group to send the push notification to. Please note that in order to achieve distribution groups we will need to add some additional information to the member objects in Umbraco which allow us to group individuals, this could be by area or building but is **TBC**.

The recipient selection area allows a CMS user to do the following:

- Select one or more tenants and create a custom list of recipients
- Select a distribution group, this could be by area or building. This will require adding a group tenancies code or estate code to the Umbraco member objects so that we can programmatically select tenants that fall into a particular category

7.5 Event triggers

System driven push notifications will require custom event triggers, this could be us polling a predefined list of tenants or some other mechanism. Further discussion is required to figure out what these triggers are and the mechanism to poll them. This is **TBC**.

8. Payment Gateway

This is **TBC**

9. Message Centre

The message center is a current feature on the web portal, it allows a staff member to instigate a two way conversation with a tenant. This functionality is powered by an API which is provided by Capita. There are two main functions that the API provides, one to initialise the conversation and one to update the thread until it is resolved. Complete conversions are archived below active ones so that tenants are able to revisit the outcomes of previous message threads.

The application version of the message centre will operate in much the same way and will also be reliant on the same Capita API calls that the web portal uses.

9.1 Active threads

All active threads are directly retrieved from the API endpoint; each action will require an API call to fetch or send data to the thread. If a message thread is started but not completed in the initial sitting, the partial message thread will be loaded in from the API when the tenant resumes the conversation.

9.2 Archived threads

After a message thread conversation is completed an API call will be triggered to mark the message as resolved and close the conversation. At this point, the entire message thread will be serialised into a JSON object and saved to the SQLite database 'message centre' table. A tenant can then always revisit the conversation to review the outcome of an action without the need for that conversation to be retrieved from Capita API. This method provides convenient offline storage for all history message threads.

10. Web Chat

The web chat (chatbot) will be hosted on the main website in a dedicated responsive webpage. The web chat page will then be rendered in a react native webview control which is set to fill the screen size (*see 1.4 Target devices and viewports*), the responsive web page will then naturally adjust to the screen view port. Please note that all web chat functionality is handled entirely in the web page and not the application screen.