

Mobile Application : Design Manual

Content

1. Technology Stack
2. Features of the application
3. Screen Stack of the application
4. Entry point to the program execution
5. Navigation Functional Component
6. Main functional component
7. Login screen implementation
8. Main screen implementation
9. Active event screen implementation
10. Active interactive screen implementation
11. Close active event screen implementation
12. Previous event screen implementation
13. Limitations and additional notes

1. Technology Stack

- The main application was done using react native framework.
- Expo was used which is a bundle of tools created around React Native to help you start an app very fast.
- Below are some of the technologies related to the mobile app , with its purpose.

Technology	Purpose
React Native	Main code base of the application.
Expo	React Native Toolset for faster deployment.
Expo-av	Recording and Playing Sounds.
Expo-font	Used as the native font set in the application.

Firebase	Acts as the cloud server of the mobile application. This is where the app downloads its data and uploads data. Also the interaction with the outsider happens through this.
Redux-thunk	Act as a centralised database system of the codebase.
Firebase module	Used to connect the application with the firebase cloud servers. Real time communication with the outsider is possible , thanks to this module.
VS Code	Developing Environment.

- The application is upgraded to the latest version of expo , which is SDK 47.0.
- Also android studio was used to emulate the application with the help of Expo servers.
- One of the main features of the mobile app is voice recording and sending it to the server. But this is not doable with the virtual emulator , therefore an actual device was used for developing this feature.

2. Features of the Application

- The features of the application were previously mentioned as well. But for the sake of explaining the implementations, it is worth mentioning them again.

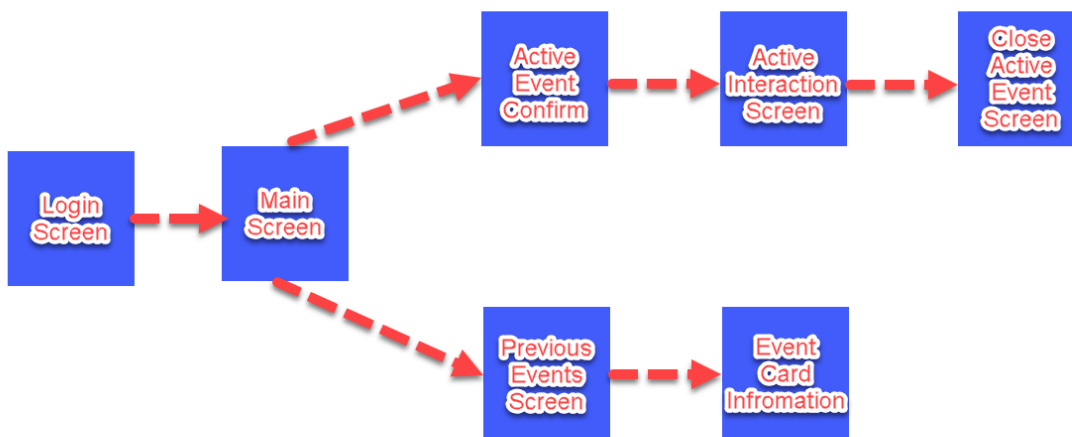
1. Log in to the user account
2. Check if there is an active event : Active event refers to an event of interaction between the outsider and the homeowners.
3. If there is an active event currently, it can be accepted or declined.
4. If declined, the homeowners have the privilege to consider accepting it again, as long as the outside person is still triggering the hardware node.
5. If accepted, the user can communicate with the outside user using the intercommunication technology. This is half duplex communication. Both the users at each end, need to record their voice and send it to the otherside. These recorded voices are used for record keeping.

6. Outsiders can request for opening up the mailbox for placing the delivery package. Then the user can open the mailbox through the app.
7. When the outsider closes the current event, the user can save the event with all the information related to the event.
8. The user can review the information related to the previous events.

- All the technical and design details related to this , will be explained in the upcoming chapters.

3. Screen Stack of the Application

- There are few main screens in the mobile application.



- *Important : All the design, technology and implementation related things are written in-terms of each screen.*

4. Entry Point to the program execution

- Entry point to any program is App.js script which resides in the root directory of a react native project.

- In this application, the App.js is responsible for the below features.

1. Setting the default fonts of the application.
 - This is done using loadAsync api for Fonts in react native.
 - Open-sans and open-sans-bold are used in the application.

2. Setting up the centralised database for the application.
 - Redux thunk centralised database is initialised here.
 - Uses applyMiddleware API for this.
3. Initializing firebase application.
 - The Firebase module needs a few information such as apiKey of the cloud server, authDomain, databaseURL and few other parameters.
 - All of these are stored as an object in .js script and from that script that object is used for initialization of the firebase module.
4. Loading fonts.
5. Returning the <Navigation> component. This component is used for handling the navigation system of the application.

5. Navigation Functional Component

- Navigation system of the application are designed and implemented from scratch without using any already available modules.
- Maps three main components.
 - Starts the navigation from the login screen.
 - Then it auto navigates the screens to the welcome screen / main screen , in case of a successful user login.
 - Then from the main screen, it will navigate the screen to either Active event Navigation System or Previous event Navigation System.
- When navigating from the login screen to the welcome screen, the name of the user and the photo of the user is passed on to the main screen , via props.
- In case of login out from the main screen, the app redirects the user back to the login screen.
- Then each screen or navigation system is handled by using props to select the correct screen to render to the display using boolean values.

- Active event navigation system and previous event navigation systems handle the navigation just like it is described above. These components basically handle the navigation in the active event screen stack and previous events screen stack.

6. Main Functional Components

- These are the main functional components of the system.
- These functional components help in code reusability as it greatly helps to reduce repetition. These components are mainly implemented for occasions where there are few situations they can be used.

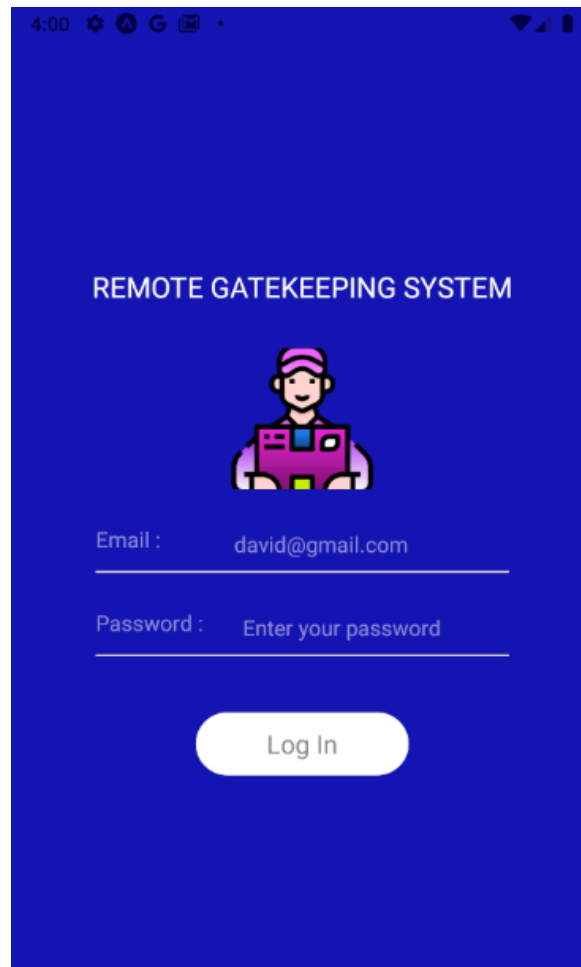
Component Name	Purpose	Implementation Details
AudioPlayButton	Handling the playing of a sound when the url of the sound source is given. This component is mainly used in the active interactive screen where the interaction between the user and the outsider happens. There, when the outsider updates the server with the url of the voice source, this button fetches the corresponding source from that url and plays in the mobile phone.	<ul style="list-style-type: none">- Uses an expo-av module for playing the sound.- URI of the sound source is passed to the component through props.- Uses this URI to create a sound using createAsync() API.- Uses playAsync() API for playing the sound.
BackButton	For implementing a basic navigation controlling back button where use is redirected back to the screen stack by popping out the current screen from the stack.	<ul style="list-style-type: none">- Implemented using <TouchableOpacity> component in react native.- Uses InDesign vector icons for enhancing the visual guide to the user.- Uses the onPress function for navigating back to the previous

		<p>screen.</p> <ul style="list-style-type: none"> - onPress function is passed through props.
CustomButton	For creating a button of a certain style. Used for enhancing the user interface.	<ul style="list-style-type: none"> - Implemented using <TouchableOpacity> component in react native. - Text for the button is passed through the props.
CustomButton2	For creating a button of a certain style. Used for enhancing the user interface.	<ul style="list-style-type: none"> - Implemented using <TouchableOpacity> component in react native. - Text for the button is passed through the props.
CustomButton3	For creating a button of a certain style. Used for enhancing the user interface.	<ul style="list-style-type: none"> - Implemented using <TouchableOpacity> component in react native. - Text for the button is passed through the props.
EventCard	For showing the details of an event on a small card. In the previous events screen, a set of event cards are placed as a list.	<ul style="list-style-type: none"> - Implemented using <TouchableOpacity> component in react native. - Photo of the outside person, name of the event, description of the event is passed as props. - onPress attribute will set the functionality to redirect the user into the EventDetailsScreen.
EventDetailsCard	For showing all the relevant information with respect to an event such as name of the event, date, rating, description,	<ul style="list-style-type: none"> - Implemented using few <View> and <Text> components. - All the details related to the event are passed

	picture of the outsider and whether mailbox is opened in the event or not.	using props.
Header	This is the default header of the screen. The header is useful for improving the user experience. Title of the screen is written on the header.	<ul style="list-style-type: none"> - Name of the header is passed through props. - Also the back button is used for the header for navigation control.
LogOutButton	For users to log out from the system.	<ul style="list-style-type: none"> - Implemented using <TouchableOpacity> component in react native. - Navigation is controlled through a function passed through props.
RatingStars	An array of 5 stars. This is used to rate the event.	<ul style="list-style-type: none"> - Each element in the array consists of an icon which is a star icon from AntDesign. - Size of the icon is passed through props.
RatingStarsInteractive	Makes the star array dynamic. Due to the dynamic behaviour, user experience is also increased. Users can select the rating by selecting a number of stars.	<ul style="list-style-type: none"> - Implemented using <TouchableOpacity> component in react native. - When a certain number of stars are pressed, those stars are colored accordingly for the user to identify the rating.

7. Login Screen Implementation

Screen Layout :



- First of all the screen checks whether to render the screen or not before rendering.
- If the authentication token obtained from the cloud is not expired, the app go straight to the main screen rather than login screen.
- If the authentication token is expired, the user needs to login again and get a new authentication token before login in.
- For getting the authentication token, useDispatch() hook was used.
- Also all the necessary error handling techniques are built into the system.
- Also the loading wheel is implemented in case network error happens or any other network related issue arises.

- Also `authAction` is implemented on another script so that it is easier to handle the authentication part.
- The whole login screen is wrapped around the `<KeyboardAvoidingView>` component so that , whenever the keyboard pops out, the screen auto adjusts itself to create the space for the keyboard. This will greatly improve the user experience.

8. Main Screen Implementation

Screen Layout :



Welcome David!

Active Event

Previous Events

Log Out

- There are 3 main options available for this screen. Those are direct the user to the active event interface, previous events interface or to log out the user.
- For improving the user experience , the profile picture of the user is displayed at the top of the screen and also the name of the user is also

displayed. For that `useEffect()` hook is used in the implementation. There, the url responsible for users' profile picture and the name is checked.

- Also `Asyncstorage` is used for the account information storing.

- Main components are `<ImageBackground>` components, and `<CustomButton2>` and `<LogOutButton>` are also used.

9. Active Event Screen Implementation

Screen Layout :



- Here the user has the privilege to either cancel or accept the order.

- Here also `useDispatch` was used to get inputs from the react-redux centralised database.

- Here the app first checks the realtime database of the firebase and gets the latest entry through its rest API. If the last message has a message type of image, then the application knows that the node module has uploaded a photo of the outsider to the cloud database.
- Then it asks the user whether the current event is accepted or declined.
- If the user declines it, then that means the user is not ready for a current active event. However the user has the privilege to come back to it and accept it anytime if the outsider is still operating at the node hardware module.
- If this request is accepted by the user, then the user will be redirected to another screen called active interactive screen.

10. Active Interactive Screen Implementation

Screen Layout :



- This is the most important and also the most complex implementation of the whole application.

- Here the implementation related to the real time communication is implemented.

- The protocol related to the real time communication implemented here. There are about 10 hooks to handle the state of the system.

- Design Protocol

- If the last message of the database has a message type of AUDIO and the status of the last message is NONE , then the application determines that there is an unlisted voice that has been sent to the database and it fetches that from the url and plays it.

- If the message has a message type of AUDIO and the status of the last message is LISTENING, then the application determines that the user needs to record a voice and send that back to the database. For that, it will make the interface for it.

- If the message has a message type of AUDIO and the status of the last message is RECORDING then the application sends the voice to the database. At the same time , it updates the message URL so that hardware node can download that from the database and plays it.

- If the last message has a message type of MAIL_BOX_ACCESS , then the application determines that the outsider needs to open the mailbox so the cloud will send the signal to open the mailbox at the other end.

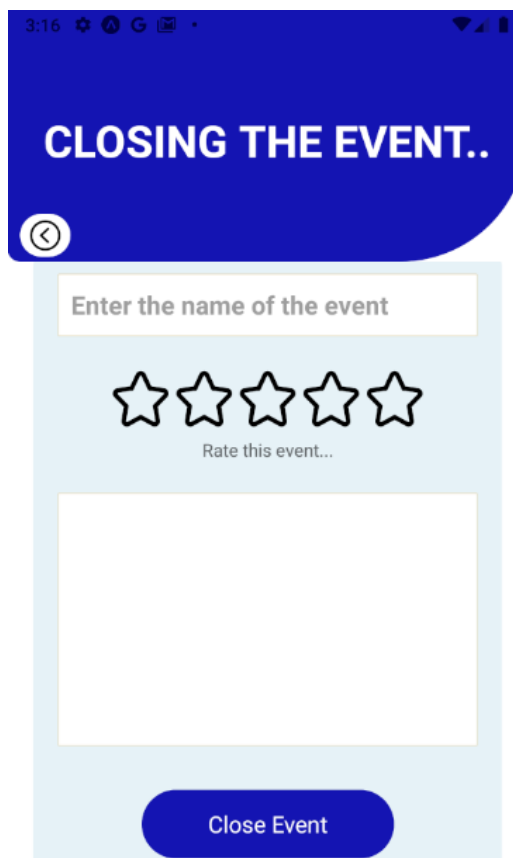
- If the last message has a message type of CLOSED , then the mobile app determines that the outsider has to open the mailbox at the other end , and then make the UI so that the user can close the current event after submitting all the information related to the current event.

- All of these are done after listening to the database instance at the firebase and whenever the status of the database changes, the application triggers different component behaviour in the mobile app UI.

- This realtime behaviour was designed using the firebase module.
- All of these states are handled using the useState() hook.
- Initial configuration of this process is handled using useEffect() React hook.
- When the audio is recorded for the first time, the permission for the user is granted using the requestPermission Async() API.
- Audio recording and sound playing is done using the expo-av module and using the Audio.Recording.createAsync() API then the recording is stopped using stopAndUnloadAsync() API.
- AudioPlayButton and CustomButton3 are used as functional components. Additional ICON packages like AntDesign and fonts like FontAwesome are used for enhancing the user experience in the UI.

11. Close Active Event Screen Implementation

Screen Layout :

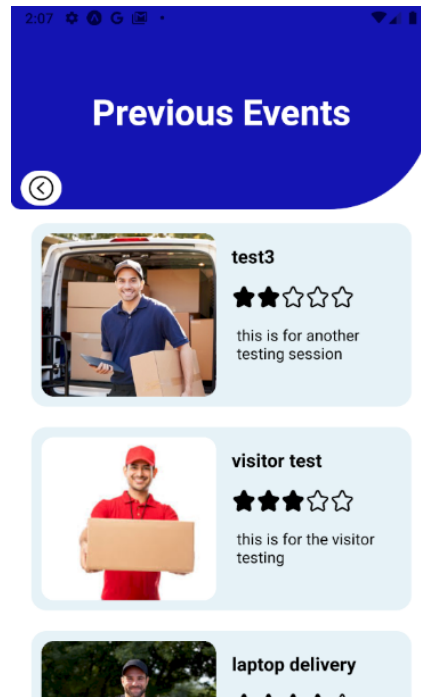


The screenshot shows a mobile application screen titled "CLOSING THE EVENT..". At the top left of the screen is a back arrow icon. Below the title is a text input field with the placeholder text "Enter the name of the event". Underneath the input field is a rating system consisting of five empty star icons, with the text "Rate this event..." centered below them. A large, empty rectangular text area is positioned below the stars. At the bottom of the screen is a blue rounded rectangular button with the text "Close Event".

- Here all the parameters related to the event are filled by the user. Main parameters are name of the event, description of the event and rating of the event.
- This new event is then inserted into the database using the `useDispatch()` hook of the react native.
- `useSelector` is used to select the correct event before dispatching the data to the database.
- Also error handling is implemented too. For example, the length of the name and the description is checked to see if those fields are filled.
- Here, for inserting details to the form, we use a `<KeyboardAvoidingView>` component. Also `<ScrollView>` is used so that users can have a window with the scrolling enabled.
- Here for getting the user inputs, `<TextInput>` component is used. Here some placeholders like, "Enter the name of the event" and others can be put in order to improve the user experience. Also autofocus on the input field is enabled.
- `<CustomButton2>` is used to close the event and submit the data to the database.
- For rating the event, `<RatingStarsInteractive>` custom component is used. Here also the user experience is improved since the user is given the option to choose the rating graphically.

12. Previous Events Screen Implementation

Screen Layout :



- Here the user is able to review the previous events that have happened.
- Some useState hooks are maintained in order to maintain the state of the system. Those are isLoading which is responsible for loading the screen, isRefreshing, which is responsible for refreshing of the screen , error which is responsible for handling errors.
- useSelector() hook is used getting each event from the previous events redux database.
- useDispatch() hook is used to get data from the database.
- Then the useCallback() hook is used to load the data on the screen as a list.
- useEffect() is used to initialise the loading to the screen.
- Here instead of data getting real time using the firebase module, the REST API of the firebase is called. This is a good method for this use case since the data is not updating real time when the user is reviewing the previous events.
- Then previous events are displayed to the screen using the <EventCard> custom components.

13. Difficulties and Additional Notes

- There were some difficulties faced when installing the firebase module and the redux thunk, due to depreciate errors.
- Also some infinity rendering cycles may occur due to useEffect() hooks. Therefore it is essential to change parameters in the dependency list of the useEffect hook when there is a chance of infinite cycles.
- The apk of the application is made using the expo without uploading to play store or app store because those will charge payments.