**Department of Computing**

**Bachelor of Science (Hons) in Software Development**

# Creche Connect

Design Document

| | |
|---|---|
| Supervisor: | Chris Meudec |
| Student Name: | Michal Gornicki |
| Student Number: | C00265618 |
| Date: | 2022/2023 |

## Abstract

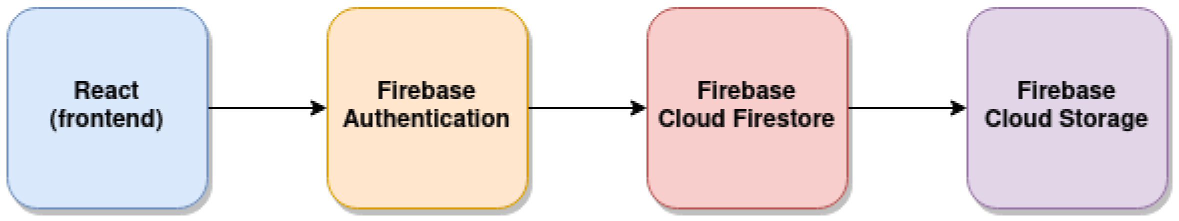
The Creche Connect application aims to create a simple and secure platform for childcare practitioners to track the pupil's progress and communicate effectively with parents/guardians about their child's development.

## Table of contents

## 1. Overview

The Creche Connect application aims to create a simple and secure platform for childcare practitioners to track the pupil's progress and communicate effectively with parents/guardians about their child's development. This design document will explain the system architecture in detail. In the paragraph Data model, the database details will be clarified. Documentation paragraph will explain in detail more important components.

## 2. System architecture



### 2.1. React

This JavaScript library builds the application's front end. React is responsible for handling the user interface and interactions, and it makes API calls to Firebase to retrieve and update data.

### 2.2. Firebase Authentication

The application uses Firebase Authentication [1] to create and manage user accounts. When a user registers, Firebase Authentication creates a new user document with their email and password in the Firebase Authentication service. When the user logs in, Firebase Authentication verifies their email and password and returns a unique user ID (UID) that the front end can use to authenticate subsequent requests.

### 2.3. Firebase Cloud Firestore

The application uses Firebase Cloud Firestore [2] to store user data. Firestore is a NoSQL document-based database that efficiently stores, retrieves, and syncs data. When a user registers, the application creates a new user document in the Firestore, with their uid, name, email and picture url.

### 2.4. Firebase Cloud Storage

The application uses Firebase Cloud Storage [3] to store the user's profile pictures. When a user uploads an image, the application uploads the picture to the storage bucket and generates a unique URL, which can then be used to display the picture in the application.

## 3. Data model

### 3.1. Users

A collection that contains documents for each registered user. Each user document would contain the following fields:

- **displayName:** This is a string containing the user's display name
- **email:** This string contains the user's email address
- **photoURL:** This string contains the URL of the user's profile picture
- **searchArray:** This is an array that contains strings that are used for searching the user's name in the application. It contains the substrings of the  user's name
- **uid:** This is a string that contains the unique identifier of the user
- **userRole:** This is a string that contains the user's role in the application

The profile picture is a storage bucket containing the profile pictures uploaded by users. Each file would have a unique name based on the user's email and the timestamp when the file was uploaded.

### 3.2. allMessages

The allMessages collection contains documents representing different chat conversations, each with a unique **messageId**. The documents contain an array called messages, representing the messages in the chat conversation. Each message is an object with properties such as:

- **id:** A unique identifier for each message within the conversation.
- **text:** The message content (text).
- **senderId:** The ID of the user who sent the message.
- **date:** A timestamp that represents the date and time the message was sent.
- **img:** A URL to the image file, if the message includes an image.

### 3.3. userMessages

The userMessages collection contains documents for each user, where each document has:

- **uid:** The user's unique ID in Firebase Authentication.

- **lastMessage:** An object representing the last message exchanged between the user and another user. This object has the following fields:
  - **text:** The text of the last message.
  - **date:** The timestamp when the last message was sent.
- **messages:** An array of objects representing the messages exchanged between the user and another user.

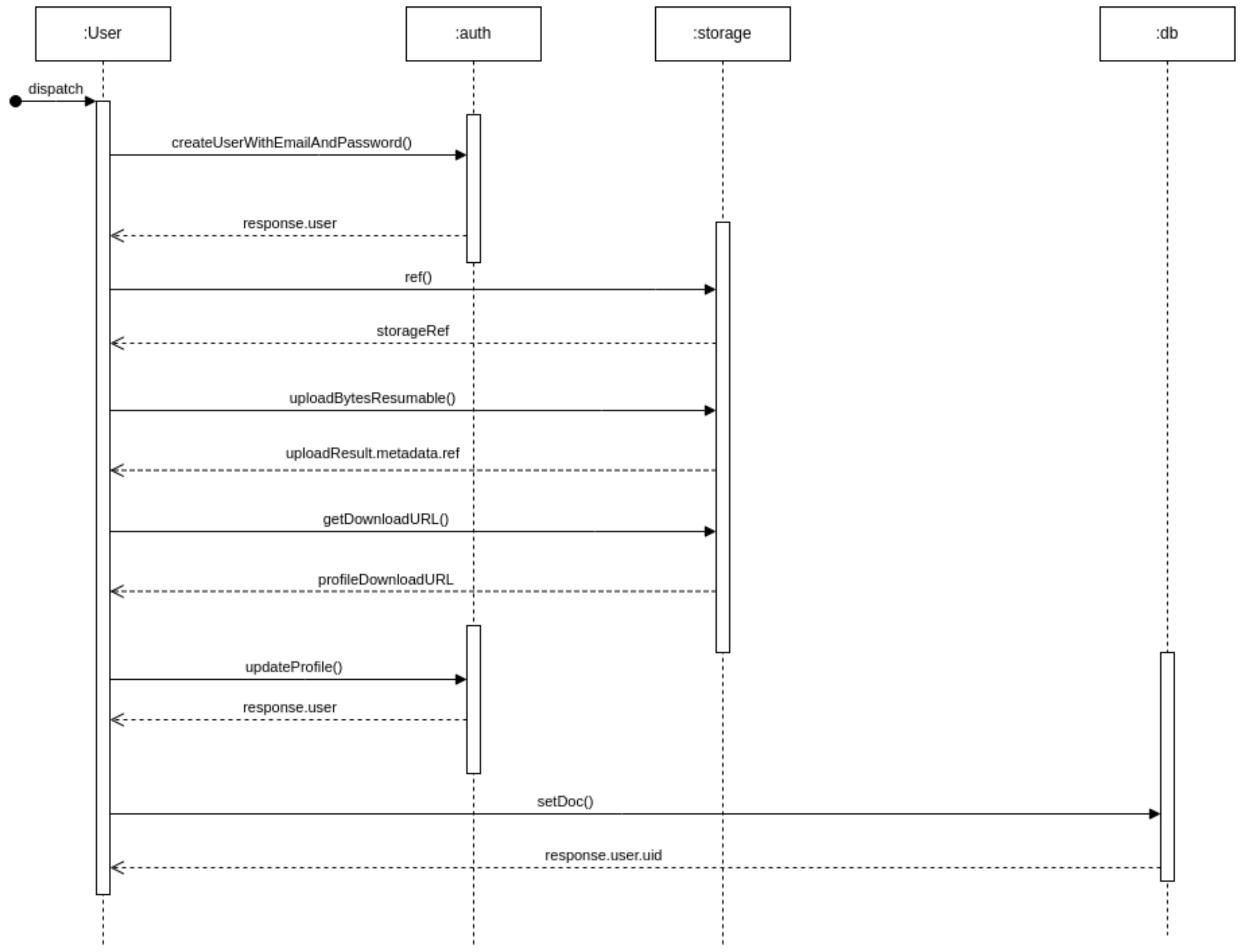
### 3.4. children

The children collection contains information about children in the system:

- **childId:** a unique identifier for the child.
- **dob:** the date of birth of the child.
- **healthInfo:** information related to the child's health, including vaccination status and medicine instructions.
- **id:** same as childId.
- **lowFirstName:** the first name of the child in lowercase.
- **lowLastName:** the last name of the child in lowercase.
- **parentEmail:** the email address of the parent or guardian of the child.
- **parentMobile:** the mobile phone number of the parent or guardian of the child.
- **parentName:** the name of the parent or guardian of the child.
- **searchArray:** an array of strings that enable a fast search of the child's name.
- **additionalInfo:** additional information related to the child, including allergies and sensitivities.
- **dailyReviews:** an array of objects representing daily reviews for the child. Each object has properties such as date, mealTime, meals, nappyStatus, nappyTime, activities, otherComments, updatedBy, and timestamp, which are used to record information related to the child's daily activities.
- **updatedBy:** the name of the user who last updated the child's information.

## 4. Sequence Diagrams

### 4.1. Register User



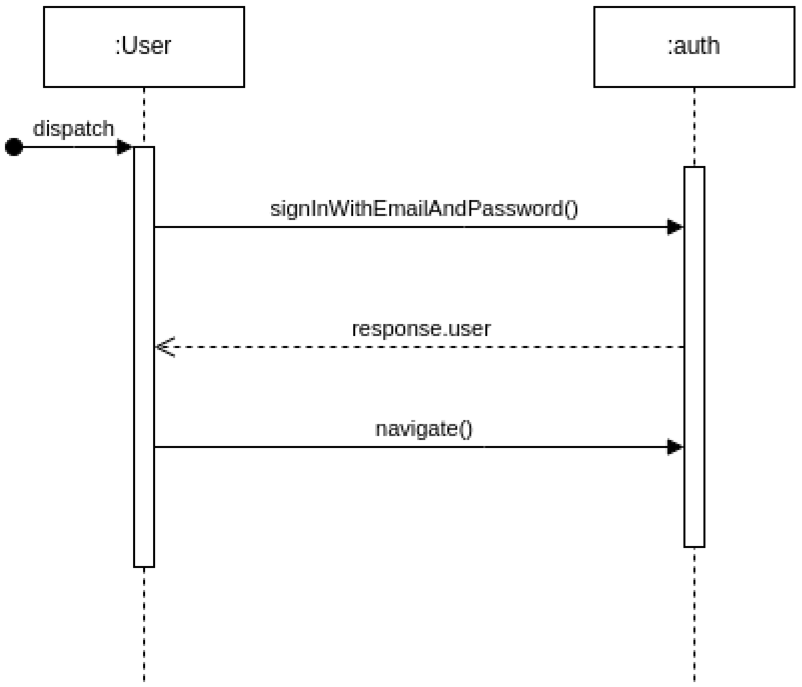
**:auth** - Firebase Authentication creates a new user document with their email and password in the Firebase Authentication service. When the user logs in, Firebase Authentication verifies their email and password and returns a unique user ID (UID) that the front end can use to authenticate subsequent requests.

**:db** - Firebase Realtime Database is a NoSQL cloud-hosted database that stores JSON data and synchronizes changes instantly across all clients.

**:storage** - Firebase Storage is a cloud-based file storage service optimized for storing and serving user-generated content, such as images, videos, and audio files.

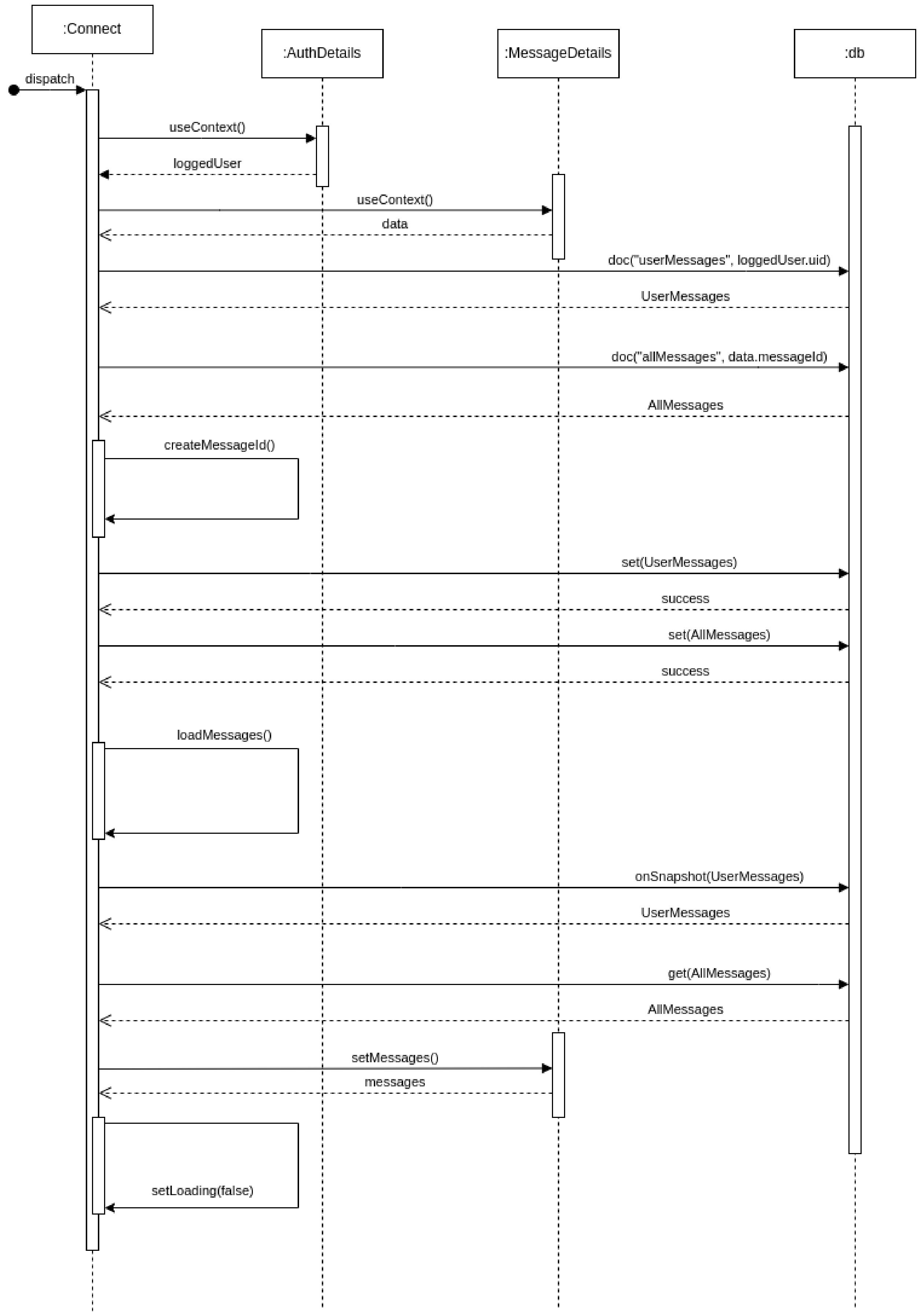
User registers for the application by providing the required information. The ***createUserWithEmailAndPassword()*** method creates a user with an email and password. The result is stored in the ***response.user***. The user's profile picture is uploaded to the storage. The storage reference is created with the ***ref()*** method, the picture is uploaded with the ***uploadBytesResumable()*** method and the ***downloadURL*** for the picture is retrieved with ***getDownloadURL()*** method. The user's profile is updated with ***updateProfile()*** method, and the user is created in the Firestore with ***setDoc()*** method.

### 4.2. Login User



User logs in to the application by providing the required information. The ***signInWithEmailAndPassword()*** method is called to check the user's credentials. The ***navigate()*** method is called to navigate to the home page.

### 4.3. Connect


:Connect
:AuthDetails
:MessageDetails
:db
dispatch
useContext()
loggedUser
useContext()
data
doc("userMessages", loggedUser.uid)
UserMessages
doc("allMessages", data.messageId)
AllMessages
createMessageId()
set(UserMessages)
success
set(AllMessages)
success
loadMessages()
onSnapshot(UserMessages)
UserMessages
get(AllMessages)
AllMessages
setMessages()
messages
setLoading(false)

Component Connect first uses useContext to get the ***loggedUser*** from ***AuthDetails*** and ***data*** from ***MessageDetails***. It then retrieves the user's message document from the database using ***doc("userMessages", loggedUser.uid)*** and sets the ***UserMessages*** variable to the result.

The component then creates a new message ID using the ***createMessageId()*** function and saves the ***UserMessages*** and ***AllMessages*** documents to the database using the **set()** method. Next, it loads the messages for the current chat by subscribing to the ***onSnapshot()*** method of the ***UserMessages*** document and retrieving the ***AllMessages*** document using the ***get()*** method. Finally, it sets the retrieved messages in ***MessageDetails*** using the ***setMessages()*** method and sets loading to ***false***.

## 5. Documentation

### 5.1. Register page

This component is used to create and update user data in Firebase Authentication and Firestore and to upload and download profile pictures from Firebase Cloud Storage.

**Functions used:**

- **createUserWithEmailAndPassword(auth, email, password)**

A function that creates a new user with the provided email and password using Firebase Authentication. This function returns a response object that contains the user's unique ID (uid) and other information about the user, such as their email address and display name.

- **updateProfile(user, profile)**

A function that updates the user's profile information using Firebase Authentication. This function takes a user object and an object containing the new profile information, such as the display name, and updates the user's profile in Firebase Authentication.

- **ref(storage, fileName)**

A function that generates a reference to a file in Firebase Cloud Storage. This function takes the storage object and the file name and returns a reference to the file.

- **uploadBytesResumable(storageRef, file)**

A function that uploads a file to Firebase Cloud Storage. This function refers to the file in storage and the file to be uploaded.

- **getDownloadURL(fileRef)**

A function that generates a download URL for a file in Firebase Cloud Storage. This function takes a reference to the file in storage and returns the download URL.

- **doc(db,collection, document)**

A function that generates a reference to a document in Firebase Cloud Firestore. This function takes the Firestore object and the collection name, and the document id and returns a reference to the document.

- **setDoc(docRef, data)**

A function that creates or updates a document in Firebase Cloud Firestore. This function takes a reference to the document and the data to be stored in the document.

## 5.2. Login page

**Functions used:**

- **handleSubmit**

This function is called when the form is submitted. It prevents the default form submit behaviour, gets the email and password values from the form, and attempts to sign in the user using the signInWithEmailAndPassword function from the Firebase Auth library. If the sign-in is successful, it navigates the user to the homepage; otherwise, it catches the error and logs it to the console.

- **signInWithEmailAndPassword(auth, email, password)**

The Firebase Auth library provides this function. It allows to sign in a user with their email address and password. It takes in 3 arguments, auth object, email, and password. It returns a promise, resolved if the sign-in is successful and rejected if there is an error.

- **useNavigate**

This is a hook provided by the react-router-dom library. It allows programmatically navigating to a different route in the application.

## 5.3. AuthDetails Component

The "AuthInfo" module is a component that provides authentication details to its child components. It creates a context object called "AuthDetails" and sets the "loggedUser" state value based on the current user's authentication status.

**Functions used**:

- **useEffect**

This hook function from the React library performs side effects in a functional component. It takes two arguments: a function to execute after the component has rendered and an optional array of dependencies to control when the function should be re-run. In this code, the useEffect hook is used to listen for changes in the user's authentication status and update the state of the loggedUser variable accordingly.

- **useState**

This hook function from the React library creates a state variable in a functional component. It takes an initial value as an argument and returns an array with two elements: the current state value and a function to update the state value. In this code, the useState hook is used to create a loggedUser state variable and its associated updater function.

- **createContext**

This is a function from the React library that creates a new context object, which can be used to share state or other data between components in a React application. In this code, the createContext function creates a new context object called AuthDetails.

- **AuthInfo**

This custom React component uses the useEffect, useState, and createContext hooks to manage the user authentication state in the application. It takes a children prop as input, a special prop representing any child elements of the component.
Inside the AuthInfo component, the useEffect hook is used to listen for changes in the user authentication status using the onAuthStateChanged function from the Firebase authentication library. When the authentication state changes, the setLoggedUser function updates the loggedUser state variable with the current user object logged into the console. The return statement in the useEffect hook includes a clean-up function that removes the listener to prevent memory leaks.
The AuthDetails.The provider component wraps the children elements and passes the loggedUser value to any child components that need it using the value prop. The AuthDetails.Provider component is exported along with the AuthDetails context object, so that other components in the application can use the loggedUser value in the AuthDetails context.

## 5.4. MessageDetails Component

**Functions used:**

- **createContext**

This function from the React library creates a new context object, which can be used to share state or other data between components in a React application. In this code, the createContext function creates a new context object called MessageDetails.

- **useContext**

This hook function from the React library allows a functional component to consume a context object and access the data that a parent component has provided. In this code, the useContext hook is used to access the loggedUser value from the AuthDetails context using the AuthDetails context object.

- **useReducer**

This hook function from the React library allows a functional component to manage state using a reducer function. It takes two arguments: a reducer function that takes the current state and an action as input and returns a new state, and an initial state value. In this code, the useReducer hook creates a state variable called state and its associated updater function dispatch. It uses the messageReducer function as the reducer and the INITIAL_STATE object as the initial state.

- **MessageInfo**

This custom React component uses the createContext, useContext, and useReducer hooks to manage the state of a message object in the application. It takes a children prop as input, a special prop representing any child elements of the component.
Inside the MessageInfo component, the useContext hook is used to access the loggedUser value from the AuthDetails context object, which determines the order of the user IDs in the messageId property of the message object.
The messageReducer function is a reducer function that takes the current state and an action as input and returns a new state based on the action type. In this code, the messageReducer function updates the user and messageId properties of the state object based on the CHANGE_USER action type.
The MessageDetails.Provider component is used to wrap the children elements, pass the data, and dispatch values to any child components that need them using the value prop.
The MessageDetails.Provider component is exported along with the MessageDetails context object so that other components in the application can use the data and dispatch values in the MessageDetails context.

## 5.5. AddChild Component

**Functions Used:**

- **useState()**

This is a hook provided by React that allows us to declare a state variable and update it. It takes the initial value of the state variable as an argument. It returns an array containing two values - the current value of the state variable and a function to update the state variable.

- **buildSearchArray(searchTerm)**

This function takes a string argument searchTerm and returns an array of substrings of searchTerm. Concatenating characters of searchTerm creates the substrings from the beginning. For example, if searchTerm is "John Doe", the returned array would be ["J", "Jo", "Joh", "John", "D", "Do", "Doe"].

- **handleSubmit(event)**

This function is called when the form is submitted. It prevents the default behaviour of the form (i.e. refreshing the page) and then creates a new child object with the input data from the form. It then adds this child object to the Firebase Firestore database using the addDoc() function. Finally, it updates the childId field of the child object with the docRef.id value and updates the state variables to clear the form inputs and display the docRef.id value.

- **AddChild()**

This main functional component renders the form for adding a new child. It uses the useState() hook to declare state variables for each input field in the form. It also uses the buildSearchArray() and handleSubmit() functions. The JSX code defines the form layout and includes a submit button that calls the handleSubmit() function when clicked.

## 5.6. ChildDetails Component

**Functions Used:**

- **useContext(AuthDetails)**

This hook provided by React allows us to access the values of the context object. In this component, we use the AuthDetails context object to access the loggedUser value.

- **useState()**

This is a hook provided by React that allows us to declare a state variable and update it. It takes the initial value of the state variable as an argument. It returns an array containing two values - the current value of the state variable and a function to update the state variable.

- **getCurrentFormattedDate()**

This function returns the current date in a formatted string. The format is "DD/MM/YYYY".

- **copyDailyReviews()**

This function is called when the "Copy Review" button is clicked. It filters the dailyReviews array of the child object to get the reviews that were added today (i.e. reviews with the current date). It then maps over this filtered array to create a string containing each review's details. The string is then copied to the clipboard using the navigator.clipboard.writeText() method. If the copy is successful, a popup message is displayed using the setShowCopyPopup() function.

- **ChildDetails({child, removeChild})**

This is the main functional component that displays the details of a child. It takes two props - child, which is an object containing the child's details, and removeChild, which is a function called when the "Remove" button is clicked. The JSX code defines the card layout that displays the child's details, including the daily reviews. It also includes buttons to add a new review, copy the daily reviews, update the child's details, and remove the child. The getCurrentFormattedDate() and copyDailyReviews() functions are used in this component. The useContext() hook also accesses the loggedUser value from the AuthDetails context object.

## 5.7. DailyReview Component

**Functions Used:**

- **useState**

This hook function provided by React allows us to define state variables in functional components. It takes an initial value as its argument and returns an array containing the current state value and a function to update it.

- **useParams**

This hook function provided by the react-router-dom library allows us to access the parameters passed in the URL.

- **useNavigate**

This hook function provided by the react-router-dom library allows us to programmatically navigate to different routes.

- **useContext**

This hook function provided by React allows us to consume data from a context created using the createContext function.

- **useEffect**

This hook function provided by React allows us to perform side effects in functional components. It takes a function as its argument and runs it after the component is rendered.

- **doc**

This function is provided by the Firestore library that creates a reference to a document in the database.

- **updateDoc**

This function is provided by the Firestore library that updates a document in the database.

- **arrayUnion**

This function provided by the Firestore library allows us to add an element to an array in the database without overwriting the existing elements.

- **getDoc**

This function provided by the Firestore library retrieves a document from the database.

## 6. References

[1] *Firebase authentication* (no date) *Google*. Google. Available at: https://firebase.google.com/docs/auth (Accessed: January 16, 2023).

[2] *Firestore | Firebase* (no date) *Google*. Google. Available at: https://firebase.google.com/docs/firestore (Accessed: January 16, 2023).

[3] *Cloud storage for Firebase* (no date) *Google*. Google. Available at: https://firebase.google.com/docs/storage (Accessed: January 16, 2023).