Design Document: Child - User Module – Complete System Architecture

1. Context and Scope :

- Each child is connected to one or more users(spouse,guardian etc..).
- This link between child and device helps the system track activity, show personalised content, and send updates to the user .
- The child's information is used by both the Oriel QML app (on the child's device) and the Parent Portal (for monitoring and reports).

2. Goals :

- Allow a User to manage many children and view device status.
- Enable analytics & safety signals (screen time, activity markers,etc..)

3 . Data Models

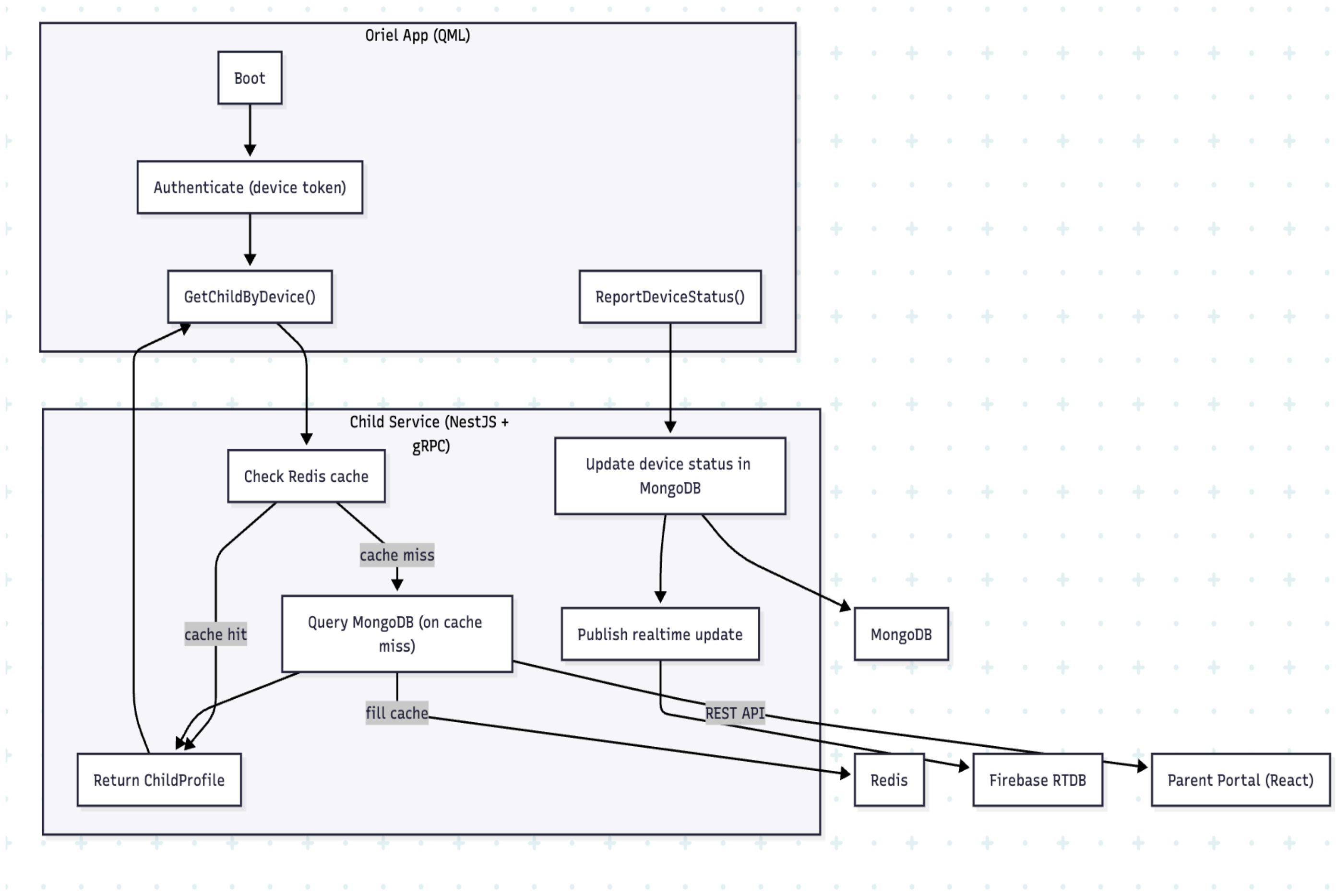
3.1 Collections & Relationship :

- **Accounts** (_id, OwnerUserID, SubscriptionPlan, SubscriptionStatus etc..)
- **Users** `(_id, firebase_id, role, email, mobileNumber,etc..)`
- **Children** `(_id, userId, name, dateOfBirth, age, avatar, deviceId, createdAt, updatedAt)`
- **Devices** `(_id, deviceId, childId, userId, model, serialNumber, screenTime, deviceStatus, batteryLevel, preferences, capabilities, settings, createdAt, updatedAt)`

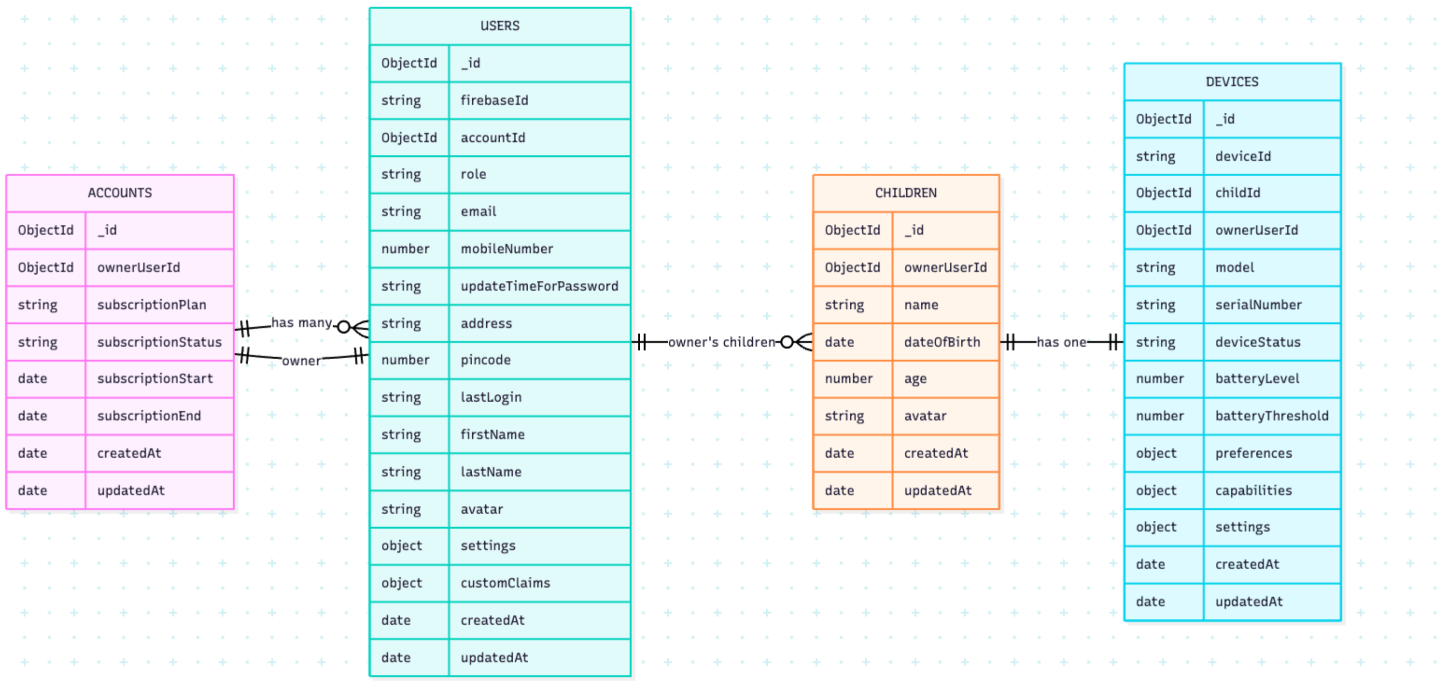
3.2 Cardinality & Constraints

- **User —>Children** (one parent can have many children)
- **Child ——> Device** (each child has exactly one active device)
- **User —>Devices** (user owns devices via children)

4. Data flow :


Oriel App (QML)
Boot
Authenticate (device token)
GetChildByDevice()
ReportDeviceStatus()
Child Service (NestJS + gRPC)
Check Redis cache
Update device status in MongoDB
cache miss
cache hit
Query MongoDB (on cache miss)
Publish realtime update
MongoDB
fill cache
REST API
Return ChildProfile
Redis
Firebase RTDB
Parent Portal (React)

5 . ERD



ACCOUNTS	
ObjectId	_id
ObjectId	ownerUserId
string	subscriptionPlan
string	subscriptionStatus
date	subscriptionStart
date	subscriptionEnd
date	createdAt
date	updatedAt

USERS	
ObjectId	_id
string	firebaseId
ObjectId	accountId
string	role
string	email
number	mobileNumber
string	updateTimeForPassword
string	address
number	pincode
string	lastLogin
string	firstName
string	lastName
string	avatar
object	settings
object	customClaims
date	createdAt
date	updatedAt

CHILDREN	
ObjectId	_id
ObjectId	ownerUserId
string	name
date	dateOfBirth
number	age
string	avatar
date	createdAt
date	updatedAt

DEVICES	
ObjectId	_id
string	deviceId
ObjectId	childId
ObjectId	ownerUserId
string	model
string	serialNumber
string	deviceStatus
number	batteryLevel
number	batteryThreshold
object	preferences
object	capabilities
object	settings
date	createdAt
date	updatedAt

6. Methods :

Backend :

6.1 CheckAccountOnLogin() - Portal

- When the user logs in, look into the user table for accountID if found abort.
- If accountID is not found, create an account table for the CurrentUser with the following.

{ OwnerUserID	: use the objectID of currentUser here,
SubscriptionPlan	: CurrentUser Plan,
SubscriptionStatus	: CurrentUser Plan status,
SubscriptionStart	: CurrentUser Plan starting date
SubscriptionEnd	: CurrentUser Plan ending date,
CreatedAT	: getCurrentTimeStamp from server,
UpdatedAt	: getCurrentTimeStamp from server, }

- Map that account table's ObjectID _id to accountID on the CurrentUser table.
- Return success.

6.2. GetChildrenByAccountId() - Portal

- Look into user table to find accountID based on this accountID find the ownerUserId
- Look in the **Child** table for all records where the OwnerUserId matches the ownerUserId(coming from above step).
- Return the array of matching children.

6.3.GetChildFullProfile(childId) - Portal

- Find the child record where _id equals childId.
- Find the device record where deviceId matches the child's deviceId.
- Combine the child’s details with the device’s info into one object.
- Return this combined profile.

6.4.UpdateDeviceStatus (aka Heartbeat)(deviceId, batteryLevel) - Device

- Called every 30 seconds
- Find the device where deviceId equals the input deviceId.
- Update the device’s batteryLevel to batteryLevel.
- Update the device’s updatedAt timestamp to the current time.

6.5. TrackEvents (on-demand - whenever it happens) - Device

- Screen_on ({event_name: “screen_on”})
- Screen_off ({event_name: “screen_off”})
- Shutdown ({event_name: “shutdown”})
- App Based Events
    - Listened to Podcast
        - Event: {event_name: “listened_podcast”,podcast_title: “”, episode_title: “”}
    - Wrote Journal
        - Event: {event_name: “written_journal”}
    - Read a Book
        - Event: {event_name: “read_book”, book_title: “”, last_page: 10, total_pages: 120}
    - Played a Game
        - Event: {event_name: “played_game”, game_title: “”}
    - Played a Quiz
        - Event: {event_name: “played_quiz”, quiz_title: “”}

- Browsed Internet
    - Event: {event_name: “browserd”,url: “”}
- AI
    - Event: {event_name: “ai_text_chat”, prompt: “”}
    - Event: {event_name: “ai_voice_chat”, url: “”}.

6.6 AddNewChild(deviceId, childName, dateOfBirth) - Portal

- Check if the device with the given deviceId exists and that it belongs to the userId.
    - If the device does not exist or does not belong to this user, return an error saying: "Device doesn’t exist”
- Create a new child record with these details:
    - userId from input
    - name set to childName from input

    - dateOfBirth from input
    - Calculate age based on the dateOfBirth
    - Link deviceId from input
    - Set createdAt and updatedAt to the current date and time

- Save this new child record into the **CHILDREN** collection.

- Update the device record with the matching deviceId to link it to the new child by setting:

    - childId to the newly created child's ID
    - updatedAt to the current date and time

- Return the newly created child record including the linked device information.

GRPC and QT :

6.1.1 Function **updateDevicePreferences**(userId, newPreferences)

- Look in the database for a device that belongs to the given user ID.
- If no device is found for that user, stop and return a message
- If a device is found replace its current preferences with the new preferences provided
- Save the updated device information back to the database
- Return success

7. Conclusion:

This document defines the complete architecture, data flow, and gRPC/Qt integration for managing child_user relationships in Oriel enabling parents to monitor control and receive real time alerts for their child’s device activity