“Juggle”

Android App to manage busy families’ calendars

FINAL REPORT

Kathleen McCarthy Kelleher
#20042361
Higher Diploma in Science in Computer Science
April 2022

Declaration

I declare that the work which follows is my own, and that any quotations from any sources (e.g. books, journals, the internet) are clearly identified as such by the use of 'single quotation marks', for shorter excerpt and identified italics for longer quotations. All quotations and paraphrases are accompanied by (date, author) in the text and a fuller citation is the bibliography. I have not submitted the work represented in this report in any other course of study leading to an academic award.

Name: Kathleen McCarthy Kelleher Date: 18/04/2022

Acknowledgements

A huge thanks to everyone involved with the Higher Diploma in Computer Science at Waterford Institute of Technology, it has been a hugely positive experience.

Thanks to my project supervisor Frank Walsh.

Special thanks to my fellow Juggler Ger, and also my Juggled Fionn, Cullen and Síomha who are a constant inspiration.

Thanks to Cullen McCarthy Kelleher for allowing me to use his original artwork within the sign in view, it made me smile every time.

Titles

COMMERCIAL TITLE

“Juggle”

ACADEMIC TITLE

Android App to manage busy families’ calendars

Preface

This document should be read in consultation with:

Project GitHub Page:	https://kathleenmk.github.io/android_app_juggle/
Project Repository:	https://github.com/KathleenMK/android_app_juggle
Trello Board:	https://trello.com/b/ggPVPCOG/juggle

Abstract

Inspired by the constant juggle of children’s activities and the assignment of related tasks for their parents.

Juggle is a native Android application that upon Google Sign-In retrieves and displays the calendars for which the user has owner access, the user can assign a Juggler(responsible adult) or Juggled(child) role to each calendar.

Juggled events can be viewed within the app and related events(getting there, getting home, other) added to one of the Jugglers calendars.

The app utilizes the Google calendar APIs and so the Android family can continue to use all the Google calendar event functionality.

Keywords

User:	The signed in user of the Juggle android app, Google Sign In is the only sign in option available and so that user will always have a Google Account.
Calendar:	A Google calendar. The Juggle app can read and write to existing Google calendars of the User and the Google calendars to which the User has “owner” access. The Juggled and Jugglers will continue to use their Google calendars in the usual way.
Juggled:	One of two roles that can be assigned by a User to the owner of a Google calendar to which the User has “owner” access. The focus of the app is to display the Google calendar events of the Juggled so that related events can be added and reviewed by the User. If the User adds a Dinner (or other event using the “Add Event” functionality), it will be added to the Juggled Google Calendars. The Juggled do not require the app. Children would be assigned the role of Juggled.
Juggler:	The other of two roles that can be assigned by a User to the owner of a Google calendar to which the User has “owner” access. Events created within the app, that are related to a Juggled calendar event, is added to one of the Juggler’s calendars. If the User adds a Dinner (or other event using the “Add Event” functionality), it will be added to the Juggler’s Calendar also. Jugglers do not require the app to be assigned events, although they would be an expected User of the app. Parents would be assigned the role of Juggler.

List of Abbreviations

UID:	‘an identifier that is guaranteed to be unique among all identifiers used for those objects and for a specific purpose’ (Wikipedia, 2022)

Table of Contents

Table of Figures

1 Introduction

1.1 CONCEPT

I wanted to build something that I would use myself. My initial idea was to build an agile type solution to help manage family and household tasks and share the load between the family to include a backlog of tasks, a shared calendar and the generation of associated tasks if a certain type of activity is added.

Considering the popularity of Google calendar and the broad range of integration options for applications that already exists it seemed unnecessary to try and replicate that functionality and that's when I stumbled upon the Google Calendar API.

I also wanted to focus on mobile app development, it was one of the most challenging of the modules but I was keen to learn more as mobile apps are so popular and easy to distribute.

I am a parent of three children, two of which are old enough to own android tablets and have a personal Google account which by default includes a Google calendar. Voice commands are their default when they interact with Google and so they can easily add their own events and access the details of their own Google Calendar events.

And so my project idea was born:

- for kids to use their existing Google Calendar functionality to add and access events
- for parents to access those events in the app using the Google Calendar API and within the app create and assign related tasks, with some persistence in a cloud database.

  (in agile speak the list of children's events being the backlog; each backlog item is then broken up into individual consequential tasks; those tasks assigned via an entry in a parent's calendar; estimate of time required per task using the start and end times of a Google calendar event.)
- And while engaged with the Google calendar functionality, I also hoped to address the repeated answering of the question "what's for dinner?" by utilizing that same Google API functionality to add a dinner event to each calendar so that they could repeatedly ask Google Assistant or Alexa that very question.

As an example, my eldest child has a once off football training session on Monday at 6pm, he adds it to his calendar using Google Assistant on his Android tablet. I see the event in the Juggle app, I add a related event in Juggle, assigned to myself, to bring him to the session with a start time of 5:45pm, I add an additional related event in Juggle, assigned to his Father, to bring him home from football with a start time of 6:45. We all get default Google calendar reminders of these events.

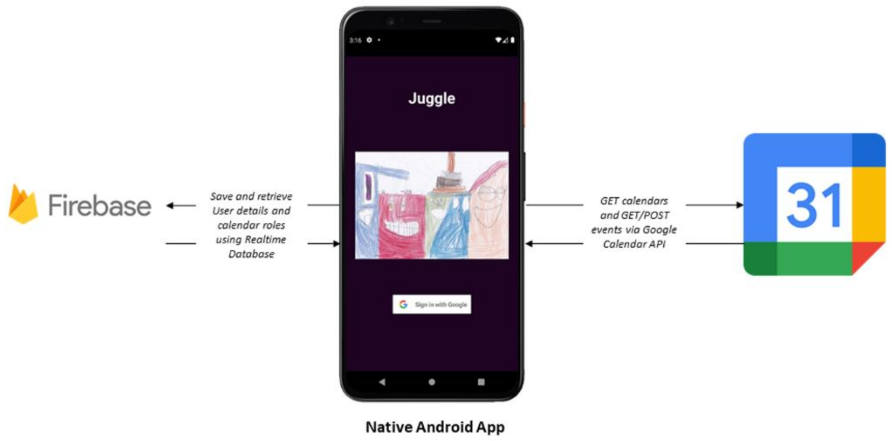

Firebase
Save and retrieve User details and calendar roles using Realtime Database
Juggle
Sign in with Google
Native Android App
GET calendars and GET/POST events via Google Calendar API
31

Figure 1: High level view of Concept

1.2 SCOPE

This app is being built to run on Android only (with Minimum SDK of 29)

A user will have a Google account and signing into the app is by Google Sign In only. The user will need to grant the Juggle app read/write access to their Google calendars via a consent screen generated upon sign in.

The Google user who signs into the app will already have been granted read/write access to the Google calendars from which events will be displayed and for which related tasks can be added or which a dinner event could be added.

Google's calendar's APIs are central to the flow of information from Google calendars to the Juggle app, and the addition of events to Google Calendars within the Juggle app.

The specific Google Calendar API HTTP requests, as outlined on the Google Calendar for Developers page (Google Developers, 2021) to be used are:

Calendar List: returns the google calendars to which the user has access

"GET https://www.googleapis.com/calendar/v3/users/me/calendarList"

Events list: returns the events of a specific calendar

"GET https://www.googleapis.com/calendar/v3/calendars/*calendarId*/events"

Create an event: creates an event in a specific calendar

"POST https://www.googleapis.com/calendar/v3/calendars/*calendarId*/events"

The User will be able to view a list of calendars to which they have access with some of the specific details of each calendar, most importantly the calendar id, being the unique but readable identifier for a calendar.

The User will be able to view the events of a calendar to which they have access.

The User will be able to view the details of an event from a calendar to which they have access.

The User will be able to add a task to a Jugglers' calendar with details as input in the app (a task in this case will be a Google event with a start and end time in a parent's calendar).

2 Analysis

The following analysis examines, at a high level, some options currently available for parent's "Juggling" children's events and why there is a gap for a calendar managing app such as Juggle.

Current "Juggling" Options:

1. Manual option, tried and tested, parent/parents remembering everything.

2. Using current Google calendar functionality, view all calendars for which the parent has access, add new events as necessary for a calendar through existing android calendar app.

3. Use one of the existing technical solutions available, for example, the top two as per the website (Develop Good Habits, 2022):

    a. "Calendar" as per their website (Calendar, 2022)
        - Users grant access to Google calendars so any calendars that the user has access to is included.
        - Users can create events for the family in any of those calendars
        - Users can view an analysis of time spent based on different metrics including time spent with each other

    b. "Cozi Family Organizer", as per their website (Cozi, 2022)
        - Users grant access to Google calendars by family member so any calendars that the user has access to is included.
        - Calendar shared by family members
        - Users can create events
        - Users can create reminders
        - To do lists shared by family members

The main gaps in option 1 above is the possibility of the mental load being distributed unevenly and also the distinct possibility of forgetting some of the juggling steps.

The main gap in options 2 and 3 are the lack of relationship between the children's events and the related parents' tasks.

The Juggle app will address those aforementioned gaps but will also further my education of Android Development, Databases and Google's APIs.

3 Tools and technologies

3.1 KOTLIN ANDROID APP

With all the devices that I own, my own android mobile phone is my first point of call and so I decided to build an android app. Many years ago I choose Android and it has always made sense to stay with it, luckily my husband had made that same decision and so my children also have android devices

Kotlin was chosen as the language for this app, we had covered it in the mobile app development module and I welcomed the opportunity to continue on that learning journey. Is it something that I questioned in the early stages of the project, whether I should develop the app in Java to practice those skills. Kotlin was deemed to be preferred, as per the following quote in response to the question of which was better, as found in a medium.com article

> *In 2021, Kotlin will be the primary language for Android development. Although both Java and Kotlin can be used to create performant and valuable apps, Google's frameworks, tools, documenting, and study materials continue to favour Kotlin, making it the better language for Android today.* (Modi, 2022)

One of the disadvantages of Kotlin, a much newer language, that I came to realize later in the project was the lesser available support when trying to overcome issues throughout the project's development.

3.2 GOOGLE SIGN IN

Google Sign In seemed the obvious choice for my app. Allowing any other authentication option would be quickly redundant as a Google Account with Google calendars is required to use the app and access to that account and those calendars are sought immediately.

3.3 FIREBASE AUTHENTICATION

Although Google Sign in does not require Firebase Authentication, I will implement Firebase so that that same authentication can be used when reading and writing to the Firebase database.

3.4 FIREBASE REALTIME DATABASE

Firebase and its ease of use with Google sign in made it another obvious choice to use for persistence. Firebase has a Realtime Database and a Cloud Firestore Database. The choice to use the Realtime database was based on the considerations as presented on the Firebase Documentation found here (Google Developers, 2022), from which the Realtime Database was deemed most suitable. See Appendix A for decision considerations.

3.5 GOOGLE CALENDAR APIS

3.5.1 OAuth Authorization

The Google Calendar for Developers page (Google Developers, 2022) includes the following: 'Your application must use OAuth 2.0 to authorize requests. No other authorization protocols are supported. If your application uses Google Sign-In, some aspects of authorization are handled for you.' That same page has the following steps outlining the authorization of requests to the Google Calendar API:

The following general process applies to all application types:

1. *When you create your application, you register it using the Google API Console. Google then provides information you'll need later, such as a client ID and a client secret.*
2. *Activate the Google Calendar API in the Google API Console. (If the API isn't listed in the API Console, then skip this step.)*
3. *When your application needs access to user data, it asks Google for a particular **scope** of access.*
4. *Google displays a **consent screen** to the user, asking them to authorize your application to request some of their data.*
5. *If the user approves, then Google gives your application a short-lived **access token**.*
6. *Your application requests user data, attaching the access token to the request.*
7. *If Google determines that your request and the token are valid, it returns the requested data.*

Google Developers website also includes a OAuth 2.0 Playground (Google Developers, n.d.) that can replicate the above steps using one's own OAuth credentials. See Appendix B for OAuth 2.0 Playground, Server-side OAuth flow steps.

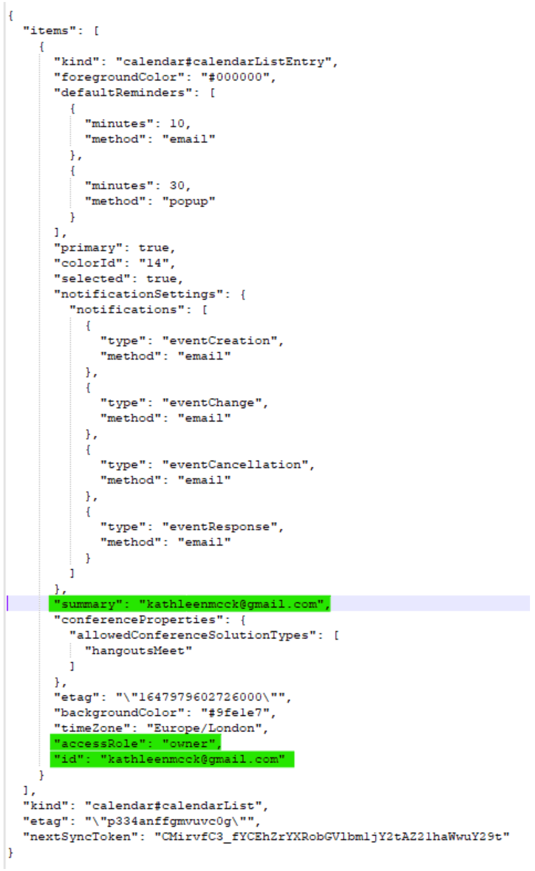
3.5.2 JSON responses

3.5.2.1 Calendar List

The HTTP GET request returns a JSON array of data ("items") from the Request URI: https://www.googleapis.com/calendar/v3/users/me/calendarList

Each item is a calendar that the User has access to, the app's Calendar model is a subset of the available JSON data consisting of: summary, access role and id.

Here is an example of the JSON data available, showing one calendar returned:



```
{
  "items": [
    {
      "kind": "calendar#calendarListEntry",
      "foregroundColor": "#000000",
      "defaultReminders": [
        {
          "minutes": 10,
          "method": "email"
        },
        {
          "minutes": 30,
          "method": "popup"
        }
      ],
      "primary": true,
      "colorId": "14",
      "selected": true,
      "notificationSettings": {
        "notifications": [
          {
            "type": "eventCreation",
            "method": "email"
          },
          {
            "type": "eventChange",
            "method": "email"
          },
          {
            "type": "eventCancellation",
            "method": "email"
          },
          {
            "type": "eventResponse",
            "method": "email"
          }
        ]
      },
      "summary": "kathleenmcck@gmail.com",
      "conferenceProperties": {
        "allowedConferenceSolutionTypes": [
          "hangoutsMeet"
        ]
      },
      "etag": "\"1647979602726000\"",
      "backgroundColor": "#9fe1e7",
      "timeZone": "Europe/London",
      "accessRole": "owner",
      "id": "kathleenmcck@gmail.com"
    }
  ],
  "kind": "calendar#calendarList",
  "etag": "\"p334anffgmvuvc0g\"",
  "nextSyncToken": "CMirvfC3_fYCEhZrYXRobGVlbmljY2tAZ21haWwuY29t"
}
```

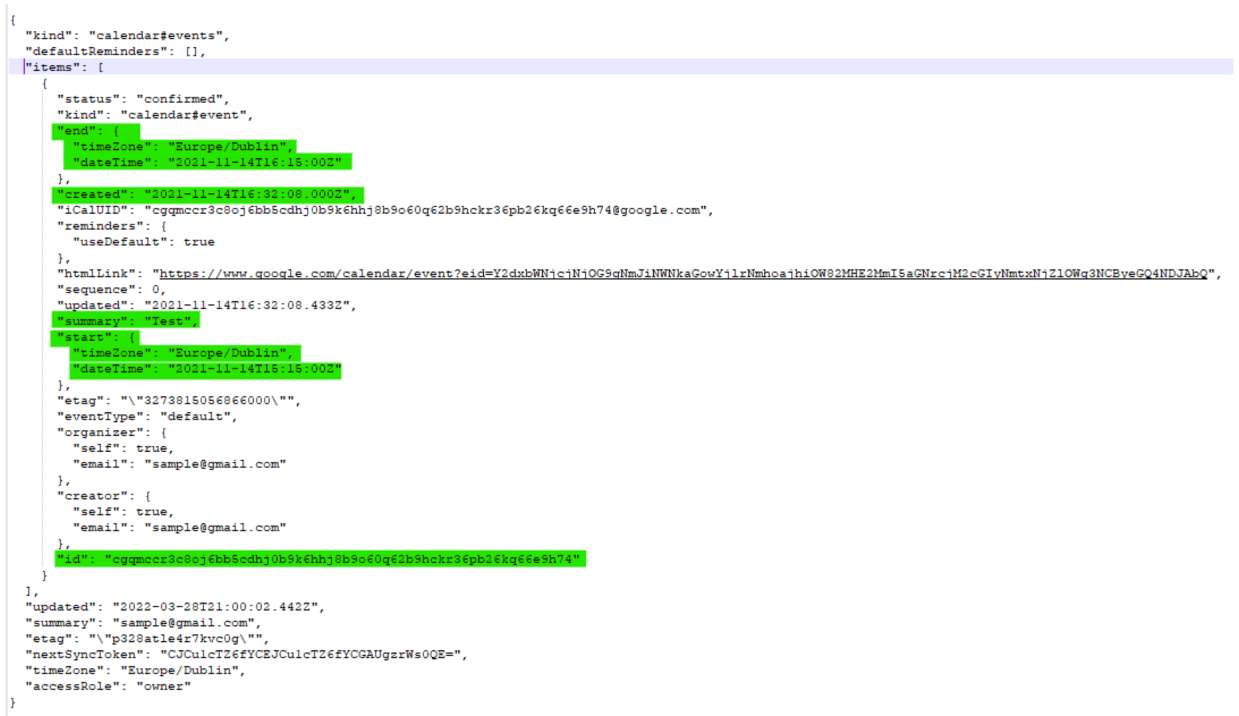
Figure 2: Calendar List sample JSON response

3.5.2.2 Events list

The HTTP GET request returns a JSON array of data ("items") from the Request URI: https://www.googleapis.com/calendar/v3/calendars/*calendarId*/events

Each item is an event in the specified calendar included in the request. The app's event model is a subset of the available JSON data consisting of: summary, id, start, end and created.

Here is an example of the JSON data available, showing one event returned for a specific calendar:



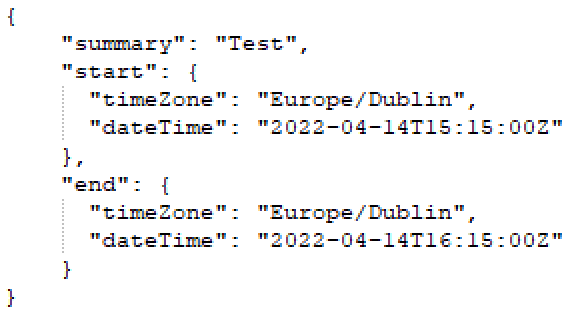
```
{
  "kind": "calendar#events",
  "defaultReminders": [],
  "items": [
    {
      "status": "confirmed",
      "kind": "calendar#event",
      "end": {
        "timeZone": "Europe/Dublin",
        "dateTime": "2021-11-14T16:15:00Z"
      },
      "created": "2021-11-14T16:32:08.000Z",
      "iCalUID": "cgqmccr3c8oj6bb5cdhj0b9k6hhj8b9o60q62b9hckr36pb26kq66e9h74@google.com",
      "reminders": {
        "useDefault": true
      },
      "htmlLink": "https://www.google.com/calendar/event?eid=Y2dxbWNjcjNjOG9qNmJiNWNkaGowYjlrNmhoajhiOW82MHE2MmI5aGNrcjM2cGIyNmtxNjZlOWg3NCByeGQ4NDJAbQ",
      "sequence": 0,
      "updated": "2021-11-14T16:32:08.433Z",
      "summary": "Test",
      "start": {
        "timeZone": "Europe/Dublin",
        "dateTime": "2021-11-14T15:15:00Z"
      },
      "etag": "\"3273815056866000\"",
      "eventType": "default",
      "organizer": {
        "self": true,
        "email": "sample@gmail.com"
      },
      "creator": {
        "self": true,
        "email": "sample@gmail.com"
      },
      "id": "cgqmccr3c8oj6bb5cdhj0b9k6hhj8b9o60q62b9hckr36pb26kq66e9h74"
    }
  ],
  "updated": "2022-03-28T21:00:02.442Z",
  "summary": "sample@gmail.com",
  "etag": "\"p328atle4r7kvc0g\"",
  "nextSyncToken": "CJCulcTZ6fYCEJCulcTZ6fYCGAUgzrWs0QE=",
  "timeZone": "Europe/Dublin",
  "accessRole": "owner"
}
```

Figure 3: Event List sample JSON response

3.5.2.3 Create an event

The HTTP POST request sends a JSON representation of an event for a specified calendar using the Request URI:
https://www.googleapis.com/calendar/v3/calendars/*calendarId*/events

For the creation of an event the app's add event model specifies a summary, start and end as the POST's request body, for example:

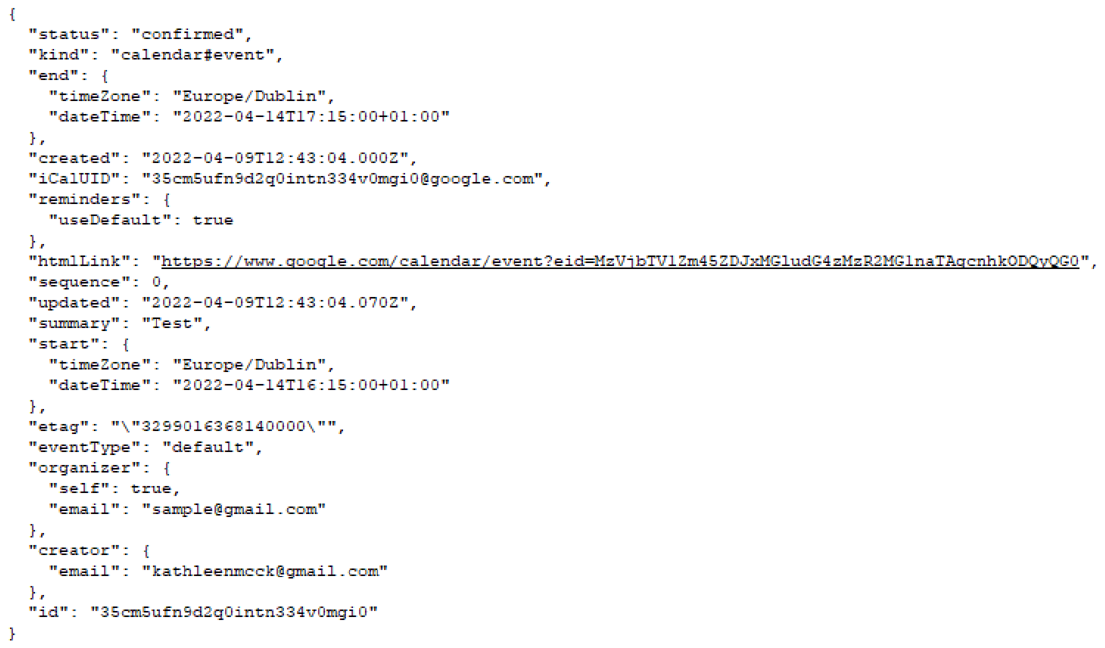


```
{
    "summary": "Test",
    "start": {
      "timeZone": "Europe/Dublin",
      "dateTime": "2022-04-14T15:15:00Z"
    },
    "end": {
      "timeZone": "Europe/Dublin",
      "dateTime": "2022-04-14T16:15:00Z"
    }
}
```

Figure 4: Event add sample JSON request body

Upon successful creation the response details the event as a JSON response in the same way as returned for the Events list above.

Here's an example of the response upon the creation of a calendar event using the API:



```
{
  "status": "confirmed",
  "kind": "calendar#event",
  "end": {
    "timeZone": "Europe/Dublin",
    "dateTime": "2022-04-14T17:15:00+01:00"
  },
  "created": "2022-04-09T12:43:04.000Z",
  "iCalUID": "35cm5ufn9d2q0intn334v0mgi0@google.com",
  "reminders": {
    "useDefault": true
  },
  "htmlLink": "https://www.google.com/calendar/event?eid=MzVjbTV1Zm45ZDJxMGludG4zMzR2MGlnaTAgcnhkODQyQG0",
  "sequence": 0,
  "updated": "2022-04-09T12:43:04.070Z",
  "summary": "Test",
  "start": {
    "timeZone": "Europe/Dublin",
    "dateTime": "2022-04-14T16:15:00+01:00"
  },
  "etag": "\"3299016368140000\"",
  "eventType": "default",
  "organizer": {
    "self": true,
    "email": "sample@gmail.com"
  },
  "creator": {
    "email": "kathleenmcck@gmail.com"
  },
  "id": "35cm5ufn9d2q0intn334v0mgi0"
}
```

Figure 5: Event add sample JSON response

4 Modelling

4.1 USER STORIES

- As a parent I want to launch and signup to the app with Google Sign In so I do not have to use a password and separately identify my Google calendar.
- As a parent I want to see a list of Google calendars, to which I have access, and choose which calendar is to be monitored for new events so that I can focus only on those calendar events.
- As a parent I want to also choose which calendar to which new tasks can be added so that I don't have to go into my Google calendar separately.
- As a parent I want to see a list of Google calendar events from children's calendars so that I don't need to check calendars individually.
- As a parent I want to see more detail of an event, as above, in a new view so that I can decide what related tasks are required.
- As a parent I want to create a task using some of the details from an event, as above, or some additional details, and add that task to a parent's calendar so that I can use the existing Google calendar functionality for reminders etc.
- As a parent I want to be able to distinguish between new events and those that have already have related tasks added so that I can focus on those that have not been considered.
- As a parent I want to create a dinner event and add it to parents and children's calendar so that I don't have to continually repeat myself.
- As a parent I want to see what calendars I have specified for each role so that I can verify or update my choices.
- As a parent I want to login and see that my choices have persisted so that I don't have to specify roles repeatedly.
- As a child I want to add an event to my Google calendar and know that my parent will see it and consider what needs to be done.
- As a child I want to ask Alexa/Google Assistant "what's for dinner?" and get the details so that I will know what's for dinner.

4.2 NON-FUNCTIONAL REQUIREMENTS

- User must have a Google account
- User must already have been granted read/write access to the Google calendars that will be monitored for new events or to which tasks can be added or which a dinner event could be added (specifically the permission "make changes and manage sharing" must be granted by the calendar owner to the User).
- User must give permission for access to Google Calendar (specifically "See, edit, share, and permanently delete all the calendars you can access using Google Calendar)
- Assumed continued use of existing Google Calendar add and read events from android and/or Alexa/Google Assistant.
- A "Routine" can be added to Google Assistant, via Settings in the Google Android app, with the following specifics:
    - *"starter": "When I say to Google Assistant: "What's for Dinner?"*
    - *"action": "Tell me about today's calendar"*

  So that Google Assistant, upon the above voice command, can read the events of that day's calendar, including the Dinner event with dinner details as added by the User in the Juggle app

4.3 SYSTEM MODEL

4.3.1 Flow Diagram

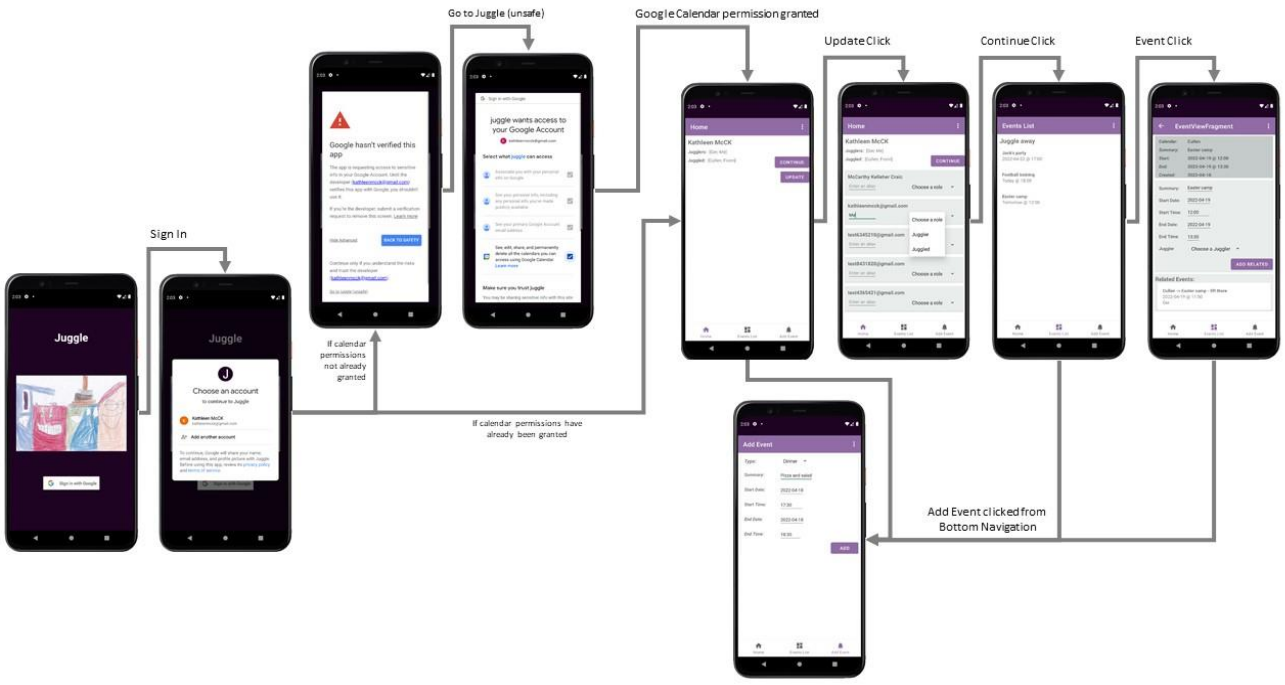

Sign In
Go to Juggle (unsafe)
If calendar permissions not already granted
Google Calendar permission granted
If calendar permissions have already been granted
Update Click
Continue Click
Event Click
Add Event clicked from Bottom Navigation
Juggle
Choose an account
Google hasn't verified this app
juggle wants access to your Google Account
Home
Events List
EventViewFragment
Add Event

Figure 6: App Flow Diagram

4.3.2 Views

4.3.2.1 Sign In

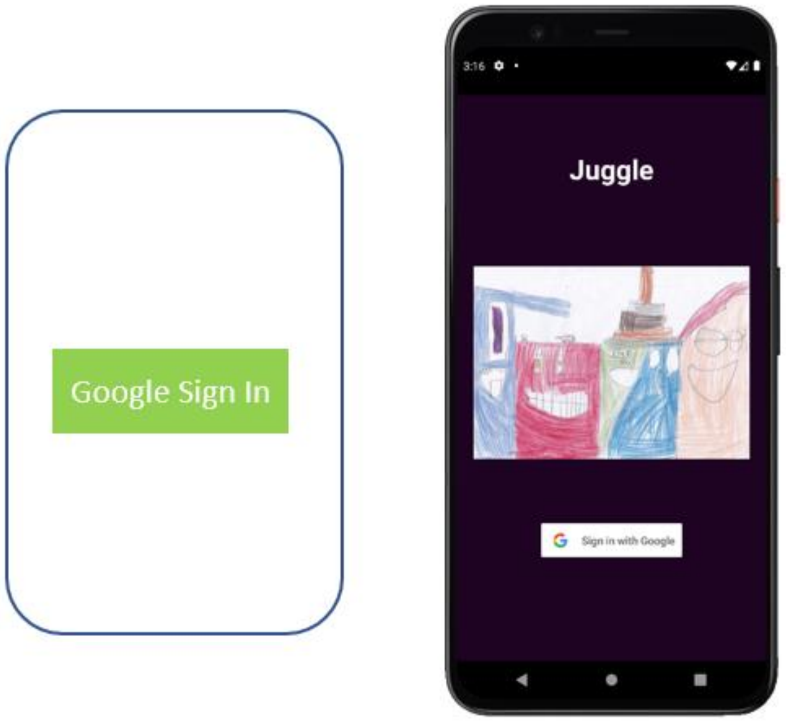


Figure 7: Sign In view - initial design vs. actual

Upon app launch, a simple view with only Google sign in option displayed. There is no alternative view or flow for a user to sign up to the app. If a user has not signed in previously and saved their Juggler and Juggled choices the heading field will display default text, otherwise the data will be populated from the User as saved in the database.

4.3.2.2 Calendar List

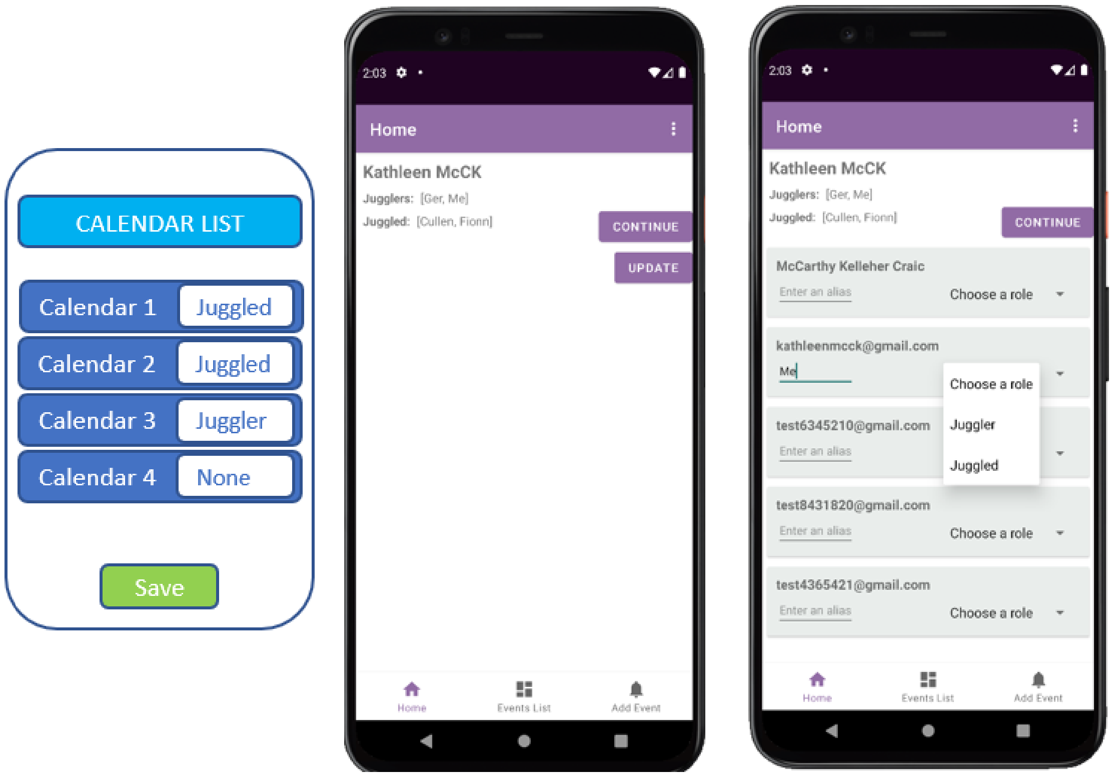


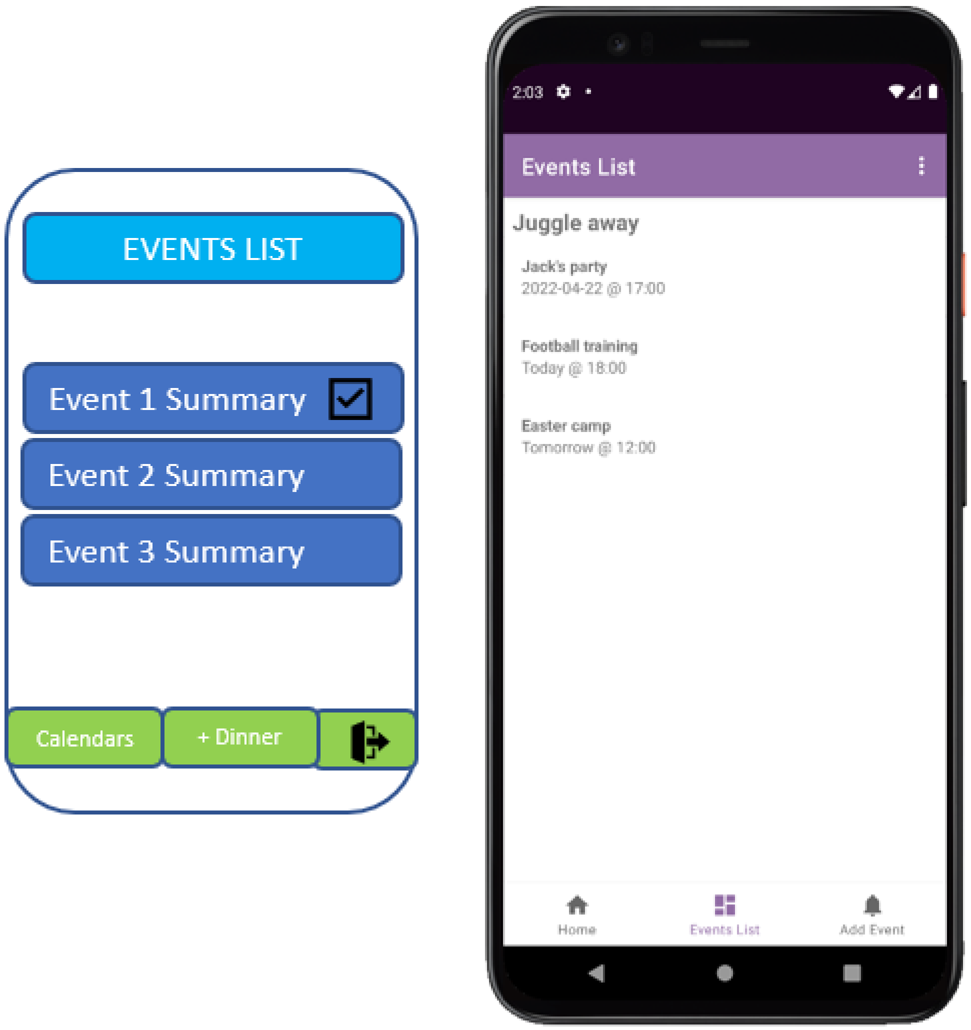
Figure 8: Calendar List view - initial design vs. actual &actual upon update button clicked

This view will display the current user's Google account name as well as any Juggler and Juggled aliases that have already been added.

Upon clicking the update button, a list of the user's Google calendars to which they have owner access are displayed. An alias and a role of Juggled(child) or Juggler(parent) can be assigned to each calendar. The aliases must be distinct from each other (ignoring case) and at least one of each role must be chosen to continue. Upon continue the Google Id, Google name, aliases and corresponding google id is saved for this user and the events list is displayed.

If the user clicks on a specific calendar in the list, the events for that calendar will be displayed in the Events List view.

4.3.2.3 Events List



Juggle away

Jack's party
2022-04-22 @ 17:00

Football training
Today @ 18:00

Easter camp
Tomorrow @ 12:00

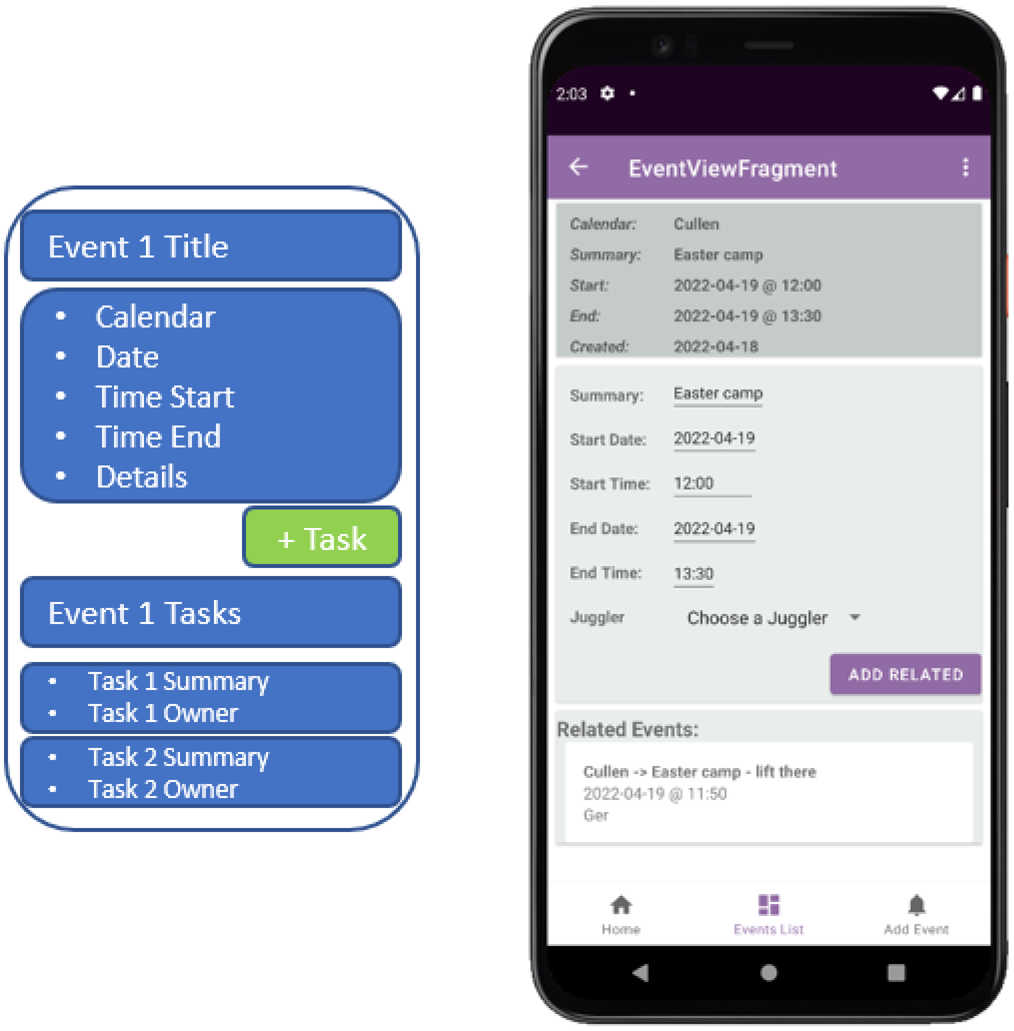
Figure 9: Events List view - initial design vs. actual

A list of events from the calendars that are to be juggled as per that user's choices. Current functionality includes only events from the first of the Juggled calendars. Future development would be required to ensure events from all Juggled calendars are displayed.

Also I have not implemented the checked event indicator (Initially designed as: a checked indicator will appear in the list with the event title. An event is deemed to have been checked if tasks have already been added or "no task required" has been chosen.)

Upon clicking an event in the events list, the detail of that event is displayed.

4.3.2.4 Events Detail



Calendar:	Cullen
Summary:	Easter camp
Start:	2022-04-19 @ 12:00
End:	2022-04-19 @ 13:30
Created:	2022-04-18

Summary:	Easter camp
Start Date:	2022-04-19
Start Time:	12:00
End Date:	2022-04-19
End Time:	13:30
Juggler	Choose a Juggler

Related Events:

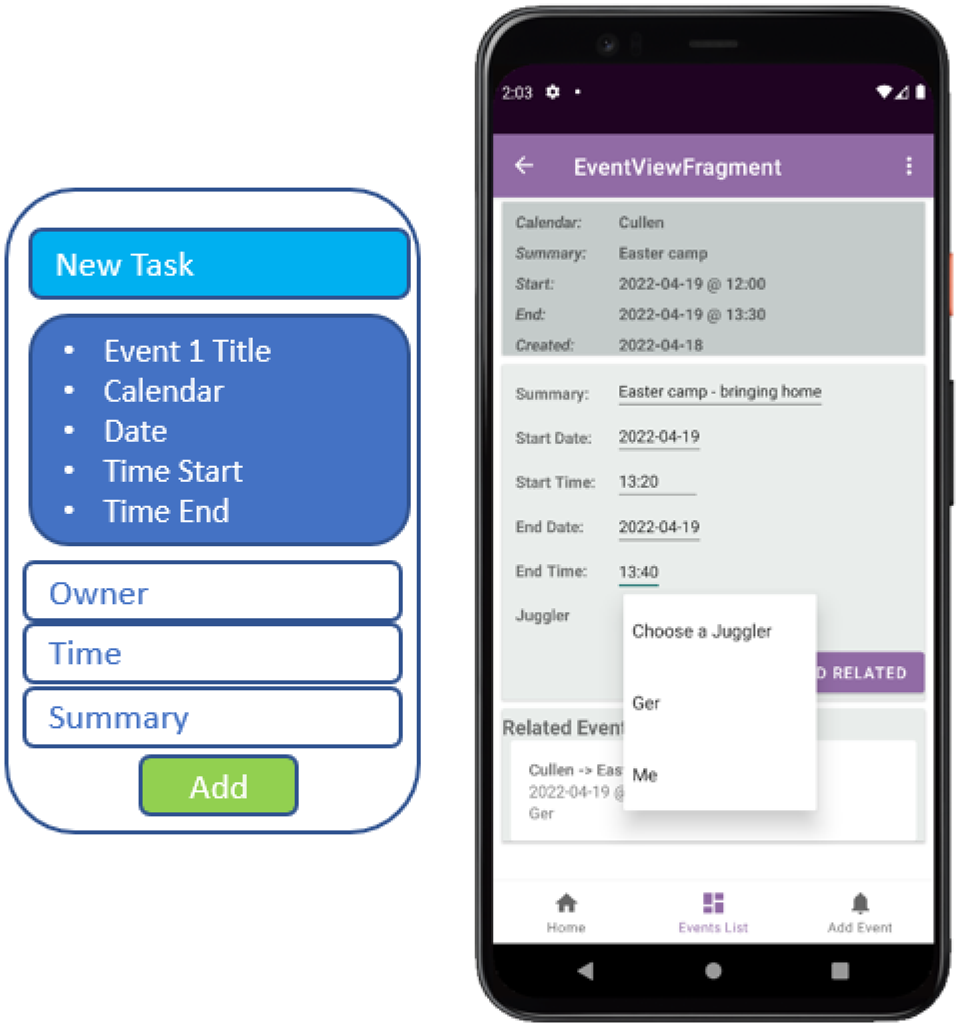
Cullen -> Easter camp - lift there
2022-04-19 @ 11:50
Ger

Figure 10: Events Detail view - initial design vs. actual

Details of the event selected is displayed in a new view, any tasks already assigned are listed.

The initial design was to click on the "+ Task" button to display the "Add Task" view, however I have implemented the above view where the related event can be added from this view.

4.3.2.5 *Add Task*



2:03

EventViewFragment

Calendar: Cullen
Summary: Easter camp
Start: 2022-04-19 @ 12:00
End: 2022-04-19 @ 13:30
Created: 2022-04-18

Summary: Easter camp - bringing home
Start Date: 2022-04-19
Start Time: 13:20
End Date: 2022-04-19
End Time: 13:40
Juggler

Choose a Juggler
Ger
Me

D RELATED

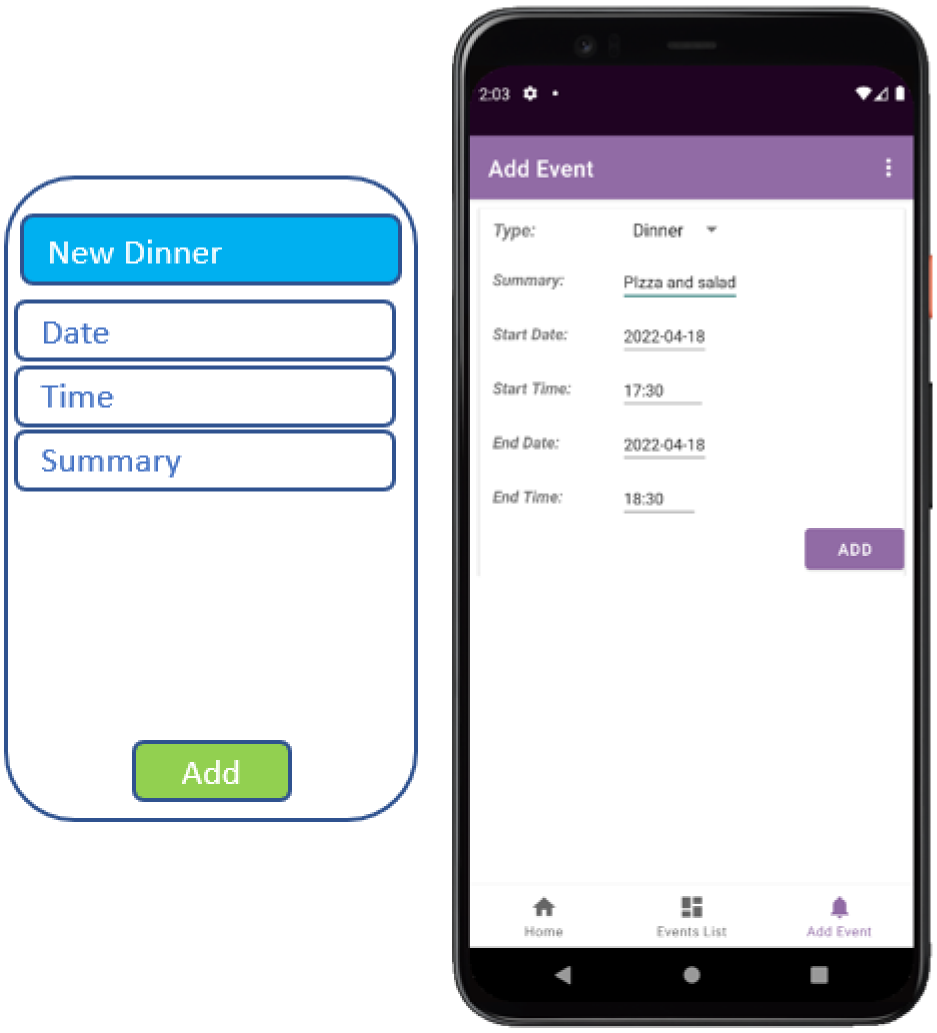
Related Event

Cullen -> Eas
2022-04-19 @
Ger

Figure 11: Add Task view - initial design vs. actual.

As above the initial design had a separate view for the add task, however this has been updated to the above, so that all the related tasks already added can be seen when deciding upon any additional related tasks that should be added. The related event input fields are prepopulated with the original event details, the "Choose a Juggler" dropdown is populated with the Juggler aliases that the user has saved to the Database.

4.3.2.6 Add Event



2:03

Add Event

Type: Dinner

Summary: Pizza and salad

Start Date: 2022-04-18

Start Time: 17:30

End Date: 2022-04-18

End Time: 18:30

ADD

Figure 12: Add Event view - initial design vs. actual

Date, time, Summary of a new event can be input and upon a click of the add button that event will be added to all the Juggled and all the Jugglers calendars that the User has specified. The initial design was for a dinner event however the addition of many different types of events could also be required for the same set of calendars, and so I have implemented a "Type" field that defaults to "Dinner", a default start time that is "17:30" and a default end time of "18:30". The dropdown currently includes a blank type but in future work could be updated to include User defined default types and default times.

4.4 DATA MODEL

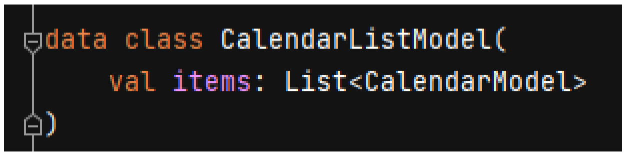
4.4.1 App

The models within the Juggle app detailed below. Any data that is passed between fragments in the app uses Parcelize. Parcelize is a Plugin that allows the serialization of a class so that it can be transferred through the android app.

4.4.1.1 Calendar List

The JSON response from the Calendar list GET request is a list of items, each of which is a calendar, modelled by the CalendarModel.

The following is the CalendarListModel used:

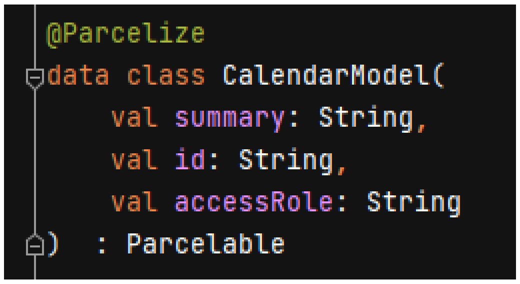


```
data class CalendarListModel(
    val items: List<CalendarModel>
)
```

Figure 13: CalendarListModel within Juggle

4.4.1.2 Calendar

Of the data returned for each item in the JSON response from the Calendar List GET request the following subset is modelled as the CalendarModel, where the data types match that of the JSON response:



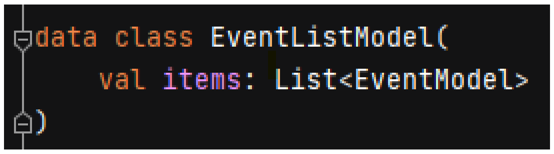
```
@Parcelize
data class CalendarModel(
    val summary: String,
    val id: String,
    val accessRole: String
)  : Parcelable
```

Figure 14: CalendarModel within Juggle

4.4.1.3 Event List

The JSON response from the Calendar Event list GET request is also a list of items, each of which is an event, modelled by the EventModel.

The following is the EventListModel used:

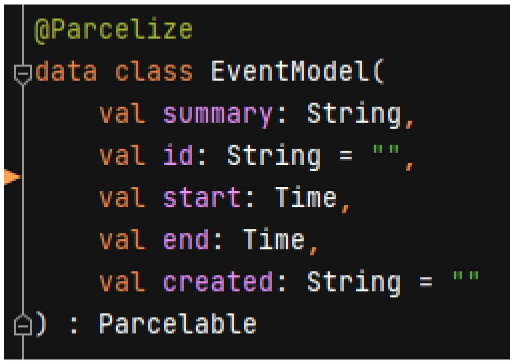


```
data class EventListModel(
    val items: List<EventModel>
)
```

Figure 15: EventListModel within Juggle

4.4.1.4 Event

Of the data returned for each item in the JSON response from the Event List GET request the following subset is modelled as the EventModel, where the data types match that of the JSON response, with the exception of "Time" for which a custom model was included in the app:

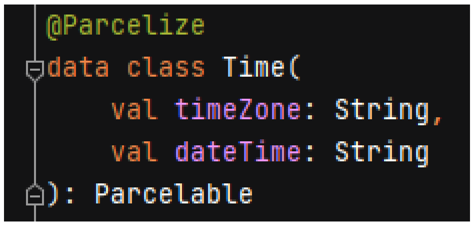


```
@Parcelize
data class EventModel(
    val summary: String,
    val id: String = "",
    val start: Time,
    val end: Time,
    val created: String = ""
) : Parcelable
```

Figure 16: EventModel within Juggle

4.4.1.5 Time

The following Time model is required to handle the start and end timeZone and dateTime returned for each event in the event's JSON response, where the data types match that of the JSON response:

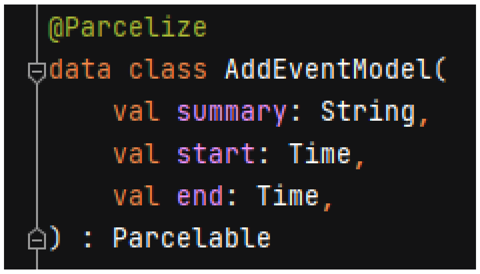


```
@Parcelize
data class Time(
    val timeZone: String,
    val dateTime: String
): Parcelable
```

Figure 17: Time model within Juggle

4.4.1.6 *Add Event*

The AddEventModel is used to compose the JSON request to add an event via the API. The above EventModel contains the fields id and created, which are not included in the POST add event request but are generated upon the addition of the event. The following is the model used for an Event being added to a calendar, where the Time model as detailed above is used for the start and end time and date details:

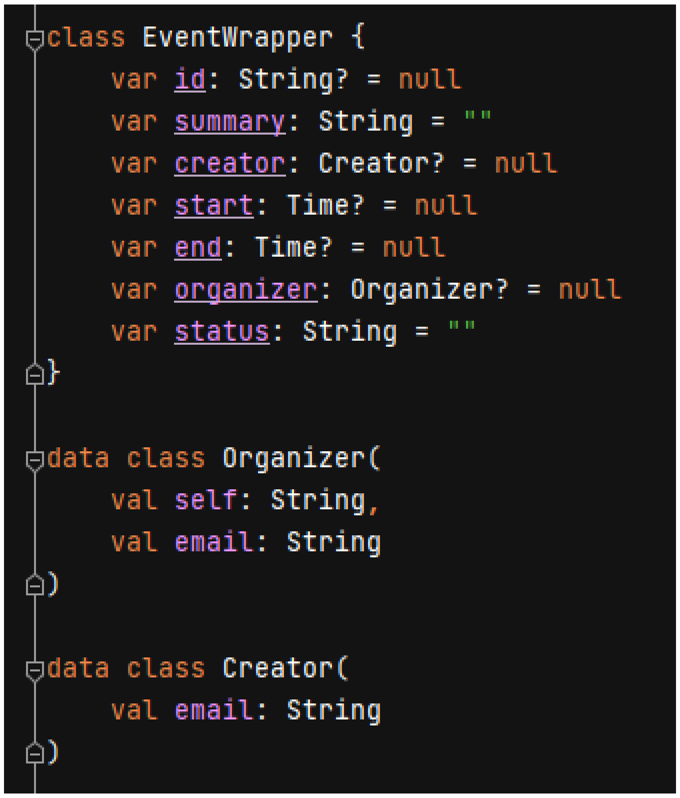


```
@Parcelize
data class AddEventModel(
    val summary: String,
    val start: Time,
    val end: Time,
) : Parcelable
```

Figure 18: AddEventModel within Juggle

4.4.1.7 *Event Wrapper*

The JSON response upon the addition of a new event is handled by the EventWrapper model, the data is then used to populate the Related Event model for saving to the database, the model is as follows:



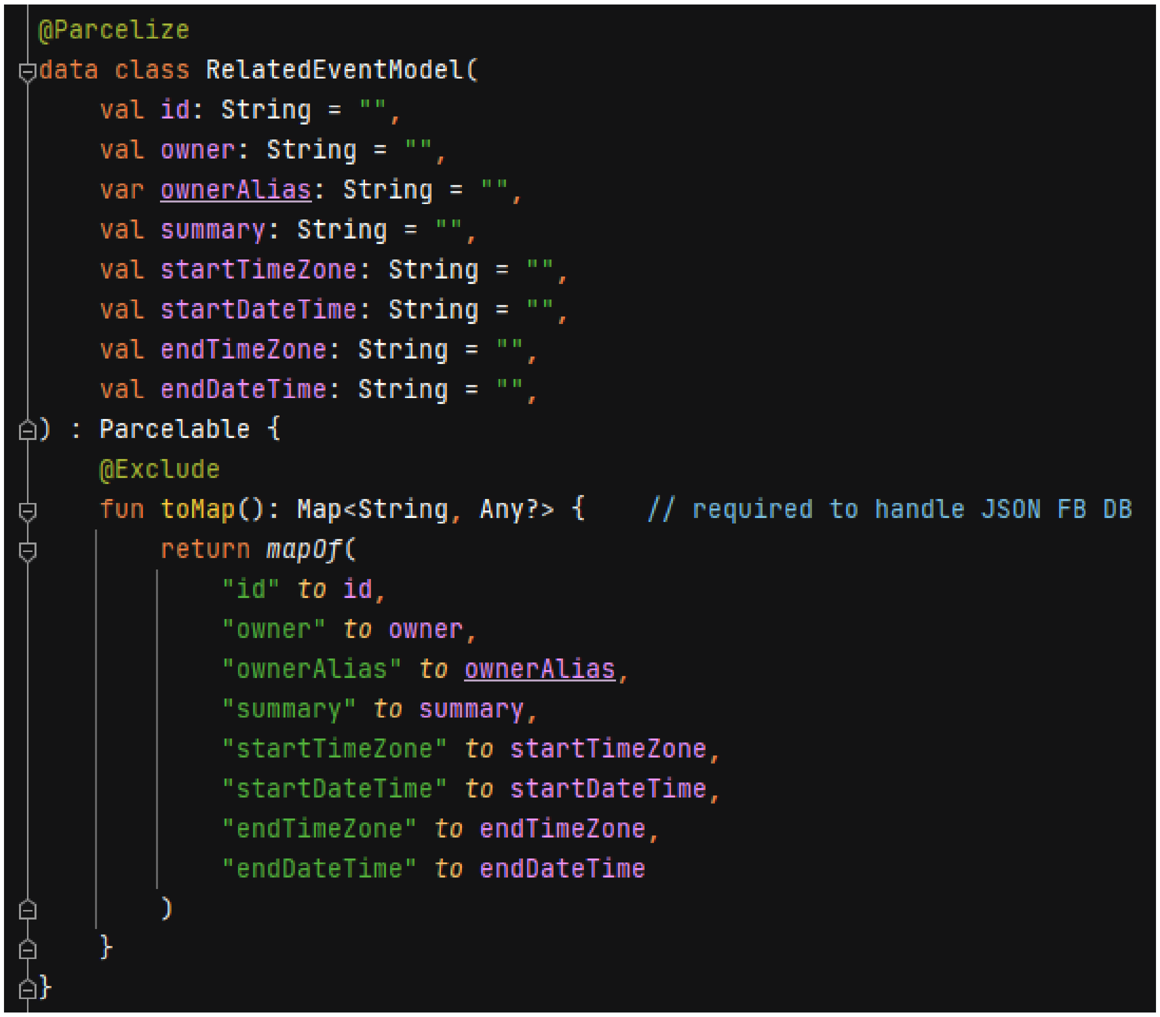
```kotlin
class EventWrapper {
    var id: String? = null
    var summary: String = ""
    var creator: Creator? = null
    var start: Time? = null
    var end: Time? = null
    var organizer: Organizer? = null
    var status: String = ""
}

data class Organizer(
    val self: String,
    val email: String
)

data class Creator(
    val email: String
)
```

Figure 19: EventWrapper model including Organizer and Creator sub models

Where Organizer and Creator are required to be modelled separately.

4.4.1.8 Related Event

The Related Event model in the app, that is also saved to the Realtime database, is as follows:



```kotlin
@Parcelize
data class RelatedEventModel(
    val id: String = "",
    val owner: String = "",
    var ownerAlias: String = "",
    val summary: String = "",
    val startTimeZone: String = "",
    val startDateTime: String = "",
    val endTimeZone: String = "",
    val endDateTime: String = "",
) : Parcelable {
    @Exclude
    fun toMap(): Map<String, Any?> {    // required to handle JSON FB DB
        return mapOf(
            "id" to id,
            "owner" to owner,
            "ownerAlias" to ownerAlias,
            "summary" to summary,
            "startTimeZone" to startTimeZone,
            "startDateTime" to startDateTime,
            "endTimeZone" to endTimeZone,
            "endDateTime" to endDateTime
        )
    }
}
```

Figure 20: RelatedEventModel within Juggle including the map for saving to the database

Where the following describes each field:

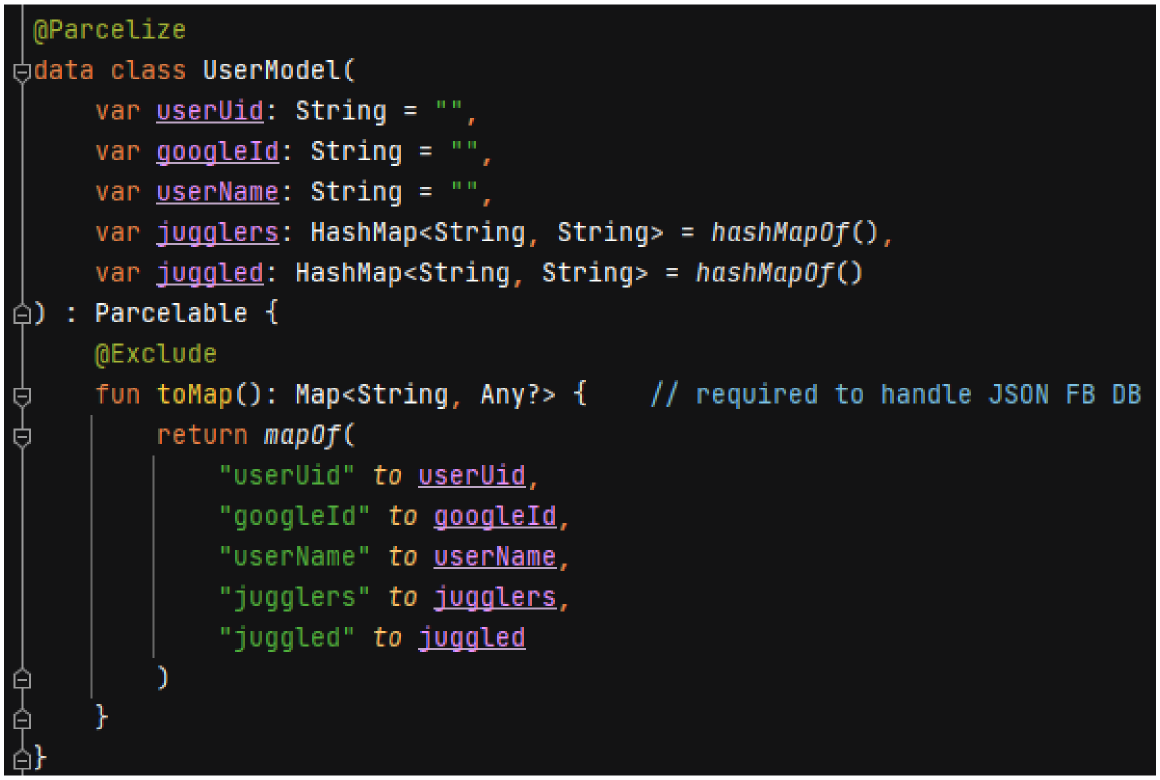
id:	the Uid of the newly created event as returned when the event is created via the API
owner:	the Google Id (one of the Jugglers) of the calendar to which the event has been added
ownerAlias:	the alias as assigned to the above owner at the time of the event creation
summary:	the summary of the new event, the alias of the juggled is appended to the summary
startTimeZone:	corresponds to the start Time.timeZone of the new event
startDateTime	corresponds to the start Time.dateTime of the new event

endTimeZone: corresponds to the end Time.timeZone of the new event

endDateTime: corresponds to the end Time.dateTime of the new event

4.4.1.9 User

The User model in the app, that is also saved to the Realtime database, is as follows:



```
@Parcelize
data class UserModel(
    var userUid: String = "",
    var googleId: String = "",
    var userName: String = "",
    var jugglers: HashMap<String, String> = hashMapOf(),
    var juggled: HashMap<String, String> = hashMapOf()
) : Parcelable {
    @Exclude
    fun toMap(): Map<String, Any?> {    // required to handle JSON FB DB
        return mapOf(
            "userUid" to userUid,
            "googleId" to googleId,
            "userName" to userName,
            "jugglers" to jugglers,
            "juggled" to juggled
        )
    }
}
```

Figure 21: UserModel within Juggle including the map for saving to the database

Where the following describes each field:

userUid: the Uid of the signed in Google User as returned when authenticated with Firebase

googleId: the email of the Google User that has been authenticated with Firebase that is currently signed in

userName the name attached to the Google account of Google User currently signed in

jugglers: a Kotlin HashMap where the key is the alias as entered by the user and the value is the calendar id of the Google calendar to which the alias is assigned and the role of Juggler is chosen

juggled:	a Kotlin HashMap where the key is the alias as entered by the user and the value is the calendar id of the Google calendar to which the alias is assigned and the role of Juggled is chosen

4.4.2 Realtime Database

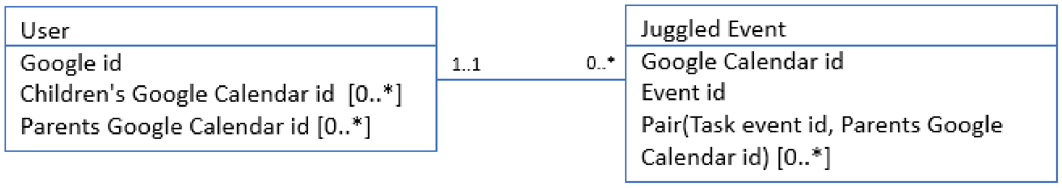

User
Google id
Children's Google Calendar id [0..*]
Parents Google Calendar id [0..*]
1..1
0..*
Juggled Event
Google Calendar id
Event id
Pair(Task event id, Parents Google Calendar id) [0..*]

Figure 22: Database - initial design

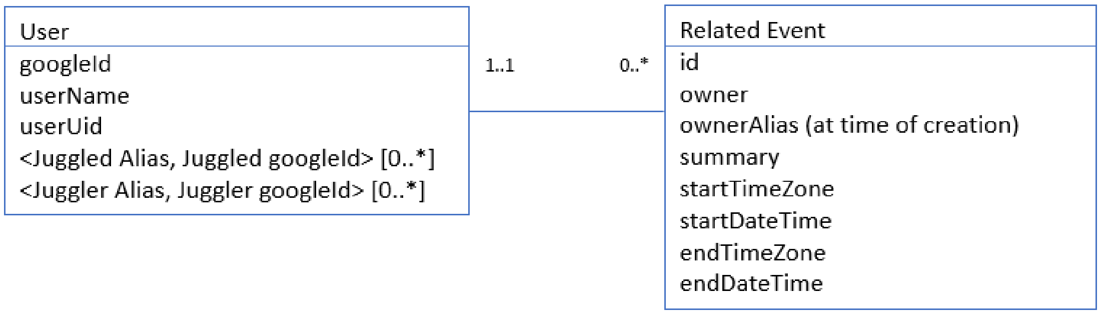
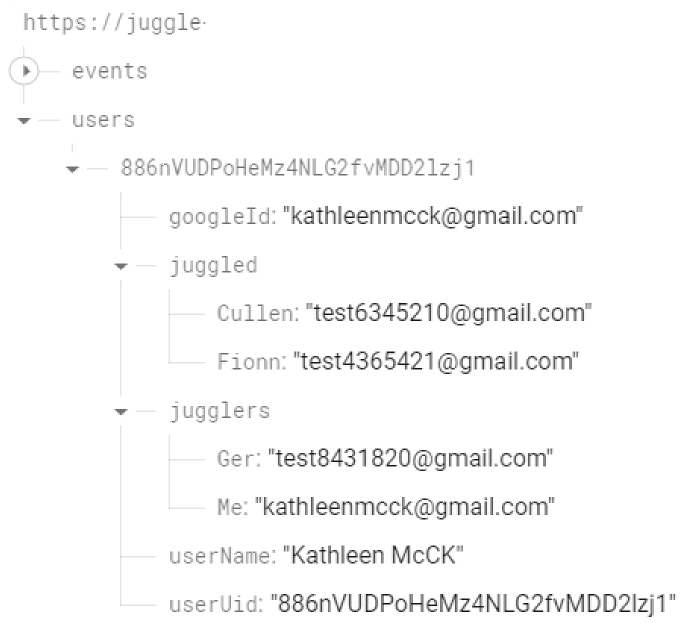

User
googleId
userName
userUid
<Juggled Alias, Juggled googleId> [0..*]
<Juggler Alias, Juggler googleId> [0..*]
1..1
0..*
Related Event
id
owner
ownerAlias (at time of creation)
summary
startTimeZone
startDateTime
endTimeZone
endDateTime

Figure 23: Database models - actual design

The User model has been expanded from the initial design, to include additional fields, googleId and userName as taken from the Google account of the signed in user but keeping them together in the database model has made that data easier for the app to access as it is from one source. The alias allows the user to specify which calendar belongs to who rather than continually use the, sometimes ambiguous, googleId.

The Juggled event entity has been replaced with the related event, and saved to the path in the database specified by the juggled event id, this again was for ease of access, as the app will require multiple related events per single juggled event in the Event view. If I'd used the Kotlin Pair the related event details available to the app via the database call would have been limited to only the event id and the google calendar to which it belonged so to display additional information, as actually stored, an API call to the related events google calendar would have been required.

4.4.2.1 User



```
https://juggle-
  events
  users
    886nVUDPoHeMz4NLG2fvMDD2lzj1
      googleId: "kathleenmcck@gmail.com"
      juggled
        Cullen: "test6345210@gmail.com"
        Fionn: "test4365421@gmail.com"
      jugglers
        Ger: "test8431820@gmail.com"
        Me: "kathleenmcck@gmail.com"
      userName: "Kathleen McCK"
      userUid: "886nVUDPoHeMz4NLG2fvMDD2lzj1"
```

Figure 24: Actual User Model in Database

The user of the app, requires a Google account and access to Google calendars to be juggled as well as access to Google calendar of other potential jugglers. The associated calendars need to be saved with that user's Google id so that the roles assigned to calendars by that user will persist.

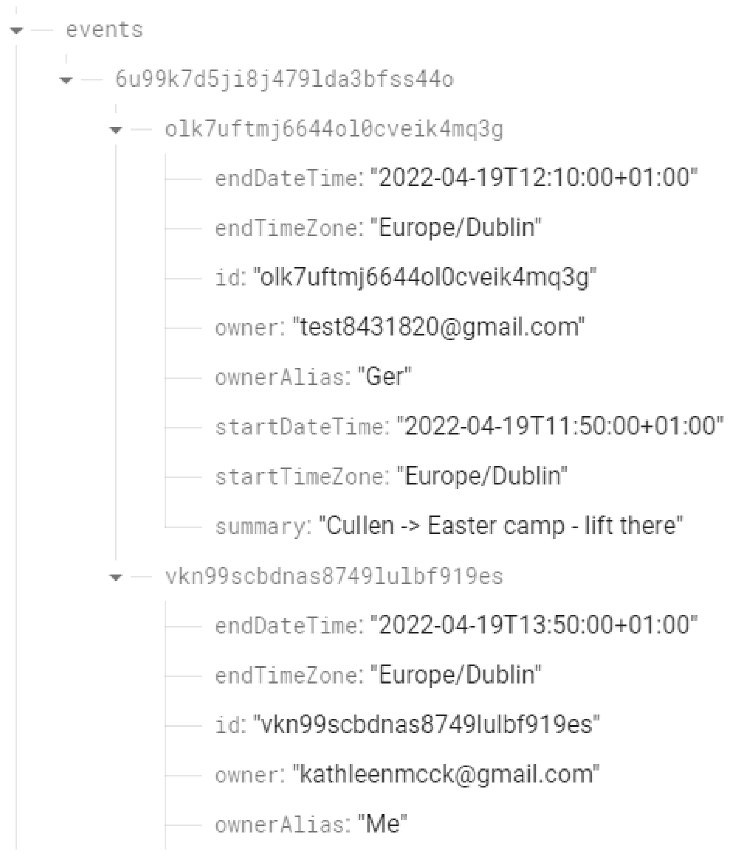
During the development of the app I expanded the initial User design to include a UID that would identify the specific child node of "users" for the logged in user and that the UID for the user in the database should be the UID of the authenticated user in Firebase.

Additionally to allow the inclusion of an alias for each Juggled or Jugglers' calendar I altered the model to use a HashMap of the alias and calendar id of each role, rather than an array of just the calendar id assigned to each role.

Lastly I added the username that is populated with the user's Google Account name for display during the app.

4.4.2.2 Juggled Event

Once a Child's event has been reviewed and tasks attached the child's Google calendar id, related task ids and some data will be saved to the data base using the path events\juggledEventId (where the Juggled event id is the id of the Google Calendar event in in the Child's calendar that is being juggler.). This will be used to display the already related events that have been added for an event in the Event details view in the app. In future development it will also be used to determine whether an event has been checked in the events list.



```
events
  6u99k7d5ji8j479lda3bfss44o
    olk7uftmj6644ol0cveik4mq3g
      endDateTime: "2022-04-19T12:10:00+01:00"
      endTimeZone: "Europe/Dublin"
      id: "olk7uftmj6644ol0cveik4mq3g"
      owner: "test8431820@gmail.com"
      ownerAlias: "Ger"
      startDateTime: "2022-04-19T11:50:00+01:00"
      startTimeZone: "Europe/Dublin"
      summary: "Cullen -> Easter camp - lift there"
    vkn99scbdnas8749lulbf919es
      endDateTime: "2022-04-19T13:50:00+01:00"
      endTimeZone: "Europe/Dublin"
      id: "vkn99scbdnas8749lulbf919es"
      owner: "kathleenmcck@gmail.com"
      ownerAlias: "Me"
```

Figure 25: Actual Related event in Database

5 Implementation

The development steps have been split out by iteration, as follows:

5.1 ITERATION 1 (19 FEB 22 -> 05 MAR 22)

- Set up a GitHub repository
- Within Android Studio created a new project with Bottom Navigation Activity
- Set up Firebase in the Firebase console with the Google Sign In method for my app
- Set up a new project in the Google console for my app and configured for Google Sign In
- Google Sign In implemented in new Activity
- Refactored the app to use Firebase authentication upon sign in
- Refactored to use the MVVM design
- Upon Google Sign In read and write access to Google Calendar requested via scopes based on the Google Sign In Options builder as discussed on StackOverflow.com (stackoverflow, 2019)
- Sign out and revoke access was also implemented
- Investigated the retrieval of the access token for use in the Google Calendar API calls however I could not retrieve that token following the Google sign-in

5.2 ITERATION 2 (06 MAR 22 - > 13 MAR 22)

Continued to investigate how to access the access token including the use of a Google calendar service (as used in the following repot on GitHub (Yuana, 2018))but to no avail.

At this point I made the decision to postpone the attempt to programmatically retrieve and use the temporary access token that I was expecting to be available upon Google Sign In and in order to continue making progress on the project I would continue to use a hardcoded access token, in the hope that I would have the time to revisit.

- Added a custom Calendar model and Event model to the app (based on a subset of the JSON data response expected)
- Added a custom Calendar List model and an Event List model to the app to handle the JSON list response.
- Added Retrofit to the android project, made a API call for calendars to which the logged user has access using a hardcoded access token (as retrieved using the Google Developers OAuth 2 playground (Google Developers, n.d.)) as the GET requests authorization header value.
- Calendar model summary and access role displayed upon login, using a recycler view.

5.3 ITERATION 3 (14 MAR 22 -> 20 MAR 22)

- Added the find calendar events functionality for a specific calendar id using an API GET request
- Added a recycler view to be displayed in the Events list fragment
- Added a Calendar on click event to the Calendar list recycler view that would navigate to the event list Fragment
- Added an event view fragment to display certain details of a specific event
- Added an Event on click event to the Event list recycler view that would navigate to the event view Fragment

5.4 ITERATION 4 (21 MAR 22 -> 27 MAR 22)

- Added the input fields for the new event to be added
- Added the post new event via API using that inputted data upon the click of a new button to the primary calendar of the logged in user.
- Refactored the event view layout

5.5 ITERATION 5 (27 MAR 22 -> 02 APR 22)

- Set up the project to use the Firebase Realtime Database in the Firebase console
- Added a Firebase object to integrate with the database
- Added a custom User model to the app
- Added a drop down (spinner) in the Calendar list recycler view with Juggled or Juggler role options as well as an input text alias
- Added the functionality to save a user to the project's Firebase database upon the click of a button in the Calendar view including both a Juggled and Jugglers HashMap of aliases and calendar ids.
- Implemented validation that the aliases were distinct as required for keys in a HashMap.
- Restricted calendars displayed in a list to those only with owner access

5.6 ITERATION 6 (03 APR 22 -> 18 APR 22)

- Creation of GitHub Page and Readme
- Added the functionality to save a related event to the project's Firebase database and the subsequent rendering of those events in a recycler view when viewing the Juggled event.
- Updated the add related event card to allow the choice of one persisted Juggler as the owner of the related task
- Add event functionality implemented, event to be added to all Jugglers and all Juggled for the User

- Multiple UI updates
- Reformatted the code, removed redundant comments.

6 Reflection

6.1 LEARNINGS

Android development is difficult to comprehend and it is difficult to find resources to help: I had a suspicion that this was the case during the mobile app development module however I was convinced that with a bit more effort, as part of the final project, the flow of android development, including the MVVM approach would become clear, as did the majority of the other course modules. However this was not the case, progress at every stage was slow and I didn't get to develop the functionality that I had originally hoped.

Android Development: Although I had already built an android app as part of the mobile app development module, I learned a lot more about: data models in both the retrieval of data using the API, posting data via the API, moving data within the app and saving and retrieving data from the Realtime database; Retrofit HTTP requests; retrieving input text and spinner position for items in a recycler view but also the Kotlin language (for example the for loop validating the alias and roles assigned to calendars available).

Google APIs: There are a vast number of Google APIs available for use by developers. The potential for using even the Calendar API is huge, for which I've only begun to explore as part of this project. I almost automatically grant Google scope access when requested to a variety of mobile and web apps without thinking too much about it, so it has been great to get a better understanding of how they all interact with Google including the use of OAuth authorization. My own app was unverified by Google, so it's been good to realize the process employed by Google to authorize an app.

Authorization tokens are very valuable: One of the main challenges of implementing the Google API functionality was the absence of a short lived access token that I was expecting as part of the Google Sign in process flow. I can't help but feel that it was very close but without it explicitly showing itself, hardcoding was the only way forward. I could have easily spent my entire development time searching for that access token so I was grateful to realise progress was possible without it and continue developing my app even though it was not as I had planned. Thankfully the "OAuth 2.0 Playground" (see Appendix B) was available to generate that access token so that I could continue to build the majority of the intended app functionality.

Timeboxing: My approach to solving a problem has always been to spend as much time as it takes to get a solution and it had worked well for me in the past. At most stages of this project my goal of a functioning app seemed unachievable and with that, sometimes panic, unretrievable time would vanish with little results. Another lesson from the elusive access token was to decide upon an amount of time to be spent on a problem and once that time was up move on to what I could make progress on.

My planning did not go to plan: Initial expectations were to plan sprints by start and end dates, neatly filling each with clear functionality to be implemented and upon completion of those plans move onto the next iteration. Thanks to the confidence inspiring Google API documentation, it was a forgone conclusion for me, that the functionality I had planned would be easily implemented. However when I began to struggle with accessing the Google API data that initial plan was abandoned for a much more short-term-goal-focused approach, I made some use of Trello to name development and documentation tasks but I struggled to estimate efforts required and so worked through them as much as I could in a sensible order. Also as a result of the lack of confidence in what I was doing, my Git repository commits were to a development branch only, many times during development I reverted all the outstanding file changes if the changes weren't translating to my expected additional functionality, so only when I was confident that the complete commit worked did I in fact commit the changes. (See Appendix C for commit history and Trello screenshot.)

6.2 ACHIEVEMENTS

I have built a native Android mobile application, using Kotlin, where a User can successfully sign in using their existing Google account and grant permission to the app for read/write access to their Google calendar, as well as being authenticated with Firebase.

The app can get calendar specifics using the Google calendar APIs and within the app handle that data response using custom built data models, it can also post event specifics using the Google calendar APIs and again handle those responses in a similar way.

The app can also save to a Firebase Realtime database, a combination of data as retrieved using those APIs and data entered by the User, as well as retrieve data from that database.

The app can display data sourced from all of the above (Google Sign In, Firebase Authentication, Google calendar APIs and Firebase Realtime database) and move it through the app and display to the User.

6.2.1 Future Development

Future development of Juggle would prioritise, the following, as per the initial design:

Get and use the API Access token programmatically:

As mentioned already, the Google API access token has so far been beyond my reach, there seems to be three possible approaches that would need to be reviewed again and a solution found, they are:

1. Upon Google Sign In use the given “short-lived access token”, this is the approach as described in the previous section “OAuth Authorization”, step 5

2. Create an API call using the returned authorization code upon Google sign in, as per the POST request in the “OAuth 2.0 Playground” (see Appendix B, specifically exchanging the authorization token for the access token)

3. Google APIs services calendar as described at (Google Developers, 2022) and used in the following Kotlin project on GitHub (Yuana, 2018). This approach would seem to handle the access token but also the API calls and presumably remove the need for the current Retrofit methodology implemented in the app.

Ensure the Juggled events are for all User’s Juggled calendars:

The app’s functionality would need to be expanded so that the GET events request would be generated for each of the Juggled calendars and the responses collated for review in the Events List view, as described here.

Determine and display whether the Event is already checked:

As per the initial design the Events List view should show a checked indicator per event, the idea here was to highlight more easily the Juggled events that need attention. This functionality would require a check on whether related events exist in the database, optionally the User could assign a checked indicator per Juggled event, this would require the addition of a Juggled Event model to the app and database.

Additional functionality notions:

- Date and time validation for new events upon User input, including time zone handling (currently default values) and all day event modelling (i.e. start and end is a date only)
- Handling of unexpected/errors in API and Database responses
- Checking Juggler’s for free time before creating new related events
- Handling the scenario where Juggled have events in common and so related tasks could also be merged
- Inviting fellow Jugglers to the app, or requesting access to calendars from the app
- Integrate the dinner event creation with adding ingredients to a shared shopping list, could also retrieve previous event details for reuse, could also assign event to a Juggler

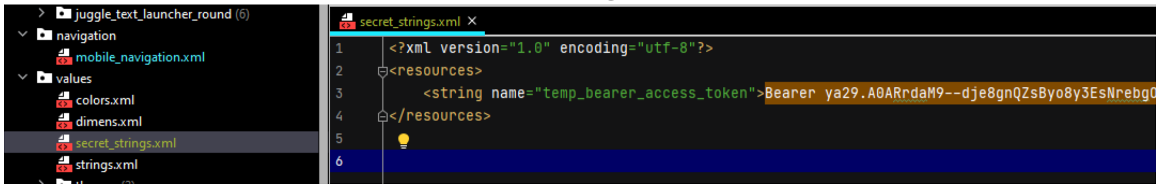
6.3 CHALLENGES

Throughout the Project effort I hit many challenges, they could be categorized as follows:

The search for the API access token:

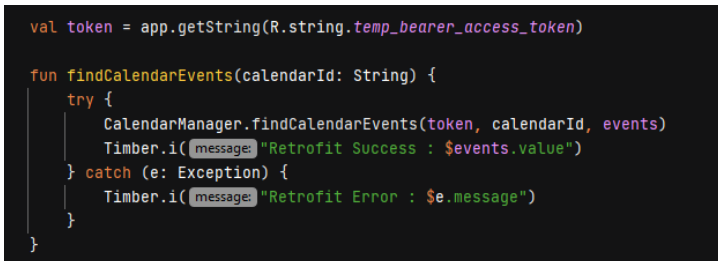
I've mentioned this challenge many times already and the possible solutions are included in Future Work above. To be able to continue with development using the calendar API in real-time with real data I:

1. Included the hardcoded value in a "secret_strings.xml" file:



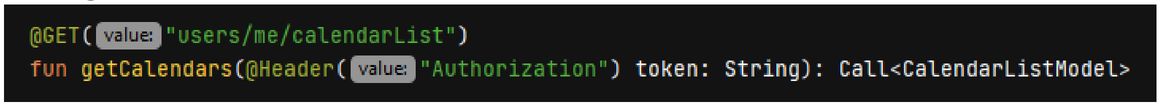
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="temp_bearer_access_token">Bearer ya29.A0ARrdaM9--dje8gnQZsByo8y3EsNrebgO
</resources>
```

2. Referenced that string resource in the View Models and included it as a parameter in relevant functions:



```
val token = app.getString(R.string.temp_bearer_access_token)

fun findCalendarEvents(calendarId: String) {
    try {
        CalendarManager.findCalendarEvents(token, calendarId, events)
        Timber.i( message: "Retrofit Success : $events.value")
    } catch (e: Exception) {
        Timber.i( message: "Retrofit Error : $e.message")
    }
}
```

3. Passing it as a Authorization Header in the Retrofit calls:



```
@GET( value: "users/me/calendarList")
fun getCalendars(@Header( value: "Authorization") token: String): Call<CalendarListModel>
```

The lack of available Kotlin examples and discussions:

Again I've mentioned this already but when no clear Kotlin solution was found at the usual online resources, I sometimes could piece something together with Android Studio prompts or converting from a Java solution, but at times had to abandon that particular effort.

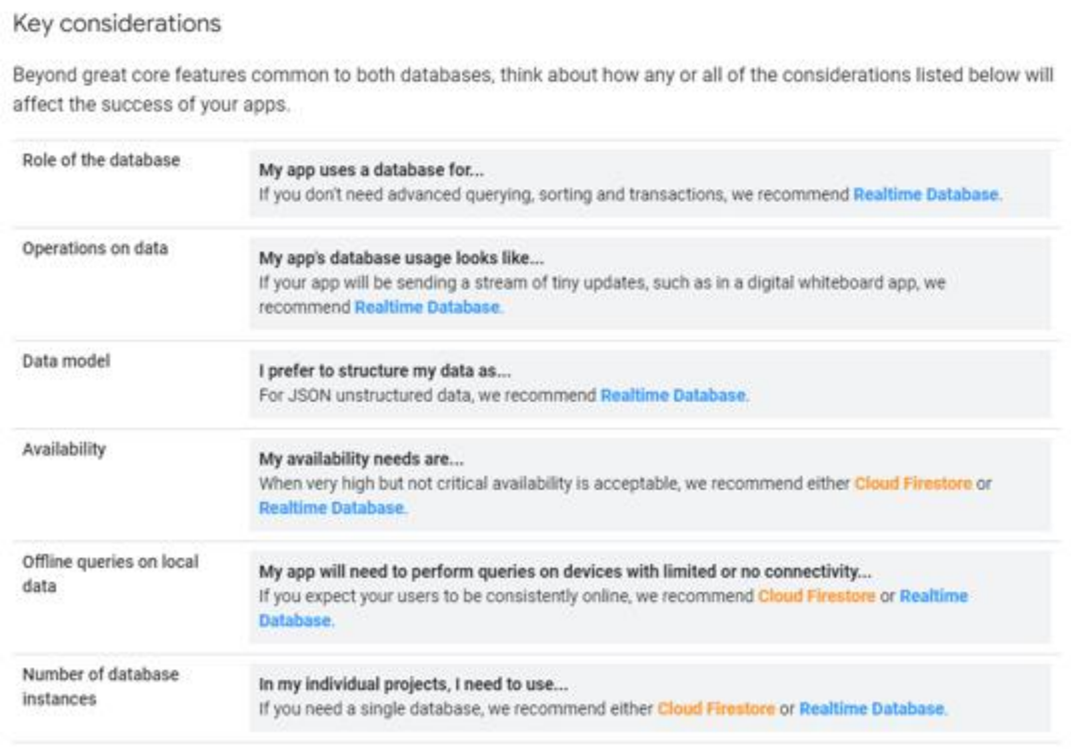
Time:

This was a big challenge for me, the "Juggle" is real!

7 Appendices

7.1 APPENDIX A

Figure 26: Choosing a Firebase Database, following screenshot as per (Google Developers, 2022)



Key considerations

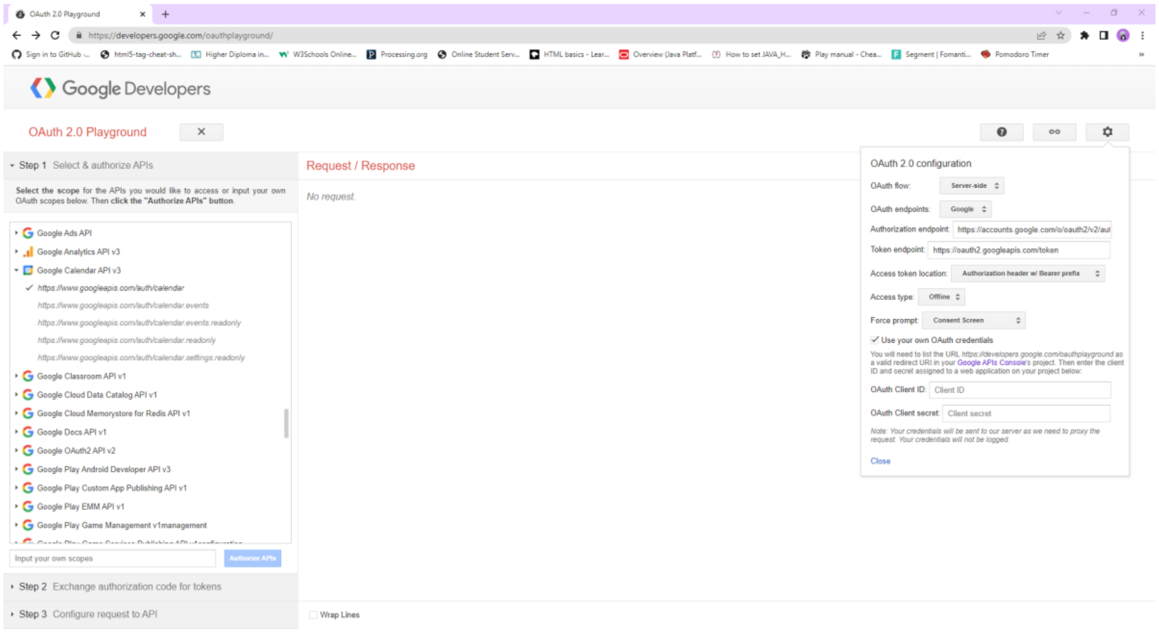
Beyond great core features common to both databases, think about how any or all of the considerations listed below will affect the success of your apps.

Role of the database	**My app uses a database for...** If you don't need advanced querying, sorting and transactions, we recommend Realtime Database.
Operations on data	**My app's database usage looks like...** If your app will be sending a stream of tiny updates, such as in a digital whiteboard app, we recommend Realtime Database.
Data model	**I prefer to structure my data as...** For JSON unstructured data, we recommend Realtime Database.
Availability	**My availability needs are...** When very high but not critical availability is acceptable, we recommend either Cloud Firestore or Realtime Database.
Offline queries on local data	**My app will need to perform queries on devices with limited or no connectivity...** If you expect your users to be consistently online, we recommend Cloud Firestore or Realtime Database.
Number of database instances	**In my individual projects, I need to use...** If you need a single database, we recommend either Cloud Firestore or Realtime Database.

7.2 APPENDIX B

Stepping through the OAuth flow using the Google Developers OAuth 2.0 Playground

a) On the right hand side, enter the Juggle app's Client ID and Client secret, on the left hand side choose the scope https://www.googleapis.com/auth/calendar, and "Authorize APIs"



OAuth 2.0 Playground

https://developers.google.com/oauthplayground/

Google Developers

OAuth 2.0 Playground

Step 1 Select & authorize APIs

Select the scope for the APIs you would like to access or input your own OAuth scopes below. Then **click the "Authorize APIs" button**.

- Google Ads API
- Google Analytics API v3
- Google Calendar API v3
  - ✓ *https://www.googleapis.com/auth/calendar*
  - *https://www.googleapis.com/auth/calendar.events*
  - *https://www.googleapis.com/auth/calendar.events.readonly*
  - *https://www.googleapis.com/auth/calendar.readonly*
  - *https://www.googleapis.com/auth/calendar.settings.readonly*
- Google Classroom API v1
- Google Cloud Data Catalog API v1
- Google Cloud Memorystore for Redis API v1
- Google Docs API v1
- Google OAuth2 API v2
- Google Play Android Developer API v3
- Google Play Custom App Publishing API v1
- Google Play EMM API v1
- Google Play Game Management v1management

Input your own scopes | Authorize APIs

Step 2 Exchange authorization code for tokens

Step 3 Configure request to API

Request / Response

No request.

Wrap Lines

OAuth 2.0 configuration

OAuth flow: Server-side

OAuth endpoints: Google

Authorization endpoint: https://accounts.google.com/o/oauth2/v2/aut

Token endpoint: https://oauth2.googleapis.com/token

Access token location: Authorization header w/ Bearer prefix

Access type: Offline

Force prompt: Consent Screen

✓ Use your own OAuth credentials

You will need to list the URL *https://developers.google.com/oauthplayground* as a valid redirect URI in your Google APIs Console's project. Then enter the client ID and secret assigned to a web application on your project below:

OAuth Client ID: Client ID

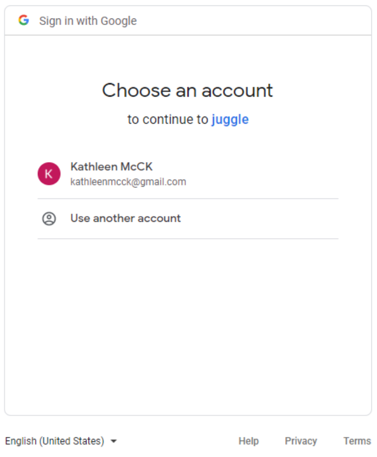
OAuth Client secret: Client secret

Note: Your credentials will be sent to our server as we need to proxy the request. Your credentials will not be logged.

Close

Figure 27: OAuth flow, add OAuth credentials and authorize APIs (Google Developers, n.d.)

b) Choose a Google account



Sign in with Google

Choose an account

to continue to juggle

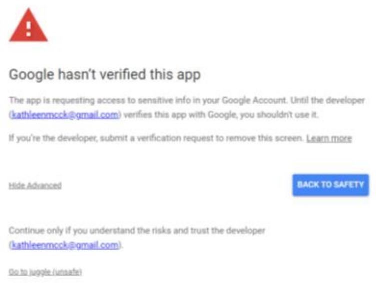
Kathleen McCK
kathleenmcck@gmail.com

Use another account

English (United States) Help Privacy Terms

Figure 28: OAuth flow, choose an account (Google Developers, n.d.)

c) Confirm continue for unverified app, "Go to Juggle (unsafe)"



Google hasn't verified this app

The app is requesting access to sensitive info in your Google Account. Until the developer (kathleenmcck@gmail.com) verifies this app with Google, you shouldn't use it.

If you're the developer, submit a verification request to remove this screen. Learn more

Hide Advanced

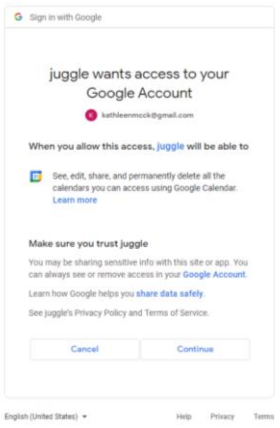
BACK TO SAFETY

Continue only if you understand the risks and trust the developer (kathleenmcck@gmail.com).

Go to juggle (unsafe)

Figure 29: OAuth flow, unverified app warning (Google Developers, n.d.)

d) Grant access to requested calendar scopes



Sign in with Google

juggle wants access to your Google Account

kathleenmcck@gmail.com

When you allow this access, juggle will be able to

See, edit, share, and permanently delete all the calendars you can access using Google Calendar. Learn more

Make sure you trust juggle

You may be sharing sensitive info with this site or app. You can always see or remove access in your Google Account.

Learn how Google helps you share data safely.

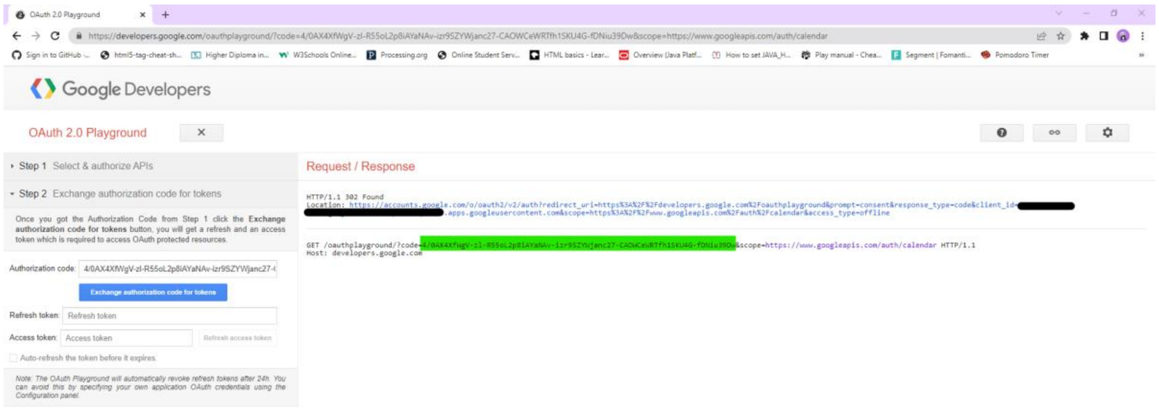
See juggle's Privacy Policy and Terms of Service.

Cancel Continue

English (United States) Help Privacy Terms

Figure 30: OAuth flow, Consent to requested scopes screen (Google Developers, n.d.)

e) Response includes authorization code



OAuth 2.0 Playground

https://developers.google.com/oauthplayground/?code=4/0AX4XfWgV-zl-R55oL2p8iAYaNAv-izr9SZYWjanc27-CAOWCeWRTfh1SKU4G-fDNiu39Dw&scope=https://www.googleapis.com/auth/calendar

Sign in to GitHub ... html5-tag-cheat-sh... Higher Diploma in... W3Schools Online... Processing.org Online Student Serv... HTML basics - Lear... Overview (Java Platf... How to set JAVA_H... Play manual - Chea... Segment | Fomanti... Pomodoro Timer

Google Developers

OAuth 2.0 Playground

Step 1 Select & authorize APIs

Step 2 Exchange authorization code for tokens

Once you got the Authorization Code from Step 1 click the **Exchange authorization code for tokens** button, you will get a refresh and an access token which is required to access OAuth protected resources.

Authorization code: 4/0AX4XfWgV-zl-R55oL2p8iAYaNAv-izr9SZYWjanc27-(

Exchange authorization code for tokens

Refresh token: Refresh token

Access token: Access token Refresh access token

Auto-refresh the token before it expires.

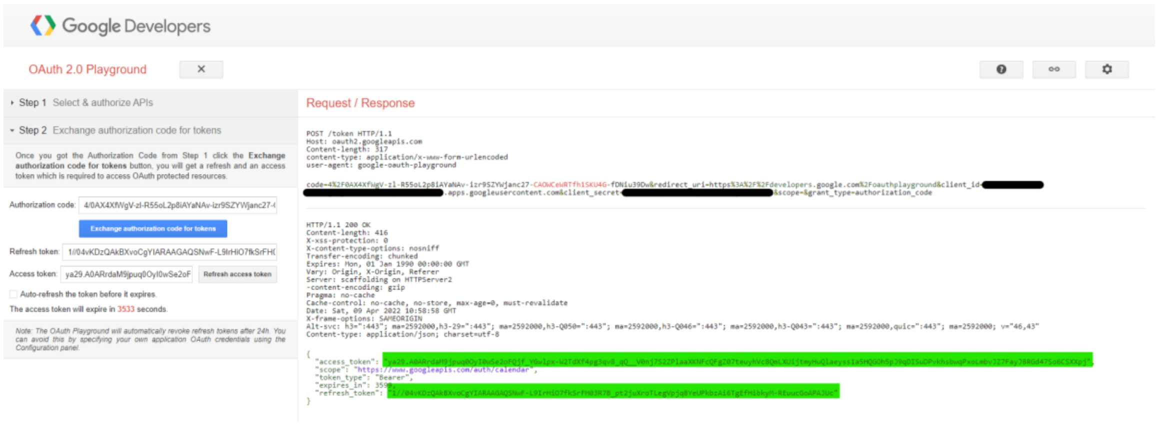
Note: The OAuth Playground will automatically revoke refresh tokens after 24h. You can avoid this by specifying your own application OAuth credentials using the Configuration panel.

Request / Response

```
HTTP/1.1 302 Found
Location: https://accounts.google.com/o/oauth2/v2/auth?redirect_uri=https%3A%2F%2Fdevelopers.google.com%2Foauthplayground&prompt=consent&response_type=code&client_id=[redacted]
[redacted].apps.googleusercontent.com&scope=https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fcalendar&access_type=offline

GET /oauthplayground/?code=4/0AX4XfWgV-zl-R55oL2p8iAYaNAv-izr9SZYWjanc27-CAOWCeWRTfh1SKU4G-fDNiu39Dw&scope=https://www.googleapis.com/auth/calendar HTTP/1.1
Host: developers.google.com
```

Figure 31: OAuth flow, retrieval of authorization code (Google Developers, n.d.)

f) Exchange authorization code for access and refresh tokens


Google Developers
OAuth 2.0 Playground
×
Step 1 Select & authorize APIs
Step 2 Exchange authorization code for tokens
Exchange authorization code for tokens
Refresh access token
Request / Response

Once you got the Authorization Code from Step 1 click the **Exchange authorization code for tokens** button, you will get a refresh and an access token which is required to access OAuth protected resources.

Authorization code: 4/0AX4XfWgV-zl-R55oL2p8iAYaNAv-izr9SZYWjanc27-(

Refresh token: 1//04vKDzQAkBXvoCgYIARAAGAQSNwF-L9IrHiO7fkSrFH(

Access token: ya29.A0ARrdaM9jpuq0OyI0wSe2oF

Auto-refresh the token before it expires.

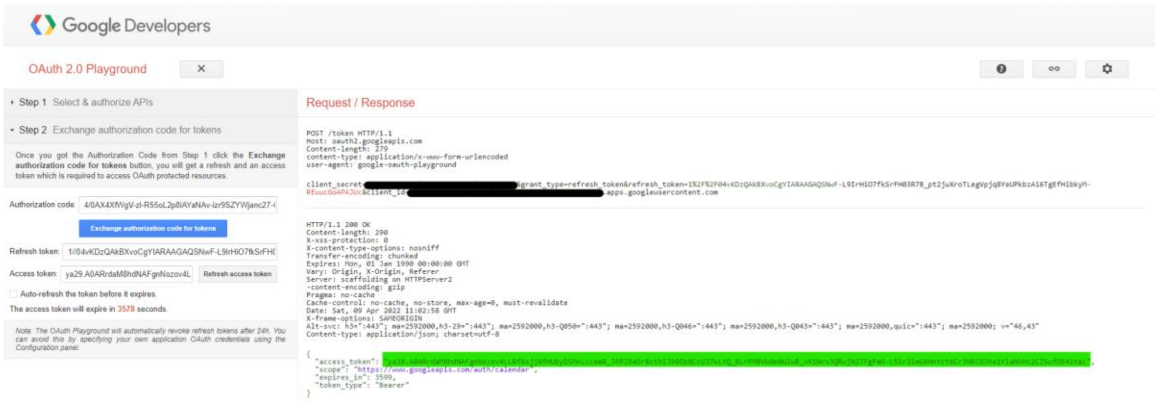
The access token will expire in 3533 seconds.

Note: The OAuth Playground will automatically revoke refresh tokens after 24h. You can avoid this by specifying your own application OAuth credentials using the Configuration panel.

```
POST /token HTTP/1.1
Host: oauth2.googleapis.com
Content-length: 317
content-type: application/x-www-form-urlencoded
user-agent: google-oauth-playground

code=4%2F0AX4XfWgV-zl-R55oL2p8iAYaNAv-izr9SZYWjanc27-CAOWCeWRTfh1SKU4G-fDNiu39Dw&redirect_uri=https%3A%2F%2Fdevelopers.google.com%2Foauthplayground&client_id=[redacted]
[redacted].apps.googleusercontent.com&client_secret=[redacted]&scope=&grant_type=authorization_code

HTTP/1.1 200 OK
Content-length: 416
X-xss-protection: 0
X-content-type-options: nosniff
Transfer-encoding: chunked
Expires: Mon, 01 Jan 1990 00:00:00 GMT
Vary: Origin, X-Origin, Referer
Server: scaffolding on HTTPServer2
-content-encoding: gzip
Pragma: no-cache
Cache-control: no-cache, no-store, max-age=0, must-revalidate
Date: Sat, 09 Apr 2022 10:58:58 GMT
X-frame-options: SAMEORIGIN
Alt-svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000,h3-Q050=":443"; ma=2592000,h3-Q046=":443"; ma=2592000,h3-Q043=":443"; ma=2592000,quic=":443"; ma=2592000; v="46,43"
Content-type: application/json; charset=utf-8

{
  "access_token": "ya29.A0ARrdaM9jpuq0OyI0wSe2oFQjf_YGwlpx-W2TdXf4pg3qv8_qQ__V0nj752ZPlaaXKNFcQFgZ07teuyhVc8QmLXUijtmyHuQlaeyss1a5HQGOh5pJ9qDISuOPvkhsbwqPxoLmbvJZ7FayJ8RGd47So6CSXXpj",
  "scope": "https://www.googleapis.com/auth/calendar",
  "token_type": "Bearer",
  "expires_in": 3599,
  "refresh_token": "1//04vKDzQAkBXvoCgYIARAAGAQSNwF-L9IrHiO7fkSrFH03R78_pt2juXroTLegVpjq8YeUPkbzAi6TgEfHibkyM-REuucGoAPAJUc"
}
```

Figure 32: OAuth flow, exchange authorization code (Google Developers, n.d.)

g) Can also refresh that token


Google Developers
OAuth 2.0 Playground
×
Step 1 Select & authorize APIs
Step 2 Exchange authorization code for tokens
Exchange authorization code for tokens
Refresh access token
Request / Response

Once you got the Authorization Code from Step 1 click the **Exchange authorization code for tokens** button, you will get a refresh and an access token which is required to access OAuth protected resources.

Authorization code: 4/0AX4XfWgV-zl-R55oL2p8iAYaNAv-izr9SZYWjanc27-(

Refresh token: 1//04vKDzQAkBXvoCgYIARAAGAQSNwF-L9IrHiO7fkSrFH(

Access token: ya29.A0ARrdaM8hdNAFgnNozov4L

Auto-refresh the token before it expires.

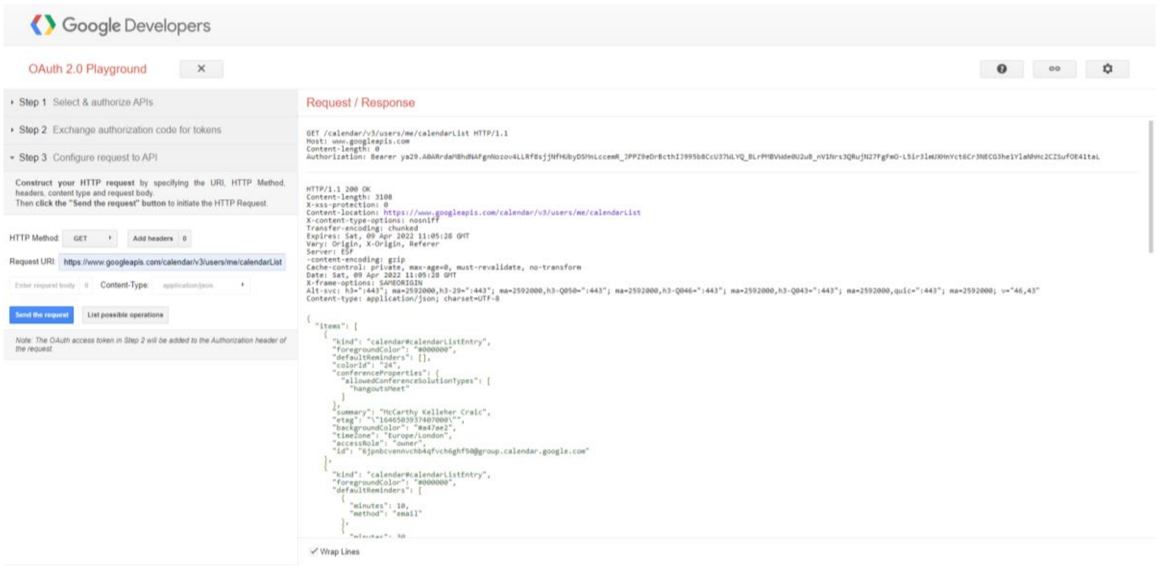
The access token will expire in 3578 seconds.

Note: The OAuth Playground will automatically revoke refresh tokens after 24h. You can avoid this by specifying your own application OAuth credentials using the Configuration panel.

```
POST /token HTTP/1.1
Host: oauth2.googleapis.com
Content-length: 279
content-type: application/x-www-form-urlencoded
user-agent: google-oauth-playground

client_secret=[redacted]&grant_type=refresh_token&refresh_token=1%2F%2F04vKDzQAkBXvoCgYIARAAGAQSNwF-L9IrHiO7fkSrFH03R78_pt2juXroTLegVpjq8YeUPkbzAi6TgEfHibkyM-
REuucGoAPAJUc&client_id=[redacted].apps.googleusercontent.com

HTTP/1.1 200 OK
Content-length: 290
X-xss-protection: 0
X-content-type-options: nosniff
Transfer-encoding: chunked
Expires: Mon, 01 Jan 1990 00:00:00 GMT
Vary: Origin, X-Origin, Referer
Server: scaffolding on HTTPServer2
-content-encoding: gzip
Pragma: no-cache
Cache-control: no-cache, no-store, max-age=0, must-revalidate
Date: Sat, 09 Apr 2022 11:02:58 GMT
X-frame-options: SAMEORIGIN
Alt-svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000,h3-Q050=":443"; ma=2592000,h3-Q046=":443"; ma=2592000,h3-Q043=":443"; ma=2592000,quic=":443"; ma=2592000; v="46,43"
Content-type: application/json; charset=utf-8

{
  "access_token": "ya29.A0ARrdaM8hdNAFgnNozov4LLRf6sjjNfMUbyDSMnLccemR_JPPZ0eOr8cthIJ995b8CcU37WLYQ_8LrPM8Vvde0U2uB_AV1Nrs3QRujN27FgFwO-L51r3lmUXHnYct6Cr3NECG3he1YlaNhHc2CZSufOE41taL",
  "scope": "https://www.googleapis.com/auth/calendar",
  "expires_in": 3599,
  "token_type": "Bearer"
}
```

Figure 33: OAuth flow, refresh access token (Google Developers, n.d.)

h) Send a request for a specified Request URI


Google Developers
OAuth 2.0 Playground
Step 1 Select & authorize APIs
Step 2 Exchange authorization code for tokens
Step 3 Configure request to API

Construct your HTTP request by specifying the URI, HTTP Method, headers, content type and request body.
Then **click the "Send the request" button** to initiate the HTTP Request.

HTTP Method: GET
Add headers 0
Request URI: https://www.googleapis.com/calendar/v3/users/me/calendarList
Enter request body 0
Content-Type: application/json

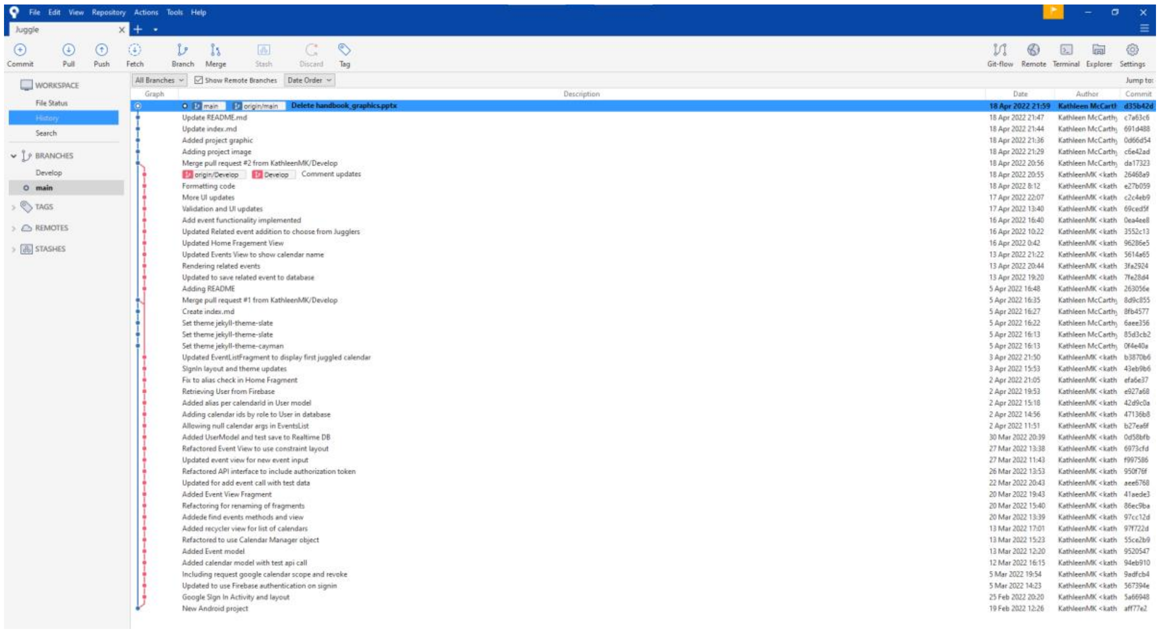
Send the request
List possible operations

Note: The OAuth access token in Step 2 will be added to the Authorization header of the request.

```
GET /calendar/v3/users/me/calendarList HTTP/1.1
Host: www.googleapis.com
Content-length: 0
Authorization: Bearer ya29.A0ARrdaM8hdNAfgnNozov4LLRf8sjjNfHUbyD5MnLccemR_JPPZ9eDrBcthIJ995b8CcU37WLYQ_BLrPMBVWde0U2uB_nV1Nrs3QRujN27FgFmO-L5ir3lmUXHnYct6Cr3NECG3he1YlaNhHc2CZSufOE41taL

HTTP/1.1 200 OK
Content-length: 3108
X-xss-protection: 0
Content-location: https://www.googleapis.com/calendar/v3/users/me/calendarList
X-content-type-options: nosniff
Transfer-encoding: chunked
Expires: Sat, 09 Apr 2022 11:05:28 GMT
Vary: Origin, X-Origin, Referer
Server: ESF
-content-encoding: gzip
Cache-control: private, max-age=0, must-revalidate, no-transform
Date: Sat, 09 Apr 2022 11:05:28 GMT
X-frame-options: SAMEORIGIN
Alt-svc: h3=":443"; ma=2592000,h3-29=":443"; ma=2592000,h3-Q050=":443"; ma=2592000,h3-Q046=":443"; ma=2592000,h3-Q043=":443"; ma=2592000,quic=":443"; ma=2592000; v="46,43"
Content-type: application/json; charset=UTF-8

{
  "items": [
    {
      "kind": "calendar#calendarListEntry",
      "foregroundColor": "#000000",
      "defaultReminders": [],
      "colorId": "24",
      "conferenceProperties": {
        "allowedConferenceSolutionTypes": [
          "hangoutsMeet"
        ]
      },
      "summary": "McCarthy Kelleher Craic",
      "etag": "\"1646503937407000\"",
      "backgroundColor": "#a47ae2",
      "timeZone": "Europe/London",
      "accessRole": "owner",
      "id": "6jpnbcvennvchb4qfvch6ghf50@group.calendar.google.com"
    },
    {
      "kind": "calendar#calendarListEntry",
      "foregroundColor": "#000000",
      "defaultReminders": [
        {
          "minutes": 10,
          "method": "email"
        },
        {
          "minutes": 30
```

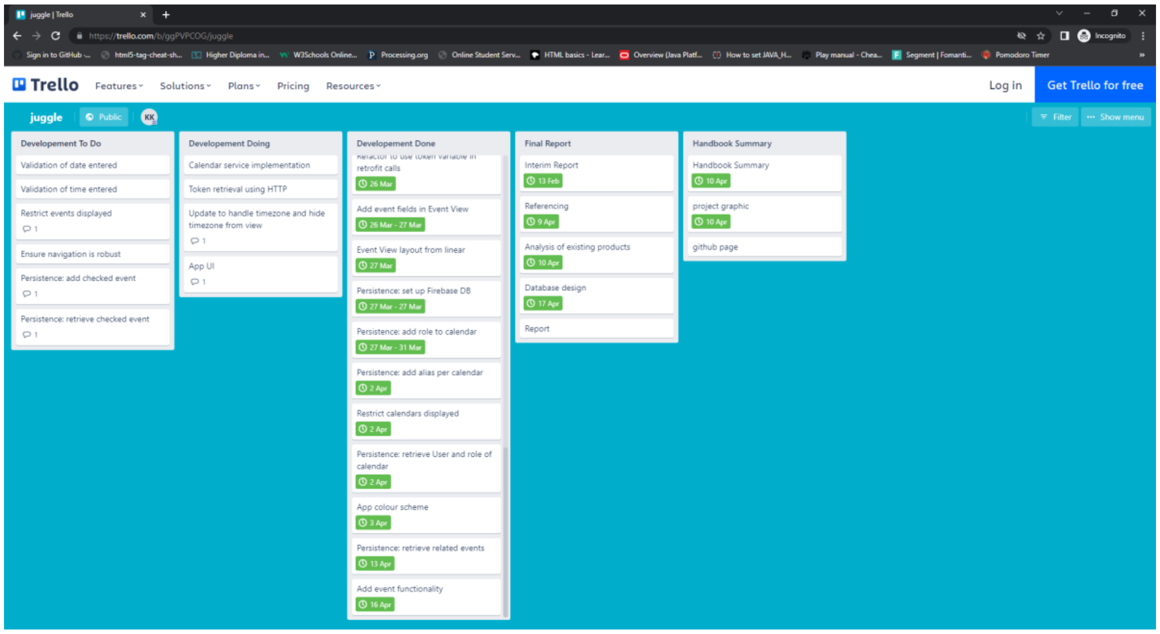
Figure 34: OAuth flow, send API request example (Google Developers, n.d.)

7.3 APPENDIX C



Description	Date	Author	Commit
main origin/main **Delete handbook_graphics.pptx**	**18 Apr 2022 21:59**	**Kathleen McCartl**	**d35b42d**
Update README.md	18 Apr 2022 21:47	Kathleen McCarthy	c7a63c6
Update index.md	18 Apr 2022 21:44	Kathleen McCarthy	691d488
Added project graphic	18 Apr 2022 21:36	Kathleen McCarthy	0d66d54
Adding project image	18 Apr 2022 21:29	Kathleen McCarthy	c6e42ad
Merge pull request #2 from KathleenMK/Develop	18 Apr 2022 20:56	Kathleen McCarthy	da17323
origin/Develop Develop Comment updates	18 Apr 2022 20:55	KathleenMK <kath	26468a9
Formatting code	18 Apr 2022 8:12	KathleenMK <kath	e27b059
More UI updates	17 Apr 2022 22:07	KathleenMK <kath	c2c4eb9
Validation and UI updates	17 Apr 2022 13:40	KathleenMK <kath	69ced5f
Add event functionality implemented	16 Apr 2022 16:40	KathleenMK <kath	0ea4ee8
Updated Related event addition to choose from Jugglers	16 Apr 2022 10:22	KathleenMK <kath	3552c13
Updated Home Fragement View	16 Apr 2022 0:42	KathleenMK <kath	96286e5
Updated Events View to show calendar name	13 Apr 2022 21:22	KathleenMK <kath	5614a65
Rendering related events	13 Apr 2022 20:44	KathleenMK <kath	3fa2924
Updated to save related event to database	13 Apr 2022 19:20	KathleenMK <kath	7fe28d4
Adding README	5 Apr 2022 16:48	KathleenMK <kath	263056e
Merge pull request #1 from KathleenMK/Develop	5 Apr 2022 16:35	Kathleen McCarthy	8d9c855
Create index.md	5 Apr 2022 16:27	Kathleen McCarthy	8fb4577
Set theme jekyll-theme-slate	5 Apr 2022 16:22	Kathleen McCarthy	6aee356
Set theme jekyll-theme-slate	5 Apr 2022 16:13	Kathleen McCarthy	85d3cb2
Set theme jekyll-theme-cayman	5 Apr 2022 16:13	Kathleen McCarthy	0f4e40a
Updated EventListFragment to display first juggled calendar	3 Apr 2022 21:50	KathleenMK <kath	b3870b6
SignIn layout and theme updates	3 Apr 2022 15:53	KathleenMK <kath	43eb9b6
Fix to alias check in Home Fragment	2 Apr 2022 21:05	KathleenMK <kath	efa6e37
Retrieving User from Firebase	2 Apr 2022 19:53	KathleenMK <kath	e927a68
Added alias per calendarId in User model	2 Apr 2022 15:18	KathleenMK <kath	42d9c0a
Adding calendar ids by role to User in database	2 Apr 2022 14:56	KathleenMK <kath	47136b8
Allowing null calendar args in EventsList	2 Apr 2022 11:51	KathleenMK <kath	b27ea6f
Added UserModel and test save to Realtime DB	30 Mar 2022 20:39	KathleenMK <kath	0d58bfb
Refactored Event View to use constraint layout	27 Mar 2022 13:38	KathleenMK <kath	6973cfd
Updated event view for new event input	27 Mar 2022 11:43	KathleenMK <kath	f997586
Refactored API interface to include authorization token	26 Mar 2022 13:53	KathleenMK <kath	950f76f
Updated for add event call with test data	22 Mar 2022 20:43	KathleenMK <kath	aee6768
Added Event View Fragment	20 Mar 2022 19:43	KathleenMK <kath	41aede3
Refactoring for renaming of fragments	20 Mar 2022 15:40	KathleenMK <kath	86ec9ba
Addede find events methods and view	20 Mar 2022 13:39	KathleenMK <kath	97cc12d
Added recycler view for list of calendars	13 Mar 2022 17:01	KathleenMK <kath	97f722d
Refactored to use Calendar Manager object	13 Mar 2022 15:23	KathleenMK <kath	55ce2b9
Added Event model	13 Mar 2022 12:20	KathleenMK <kath	9520547
Added calendar model with test api call	12 Mar 2022 16:15	KathleenMK <kath	94eb910
Including request google calendar scope and revoke	5 Mar 2022 19:54	KathleenMK <kath	9adfcb4
Updated to use Firebase authentication on signin	5 Mar 2022 14:23	KathleenMK <kath	567394e
Google Sign In Activity and layout	25 Feb 2022 20:20	KathleenMK <kath	5a66948
New Android project	19 Feb 2022 12:26	KathleenMK <kath	aff77e2

Figure 35: Sourcetree screenshot of commit history



https://trello.com/b/ggPVPCOG/juggle

juggle

Developement To Do
- Validation of date entered
- Validation of time entered
- Restrict events displayed (1)
- Ensure navigation is robust
- Persistence: add checked event (1)
- Persistence: retrieve checked event (1)

Developement Doing
- Calendar service implementation
- Token retrieval using HTTP
- Update to handle timezone and hide timezone from view (1)
- App UI (1)

Developement Done
- Refactor to use token variable in retrofit calls — 26 Mar
- Add event fields in Event View — 26 Mar - 27 Mar
- Event View layout from linear — 27 Mar
- Persistence: set up Firebase DB — 27 Mar - 27 Mar
- Persistence: add role to calendar — 27 Mar - 31 Mar
- Persistence: add alias per calendar — 2 Apr
- Restrict calendars displayed — 2 Apr
- Persistence: retrieve User and role of calendar — 2 Apr
- App colour scheme — 3 Apr
- Persistence: retrieve related events — 13 Apr
- Add event functionality — 16 Apr

Final Report
- Interim Report — 13 Feb
- Referencing — 9 Apr
- Analysis of existing products — 10 Apr
- Database design — 17 Apr
- Report

Handbook Summary
- Handbook Summary — 10 Apr
- project graphic — 10 Apr
- github page

Figure 36: Project Trello Board screenshot

8 References

Calendar, 2022. [Online]
Available at: https://www.calendar.com/
[Accessed 8 Apr 2022].

Cozi, 2022. [Online]
Available at: https://www.cozi.com/calendar/
[Accessed 8 Apr 2022].

Develop Good Habits, 2022. *https://www.developgoodhabits.com.* [Online]
Available at: https://www.developgoodhabits.com/family-calendar-apps/
[Accessed 8 Apr 2022].

Google Developers, 2021. *Calendar API, API Reference.* [Online]
Available at: https://developers.google.com/calendar/api/v3/reference
[Accessed 13 February 2022].

Google Developers, 2021. *Using OAuth 2.0 to Access Google APIs.* [Online]
Available at: https://developers.google.com/identity/protocols/oauth2
[Accessed 8 April 2022].

Google Developers, 2022. *Authorizing Requests to the Google Calendar API.* [Online]
Available at: https://developers.google.com/calendar/api/guides/auth
[Accessed 8 April 2022].

Google Developers, 2022. *Firebase Documentation.* [Online]
Available at: https://firebase.google.com/docs/database/rtdb-vs-firestore
[Accessed 27 March 2022].

Google Developers, 2022. *Google Calendar for Developers.* [Online]
Available at: https://developers.google.com/calendar/api/quickstart/java
[Accessed Feb, Mar, Apr 2022].

Google Developers, 2022. *Using OAUTH 2.0 to Access Google APIs > native app.* [Online]
Available at: https://developers.google.com/identity/protocols/oauth2/native-app
[Accessed 8 April 2022].

Google Developers, n.d. *OAuth 2.0 Playground.* [Online]
Available at: https://developers.google.com/oauthplayground/
[Accessed Feb, Mar, Apr 2022].

Modi, M., 2022. *https://medium.com/.* [Online]
Available at: https://medium.com/mqos-technologies/kotlin-vs-java-which-is-better-for-you-in-2022-7ce97790c20
[Accessed 08 Apr 2022].

stackoverflow, 2019. *Google Calendar API on Android access token.* [Online]
Available at: https://stackoverflow.com/questions/55657777/google-calendar-api-on-android-access-token
[Accessed 20 Feb 2022].

Wikipedia, 2022. *https://en.wikipedia.org/*. [Online]
Available at: https://en.wikipedia.org/wiki/Unique_identifier
[Accessed 10 Apr 2022].

Yuana, A., 2018. *https://github.com.* [Online]
Available at: https://github.com/andhikayuana/google-calendar-demo
[Accessed 13 Mar 2022].