Go quickstart

Create a Go command-line application that makes requests to the Google Calendar API.

Quickstarts explain how to set up and run an app that calls a Google Workspace API. This quickstart uses a simplified authentication approach that is appropriate for a testing environment. For a production environment, we recommend learning about authentication and authorization (/workspace/guides/auth-overview) before choosing the access credentials (/workspace/guides/create-credentials#choose_the_access_credential_that_is_right_for_you) that are appropriate for your app.

This quickstart uses Google Workspace's recommended API client libraries to handle some details of the authentication and authorization flow.

Objectives

- Set up your environment.
- Set up the sample.
- Run the sample.

Prerequisites

- Latest version of Go (https://golang.org/).
- Latest version of Git (https://git-scm.com/).
- A Google Cloud project (/workspace/guides/create-project).
- A Google account with Google Calendar enabled.

Set up your environment

To complete this quickstart, set up your environment.

Enable the API

Before using Google APIs, you need to turn them on in a Google Cloud project. You can turn on one or more APIs in a single Google Cloud project.

- In the Google Cloud console, enable the Google Calendar API.

  Enable the API (https://console.cloud.google.com/flows/enableapi?apiid=calendar-json.googleapis.com)

Configure the OAuth consent screen

If you're using a new Google Cloud project to complete this quickstart, configure the OAuth consent screen. If you've already completed this step for your Cloud project, skip to the next section.

1. In the Google Cloud console, go to Menu ☰ > **Google Auth platform** > **Branding**.

   Go to Branding (https://console.cloud.google.com/auth/branding)

2. If you have already configured the Google Auth platform, you can configure the following OAuth Consent Screen settings in Branding (https://console.cloud.google.com/auth/branding), Audience (https://console.cloud.google.com/auth/audience), and Data Access (https://console.cloud.google.com/auth/scopes). If you see a message that says **Google Auth platform not configured yet**, click **Get Started**:

   a. Under **App Information**, in **App name**, enter a name for the app.

   b. In **User support email**, choose a support email address where users can contact you if they have questions about their consent.

   c. Click **Next**.

   d. Under **Audience**, select **Internal**.

   e. Click **Next**.

   f. Under **Contact Information**, enter an **Email address** where you can be notified about any changes to your project.

   g. Click **Next**.

   h. Under **Finish**, review the Google API Services User Data Policy (https://developers.google.com/terms/api-services-user-data-policy) and if you agree, select **I agree to the Google API Services: User Data Policy**.

   i. Click **Continue**.

   j. Click **Create**.

3. For now, you can skip adding scopes. In the future, when you create an app for use outside of your Google Workspace organization, you must change the **User type** to **External**. Then add the authorization scopes that your app requires. To learn more, see the full Configure OAuth consent (/workspace/guides/configure-oauth-consent) guide.

Authorize credentials for a desktop application

To authenticate end users and access user data in your app, you need to create one or more OAuth 2.0 Client IDs. A client ID is used to identify a single app to Google's OAuth servers. If your app runs on multiple platforms, you must create a separate client ID for each platform.

1. In the Google Cloud console, go to Menu ☰ > **Google Auth platform** > **Clients**.

   Go to Clients (https://console.cloud.google.com/auth/clients)

2. Click **Create Client**.
3. Click **Application type** > **Desktop app**.
4. In the **Name** field, type a name for the credential. This name is only shown in the Google Cloud console.
5. Click **Create**.

   The newly created credential appears under "OAuth 2.0 Client IDs."

6. Save the downloaded JSON file as `credentials.json`, and move the file to your working directory.

Prepare the workspace

1. Create a working directory:

```
mkdir quickstart
```

2. Change to the working directory:

```
cd quickstart
```

3. Initialize the new module:

```
go mod init quickstart
```

4. Get the Google Calendar API Go client library and OAuth2.0 package:

```
go get google.golang.org/api/calendar/v3
go get golang.org/x/oauth2/google
```

Set up the sample

1. In your working directory, create a file named **`quickstart.go`**.
2. In the file, paste the following code:

calendar/quickstart/quickstart.go

ew on GitHub (https://github.com/googleworkspace/go-samples/blob/main/calendar/quickstart/quickstart.go)

```
package main

import (
        "context"
        "encoding/json"
        "fmt"
        "log"
        "net/http"
        "os"
        "time"

        "golang.org/x/oauth2"
        "golang.org/x/oauth2/google"
        "google.golang.org/api/calendar/v3"
        "google.golang.org/api/option"
)

// Retrieve a token, saves the token, then returns the generated client.
func getClient(config *oauth2.Config) *http.Client {
        // The file token.json stores the user's access and refresh tokens, an
        // created automatically when the authorization flow completes for the
        // time.
        tokFile := "token.json"
        tok, err := tokenFromFile(tokFile)
        if err != nil {
                tok = getTokenFromWeb(config)
                saveToken(tokFile, tok)
        }
        return config.Client(context.Background(), tok)
}

// Request a token from the web, then returns the retrieved token.
func getTokenFromWeb(config *oauth2.Config) *oauth2.Token {
        authURL := config.AuthCodeURL("state-token", oauth2.AccessTypeOffline)
        fmt.Printf("Go to the following link in your browser then type the "+
                "authorization code: \n%v\n", authURL)

        var authCode string
        if _, err := fmt.Scan(&authCode); err != nil {
                log.Fatalf("Unable to read authorization code: %v", err)
        }

        tok, err := config.Exchange(context.TODO(), authCode)
        if err != nil {
                log.Fatalf("Unable to retrieve token from web: %v", err)
        }
        return tok
}

// Retrieves a token from a local file.
func tokenFromFile(file string) (*oauth2.Token, error) {
        f, err := os.Open(file)
        if err != nil {
                return nil, err
        }
        defer f.Close()
        tok := &oauth2.Token{}
        err = json.NewDecoder(f).Decode(tok)
        return tok, err
}
```

```
// Saves a token to a file path.
func saveToken(path string, token *oauth2.Token) {
        fmt.Printf("Saving credential file to: %s\n", path)
        f, err := os.OpenFile(path, os.O_RDWR|os.O_CREATE|os.O_TRUNC, 0600)
        if err != nil {
                log.Fatalf("Unable to cache oauth token: %v", err)
        }
        defer f.Close()
        json.NewEncoder(f).Encode(token)
}

func main() {
        ctx := context.Background()
        b, err := os.ReadFile("credentials.json")
        if err != nil {
                log.Fatalf("Unable to read client secret file: %v", err)
        }

        // If modifying these scopes, delete your previously saved token.json.
        config, err := google.ConfigFromJSON(b, calendar.CalendarReadonlyScope
        if err != nil {
                log.Fatalf("Unable to parse client secret file to config: %v",
        }
        client := getClient(config)

        srv, err := calendar.NewService(ctx, option.WithHTTPClient(client))
        if err != nil {
                log.Fatalf("Unable to retrieve Calendar client: %v", err)
        }

        t := time.Now().Format(time.RFC3339)
        events, err := srv.Events.List("primary").ShowDeleted(false).
                SingleEvents(true).TimeMin(t).MaxResults(10).OrderBy("startTim
        if err != nil {
                log.Fatalf("Unable to retrieve next ten of the user's events:
        }
        fmt.Println("Upcoming events:")
        if len(events.Items) == 0 {
                fmt.Println("No upcoming events found.")
        } else {
                for _, item := range events.Items {
                        date := item.Start.DateTime
                        if date == "" {
                                date = item.Start.Date
                        }
                        fmt.Printf("%v (%v)\n", item.Summary, date)
                }
```

```
            }
    }
```

Run the sample

1. In your working directory, build and run the sample:

   ```
   go run quickstart.go
   ```

2. The first time you run the sample, it prompts you to authorize access:

   a. If you're not already signed in to your Google Account, sign in when prompted. If you're signed in to multiple accounts, select one account to use for authorization.

   b. Click **Accept**.

   Your Go application runs and calls the Google Calendar API.

   Authorization information is stored in the file system, so the next time you run the sample code, you aren't prompted for authorization.

Next steps

- Create events (/workspace/calendar/create-events)
- Troubleshoot authentication and authorization issues (/workspace/calendar/api/troubleshoot-authentication-authorization)
- Calendar API reference documentation (/workspace/calendar/v3/reference)
- `google-api-go-client` section of GitHub (https://github.com/google/google-api-go-client)