Contents

Getting started with the GIRAF project - Revised

Updated December 2022

This guide will help you get started with the GIRAF project, currently consisting of the Weekplanner app, the api client package and the web api backend. This guide will take you through every step from a clean installation to a running app on the 3 major platforms, Windows, Mac (OS X and up) and Linux (Ubuntu 20.04.2.0).

Optional steps are marked with *

1 Getting a proper IDE

For the weekplanner app and the api client package, we would recommend Android Studio or Visual Studio Code as they are free and include everything needed for android (and iOS) app development.

On Ubuntu there was a problem with the newest version of Android Studio throwing weird Java errors when I tried to build. I got it to work with version 4.0.2. You can download any version from https://developer.android.com/studio/archive.

For the web api you can use any C# IDE you like, e.g., Microsoft Visual Studio or JetBrains Rider.

You will also need Git installed on your PC if you do not have it already.

2 Weekplanner

2.1 Installing the flutter framework

Download the flutter framework from https://flutter.dev/docs/get-started/install. The GIRAF project is guaranteed to work with version 3.3.8

Just follow the installation guide on the flutter website and you should be good to go.

If you are using Android Studio, IntelliJ or Visual Studio Code, be sure to install the flutter and dart plugins to get full flutter support.

2.2 Adding flutter to your path*

Adding flutter to your path is **highly** recommended as it allows you to run flutter commands from any terminal window. How to do this is very dependent on your OS so make sure to check the flutter installation guide on how to do this on your setup if you did not do it already as part of your installation or *see guide below.*

2.3 Installing a java JDK

Android needs a java JDK to compile and run. The weekplanner project currently does not work with java newer than version 11, which can be downloaded here: https://jdk.java.net/java-se-ri/11, for MacOS, download the Linux version.

After installing java 11, make sure to set your `JAVA_HOME` environment variable to the java root directory, i.e. `.../jdk-11`, and put java in your PATH environment variable, i.e. `.../jdk-11/bin`, *see guide below.*

You might have to restart your PC or IDE to make the new environment variables work.

2.4 Installing android emulator and android SDKs

Installing Android Studio is the easiest way to get an android emulator, even if you are not going to use the IDE itself, *see guide below.*

2.5 Clone the weekplanner repository

In Android Studio, choose project from version control and enter the GIRAF weekplanner url: https://github.com/aau-giraf/weekplanner.git

On other IDE´s, you can use GitHub's desktop app to clone the repository, after which you can open it with your own specified IDE

NB: Note that you will need to be added to aau-giraf GitHub organization to be able to contribute.

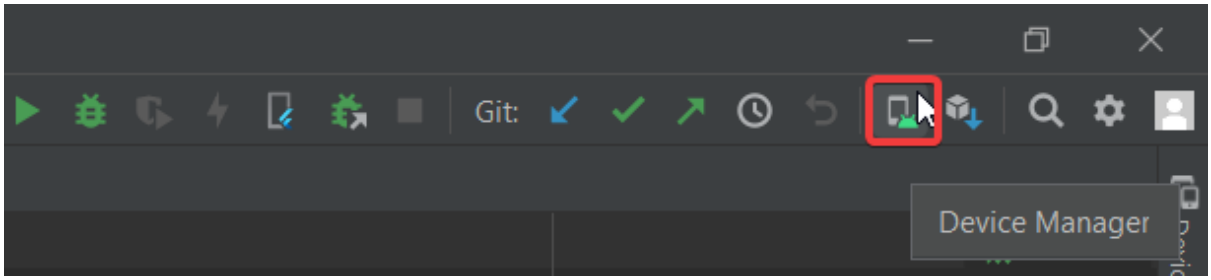
NB: Android Studio has built-in GitHub support if you link it to your GitHub account.

2.6 Download packages

Download all necessary packages for the project by running flutter command `flutter pub get` in a terminal window at the project root i.e., .../weekplanner

2.7 Setting up a virtual device with Android Studio

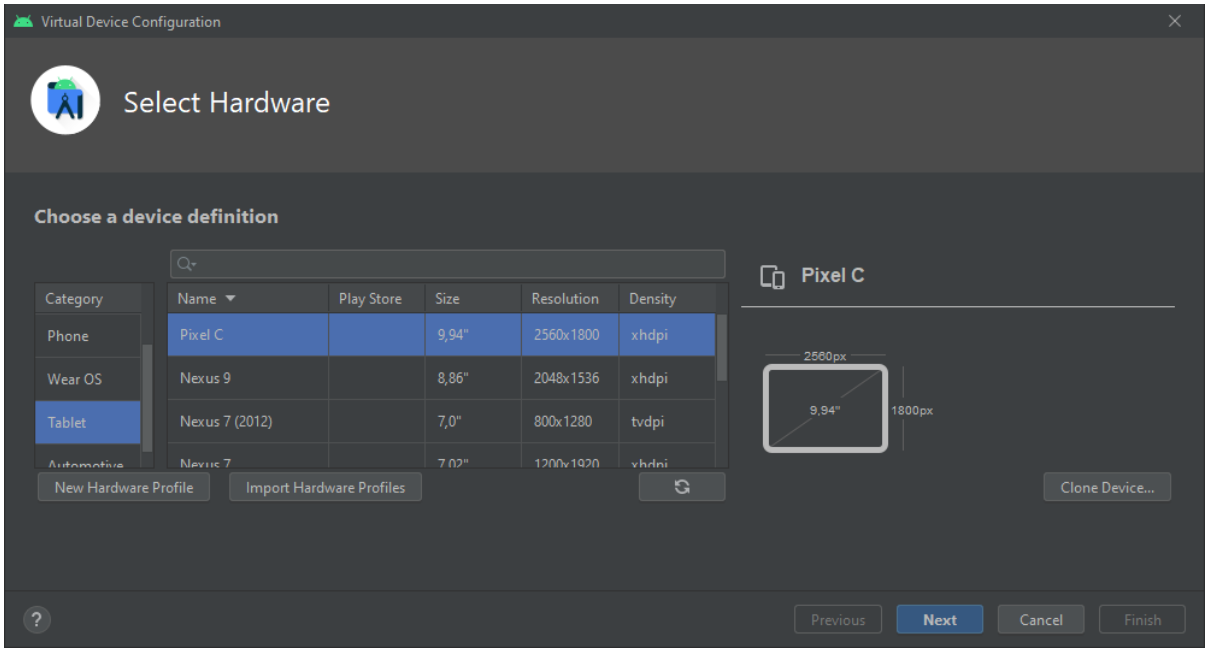
In Android Studio, click the "Device manager" button in the top right corner.



Create a new device by clicking Add Virtual Device in the bottom right corner of the popup

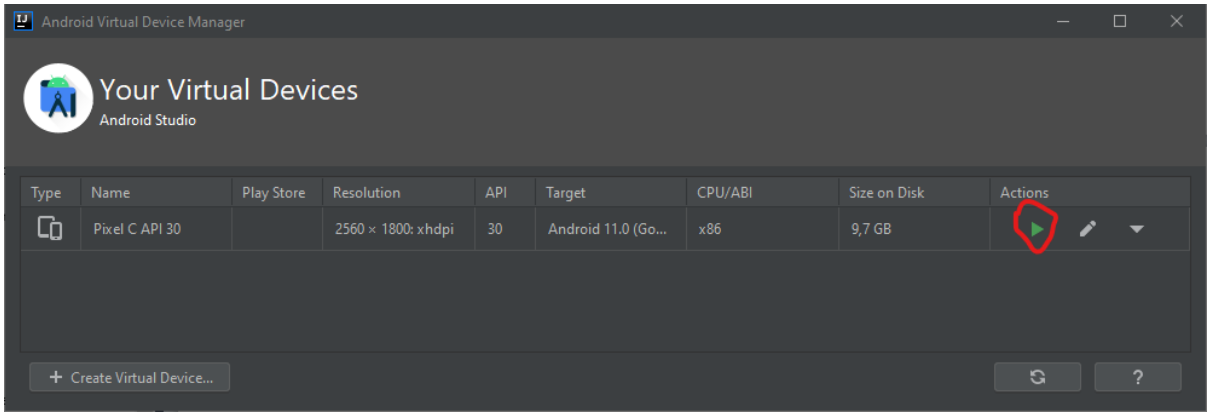
As the GIRAF weekplanner app is supposed to be used on a tablet, it makes the most sense for you to have a tablet virtual device.

Choose tablet in the category tab and choose any device, e.g., pixel C



Choose an android version to run on your virtual device e.g., android R API level 30.

After finishing the installation, confirm that your virtual device works. Go to AVD-manager and click the green arrow in the row of the virtual device you want to start.



2.7.1 MacOS: iOS emulator*

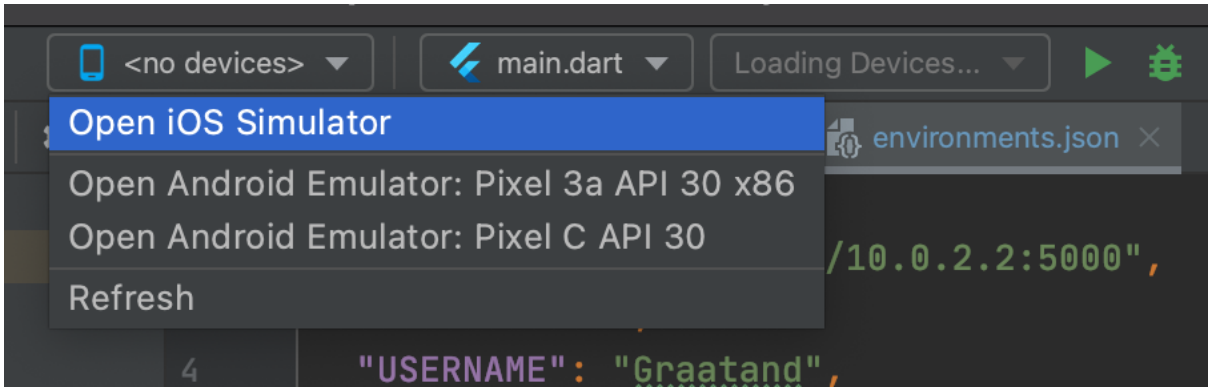
If you are running MacOS, you also have the possibility of using iOS emulator instead of Android Emulator, allowing you to test the app on iOS devices. To do this requires XCode and CocoaPods to be installed.

Install XCode from the app store at https://apps.apple.com/us/app/xcode/id497799835?mt=12, then open a terminal and execute `sudo gem install cocoapods`.

You might need to reboot your PC.

Open a terminal and run `flutter doctor` which should show that XCode is visible to flutter and will show a warning if CocoaPods is not installed.

Then in Android Studio, in the device selection list should be an option “Open iOS Simulator”. Clicking it should open the iOS simulator with a default iPhone. As the weekplanner app is meant for tablet devices I would suggest opening an iPad instead by choosing File→Open Simulator→iOS X.Y→any iPad.



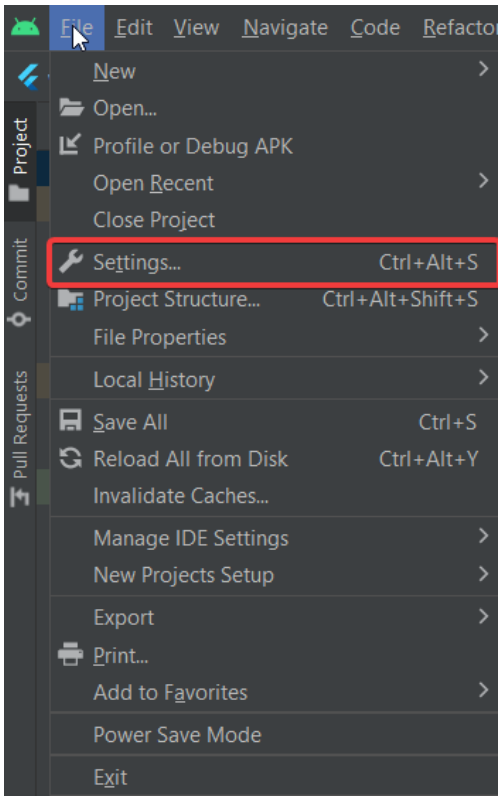
Your chosen device should then be visible in the device selection list.

2.8 Make a test run

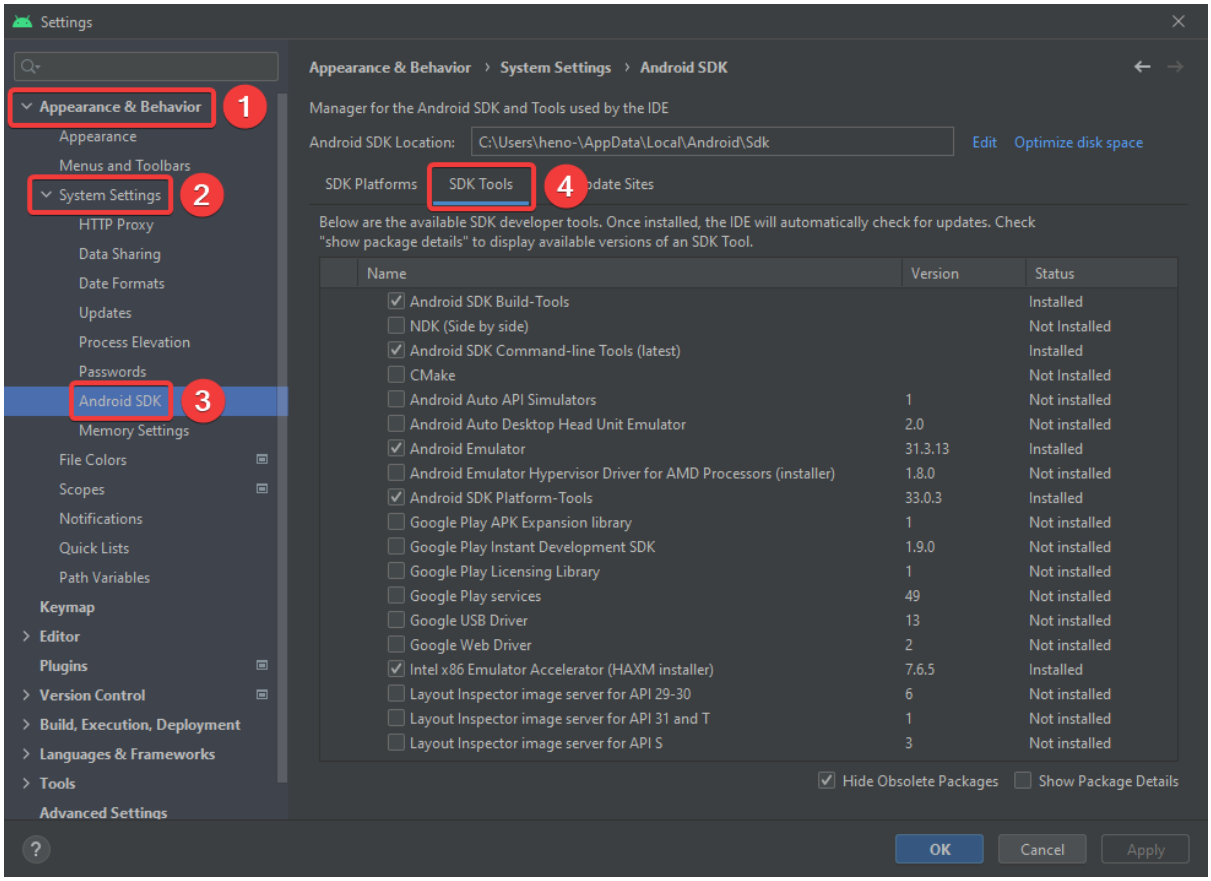
To make sure everything is as it should be, run the command `flutter doctor` in a terminal.

The first time you do this, it should give a warning at the Android tool-chain step that you need to accept android licenses. Do this by running the command `flutter doctor --android-licenses`.

If you get any errors, go to File→Settings

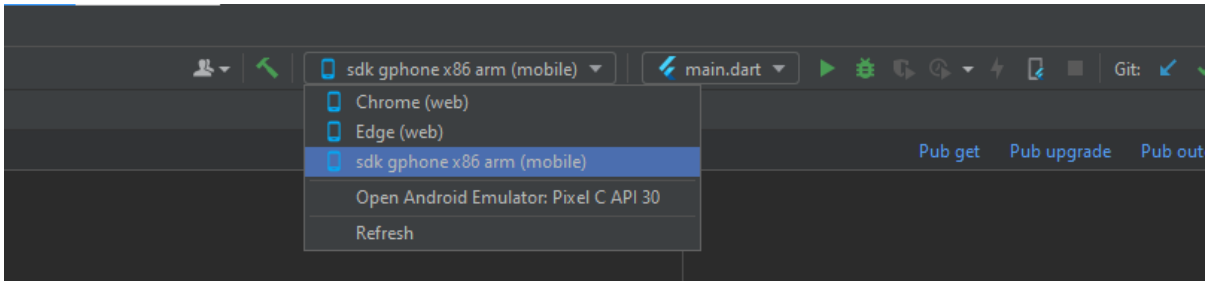


Now direct to Appearance & Behaviour→System Settings→Android SDK, then go to the SDK-tools tab.



Here make sure that you have the newest versions of at least Android SDK buildtools, Android Emulator, Android SDK platform tools, Intel x86 emulator accelerator and Andorid SDK Command-line Tools.

When flutter doctor is happy, launch your android or iOS virtual device. It should now show up in flutter devices. Select it and click Run (the green arrow right of flutter devices)



The GIRAF weekplanner app should then start compiling, this may take several minutes, and even longer the first time. When it is done, it should print something along the lines of "*Syncing files to device sdk gphone x86 arm..".* in the Run console, and the app will be launched in your emulator.

2.8.1 For MacOS using iOS emulator

If you get an error related to pod install, try deleting the file weekplanner→ios→Podfile

2.9 Debugging on your android device*

If you own an android device, it is also possible to use that as a debugging device, i.e., testing new changes on a physical device instead of a virtual one.

Before you can use your physical android device for debugging, you need to enable developer options on your device and allow USB debugging. Check this guide on how to do that for your device. https://developer.android.com/studio/debug/dev-options

Then when you plug your android device into your PC, a pop-up might appear on the device asking to allow USB debugging.

When this is done correctly, your physical device should appear in the list of available devices in Android Studio and can be selected as any virtual device.

3 Api client

3.1 Installing flutter

Make sure you have the flutter framework installed, *see guide above.*

3.2 Clone the api client repository

In your IDE, create a new repository, in Android Studio choose project from version control and enter the api_client url: https://github.com/aau-giraf/api_client.git

3.3 Download project packages

Run the flutter command `flutter pub get` in a terminal window at the project root.

3.4 Set the weekplanner to point at your branch

If you want to test out your changes to the api_client, open weekplanner→pubspec.yaml in the weekplanner, find the api_client entry and change ref from `develop` to your branch e.g. `feature/73` . Then run flutter pub get again in the weekplanner project to update the package.

NB: due to caching, simply running pub get is not always enough. If the package is not properly updated, delete the build directory from the weekplanner project directory and build the application again. It is possible to check the current version of api client your local weekplanner uses in Android Studio by opening External Libraries → Dart Packages.

NB: Do not push this change to pubspec unless you know what you are doing, and make sure to change it back to develop before merging.

4 Web-api

Make sure you have a C# compatible IDE e.g., Microsoft Visual Studio or JetBrains Rider.

4.1 Installing dotnet core

The web-api is a dotnet core web app that runs on dotnet core 6.0.x so make sure that you have a compatible version installed. It can be downloaded here: https://dotnet.microsoft.com/en-us/download/dotnet/6.0

4.2 Installing MySQL

The web api runs a local instance of a MySQL server.

Install MySQL server 8.0 from https://dev.mysql.com/downloads/mysql/. MySQL workbench and MySQL shell are useful but optional.

During the setup, make sure to give the root user a password you can remember e.g., ”password” as this is only a development instance. Additionally, create a user with username “user” and password ”password” that have administrative rights.

NB: Make sure that your MySQL server is running (this is where MySQL workbench is useful)

4.3 For Linux - Install additional libraries

Run the following commands to install additional libraries (might require sudo privileges):

- apt install libc6-dev
- apt install libgdiplus

4.4 For MacOS - Install additional libraries

Run the following commands to install additional libraries (requires sudo privilegess):

- /bin/bash -c "(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
- brew install mono-libgdiplus

4.5 Clone the web-api repository

In your IDE, create a new repository, in Rider select Get from version control and enter the web-api url:

https://github.com/aau-giraf/web-api.git

4.6 Connect to your local MySQL server

Navigate to the web-api→GirafRest folder.

Copy the ”appsettings.template.json” file to the same folder and name the copy ”appsettings.Development.json”.

In the new file, set line 3 to "DefaultConnection": "server=localhost; port=3306; userid=root; password=password; database=giraf; Allow User Variables=True".

On line 24, set the values of Jwt.JwtKey to a random string of at least 40 characters.

On line 25 set the value of Jwt.JwtIssuer to "Aalborg University”.

Open a terminal window at web-api→GirafRest and enter the following commands:

- dotnet tool install --global dotnet-ef
- dotnet restore
- dotnet ef database update
- dotnet run --sample-data

If you get any errors when running these commands, verify that you have dotnet core 6.0.x SDK installed and in your path, and that your MySQL server is running at localhost:3306

NB: Note that in some cases, you might need to set the environment variable ”DOTNET_SYSTEM_GLOBALIZATION_INVARIANT=1”

4.7 Establish a query interface to your database*

Many IDE’s support a GUI connection to databases e.g., Visual Studio and Rider.

Select the database tab, add an instance, and select MySQL. Use the connection settings:

- Host: localhost

- Port: 3306
- Username: root
- Password: password
- Database: giraf

This should establish a connection to your database that allows you to manually edit the tables and schemas.

4.8 Connecting the local web-api to the local weekplanner

In order to test new changes to the web-api locally, you need to make the weekplanner application connect to your local web-api. The is done in the weekplanner project by locating the file weekplanner→assets→environments.json and changing the value of SERVER_HOST to "http://10.0.2.2:5000" for an Android simulator device or "http://localhost:5000" for an iOS simulator device. Using a physical device, the device needs to be connected to the same Wi-Fi network as the PC running the local web-api, then look up the ip address of the PC and set the SERVER_HOST to "http://<your ip>:5000". Alternatively, if the physical device is an android phone and is only connected to the pc through a USB "http://<your ip>:5000" should be set to "http://localhost:5000". Then while the phone is connected open a terminal and write adb reverse tcp:5000 tcp:5000. This requires adb(android debug bridge) is installed, should potentially already be installed by installing android studio. This will cause your phone to connect to your PC's localhost. Writing adb reverse tcp:5000 tcp:5000 must be repeated whenever the phone is disconnected from the PC, or the PC is turned off. This solution with adb has only been tested on a windows pc.

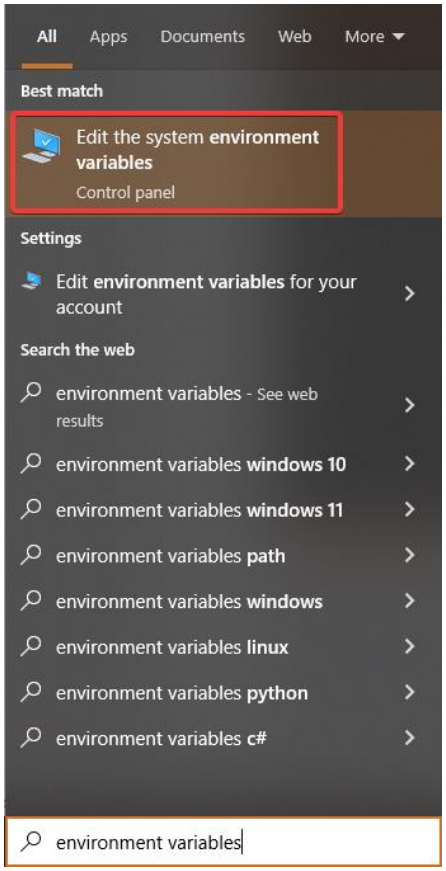
Then when you launch the weekplanner app it should connect to your local web-api. Thus, the web-api needs to be running locally.

NB: This change should ALWAYS remain a local change and should NEVER be pushed to GitHub.

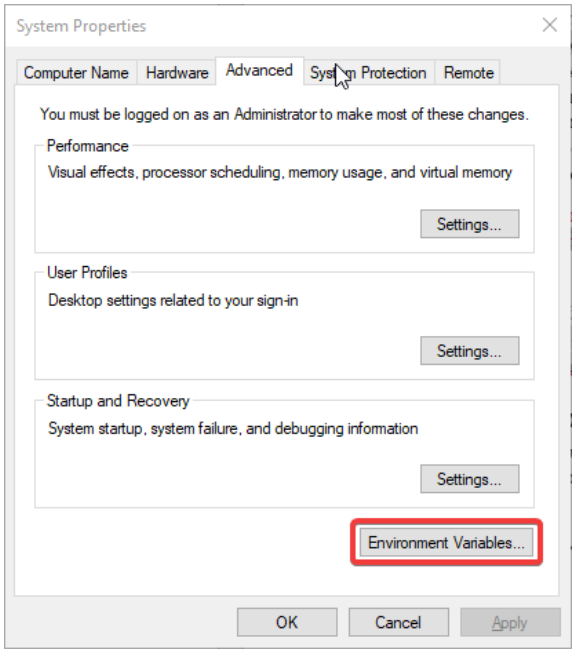
5 Setting environment variables*

5.1 Windows

In Windows, editing environment variables is quite simple. Open the windows menu or the settings menu and search for environment variables.



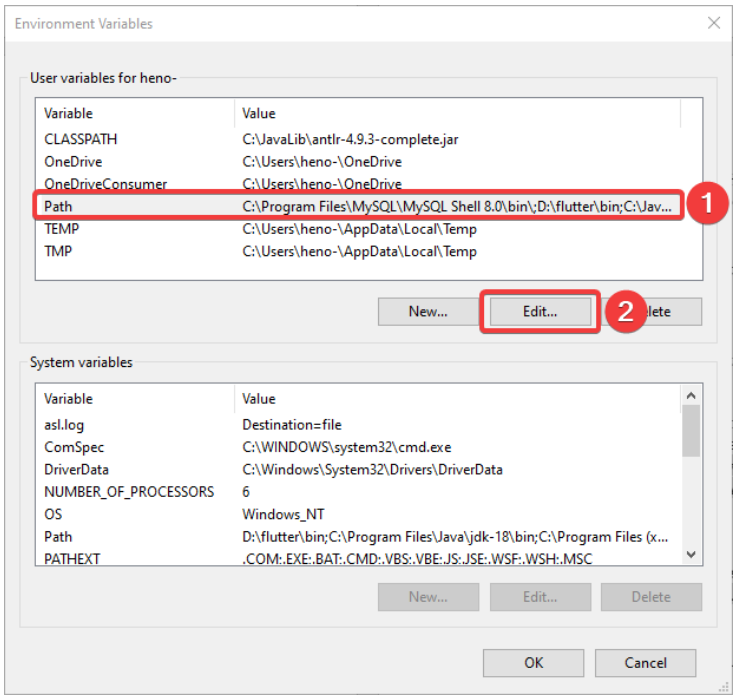
Here click on the button in the bottom labeled “Environment variables...”



This will open a new window with two lists of environment variables, for the system and for the user. Here it is possible to add new and edit existing variables.

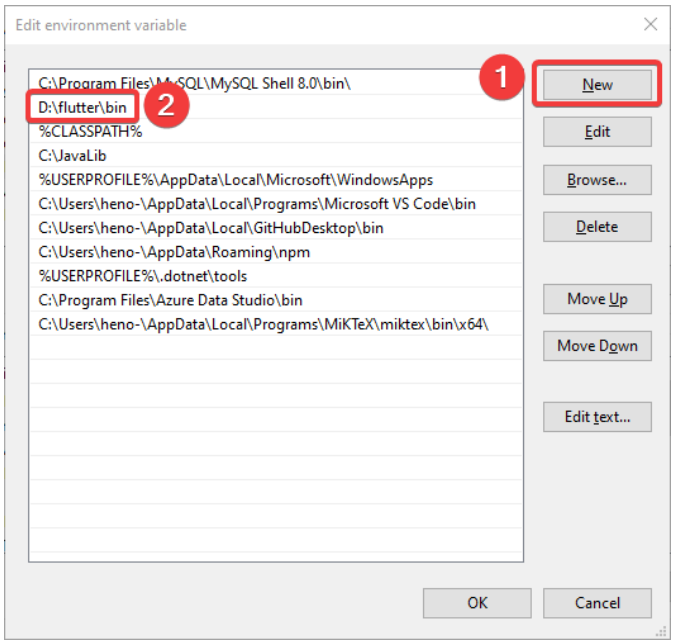
Most of the variables related to the GIRAF project should be in the upper list, specific for your user.

To add a directory to your PATH environment variable, you need to locate the variable Path in your user variables and edit it, then add a new line with the path to the directory you want to add.



If a variable named Path does not exist in the list of user variables, click the button New and create one.

If your Path variable exists but only has one value, you can add another value by editing the variable and adding the next value separated by a semi colon.



Finally, when you reach this point, you create a new path, which has the location of your flutter/bin folder. This now allows you to run flutter commands from any location.

5.2 Ubuntu

On Linux (Ubuntu) there are many ways to edit environment variables. One of the simpler is to open a terminal and edit the file etc→environment with your favorite editor e.g., emacs, nano, vim, etc. e.g., `sudo nano /etc/environment`. Add or edit the variables that you need. There is one variable per line of the form `NAME="value1:value2:value3"`. Save and exit, then reboot your PC for it to work.

5.3 Mac

How to set environment variables on Mac depends on which shell your terminal is using. Open a terminal and type `echo $SHELL` and it will tell you which you are using, typically Bash or Z Shell. Use your favorite text editor to edit the file `$HOME/.bashrc` if you're using Bash and `$HOME/.zshrc` if you're using Z shell, e.g. `sudo nano ${HOME}/.zshrc` This file might not already exist, so this might create it, thus opening an empty file. This is OK. Add and edit the variables you need. There is one variable per line of the form `NAME="value1:value2:value3"`. Save and exit and reboot your PC.

NB: If the file did not already exist or did not already contain the PATH environment variable, you need to add it as `PATH="${PATH}:your-new-path"`, to add paths instead of replacing them.

5.4 Verify

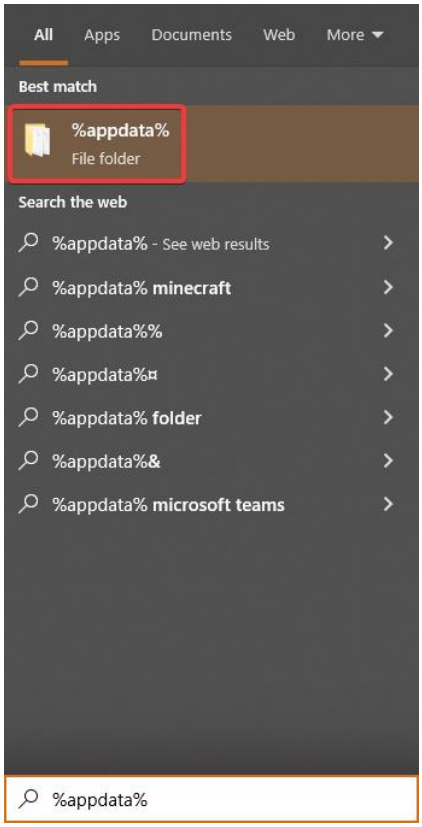
You can always verify that your changes work by opening a terminal and typing echo `$var_name` e.g., `echo $PATH` or `echo $JAVA_HOME` for Linux and Mac. For Windows using PowerShell, type `echo $Env:var_name` e.g. `echo $Env:Path`

6.0 Using the android emulator without Android Studio

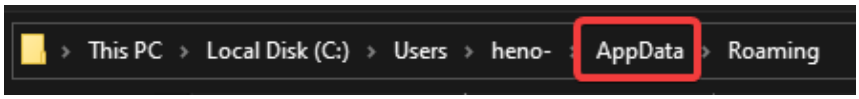
We found it especially useful to use other IDE's than Android Studio while using the emulator outside Android Studio, so here is a guide on how to run the emulator outside Android Studio.

6.1 Windows

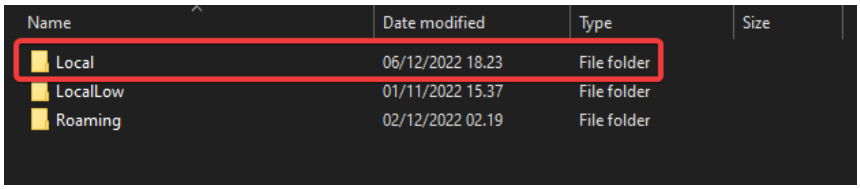
First of all, search for `%appdata%` in the search bar and open the appdata folder.

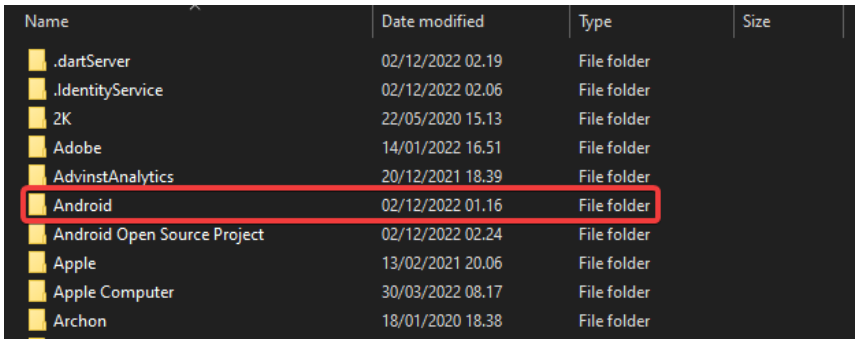


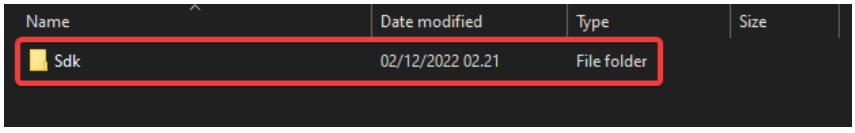
Now exit the "Roaming" folder and enter the "AppData" folder.

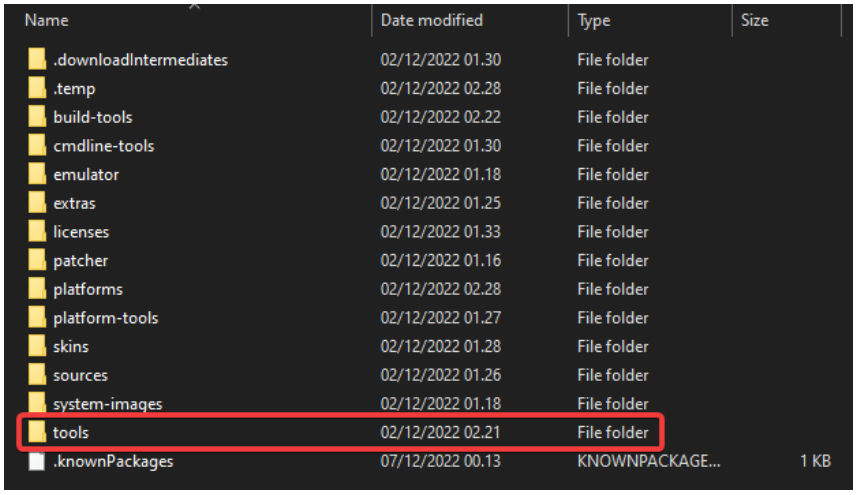


Once you are in this folder, locate to "\Local\Android\Sdk\tools".

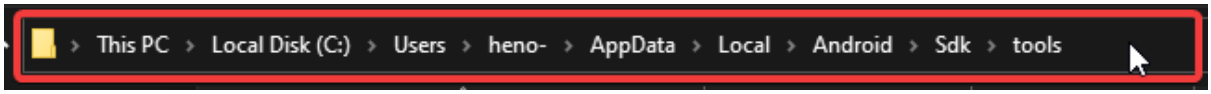


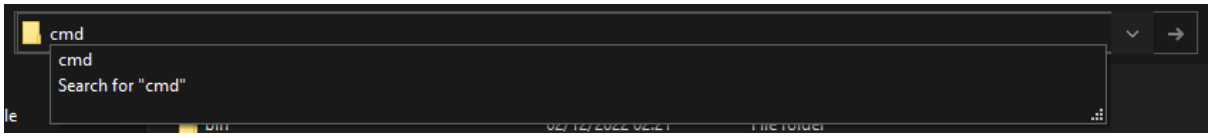


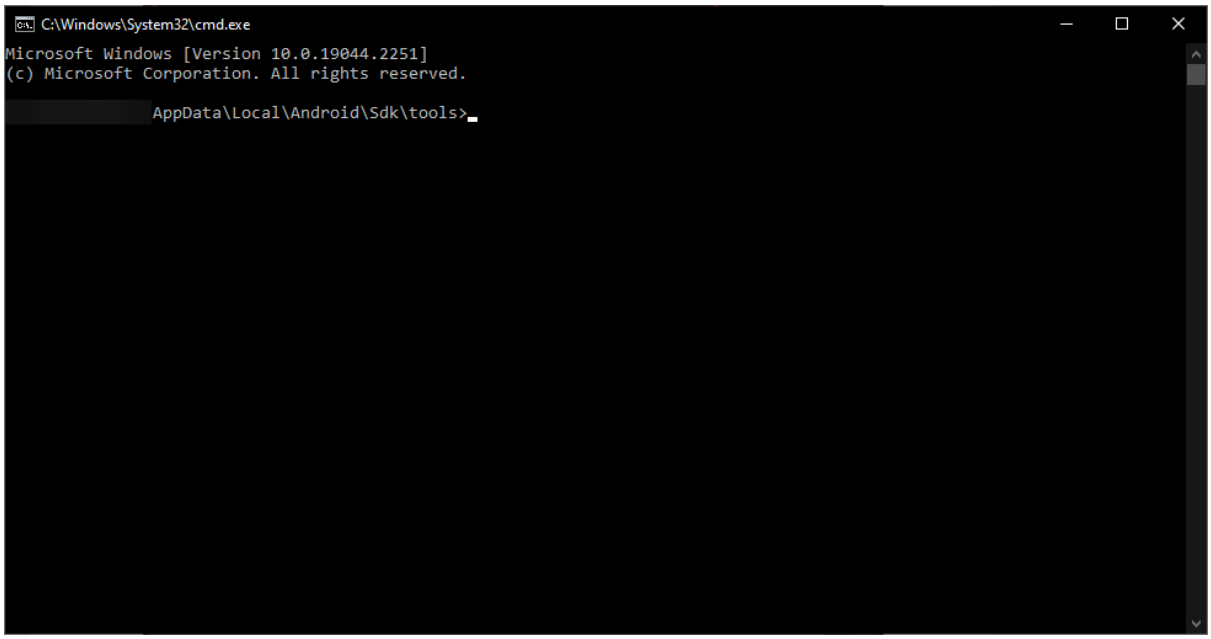




Now once you have made your way to this folder, go into the directory and type "cmd". This should open a command prompt at this location.

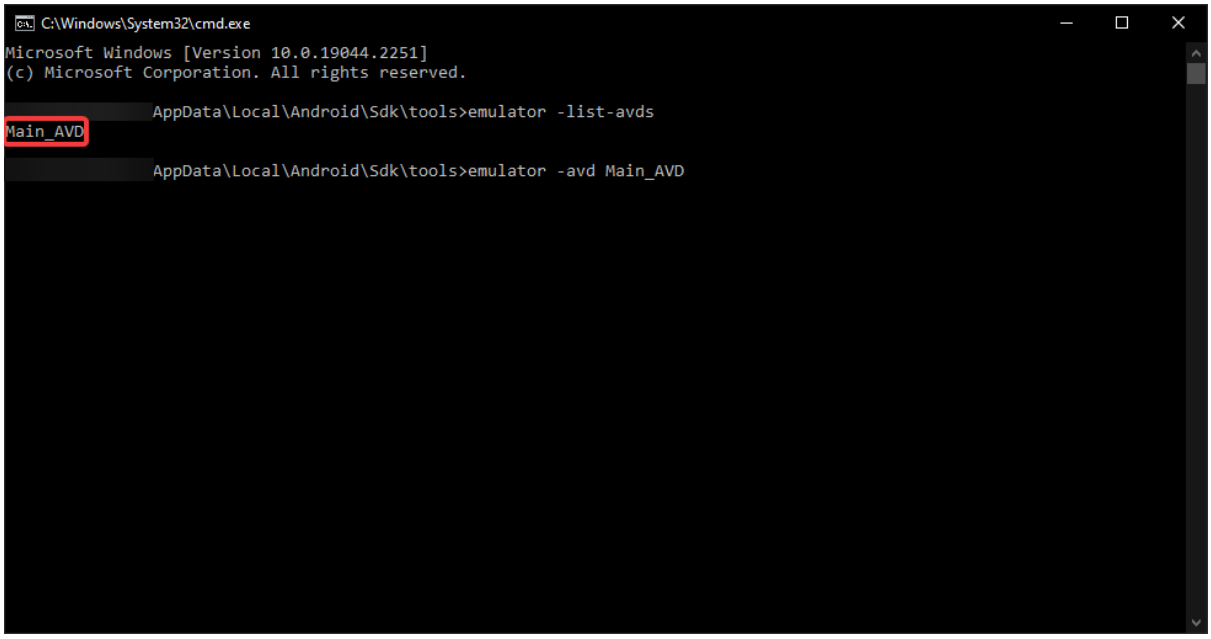




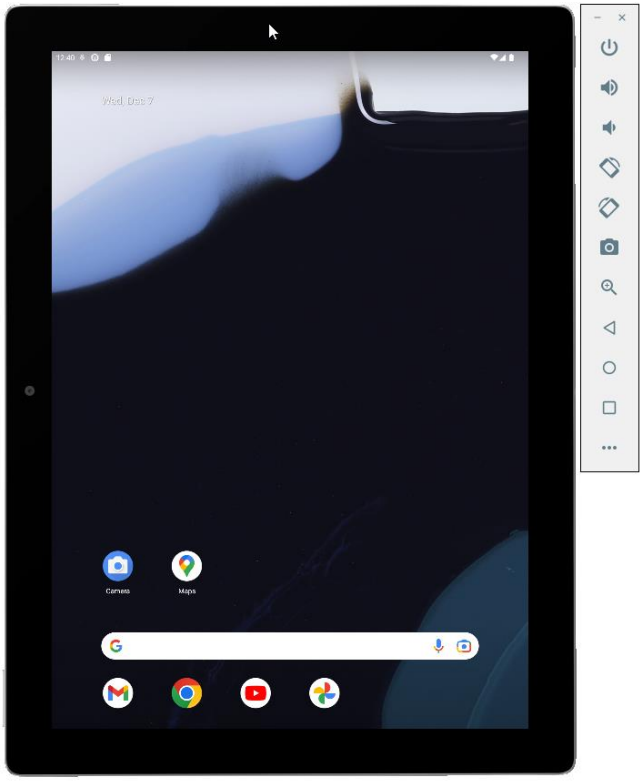


Once you have the command prompt open at the specified location, type in "emulator -list-avds". This should return a list with the name of the AVD you have created.

After this, run the command "emulator -avd <YOUR_AVD_NAME>".



This should launch your virtual device.



You are now ready to run the weekplanner. IDE´s such as Visual Studio Code should automatically recognize that you have an AVD running.

6.2 Mac

1) Install android-sdk `brew install android-sdk`

To ease the use of Android please add the following to your environment variables:

- A new variable called ANDROID_HOME, which should point to your Android installation directory (It will probably be located at "~/Library/Android/sdk)

- A new path to your PATH variable, which should point to the sdk/platform-tools and sdk/cmdline-tools/latest/bin folder of your Android installation

2) Install Flutter **3.3.8** using the steps described here:

https://flutter.dev/docs/get-started/install

The **3.3.8** version of Flutter can be found here:

https://flutter.dev/docs/development/tools/sdk/releases

3) Install dotnet **6.0** from this link: https://dotnet.microsoft.com/en-us/download/dotnet/6.0, if not done automatically then add the .dotnet/tools to your PATH.

4) Install additional libraries
Use the following commands to install needed libraries for dotnet:

```
/bin/bash -c "(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

```
brew install mono-libgdiplus
```

5) Install MySQL Server **8.0** from this link: https://dev.mysql.com/downloads/mysql/ or run this command: `brew install mysql`
When installing/setting up set the root user's password to *password*

6) Install android system image and tools:

For **x86-64** macs:

```
sdkmanager "platform-tools" "platforms;android-33" “emulator” “extras;google;auto” “system-images;android-33;google_apis;x86_64” “system-images;android-33;google_apis_playstore;x86_64”
```

For **ARM** macs:

```
sdkmanager "platform-tools" "platforms;android-33" “emulator” “extras;google;auto” “system-images;android-33;google_apis;arm64-v8a” “system-images;android-33;google_apis_playstore;arm64-v8a”
```

7) Create virtual device (Pixel C with device id 27, which is a tablet):

For **x86-64** macs:

```
avdmanager create avd -n pixel_c -k “system-images;android-33;google_apis;x86_64” -d 27
```

For **ARM** macs:

```
avdmanager create avd -n pixel_c -k “system-images;android-33;google_apis; arm64-v8a” -d 27
```

8) Set the available RAM for the virtual device to 2GB

```
emulator -avd pixel_c -partition-size 2048 -wipe-data
```

9) Use the keyboard to type instead of the emulated touchscreen

Navigate to /Users/you/.android/avd/pixel_c.avd

Use a text editor to edit “config.ini”

Find the line that says hw.keyboard=no

Change it to hw.keyboard=yes

10) Run the emulator

emulator -avd pixel_c

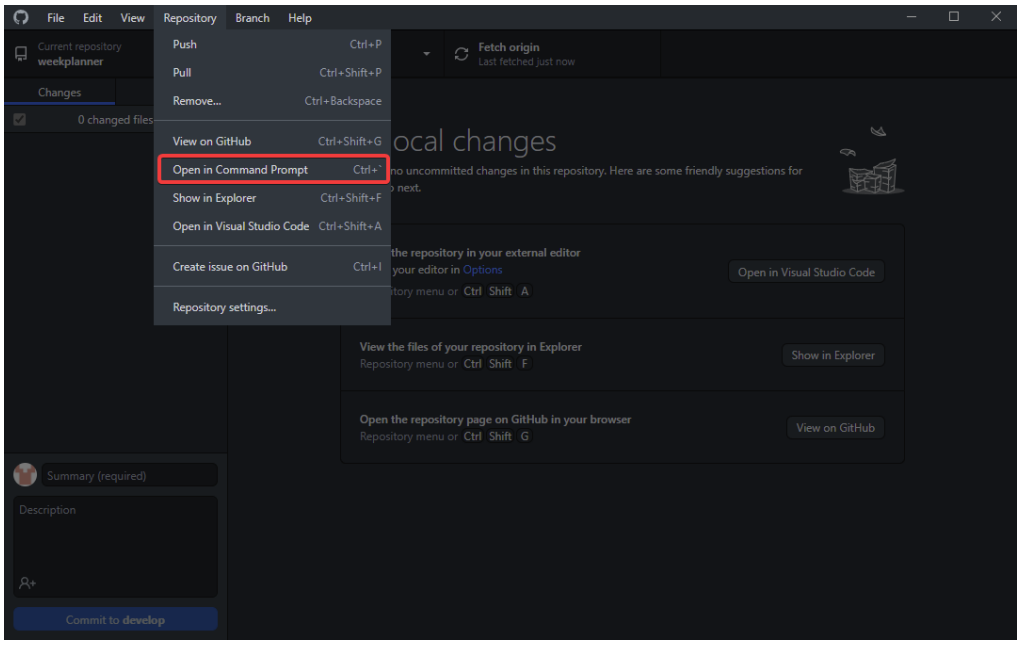
6.3 Linux

7.0 Random Issues We Found

This section is for minor issues we found when setting up GIRAF. You might not encounter any of these issues.

7.1 Using GitHub to open command prompt and run flutter

We found out that opening your command prompt via. the option on GitHub, causes your flutter commands to not be recognized.



However, this problem can be fixed by either restarting your computer to allow your user path variables to take effect or simply locating the weekplanner folder manually and then typing “cmd” in the directory.