

# Laboratory – Networking

---

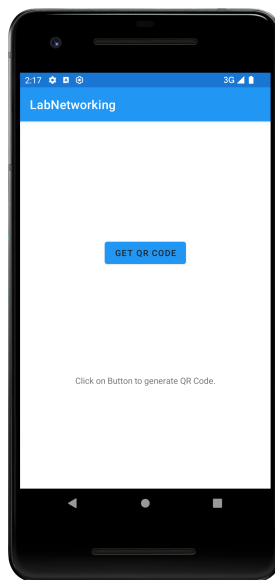
*Learn about using Kotlin coroutines to receive data over a network without blocking the Main Thread*

## Objectives

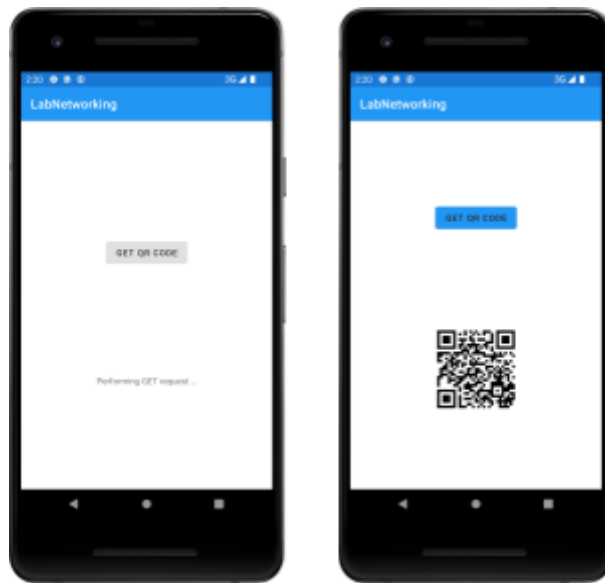
Familiarize yourself with the receiving data over a network from an https endpoint. Create an application that requests and displays a QR code image from a networked server. The app will request this data outside of the main Thread. When the app receives the image, it updates its UI to display it. Once you've completed this Lab you should understand how to create coroutines, to issue network requests within a coroutine, and to update the app UI with data produced by the coroutine.

## LabNetworking

This lab involves an app called LabNetworking. When it runs, the app displays a user interface like that shown below..



When the user clicks on the “Get QR Code” button, the app issues a request to a networked server to get a QR code containing a link to the CMSC436 class webpage, as shown below.



This application includes a single Activity called “MainActivity.” MainActivity hosts a Fragment called MainFragment, which manages the application’s UI. The application also contains a ViewModel class called MainViewModel. MainViewModel manages the network requests and publishes the QR code image and status information to the MainFragment.

See the screencast, LabNetworking.mp4, that's included in the Lab directory.

## Testing

There are two test cases with several evaluation points. Each evaluation occurs at a step labelled “evaluation point”

The first test case operates as follows:

1. Start the app in portrait mode.
2. Check that the UI shows the button and some instruction text (evaluation point 1)
3. Click on the Get QR CODE button.
4. Check that the text changes to indicate that the network request is in progress (evaluation point 2)
5. After approximately 5 seconds, check that the QR code appears (evaluation point 3)
6. Use another QR code reader to follow link to the CMSC436 class website. You can take a screenshot and then use Google Lens from the photos app. (evaluation point 4)

The second test case operates as follows:

1. Start the app in portrait mode.
2. Check that the UI shows the button and some instruction text (evaluation point 1)
3. Click on the Get QR CODE button.
4. Check that the text changes to indicate that the network request is in progress (evaluation point 2)
5. In approximately 1 second, rotate the device to landscape mode.
6. After approximately 5 seconds, check that the QR code appears (evaluation point 3)
7. Rotate the device back to portrait mode.
8. Use another QR code reader to follow link to the CMSC436 class website. You can take a screenshot and then use Google Lens from the photos app. (evaluation point 4)

After completing your solution, you will record a screencast while performing the manual test. Afterward, you will submit your code and the screencast via git. You can record a screencast using services available in the Logcat console. See:  
<https://developer.android.com/studio/debug/am-video>

## Submission

When you are ready just commit your solution to your repo on GitLab by running the following commands:

```
% git add path/to/changed/files  
% git commit -m "completed Lab8_NetworkingLab"  
% git push origin main
```

Note: if you have not already pushed this branch to your repo on GitLab you will need to make a slight modification for this first time and run this instead:

```
git push -u origin main
```

This sets up tracking between your local branch and a branch with the same name on your repo in GitLab.

## Some Implementation Notes

We are providing template code and layout resources for this application. Only modify the areas marked with the word TODO.

We have done our testing on an emulator using a Pixel 5 AVD with API level 31. To limit configuration problems, you should test your app against a similar AVD.