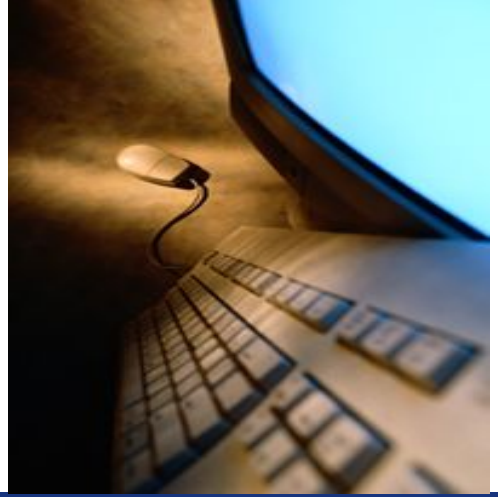
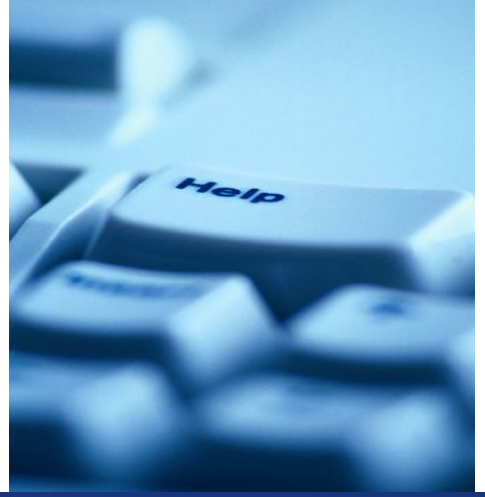
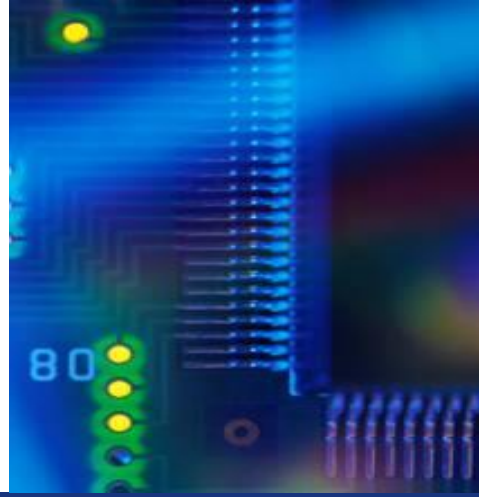
UNIVERSITAS
BUDI LUHUR



FAKULTAS TEKNOLOGI INFORMASI

Mobile Programming

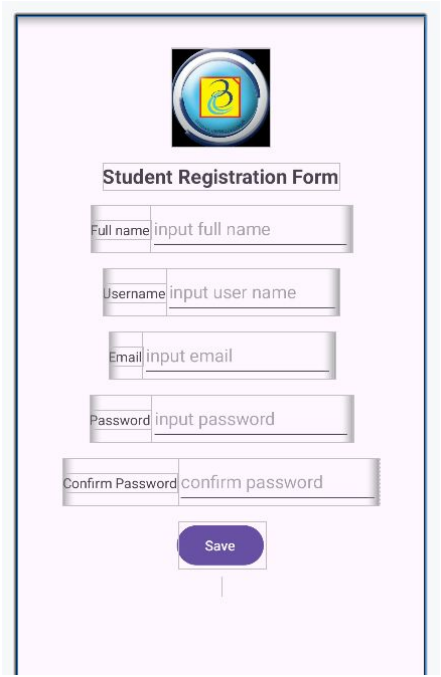
[PG119/ 3 SKS]

Riskiana Wulan, MKom

USER INTERACTION AND EVENT HANDLING
GIT and GITHUB

Cont. previous session

Layout



FindViewById

```
EditText fullNameText = findViewById(R.id.fullName);
EditText usernameText = findViewById(R.id.username);
EditText emailText = findViewById(R.id.email);
TextView txtResultView =
findViewById(R.id.txtResult);
Button btnSave = findViewById(R.id.btnSave);
```


Handling User Input

Text Fields: For inputting text (e.g., EditText)

Buttons: Trigger actions (e.g., Button)

Gestures: Touch and swipe detection (e.g., OnTouchListener)

Example:

```
<EditText
    android:id="@+id/fullName"
    android:inputType="textWebEditText"
    android:layout_width="203dp"
    android:layout_height="wrap_content"
    android:hint="input full name"
    android:minHeight="48dp" />
```

Event Listeners

What are Event Listeners?: They listen for user actions and respond to them.

Common Listeners:

- OnClickListener: Handles button clicks
- OnTouchListener: Detects touch events
- OnLongClickListener: Detects long press actions

Handling Button Clicks with OnClickListener

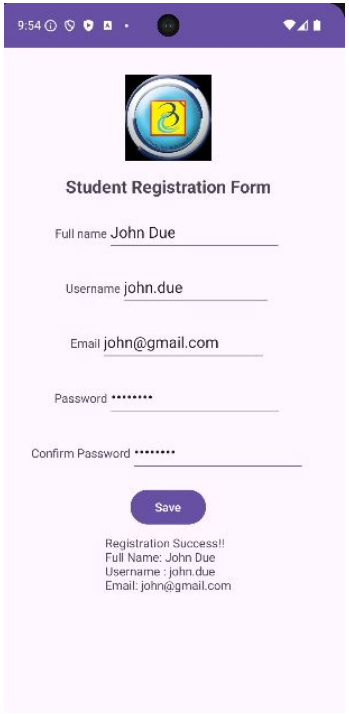
MainActivity.java

```
btnSave.setOnClickListener( new View.OnClickListener () {
    @Override
    public void onClick(View v) {
        String fullNameValue =
fullNameText.getText().toString();
        String username = usernameText.getText().toString();
        String email = emailText.getText().toString();
        txtResultView.setText("Registration Success!! \n" +
                "Full Name: " + fullNameValue + "\nUsername : "
+
                username + "\nEmail: " + email);
    }
});
```

- new View.OnClickListener() creates an instance of OnClickListener and defines what should happen when the button is clicked. The onClick(View v) method inside it gets executed when the button is clicked.

Result

Upon clicking save button, the textview result showing the information that Registration is Success



Alerts and Notifications

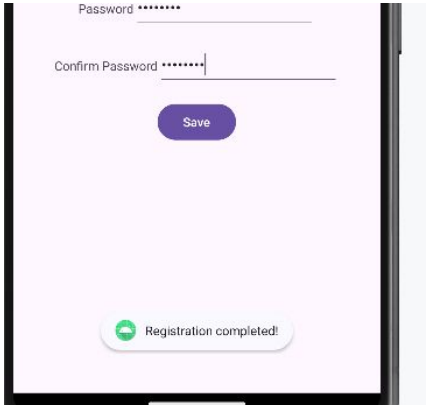
Toasts: Quick notifications that disappear automatically

Snackbars: Short messages with optional actions

Dialogs: Popup windows requiring user interaction (e.g., confirmation)

Toast example

```
Toast.makeText(getApplicationContext(),
"Registration completed!",
Toast.LENGTH_LONG).show();
```

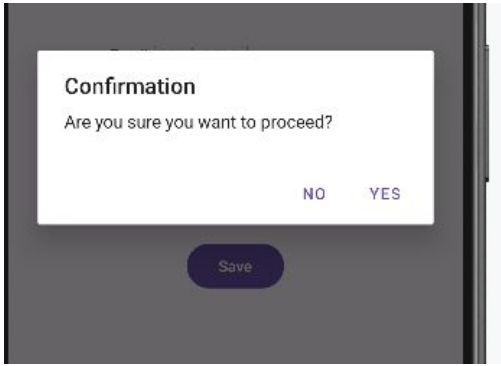


Snackbar example

```
Snackbar.make(findViewById(R.id.btnSave), "item
deleted", Snackbar.LENGTH_LONG)
        .setAction("Undo", new View.OnClickListener()
{
            @Override
            public void onClick(View v) {
                // Undo action code
            }
        }).show();
```

Dialog example

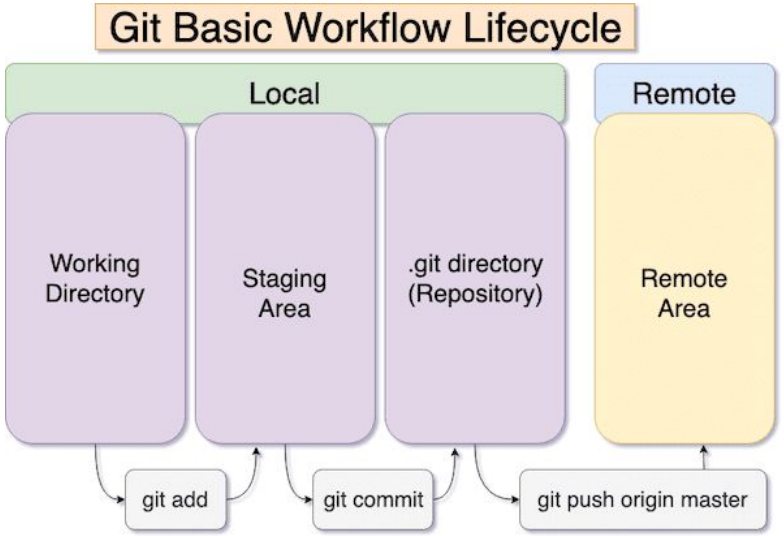
```
// When the button is clicked, show the confirmation dialog
new AlertDialog.Builder(MainActivity.this)  // Use MainActivity.this to refer to the activity context
        .setTitle("Confirmation")
        .setMessage("Are you sure you want to proceed?")
        .setPositiveButton("Yes", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                // Action to perform when user clicks "Yes"
                Toast.makeText(MainActivity.this, "You clicked Yes", Toast.LENGTH_SHORT).show();
            }
        })
        .setNegativeButton("No", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                // Action to perform when user clicks "No"
                Toast.makeText(MainActivity.this, "You clicked No", Toast.LENGTH_SHORT).show();
                dialog.dismiss();  // Close the dialog
            }
        })
        .show();  // Show the dialog
```



GIT AND GITHUB OVERVIEW

GIT and GITHUB Overview

- **Git**: A version control system to track code changes.
- **GitHub**: A cloud platform to host and share repositories.
- **Why use Git?**
  - Collaboration
  - Version control
  - Backup and security



https://www.gyanblog.com/git/practical-guide-how-work-git-basic-commands-workflows/

Simple git workflow

Master : the default branch where the final production code is stored.

Feature : not yet been release to production until it merged into master branch

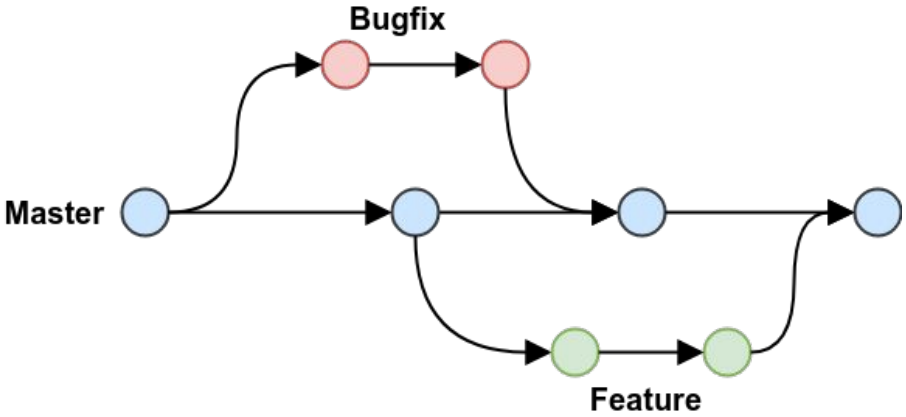
Bugfix . Developers create feature branches off of the main branch to work on new features or changes.

Each blue circle represents a commit in that branch. Once the work on the feature is complete, these branches are typically merged back into the master branch.

Bugfix: The red circles/lines represent a **hotfix branch**.

This is usually created when there's an urgent issue in the main branch that needs to be fixed immediately, without waiting for the next major feature release.

The hotfix is merged back into the main branch after the issue is resolved, ensuring the main codebase has the fix.



Setting Up GIT

1. **Install GIT: Download and install GIT from**

https://git-scm.com/downloads

2. Check GIT Version

git --version

```
riskiana@riskiana-workstation:~$ git --version
git version 2.34.1
```


Basic GIT Commands

git status: Check the current status of your project.

git add: Add files to be committed.

git commit -m: Save changes with a message.

git pull: Fetch changes from the remote repository.

git push: Upload changes to the remote repository.

```
// GIT Cheatsheet                                    @DevKhan03

// Repository
git init                          // Initialize a new Git repo
git clone <repo-url>              // Clone a repo from a URL

// Basics
git status                        // Show changes status
git add <file>                    // Add changes to staging
git commit -m "Message"           // Commit changes with a message
git log                           // View commit history

// Branching
git branch                        // List branches
git branch <branch-name>          // Create a new branch
git checkout <branch-name>        // Switch to a branch
git merge <branch-name>           // Merge changes from a branch
git branch -d <branch-name>       // Delete a branch

// Remote Repositories
git remote                        // List remotes
git remote add <name> <url>       // Add a remote
git push <remote> <branch>        // Push changes to a remote
git pull <remote> <branch>        // Pull changes from a remote

// Undoing Changes
git pull                          // Fetch and merge changes
git fetch                         // Fetch changes without merging
git reset --hard HEAD             // Discard changes
git revert <commit-hash>          // Revert changes in a commit

```

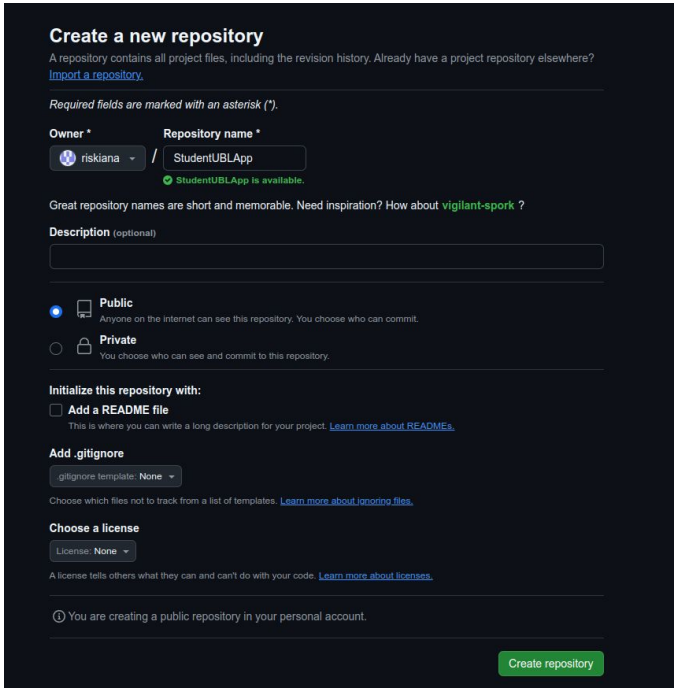
GitHub Integration in Android Studio

1. **Create a new repository**

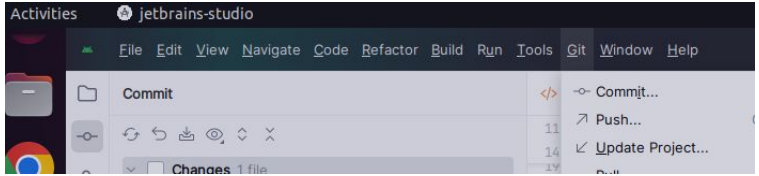
Repository Name: use the clear name according to the project purpose

Type :

- **Public : anyone can see the repo**
- **Private : We can manage who can see the repo**

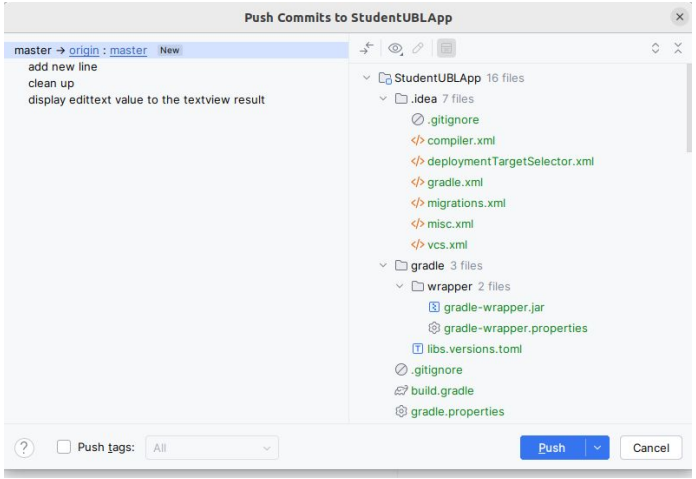


3. in android studio, choose GIT menu

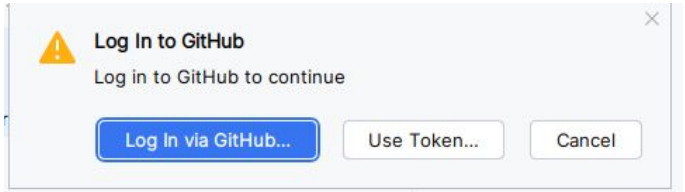


4. Define Remote

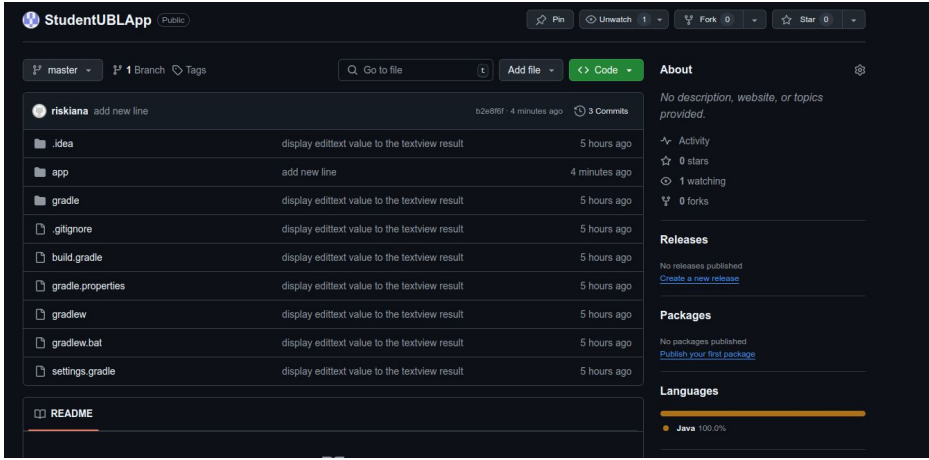
4. Push



Login according user and password in github



Source code has successfully uploaded to the github



HANDS ON LAB

1. Add validation for each fields cannot be empty
2. Add validation that minimum student age is 16 years
3. Add validation password and confirmation password should be match

4. Push the changes to github