**MINISTRY OF EDUCATION AND TRAINING**

**SCHOOL OF INFORMATION & COMMUNICATION TECHNOLOGY**

**🙞 🕮 🙜**

A blue diamond shaped sign with a torch and a globe

Description automatically generated

**MOBILE APPLICATION PROGRAMMING**

**Student name: Tran Dang Khoa**

**Student ID : B2014926**

**Courses: 46**

Can Tho, 01/2024

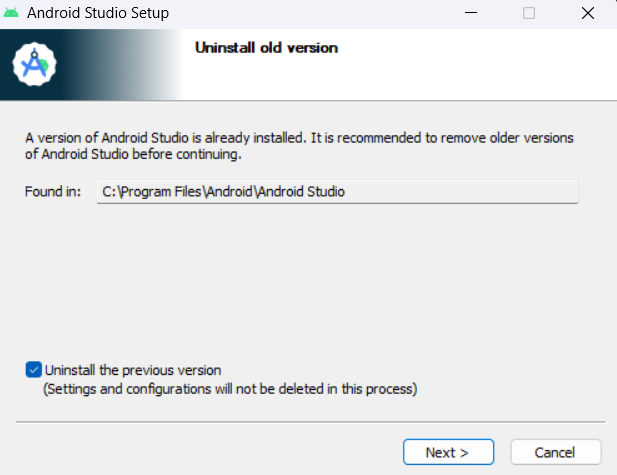
**Subject: Mobile programming (Android)**

**Lab 1. Introduction to ANDROID STUDIO**

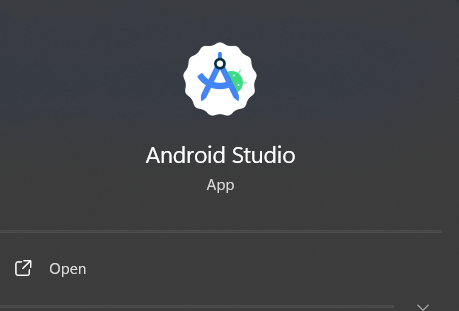
refer (or any other link):

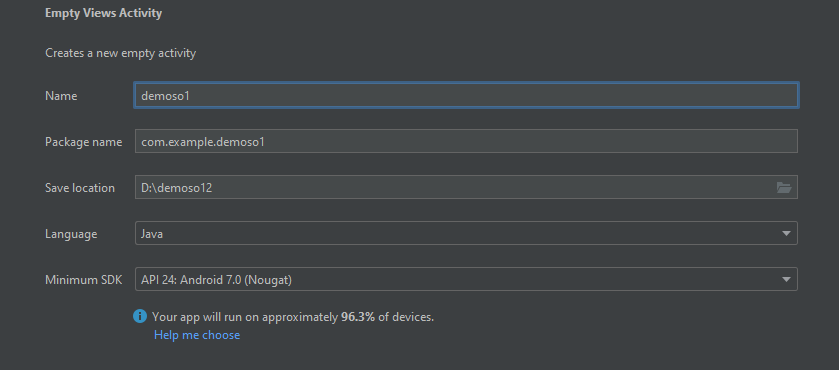
* <https://www.tutorialspoint.com/android/android_hello_world_example.htm>
* <https://developer.android.com/courses/android-basics-compose/unit-1>
* <https://developer.android.com/studio/intro/user-interface>

1. Android studio installation



To install Android Studio, we need to download file installation from the original website, then open this file and install Android Studio as a normal window application.





1. **Activity**

* In the vibrant world of Android app development, activities stand as the cornerstone of user interaction and engagement. Serving as self-contained units of functionality, an activity encapsulates a specific task or interaction within an application, providing users with focused and intuitive experiences. At the heart of each activity lies the Activity class, a fundamental component responsible for orchestrating the creation of a dedicated window to host the user interface elements.
* When the Android system launches an application, it initiates the program by invoking the onCreate() callback method within the designated activity. This method marks the inception of the activity's lifecycle, where developers initialize essential components, set up the user interface, and prepare the activity to interact with the user. As the activity progresses through its lifecycle, a sequence of callback methods is invoked, each serving a distinct purpose in managing the activity's state and behavior.
* Following the onCreate() method, the activity transitions through various lifecycle stages, including onStart(), onResume(), onPause(), onStop(), and onDestroy(). These callback methods orchestrate the activity's interactions with the user and the system, handling events such as activity creation, user input, backgrounding, and termination. By leveraging these lifecycle methods, developers can effectively manage resources, maintain state consistency, and ensure seamless transitions between different states of the activity.
* Furthermore, activities have the ability to interact with other activities within the application or even across different applications using intents. Intents serve as a powerful mechanism for initiating inter-component communication, allowing activities to launch other activities, pass data between them, and facilitate seamless navigation within the application ecosystem. Whether it's starting a new activity to perform a specific task, responding to user interactions, or navigating between different screens, intents play a pivotal role in shaping the flow and functionality of Android applications.

* In essence, activities embody the essence of user-focused interaction within Android applications, providing developers with a versatile framework to craft immersive and intuitive experiences. By mastering the lifecycle management and event handling mechanisms inherent to activities, developers can create engaging and responsive applications that captivate users and fulfill their diverse needs and expectations. With each activity representing a distinct journey for users to embark upon, Android development continues to thrive as a dynamic realm of innovation and creativity.

A green android figure with many circles around it

Description automatically generated

1. **Service**

* Services in Android serve as vital components that enable developers to execute long-running tasks or operations in the background, independent of any user interface. These components play a crucial role in enhancing the functionality and responsiveness of applications by handling tasks that persist beyond the lifecycle of individual activities or fragments. By leveraging services, developers can ensure that critical operations continue to run seamlessly, even when the application is not actively in use.
* There are two primary types of services in Android: foreground services and background services, each catering to different use cases and user experience considerations.
* Foreground Service: A foreground service is a type of service that operates with a visible notification, providing a persistent indication of ongoing activity to the user. This notification ensures that users are aware of the service's presence and activity, making it suitable for tasks that require continuous user attention. Foreground services are commonly employed for performing long-running operations that users actively engage with or monitor, such as music playback, navigation, or file downloads. By prioritizing user awareness and visibility, foreground services enhance the user experience and ensure that users remain informed about important activities within the application.

* Background Service: In contrast, a background service operates without a visible notification, making it less intrusive to the user experience. Unlike foreground services, background services do not require immediate user attention and are often utilized for performing background tasks that run discreetly without user interaction. These tasks may include periodic data synchronization, content indexing, or system maintenance operations. Although background services do not prioritize user visibility, they remain essential for maintaining the seamless operation of applications by handling non-urgent tasks efficiently. However, it's important to note that background services have a lower priority than foreground services and may be subject to termination by the system when resources are limited, such as during low memory conditions.
* In summary, services in Android provide a powerful mechanism for executing long-running tasks or operations in the background, ensuring the continued functionality and responsiveness of applications. By distinguishing between foreground and background services based on user visibility and priority, developers can design robust and user-friendly applications that effectively balance performance and user experience considerations. Whether it's delivering real-time updates to users through foreground services or silently managing background tasks without interrupting user interaction, services play a pivotal role in shaping the behavior and functionality of Android applications.



1. **Content provider**

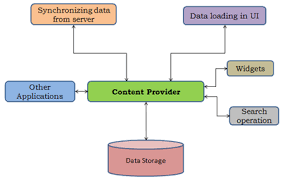
* In the intricate ecosystem of Android app development, Content Providers stand as indispensable components facilitating seamless data sharing and access management among applications. Acting as intermediaries between applications and data sources, Content Providers serve as the backbone for organizing and controlling access to structured sets of data, thus promoting data consistency, security, and interoperability.

* At its core, a Content Provider acts as an abstraction layer, shielding the underlying data source - whether it be a database, file, or any other structured storage mechanism - from direct manipulation by applications. By encapsulating data access logic within a standardized interface, Content Providers offer a unified and consistent method for applications to interact with shared data, regardless of the underlying implementation details.

* One of the primary objectives of Content Providers is to centralize and regulate access to a specific data source, ensuring that multiple applications can securely access and manipulate the data without compromising its integrity. This centralized approach not only streamlines data management and maintenance but also enhances data security by enforcing access controls and permissions
* Moreover, Content Providers foster collaboration and interoperability among disparate applications by providing a common platform for data exchange. Applications can seamlessly query, insert, update, and delete data through Content Providers, without needing to understand the intricacies of the underlying data storage mechanism. This standardized approach simplifies the development process and promotes code reusability across different applications, ultimately enhancing developer productivity and accelerating time-to-market.

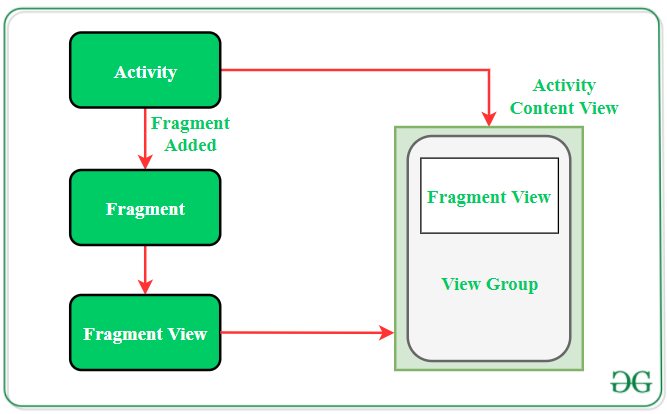
* Furthermore, Content Providers play a crucial role in enabling data sharing and integration across the Android ecosystem. By exposing a well-defined interface, Content Providers allow applications to securely share data with each other, facilitating a wide range of use cases such as content synchronization, cross-application communication, and data aggregation.
* In essence, Content Providers serve as linchpins in the Android platform, fostering a collaborative environment where applications can seamlessly interact with shared data sources. By offering a standardized approach to data access and sharing, Content Providers empower developers to build robust and interoperable applications that deliver compelling user experiences while adhering to best practices in data management and security. As the Android ecosystem continues to evolve, Content Providers remain essential components for enabling seamless data integration and interoperability across diverse applications and use cases.
* A Content Provider in Android is a component that manages the access to a structured set of data. It acts as an abstraction layer that allows applications to securely share data with each other, providing a standardized way to interact with a particular data source.

* The primary purpose of a Content Provider is to centralize and control access to a shared data source. This data source can be a database, file, or any other structured storage mechanism. Content Providers facilitate data sharing between different applications, allowing them to query, insert, update, and delete data using a common interface.



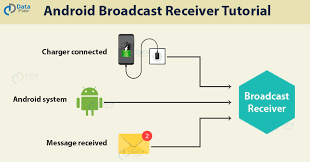
1. **Fragments**

* Comprise modular UI components that can be embedded within activities to create complex and dynamic layouts.
* Offer greater flexibility and responsiveness, particularly on larger screens or foldable devices.
* Possess the ability to be dynamically added, removed, or replaced based on user actions or program logic.
* Share the lifecycle of their hosting activity but also maintain independent lifecycle events for finer control over behavior.



1. **Broadcast Receivers**

* Act as passive listeners for system-wide events and announcements, such as battery level changes, incoming calls, or network connectivity modifications.
* Enable your application to react and adapt to these events even when not actively running in the foreground.
* Each receiver registers for specific event types and triggers predefined actions or updates the user interface accordingly, enhancing context awareness and responsiveness.
* Can be dynamically registered and unregistered, optimizing battery consumption by listening only for relevant events.



1. **Manifest File (AndroidManifest.xml)**

* Serves as the official declaration of your application's structure, components, permissions, and configurations.
* Informs the Android system about the application's composition and interaction mechanisms.
* Declares activities, services, content providers, and other essential components, specifying their names, launch modes, and required permissions.
* Acts as a roadmap for the Android system to understand the application's capabilities and facilitate component launch, user interaction handling, and security enforcement.



1. **Resources**

* Encompass the visual and audio building blocks of your application, including layouts, images, icons, strings, and styles defining the user interface's look and feel.
* Organized in dedicated folders for efficient management and separation of concerns.
* Accessed from code using unique identifiers, enabling consistent management and centralized control over the application's visual and audio elements.

1. **Build Files (build.gradle)**

* Define and automate the build and preparation process for your application's final deployment package.
* Specify dependencies, configurations, and build instructions for the project.
* Define the project structure, including source code directories, libraries used, and plugins required for development and testing.
* Ensure consistent and reliable creation of the final application package ready for installation on devices.

1. **Code Files (Java or Kotlin)**

* Represent the core logic and functionality of your application, breathing life into the components and user interface.
* Implement the behavior of activities, services, and other components.
* Utilize Android APIs to interact with the device's features and resources, such as sensors, cameras, network access, and storage.
* Mastering the programming language and Android APIs is crucial for building robust and efficient applications that leverage the platform's full potential.

**3. Folder and files in an Android project ( new project, select “Empty View: Activity”)**

First app: <https://developer.android.com/training/basics/firstapp>

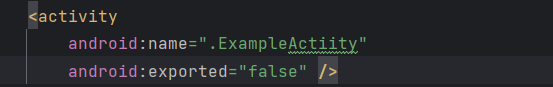
* **Key Files:**

A screenshot of a computer

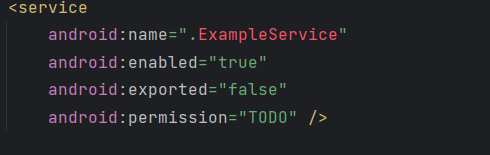
Description automatically generated

|  |  |
| --- | --- |
| **Sr.No.** | **Folder, File & Description** |
| 1 | **Java**  This contains the **.java** source files for your project. By default, it includes an MainActivity.java source file having an activity class that runs when your app is launched using the app icon. |
| 2 | **res/drawable-hdpi**  This is a directory for drawable objects that are designed for high-density screens. |
| 3 | **res/layout**  This is a directory for files that define your app's user interface. |
| 4 | **res/values**  This is a directory for other various XML files that contain a collection of resources, such as strings and colours definitions. |
| 5 | **AndroidManifest.xml**  This is the manifest file which describes the fundamental characteristics of the app and defines each of its components. |
| 6 | **Build.gradle**  This is an auto generated file which contains compileSdkVersion, buildToolsVersion, applicationId, minSdkVersion, targetSdkVersion, versionCode and versionName |

1. Components in an Android Project
   1. Activity:

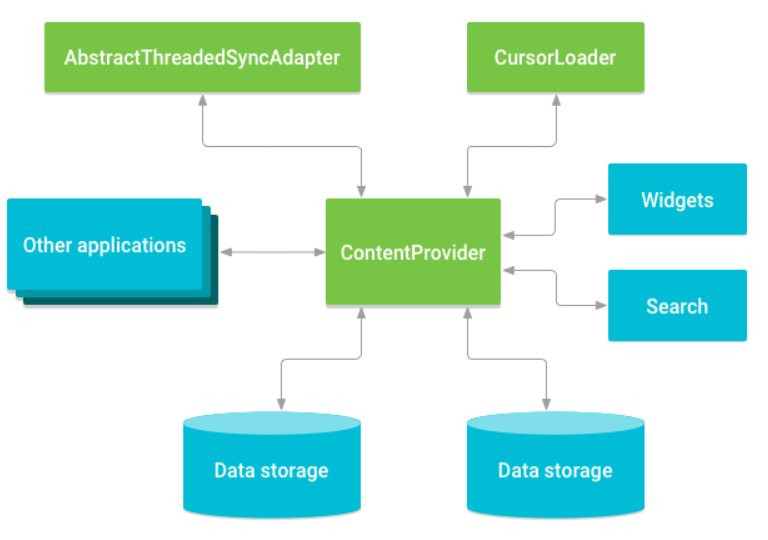


* + - An activity represents a single screen with a user interface,in-short Activity performs actions on the screen.
    - If an application has more than one activity, then one of them should be marked as the activity that is presented when the application is launched.
  1. Service:

****

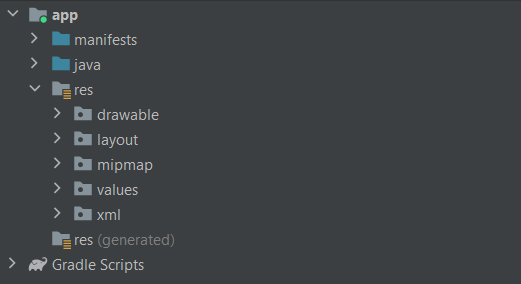
A service is a component that runs in the background to perform long-running operations.

* 1. Content provider:



Broadcast Receivers simply respond to broadcast messages from other applications or from the system

* 1. Addications Components



There are additional components which will be used in the construction of above mentioned entities, their logic, and wiring between them.

* 1. Intents:

It is a powerful inter-application message-passing framework. They are extensively used throughout Android. Intents can be used to start and stop Activities and Services, to broadcast messages system-wide or to an explicit Activity, Service or Broadcast Receiver or to request action be performed on a particular piece of data.

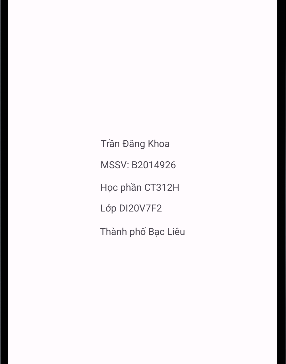
* 1. Widgets:

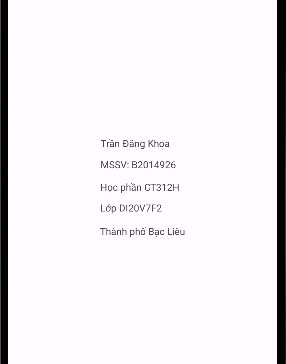
These are the small visual application components that you can find on the home screen of the devices. They are a special variation of [Broadcast Receivers](https://www.geeksforgeeks.org/broadcast-receiver-in-android-with-example/) that allow us to create dynamic, interactive application components for users to embed on their Home Screen.

1. Folders and files in an Android project (new project, select “Empty Views Activity”)

First app: <https://developer.android.com/training/basics/firstapp>

Code: A simple app to introduce your info





**Lab 2. LAYOUT and Resolutions**

Refer:

* <https://developer.android.com/guide/topics/ui/declaring-layout>
* <https://developer.android.com/training/constraint-layout>

* <https://www.tutorialspoint.com/android/android_user_interface_layouts.htm>

* <https://developer.android.com/training/multiscreen/screendensities>
* In Android development, a layout serves as the blueprint for defining the visual structure and arrangement of user interface elements within an application, such as an activity or fragment. It provides a framework for organizing and presenting various components, including text views, buttons, images, and more, in a visually appealing and intuitive manner.
* At the heart of every layout lie View and ViewGroup objects, which form the building blocks of the user interface hierarchy. Views represent individual UI elements, such as buttons or text fields, while ViewGroups act as containers that hold and arrange multiple views within them. By nesting views within view groups, developers can create complex and flexible layouts that adapt to different screen sizes, orientations, and device types.

* In Android, developers have the flexibility to declare layouts using two primary approaches: declarative layout XML (Extensible Markup Language) and dynamic layout instantiation at runtime.

* Declarative Layout XML: This approach involves defining the layout structure and properties of UI elements using XML markup within layout resource files. Developers can specify the arrangement, positioning, styling, and behavior of UI components by declaring attributes and nesting views within ViewGroup containers. Declarative XML layouts offer several advantages, including improved code readability, separation of concerns between layout and logic, and efficient design iteration through visual editors and preview tools provided by integrated development environments (IDEs) like Android Studio.

* Dynamic Layout Instantiation at Runtime: Alternatively, developers can programmatically create and configure layout elements at runtime using Java or Kotlin code. This method allows for greater flexibility and control over the user interface, enabling dynamic layout generation based on runtime conditions, user interactions, or data-driven requirements. While dynamic layout instantiation offers versatility and adaptability, it may require more complex code and incur additional overhead compared to declarative XML layouts.

* By leveraging both approaches judiciously, developers can create rich, responsive, and user-friendly interfaces that cater to diverse user needs and device configurations. Whether opting for declarative XML layouts to streamline design workflows or employing dynamic layout instantiation for enhanced runtime flexibility, mastering the art of layout design is essential for crafting compelling and intuitive Android applications that delight users and drive engagement.
* In Android, a layout defines the visual structure for a user interface in your app, such as in an activity. All elements in the layout are built using a hierarchy of *View* and *ViewGroup* objects. You can declare a layout in two ways: declare UI elements in XM and instantiate layout elements at runtime1

A diagram of a company

Description automatically generated

Types of Android Layout:

* Constraint Layout: can create large, complex layouts with a flat view hierarchy - no nested view groups. It is also a ViewGroup subclass, used to specify the position of layout constraints for every child View relative to other views present. A ConstraintLayout is similar to a RelativeLayout but in that all views are laid out according to relationships between sibling views and the parent layout, but it's more flexible than RelativeLayout and easier to use with Android Studio's Layout Editor.

To define a view's position in ConstraintLayout, add at least one horizontal and one vertical constraint for the view. Each constraint represents a connection or alignment to another view, the parent layout, or an invisible guideline. Each constraint defines the view's position along the vertical or horizontal axis. Each view must have a minimum of one constraint for each axis, but often more are necessary. If a view has no constraints, the layout on a device will drawn at position [0,0] (the top-left corner).

  In figure, the layout looks good in the editor, but there is no vertical constraint on view C. When this layout runs on a device, view C horizontally aligns with the left and right edges of view A, but it appears at the top of the screen because it has no vertical constraint.

A diagram of a diagram

Description automatically generated

* Linear Layout: is a ViewGroup subclass, used to provide child View elements in a particular direction either horizontally or vertically based on the orientation property. It can specify the layout direction with the android:orientation attribute.
* Table Layout: is a ViewGroup subclass, used to display the child View elements in rows and columns. TableLayout positions its children into rows and columns. TableLayout containers do not display border lines for their rows, columns, or cells. The table will have as many columns as the row with the most cells. A table can leave cells empty. Cells can span multiple columns, as they can in HTML.
* FrameLayout: is designed to block out an area on the screen to display a single item. Generally, FrameLayout should be used to hold a single child view, because it can be difficult to organize child views in a way that's scalable to different screen sizes without the children overlapping each other. You can, however, add multiple children to a FrameLayout and control their position within the FrameLayout by assigning gravity to each child, using the [android:layout\_gravity](https://developer.android.com/reference/android/widget/FrameLayout.LayoutParams#attr_android:layout_gravity) attribute.

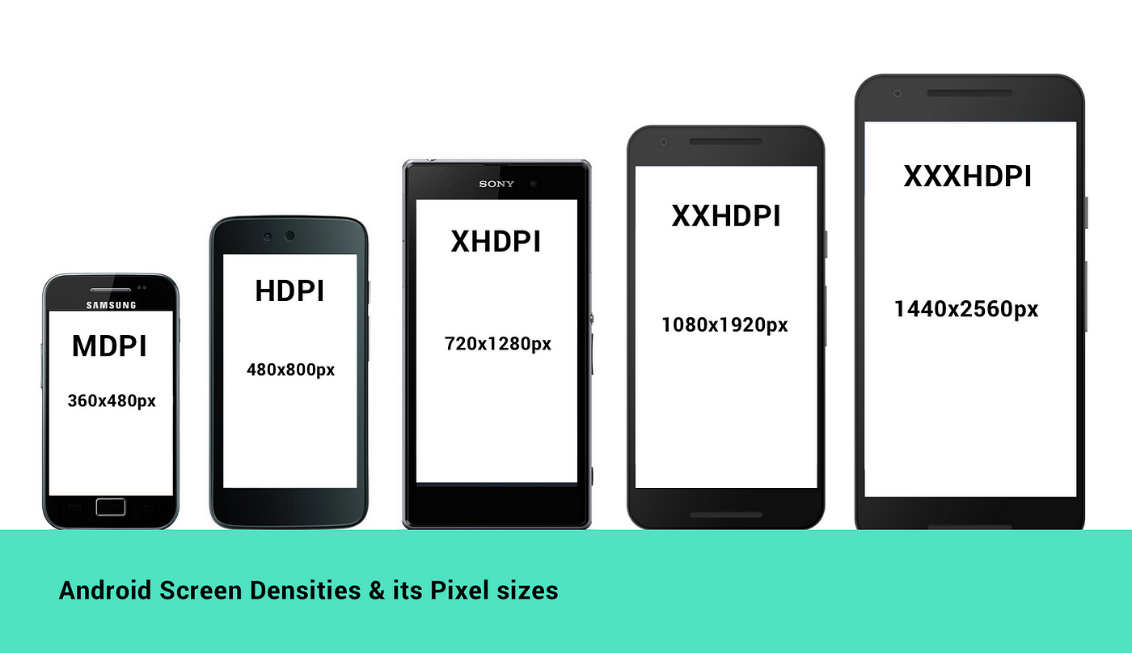
1. **Introduction to resolutions (dp, dpi, hdpi,...)**

In the realm of Android development, understanding resolutions is paramount to crafting user interfaces that seamlessly adapt to the diverse array of devices in the market. Key to this comprehension are concepts like density-independent pixels (dp or dip) and dots per inch (dpi), alongside their associated qualifiers such as mdpi, hdpi, xhdpi, and beyond.

**Density-independent Pixels** (dp or dip) serve as the cornerstone for designing layouts that remain consistent across various screen sizes and resolutions. These virtual pixels provide a standardized unit of measurement, allowing developers to specify dimensions and positions in a manner that scales proportionally across different screen densities. With one dp roughly equivalent to one pixel on a medium-density screen (160 dpi), Android dynamically adjusts the actual pixel count for each screen density, ensuring uniform visual presentation across devices.

**Dots Per Inch** (dpi) represents the density of pixels packed within one inch of a screen, serving as a metric for screen resolution. Android accommodates a spectrum of screen densities, ranging from low-density screens (ldpi) to high-density screens (xhdpi, xxhdpi, xxxhdpi), with each density level correlating to a specific dpi value. As dpi increases, so does the pixel density within each inch of the screen, resulting in crisper and more detailed visual output.

Moreover, **scalable pixels** (sp) offer a versatile alternative to dp for defining text sizes within an application. While sp initially mirrors the size of dp, it dynamically adjusts based on the user's preferred text size settings. This adaptive behavior ensures that text remains legible and accessible across different devices and user preferences, catering to diverse user needs and accessibility requirements.

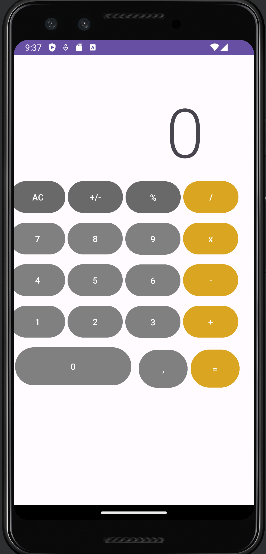


In Android development, developers leverage a range of density buckets to tailor their apps to different screen densities and resolutions. These buckets, including mdpi, hdpi, xhdpi, xxhdpi, and xxxhdpi, allow developers to provide optimized graphical assets and layouts for devices with varying pixel densities. By targeting specific density buckets, developers can ensure that their apps deliver an optimal user experience across a wide spectrum of devices, from entry-level smartphones to high-resolution tablets and beyond

A screenshot of a computer

Description automatically generated

Code: Design a mini calculator (just for the interface, don't need event handler)



**Lab 3: Event handling (1): TEXTVIEW, EDITTEXT, BUTTON**

References:

* <https://www.tutorialspoint.com/android/android_event_handling.htm>

**1. Introduction**

Event handling is a critical aspect of Android application development, enabling developers to respond to user interactions and system events effectively. This report provides an in-depth overview of event handling in Android, covering the fundamental concepts, mechanisms, and best practices.

At the core of event handling lies the Event Listener, an interface nestled within the View class. This interface embodies a singular callback method that triggers when a user interacts with a UI element. Think of Event Listeners as vigilant sentinels, eagerly awaiting events to occur, thereby enabling developers to define precise actions corresponding to user inputs or system events.

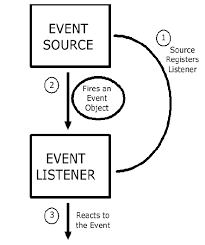
The crux of event handling lies within Event Handlers, designated methods encapsulating the logic to be executed upon event detection. These handlers act as the responsive elements of an application, interpreting user inputs or system-triggered events and orchestrating appropriate actions. Whether it's updating UI elements, triggering animations, or invoking backend processes, Event Handlers serve as the linchpin in delivering a dynamic and engaging user experience.

Event Listeners, however, do not operate in isolation; they require registration with an Event Handler to facilitate seamless communication. Event Handlers serve as conduits between Event Listeners and the associated logic that executes upon event detection. Through registration, developers establish a symbiotic relationship, akin to subscribing to a specific channel of events, ensuring that the handler responds promptly and accurately to user interactions.

Efficiency is paramount in event handling to ensure optimal performance and responsiveness. Adopting best practices such as utilizing asynchronous event handling mechanisms, minimizing the scope of event listeners, and implementing event batching techniques can mitigate latency and enhance overall application robustness. Additionally, adhering to the principles of separation of concerns and modularization fosters maintainability and scalability, facilitating streamlined event handling across diverse application components.

Efficient event handling is crucial for creating responsive and interactive Android applications. Developers must understand the underlying mechanisms to ensure a seamless user experience.

* **An Event Listen**er is an interface within the View class, encompassing a singular callback method. When a user interacts with a UI element, the Android framework calls this method, enabling developers to define specific actions. Essentially, Event Listeners serve as the ears of the application, eagerly waiting for events to occur.



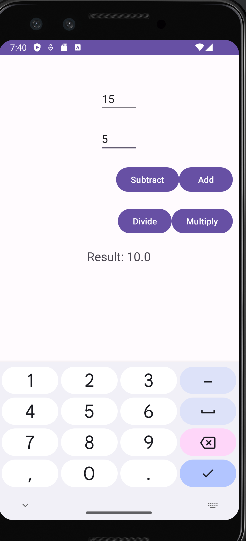
* **Event Listeners Registration** is the mechanism through which an Event Handler aligns itself with an Event Listener. By registering a handler with a listener, developers establish a connection that ensures the handler responds appropriately when the associated event takes place. It's akin to subscribing to a specific channel of events.
* **Event Handlers** are the designated methods that execute when a registered Event Listener detects a corresponding event. These handlers encapsulate the logic to be executed when the event occurs. They act as the responsive elements of the application, taking action based on the user's input or system events.

| Event Handler | Event Listener & Description |
| --- | --- |
| onClick() | **OnClickListener()**  This is called when the user either clicks or touches or focuses upon any widget like button, text, image etc. You will use onClick() event handler to handle such event. |
| onLongClick() | **OnLongClickListener()**  This is called when the user either clicks or touches or focuses upon any widget like button, text, image etc. for one or more seconds. You will use onLongClick() event handler to handle such event. |
| onFocusChange() | **OnFocusChangeListener()**  This is called when the widget looses its focus ie. user goes away from the view item. You will use onFocusChange() event handler to handle such event. |
| onKey() | **OnFocusChangeListener()**  This is called when the user is focused on the item and presses or releases a hardware key on the device. You will use onKey() event handler to handle such event. |
| onTouch() | **OnTouchListener()**  This is called when the user presses the key, releases the key, or any movement gesture on the screen. You will use onTouch() event handler to handle such event. |
| onMenuItemClick() | **OnMenuItemClickListener()**  This is called when the user selects a menu item. You will use onMenuItemClick() event handler to handle such event. |
| onCreateContextMenu() | **onCreateContextMenuItemListener()**  This is called when the context menu is being built(as the result of a sustained "long click) |

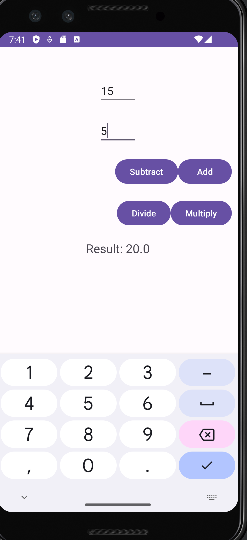
**Design a simple app to perform Event handling using 2 methods.**

* 2 text box (a, b)
* 5 buttons: +, -, \*, / and combine a&b (e.g., a = 5 and b = 7: a&b = 57)

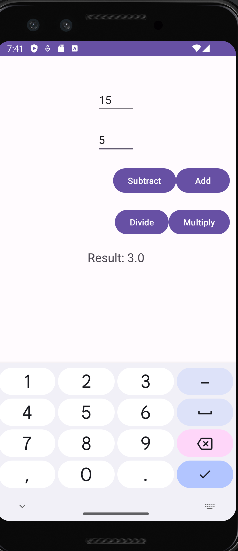
A-B:



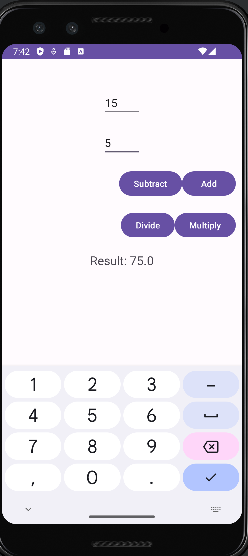
A+B:



A/B:



A\*B:



**Lab 4: Event handling (2): RADIO BUTTON, CHECKBOX, TOAST and IMAGEBUTTON**

References:

- <https://www.tutorialspoint.com/android/android_user_interface_controls.htm>

**1. Introduction:**

User Interface (UI) controls serve as the fundamental building blocks of any software application's graphical user interface (GUI), facilitating seamless interaction between users and the system. These elements or components are meticulously crafted to provide an intuitive and visually appealing means for users to input data, make selections, and trigger actions, thereby enhancing the overall user experience.

At the heart of every well-designed UI lies a diverse array of controls, each tailored to fulfill specific functions and cater to varied user needs. From simple text input fields and checkboxes to complex dropdown menus and sliders, these controls are strategically positioned within the interface to streamline navigation and empower users to effortlessly accomplish tasks.

Text input fields stand as the ubiquitous entry points where users can type in information, ranging from basic text to numerical values, dates, or even complex formulas, depending on the application's requirements. Meanwhile, checkboxes and radio buttons offer users discrete options to select from, ideal for scenarios where choices are mutually exclusive or where multiple selections are permissible.

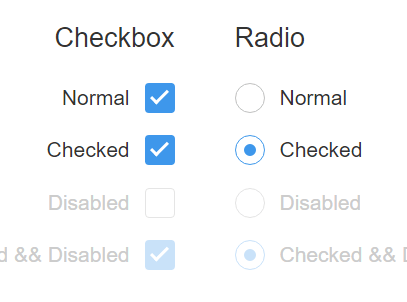
Dropdown menus and list boxes provide users with a structured set of options, conserving screen space while accommodating a wide range of choices. These controls are particularly useful in situations where the available options are extensive or hierarchical in nature, allowing users to navigate through nested categories with ease.

Furthermore, sliders and range selectors empower users to specify numerical values within predefined ranges, offering a dynamic and visually intuitive method for setting parameters or adjusting settings. Such controls are invaluable in applications where precision and fine-grained control are paramount, such as in image editing software or financial analysis tools.

Beyond mere data input, UI controls also serve as triggers for actions, enabling users to initiate processes or invoke functionalities within the application. Buttons, for instance, come in various forms, including standard push buttons, toggle buttons, and icon buttons, each tailored to suit different interaction contexts. These controls provide clear visual cues, prompting users to perform specific actions such as submitting a form, saving changes, or initiating a search operation.

Moreover, UI controls often incorporate feedback mechanisms to enhance usability and provide users with real-time information about their interactions. Tooltips, for instance, offer contextual guidance by displaying explanatory text when users hover over or interact with a control, clarifying its purpose or functionality. Similarly, validation messages and error indicators alert users to any input errors or inconsistencies, guiding them towards corrective actions and minimizing frustration.

In essence, UI controls play a pivotal role in shaping the user experience, serving as the conduit through which users engage with software applications. By leveraging intuitive design principles, thoughtful placement, and responsive feedback mechanisms, developers can create interfaces that empower users, foster productivity, and ultimately, elevate the overall quality of the software product.



**1.1 TextView**

The TextView is a pivotal component in Android app development, serving as the primary means to present text-based information to users. Its versatility lies not only in displaying plain text but also in supporting various formatting options, including font styles, sizes, colors, and alignments. This flexibility enables developers to create visually appealing interfaces that align with the app's design aesthetic and branding guidelines. Additionally, the TextView's rich text formatting capabilities allow for the inclusion of styling elements such as bold, italic, underline, and strikethrough text, as well as multimedia content like images and hyperlinks, enriching the user experience.

Furthermore, the TextView plays a crucial role in ensuring accessibility by offering features such as content descriptions, screen reader compatibility, and text scaling options. These accessibility features cater to users with diverse needs and preferences, fostering inclusivity and usability across a wide range of demographics. By prioritizing accessibility, developers can create apps that are accessible to all users, regardless of their abilities or assistive technology requirements. In essence, the TextView serves as a cornerstone of Android UI development, empowering developers to deliver informative, engaging, and accessible textual content within their applications.

**1.2 EditText**

EditText, a subclass of TextView, stands out in Android development for its rich editing capabilities, enabling users to input and manipulate text within the app interface. This dynamic component is particularly valuable in scenarios demanding user interaction with textual data, such as filling out forms or conducting searches. Unlike TextView, EditText allows users to not only view but also directly modify text content, offering features like text selection, cursor placement, and keyboard input.

The versatility of EditText extends beyond basic text input, offering developers a range of customization options to tailor the editing experience to specific application requirements. Developers can define input types to restrict the kind of data users can enter, such as numeric input for phone numbers or email addresses, or password input for sensitive information. Additionally, EditText supports various attributes for controlling text appearance, behavior, and validation, empowering developers to create intuitive and user-friendly input interfaces.

Moreover, EditText seamlessly integrates with other UI elements and functionalities, enabling developers to implement features like auto-complete suggestions, input validation, and real-time text formatting. This enhances the user experience by providing immediate feedback and assistance during text input, thereby improving efficiency and reducing errors. Overall, EditText serves as a cornerstone of interactive Android app development, facilitating seamless user interaction with textual data and enhancing the usability and functionality of applications across diverse use cases.

EditText is a predefined subclass of TextView that includes rich editing capabilities. It allows users to input and modify text, making it suitable for scenarios where user interaction with textual data is required, such as forms or search fields.

**1.3 AutoCompleteTextView**

AutoCompleteTextView, an extension of EditText, offers a sophisticated enhancement to user input by displaying a list of completion suggestions in real-time as the user types. This feature streamlines the input process by presenting relevant options based on the entered text, greatly improving efficiency and accuracy, particularly in search functionalities and other contexts requiring data input.

Unlike traditional EditText views, AutoCompleteTextView dynamically populates a dropdown list with potential completions as the user types, leveraging a data source such as a predefined list, a database query, or an API response. These suggestions are displayed based on matching criteria, such as prefix matching or fuzzy matching, ensuring that the presented options closely align with the user's input.

**1.4 Button**

In Android development, Buttons are indispensable elements that facilitate user interaction by providing a clear and direct means to initiate actions within an application. With a tap or click, users can trigger various operations, from submitting forms to navigating between different app screens. Buttons are designed for simplicity and ease of use, featuring easily recognizable labels or icons that convey their purpose, thus ensuring intuitive navigation throughout the app.

Moreover, Buttons offer developers extensive customization options, allowing them to adapt the appearance and behavior of Buttons to suit the specific requirements and design aesthetics of their applications. Whether adjusting colors, sizes, or animations, developers can tailor Buttons to seamlessly integrate into the overall user interface, enhancing both usability and visual appeal. In summary, Buttons serve as essential components in Android apps, empowering users to interact effortlessly while contributing to a cohesive and engaging user experience.

**1.5 ImageButton**

An ImageButton in Android provides developers with precise control over the placement of its children within an AbsoluteLayout. Unlike conventional buttons, an ImageButton displays an image rather than text, offering users a visually engaging and interactive means to trigger actions or access features within an application. This unique characteristic makes ImageButton particularly suitable for scenarios where graphical icons or symbols are more effective than textual labels in conveying functionality or intent.

By allowing developers to specify the exact location of its children within the layout, ImageButton offers flexibility in designing user interfaces that prioritize visual appeal and intuitive interaction. Whether positioned as standalone elements or integrated seamlessly within complex layouts, ImageButtons enable developers to create compelling user experiences that resonate with modern design trends and user expectations.

**1.6 CheckBox**

A CheckBox in Android presents users with an intuitive on/off switch that they can toggle to select or deselect options within a group. Unlike radio buttons, which allow only single selections from a group of options, CheckBoxes enable users to make multiple selections simultaneously. This functionality is particularly useful when users need to choose from a set of options that are not mutually exclusive, such as selecting multiple items from a list or enabling various features or preferences.

CheckBoxes offer a straightforward and familiar interface, featuring a small square box that can be checked or unchecked with a tap or click. The visual indicator provides immediate feedback to users, clearly indicating the current selection status of each option. Additionally, CheckBoxes can be customized with labels or descriptions to provide context and clarity about the purpose of each selectable item, enhancing usability and comprehension.

Overall, CheckBoxes play a crucial role in Android app interfaces, offering users a convenient and efficient way to make multiple selections from a group of options. Whether used in forms, settings menus, or list views, CheckBoxes contribute to a seamless and intuitive user experience, allowing users to interact with applications effortlessly and efficiently.

**1.7 ToggleButton**

A ToggleButton in Android is a specialized button that serves as a binary switch, allowing users to toggle between two states: on and off. It is equipped with a light indicator that visually communicates its current state, providing users with immediate feedback about the activation status of a particular feature or setting. ToggleButtons are particularly well-suited for scenarios where a simple binary toggle is sufficient, such as enabling or disabling a specific functionality or setting within an application.

The visual indicator of a ToggleButton typically changes in response to user interaction, clearly indicating whether the button is currently in the "on" or "off" position. This visual feedback enhances user comprehension and helps users quickly grasp the current state of the toggled feature. Additionally, ToggleButtons are designed to be highly intuitive, with a straightforward interface that encourages users to interact with them effortlessly.

**1.8 RadioButton**

The RadioButton in Android represents a selectable option with two states: checked or unchecked. It is commonly employed in scenarios where users are presented with multiple choices, but only one selection is permitted from the available options. RadioButton groups are designed to ensure mutual exclusivity, meaning that selecting one RadioButton automatically deselects any previously selected option within the same group. This behavior provides users with a clear and concise method for making single selections from a list of choices, enhancing usability and minimizing ambiguity.

The RadioButton's visual representation typically includes a small circular indicator that denotes its current state, along with a label or description to provide context about the selectable option. When a RadioButton is selected, the indicator is filled, indicating its checked state, while all other options in the group remain unchecked. This visual feedback helps users easily discern the selected option from the available choices, facilitating informed decision-making and reducing the likelihood of errors.

**1.9 RadioGroup**

In Android development, a RadioGroup is a container view used to group together one or more RadioButtons. Its primary function is to enforce mutual exclusivity among the RadioButtons it contains, ensuring that only one RadioButton within the group can be selected at any given time. This functionality facilitates the presentation of a set of exclusive options to the user, where only a single selection is permitted from among the available choices.

When RadioButtons are added to a RadioGroup, they automatically become part of the group, and the RadioGroup manages their selection states. As a result, selecting one RadioButton within the group automatically deselects any previously selected RadioButton, ensuring that only one option remains checked at any given time. This behavior simplifies user interaction and reduces the risk of ambiguity or errors when making selections from a list of options.

Overall, the RadioGroup serves as a crucial component in Android user interfaces, providing a convenient and intuitive mechanism for presenting exclusive choices to users. By enforcing mutual exclusivity among RadioButtons, the RadioGroup promotes clarity, consistency, and ease of use, enhancing the overall user experience in applications where single selections from a predefined set of options are required.

**1.10 ProgressBar**

The ProgressBar view in Android serves as a vital tool for providing visual feedback to users about the progress of ongoing tasks, particularly those occurring in the background. It offers a graphical representation of the progression of a task, reassuring users that the application is actively working on their request.

By displaying a dynamic visual indicator, such as a moving bar or a spinning wheel, the ProgressBar communicates to users that their action has been acknowledged and that the application is processing their request. This feedback mechanism is essential for enhancing user confidence and satisfaction, especially in situations where tasks may take some time to complete, such as loading data from a server or performing complex calculations.

**11. Spinner**

A Spinner in Android presents users with a compact and interactive drop-down list, offering them a convenient way to select a single value from a predefined set of options. This UI component provides users with a visually appealing and intuitive interface for making selections, especially when the available options are limited and can be displayed succinctly.

When users interact with a Spinner, they can tap or click on it to reveal the list of options, which typically appears as a drop-down menu below the Spinner's control. From there, users can scroll through the list and choose the desired option, which then becomes the selected value displayed in the Spinner's main area.

Spinners are commonly used in various contexts within Android applications, such as selecting a country from a list, choosing a category for an item, or picking a date from a list of available dates. Their versatility and ease of use make them a popular choice for presenting selectable options in a visually appealing and space-efficient manner.

**1.12 TimePicker**

The TimePicker view in Android offers users a convenient way to select a specific time of day, whether in 24-hour mode or AM/PM mode. This UI component is particularly useful in scenarios where precise time input is needed, such as setting alarms, scheduling events, or configuring reminders.

With the TimePicker, users can easily adjust the hour and minute values using intuitive touch-based controls. They can switch between 24-hour and AM/PM modes based on their preferences or the requirements of the application. Additionally, the TimePicker often provides visual indicators, such as arrows or scrolling wheels, to aid users in selecting the desired time accurately.

By integrating the TimePicker into their applications, developers can streamline the process of time selection, enhancing user efficiency and reducing input errors. Whether used in productivity apps, calendar applications, or alarm clocks, the TimePicker simplifies the task of setting specific times, contributing to a more intuitive and user-friendly experience overall.

**1.13 DatePicker**

The DatePicker view in Android provides users with a straightforward method to select a specific date. It's a fundamental component in applications where date input is crucial, such as in calendars, scheduling tools, or date-based reminders. With the DatePicker, users can easily navigate through months, days, and years to choose the desired date.

This UI element offers an intuitive and user-friendly interface, typically presenting users with a calendar-like layout or a combination of dropdown menus for month, day, and year selection. Users can tap or swipe to navigate between different months and years, ensuring ease of use and efficiency when selecting dates.

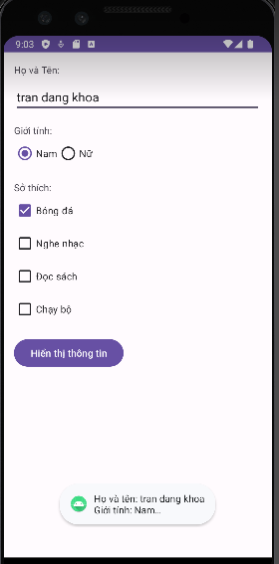
The DatePicker enhances the overall user experience by simplifying the process of date selection within applications. By integrating this component, developers can provide users with a seamless and intuitive way to input dates, thereby improving usability and reducing user frustration. Overall, the DatePicker facilitates efficient date selection, making it an essential tool in various Android applications.

**1.14 Toast**

In Android development, a Toast serves as a brief, non-intrusive pop-up message that appears at the bottom of the screen to provide users with simple feedback or notifications. Toasts are commonly utilized to display information momentarily, without necessitating any user interaction. They offer a lightweight and unobtrusive means of communicating essential messages, such as informing users about the success of an action, displaying error notifications, or providing contextual hints.

Toast messages typically consist of a short text message or an icon accompanied by a brief explanatory text. They appear briefly on the screen before automatically disappearing, ensuring that they do not disrupt the user's workflow or obstruct the view of the underlying content. This minimalistic approach to feedback delivery helps maintain the fluidity and responsiveness of the user interface while still keeping users informed about relevant events or actions.

Overall, Toasts play a vital role in enhancing the user experience in Android applications by providing timely and informative feedback to users. Whether used to acknowledge user input, alert users to system events, or offer guidance within the app, Toast messages contribute to a more intuitive and user-friendly interaction environment.

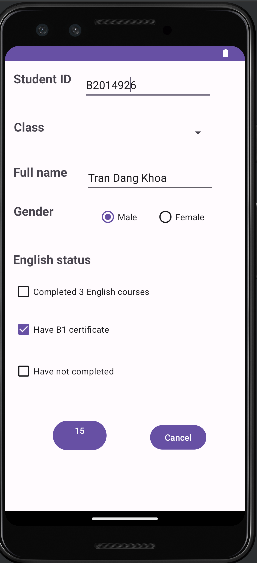


**Lab 5: Progress test 1**

English status app

* Student ID (textview + edittext)
* Class (textview + [Spinner](https://www.tutorialspoint.com/android/android_spinner_control.htm))
* Full name (textview + edittext)
* Gender (Radio button)
* birthDate (date picker), [example](https://www.tutorialspoint.com/android/android_datepicker_control.htm)
* English status:
  + Completed 3 English courses (checkbox)
  + Have B2 certificate (checkbox)
  + Have not completed (checkbox)

OK, CANCEL (button)



**Lab 6: Event handler (3): LISTVIEW**

References:

* <https://www.tutorialspoint.com/android/android_list_view.htm>
* <https://abhiandroid.com/ui/listview>

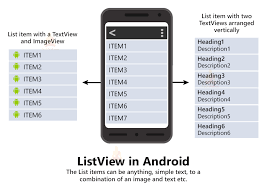
Ex1: reproduce the app at <https://xuanthulab.net/su-dung-listview-hien-thi-du-lieu-dang-danh-sach-trong-android.html>

**1. Introduction:**

**-  ListView**

The ListView component in Android serves as a fundamental building block for presenting scrollable lists of items within mobile applications. Widely utilized across a spectrum of applications, ListView offers developers a versatile tool to display various types of data, ranging from simple text entries to complex multimedia content. This report provides an overview of ListView, detailing its functionalities, customization options, and its significance in Android development.

ListView, a subclass of ViewGroup, facilitates the presentation of data in a vertically scrollable list format. It enables users to navigate through a collection of items efficiently, accommodating diverse content types with ease. Whether it's showcasing contacts in a messaging app or cataloging products in an e-commerce platform, ListView caters to a multitude of use cases, making it a cornerstone in Android UI design.



**Customization Capabilities**

One of ListView's notable features is its flexibility in customization. Developers can tailor the appearance of list items by leveraging custom layouts and adapters. This allows for the seamless integration of various UI elements, including text fields, images, buttons, and more. By harnessing adapters, ListView adapts data from a source and dynamically populates the list, ensuring a cohesive and interactive user experience.

**Efficient Scrolling Mechanism**

ListView excels in handling large datasets by implementing efficient scrolling mechanisms. Through the concept of view recycling, ListView optimizes memory usage and rendering performance. As users navigate through the list, ListView dynamically reuses existing views, minimizing resource consumption and mitigating performance bottlenecks. This enables smooth and responsive scrolling, even when dealing with extensive datasets, enhancing the overall usability of the application.

**User Interaction and Event Handling**

Incorporating user interaction features, ListView provides support for item click events, enabling developers to respond to user actions effectively. By attaching ItemClickListeners to individual list items, developers can implement custom behavior, such as navigating to detailed views or performing data manipulation operations. This enhances the interactivity of the application, empowering users to engage with the content seamlessly.

In conclusion, ListView stands as a foundational component in Android development, offering a robust solution for presenting lists of data. Its versatility, customization options, and optimization techniques make it indispensable for crafting engaging and intuitive user interfaces. As Android applications continue to evolve, ListView remains a vital tool for developers seeking to deliver compelling user experiences across a diverse range of contexts.

* **Custom ListView**

+ Within the realm of Android development, the context menu stands as a pivotal user interface element, offering users a convenient means of accessing contextually relevant actions or operations within an application. Rooted in the principle of user-centric design, the context menu dynamically adapts to the user's interactions, presenting a curated list of options tailored to the specific context of their current activity. This comprehensive introduction delves into the nuanced functionality, implementation intricacies, and overarching significance of context menus in Android applications, illuminating their pivotal role in optimizing user experience and interaction efficiency.



* **Functionality of Context Menu**

+ At its core, the context menu serves as a gateway to a plethora of actions or functionalities closely aligned with the user's immediate context within the application interface. Triggered by long-press gestures on supported UI elements, such as list items, images, or text fields, the context menu surfaces an array of actionable items pertinent to the selected element. These actions encompass a diverse spectrum, ranging from fundamental operations like editing, deleting, or sharing content to more intricate functionalities like initiating navigation or invoking contextual help resources. By distilling pertinent actions into a concise and accessible menu, the context menu empowers users to accomplish tasks expediently, fostering a seamless and intuitive interaction paradigm.

* **Implementation in Android Applications**

+ Efficient integration of the context menu into Android applications necessitates a systematic approach encompassing various stages of development. Initially, developers designate the relevant UI components as context menu targets by employing the registerForContextMenu() method, thereby enabling context menu activation upon long-press interactions. Subsequently, through the onCreateContextMenu() method override, developers populate the context menu with a curated selection of actions reflective of the current context. This entails defining menu items, assigning appropriate labels, and associating them with corresponding event handlers to execute desired functionalities. Furthermore, developers harness the onContextItemSelected() method to process user selections from the context menu, orchestrating seamless execution of the chosen action. By adhering to established design patterns and leveraging platform-specific APIs, developers ensure the seamless integration and intuitive operation of context menus within their applications.

* + **Significance in User Experience**

+ The overarching significance of the context menu transcends its utilitarian function, permeating the core tenets of user experience design. Through its judicious deployment, context menus engender a paradigm shift in interaction dynamics, catalyzing heightened efficiency, and fluidity in user workflows. By surfacing contextually relevant actions in close proximity to the user's point of interaction, context menus mitigate cognitive load, obviating the need for exhaustive navigation through hierarchical menus or settings screens. Moreover, context menus foster serendipitous discovery of secondary functionalities, augmenting user engagement and exploration within the application ecosystem. This intrinsic affordance for contextual relevance not only enhances user satisfaction but also imbues applications with a sense of responsiveness and intuitiveness, elevating the overall user experience to unprecedented heights.

+ In summation, the context menu emerges as a cornerstone of Android application design, embodying the ethos of user-centricity and interaction refinement. Through its multifaceted functionality, meticulous implementation, and profound impact on user experience, the context menu epitomizes a convergence of design elegance and functional efficacy. As developers continue to navigate the ever-evolving landscape of mobile application development, the context menu remains a steadfast ally, facilitating the seamless orchestration of user interactions and fostering a symbiotic relationship between users and applications.

* **Long click listener**

+ In Android application development, the long click listener is a crucial event handler that allows developers to detect prolonged touch interactions on specific UI elements. Unlike regular click events triggered by short taps, long click events are activated when the user presses and holds down on a view for an extended period. This introduction aims to elucidate the functionality, implementation methods, and practical applications of the long click listener in Android development, shedding light on its significance in enhancing user interaction and interface responsiveness.

* **Functionality of Long Click Listener**

+ The primary function of the long click listener is to capture extended touch interactions initiated by the user within the application interface. When a user presses and holds down on a supported UI element, such as a button, image, or list item, for a predefined duration, the long click listener is invoked, triggering a corresponding event. This event serves as a signal to the application that the user has engaged in a long-press action, prompting the execution of designated actions or behaviors associated with the long click event. By providing developers with the means to discern between short taps and prolonged presses, the long click listener enables the implementation of context-sensitive functionalities and gesture-based interactions tailored to the user's intent.

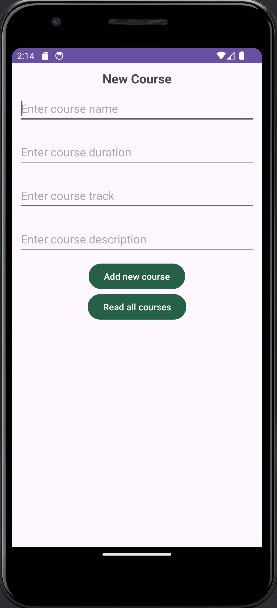
* **Implementation Methods**

+ Implementing the long click listener in Android applications entails a straightforward process facilitated by the Android framework's event handling mechanisms. Developers typically assign the long click listener to target UI elements programmatically or declaratively within the application's layout XML files. Through programmatic assignment, developers register the long click listener on the desired view object using the setOnLongClickListener() method, specifying the corresponding event handler to be invoked upon long click detection. Alternatively, declarative assignment involves defining the long click listener directly within the XML layout file by leveraging attributes such as android:longClickable and android:onClick, thereby streamlining the integration process. By adopting these implementation methods, developers can seamlessly integrate long click functionality into their applications, enriching user interactions with intuitive touch-based gestures.

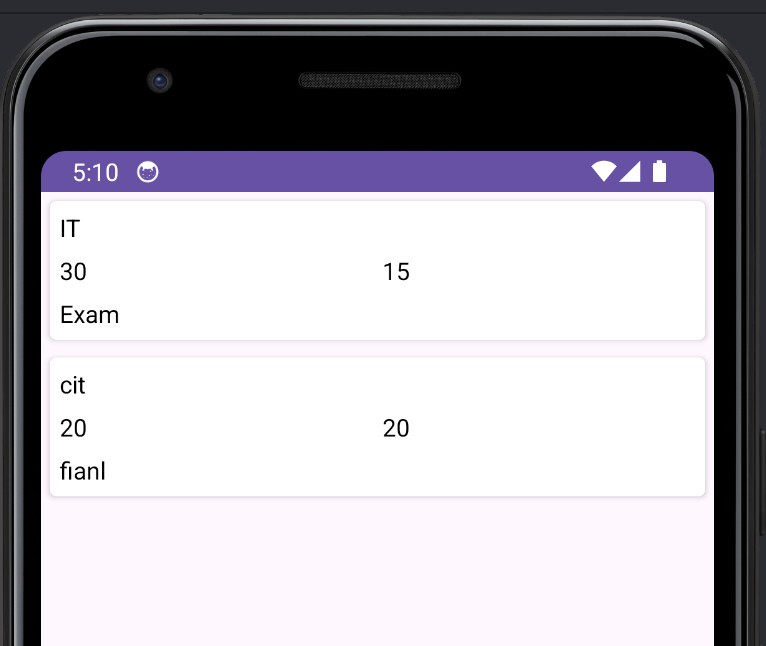
* **Practical Applications**

+ The long click listener finds widespread utility across a myriad of Android applications, serving as a versatile tool for facilitating gesture-based interactions and contextually relevant actions. Common applications of the long click listener include enabling selection modes in list-based interfaces, where long-pressing on list items triggers item selection or multi-selection functionality. Additionally, long click events are instrumental in initiating drag-and-drop operations, allowing users to rearrange items within lists or grids through intuitive touch gestures. Furthermore, the long click listener can be employed to implement secondary context menus, presenting users with a curated set of options tailored to the long-pressed UI element. By harnessing the long click listener, developers unlock a plethora of innovative interaction patterns and user experiences, fostering engagement and usability within their applications.

+ In conclusion, the long click listener stands as a foundational component in Android application development, empowering developers to capture and respond to prolonged touch interactions with precision and efficiency. Through its intuitive functionality, streamlined implementation methods, and diverse practical applications, the long click listener enriches user interactions, elevating the responsiveness and usability of Android applications. As developers continue to innovate and iterate upon user interface design paradigms, the long click listener remains a steadfast ally, enabling the seamless integration of gesture-based interactions and fostering immersive user experiences across a wide spectrum of applications and use cases.



* The results of the entered information:



Ex2: Modify the Lab 6 to store all information to a listview and display them.

**Lab 7: Custom listview**

[**https://www.youtube.com/watch?v=po3EET8uK0g**](https://www.youtube.com/watch?v=po3EET8uK0g)

[**https://www.tutorialspoint.com/how-to-implement-a-long-click-listener-on-a-android-listview**](https://www.tutorialspoint.com/how-to-implement-a-long-click-listener-on-a-android-listview)

[**https://www.tutorialspoint.com/how-to-dynamically-remove-items-from-listview-on-a-click**](https://www.tutorialspoint.com/how-to-dynamically-remove-items-from-listview-on-a-click)

Using the Lab7 app, delete an item in the listview when long pressing the item (long click). You should use context menu to display “Delete”: [**https://www.javatpoint.com/android-context-menu-example**](https://www.javatpoint.com/android-context-menu-example)

**Lab 8: SQLite database**

**Introduction:**

SQLite is an open-source database management system, also known as a compact relational database system, different from other management systems such as MySQL, SQL Server, Ocracle, PostgreSQL... SQLite is a software library that deploys a traditional SQL Database Engine, does not need a client-server model, we don't need to establish any kind of connections. It is an embedded SQL database system, can be used immediately without configuration, without needing a separate SQL Server so it is very compact. Also, SQLite is widely used on many platforms (Mobile, Desktop, Webserver...).

**Refer:**

* <https://www.tutorialspoint.com/android/android_sqlite_database.htm>
* <https://abhiandroid.com/database/sqlite>
* <https://abhiandroid.com/database/add-retrieve-image-sqlite-database-example-android-studio.html>
* <https://xuanthulab.net/su-dung-sqlite-trong-android-phan-1.html>

<https://github.com/xuanthulabnet/android-sqlite-example1>

* <https://www.javatpoint.com/android-sqlite-tutorial>

**1. Introduction:**

SQLite is a versatile, lightweight, and open-source relational database management system (RDBMS) that operates without the need for a separate server process. Unlike client-server database management systems, SQLite is self-contained, meaning it requires minimal setup and administration, making it an ideal choice for embedded systems, mobile applications, and small-scale deployments.

One of the key features of SQLite is its simplicity and ease of use. It's designed to be embedded directly into applications, allowing developers to incorporate database functionality seamlessly into their software without relying on external dependencies. This integration is particularly advantageous for mobile platforms like Android, where efficient resource utilization and offline data storage are essential.



In the context of Android development, SQLite serves as the default database engine, providing a reliable and efficient means of storing and managing application data locally on the device. This built-in support eliminates the need for developers to integrate third-party database solutions, simplifying the development process and reducing potential compatibility issues.

SQLite databases in Android are stored in the application's private storage directory,typically located at data/data/APP\_Name/databases/DATABASE\_NAME. This ensures that each application has its own isolated storage space, enhancing data security and preventing unauthorized access. Furthermore, the data within SQLite databases is stored in text format, making it easily readable and accessible for developers.

Despite its lightweight nature, SQLite offers robust relational database capabilities, including support for tables, indexes, triggers, and transactions. Developers can interact with SQLite databases using standard SQL queries, making it familiar to those accustomed to working with relational databases. This SQL-based interface facilitates seamless data manipulation, retrieval, and management within Android applications.

Moreover, SQLite databases are highly portable and can be easily transferred or shared between different devices or platforms. This flexibility allows developers to build cross-platform applications that leverage the same underlying database infrastructure across various environments.

In summary, SQLite is a powerful, lightweight, and user-friendly database solution that is well-suited for mobile and embedded applications, particularly on the Android platform. Its seamless integration, robust features, and efficient resource utilization make it a popular choice among developers for local data storage and management.

SQLite is a Structure query base database, open source, light weight, no network access and standalone database. It support embedded relational database features.

**Refer:**

* <https://www.tutorialspoint.com/android/android_sqlite_database.htm>
* <https://abhiandroid.com/database/sqlite>
* <https://abhiandroid.com/database/add-retrieve-image-sqlite-database-example-android-studio.html>
* <https://xuanthulab.net/su-dung-sqlite-trong-android-phan-1.html>

<https://github.com/xuanthulabnet/android-sqlite-example1>

* <https://www.javatpoint.com/android-sqlite-tutorial>

* Create and Add Data to SQLite Database:

In Android development, managing data with SQLite databases is a fundamental task. To create and populate a SQLite database, developers typically start by defining a database helper class that extends the SQLiteOpenHelper class. Within this class, the database schema, including table structure and column definitions, is established. When adding data, developers utilize the SQLiteDatabase class to obtain a writable database instance. Data insertion is performed using ContentValues, which contain key-value pairs representing column names and corresponding values. The insert() method of the SQLiteDatabase class is then invoked to add the data into the specified table. Once the data insertion is completed, it's essential to close the database connection to ensure proper resource management and data integrity within the Android application

<https://www.geeksforgeeks.org/how-to-create-and-add-data-to-sqlite-database-in-android/>

* Read Data from SQLite Database

In Android development, retrieving data from SQLite databases is a common operation. To read data, developers typically start by obtaining a readable instance of the SQLiteDatabase class. This is usually done through a database helper class that extends SQLiteOpenHelper. Once the database is opened in read mode, developers execute SQL queries using the query() method, which returns a Cursor object containing the result set. The Cursor allows developers to iterate over the rows of the result set and extract the desired data using methods such as getString(), getInt(), etc. Finally, it's important to close the Cursor and the database connection to release system resources and ensure proper memory management

<https://www.geeksforgeeks.org/how-to-read-data-from-sqlite-database-in-android/?ref=rp>

* Delete Data in SQLite Database in Android

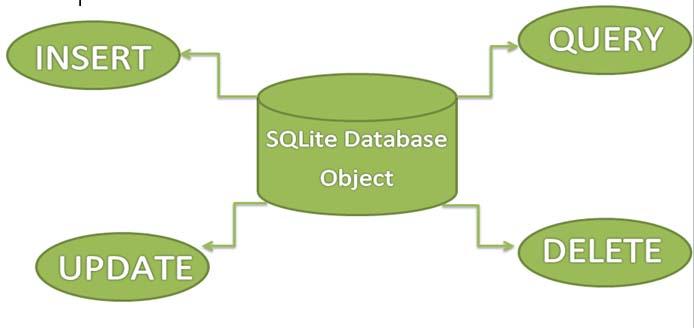
[**https://www.geeksforgeeks.org/how-to-delete-data-in-sqlite-database-in-android/?ref=rp**](https://www.geeksforgeeks.org/how-to-delete-data-in-sqlite-database-in-android/?ref=rp)

     In Android development, managing data in SQLite databases is a common task. When it comes to deleting data, developers utilize the SQLiteDatabase class provided by the Android framework. By opening a connection to the database and constructing a delete query, specific records can be removed from a table based on certain conditions. Once the deletion operation is executed, developers typically close the database connection to ensure proper resource management. This process is fundamental for maintaining the integrity and efficiency of data storage within Android applications

* Update Data to SQLite Database in Android

[**https://www.geeksforgeeks.org/how-to-update-data-to-sqlite-database-in-android/?ref=rp**](https://www.geeksforgeeks.org/how-to-update-data-to-sqlite-database-in-android/?ref=rp)

In Android development, updating data in SQLite databases is a routine task. To perform an update operation, developers typically begin by acquiring a writable instance of the SQLiteDatabase class. This is commonly done within a database helper class that extends SQLiteOpenHelper. Once the database is opened in write mode, developers construct an SQL UPDATE statement specifying the table to update, the columns to modify, and any conditions for the update. This query is then executed using the update() method of the SQLiteDatabase class, which applies the modifications to the database. It's important to ensure proper error handling and transaction management during the update process to maintain data integrity within the Android application



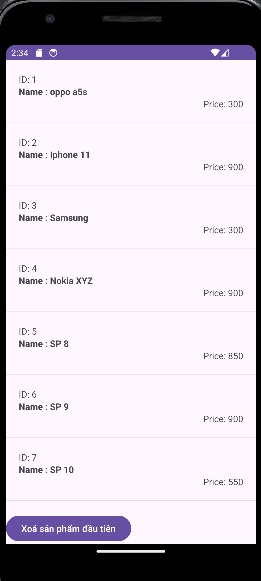
Reproduce the app in the following links:

* Create and Add Data to SQLite Database:

<https://www.geeksforgeeks.org/how-to-create-and-add-data-to-sqlite-database-in-android/>

* Read Data from SQLite Database

<https://www.geeksforgeeks.org/how-to-read-data-from-sqlite-database-in-android/?ref=rp>

****

**Lab 9: SQLite database (2)**

* Delete Data in SQLite Database in Android

[**https://www.geeksforgeeks.org/how-to-delete-data-in-sqlite-database-in-android/?ref=rp**](https://www.geeksforgeeks.org/how-to-delete-data-in-sqlite-database-in-android/?ref=rp)

* Update Data to SQLite Database in Android

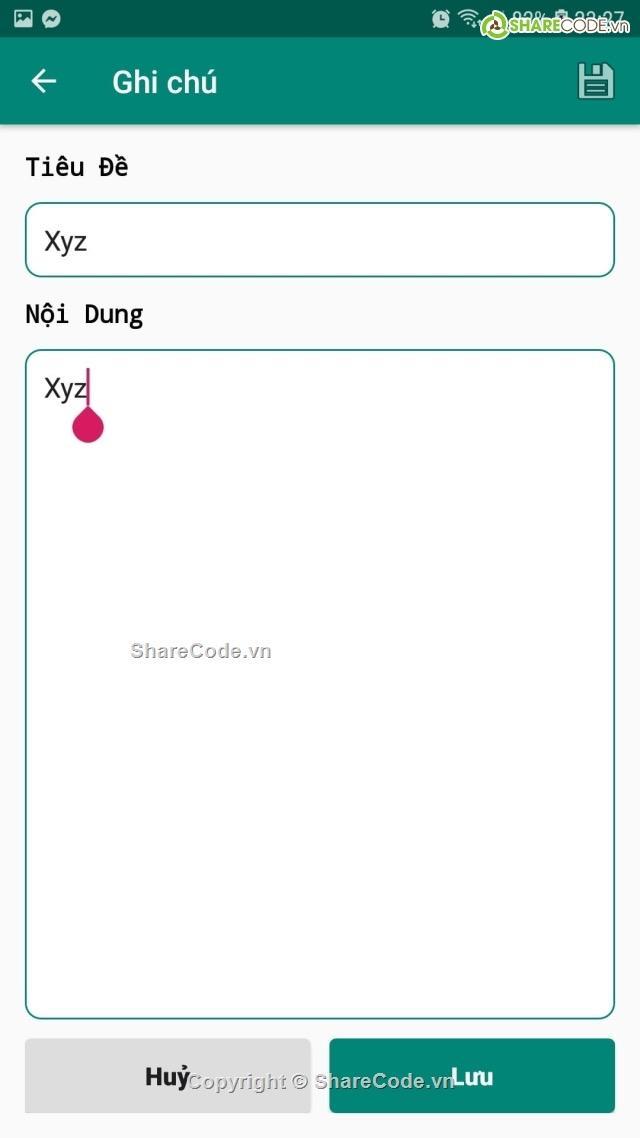
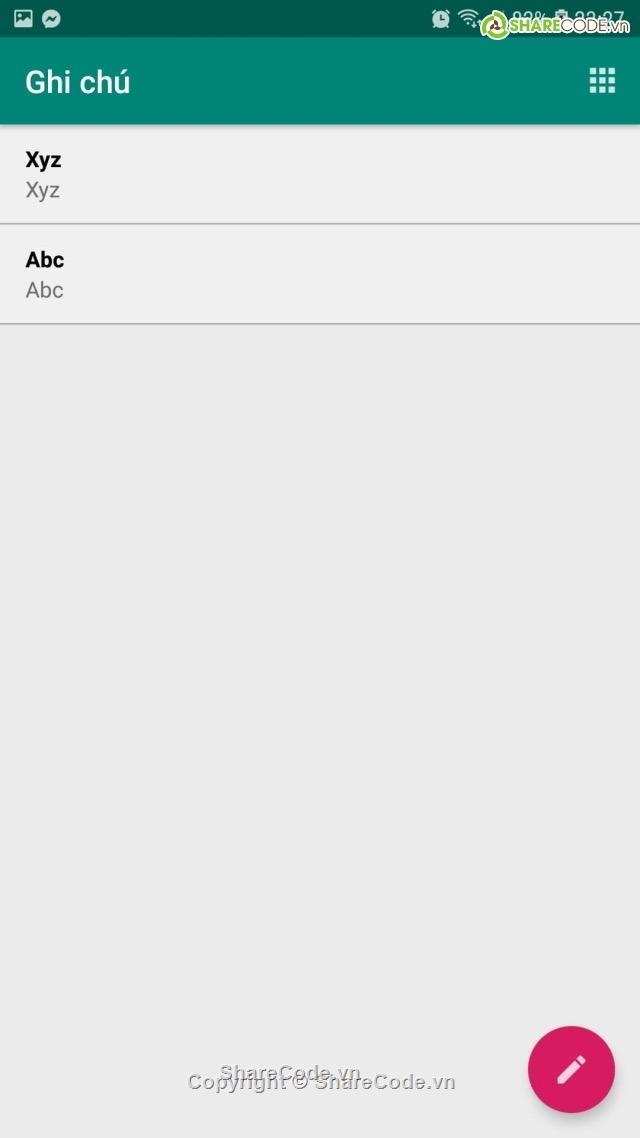
[**https://www.geeksforgeeks.org/how-to-update-data-to-sqlite-database-in-android/?ref=rp**](https://www.geeksforgeeks.org/how-to-update-data-to-sqlite-database-in-android/?ref=rp)

**Lab 10: Progress test 2**

Thiết kế phần mềm quản lý ghi chú tương tự như hình, lưu vào csdl sqlite. Cho phép thêm, sửa, xóa ghi chú.

* Tiêu đề
* Nội dung
* Thời gian (ngày, giờ)

Khi ấn lâu vào mục ghi chú nào, menu ngữ cảnh (xem lại Lab 8) hiện ra cho phép chọn Xóa hoặc Sửa 1 ghi chú.

**.   **

**1.  Create application with following requirements**

For this progress test, I will create an application to manage Notes. Users can add new note with title, content of note and time. For choosing time, user can click on calendar button and the DatetimeDialog will display.

- Create database to store Note information

A screenshot of a computer program

Description automatically generated

- After this creation UI to add new note in main activity

- Create model object to store information

A screen shot of a computer program

Description automatically generated

- Create an adapter to binding information to list view A screen shot of a computer program

Description automatically generated

- Create a context menu items

A screenshot of a computer

Description automatically generated

- Create methods in DBHandler to manipulate store information

- Create new resource activity for reviewing the Note information and a listview to show all note item

A screenshot of a computer

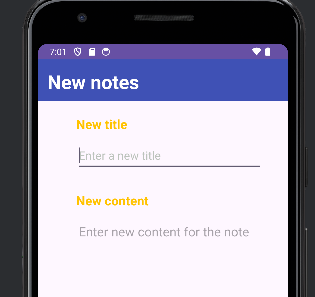
Description automatically generated

A screenshot of a phone

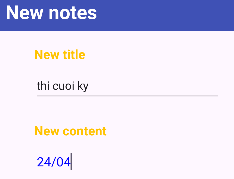
Description automatically generated

- A main page with this interface and a button to show all saved notes

- Calendar button help us choosing date and time is current system time.



* Enter the information:



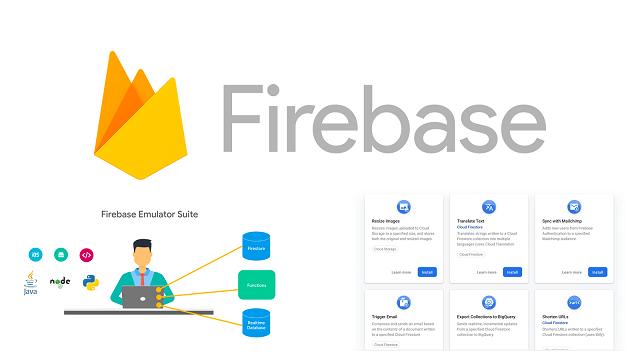
**Lab 11: Google Firebase database**

**1.  Introduction about Firebase**

Firebase, a flagship platform from Google, represents a paradigm shift in app development, offering an integrated ecosystem of tools and services that revolutionize the way developers build, manage, and scale their applications. At its core, Firebase is designed to empower developers by providing a comprehensive suite of backend solutions that handle crucial aspects of app development, ranging from user authentication to real-time data synchronization, cloud storage, hosting, and analytics.

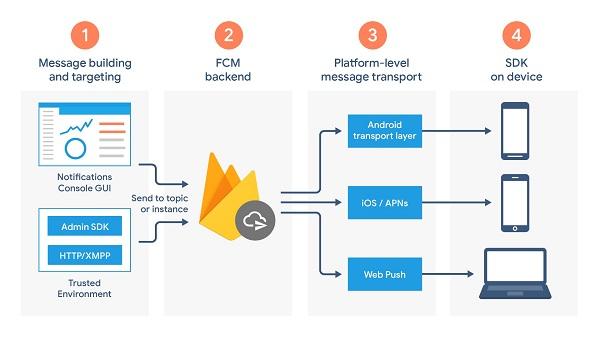
One of the hallmark features of Firebase is its seamless integration with both web and mobile applications. Whether you're developing a native Android or iOS app, a web application, or even a backend service, Firebase offers a unified platform that simplifies the development process. Developers can leverage Firebase's APIs and SDKs to incorporate powerful functionality into their applications with minimal effort, eliminating the need for complex infrastructure setup and maintenance.

Authentication is a fundamental component of any modern application, and Firebase excels in this area by providing robust authentication services out of the box. With Firebase Authentication, developers can easily implement user sign-in, sign-up, and password management features, as well as support for popular identity providers such as Google, Facebook, and Apple.



Real-time data synchronization is another key strength of Firebase, enabled by its Realtime Database and Firestore services. These NoSQL databases offer seamless data synchronization across clients in real time, allowing for collaborative and interactive experiences in applications. Whether it's a chat app, a collaborative document editor, or a multiplayer game, Firebase's real-time capabilities enable developers to build dynamic and responsive experiences for their users.

Firebase also offers scalable cloud storage solutions for storing and serving user-generated content, such as images, videos, and files. With Firebase Storage, developers can securely upload, download, and manage files in the cloud, with built-in security features and support for fine-grained access control.



Hosting is another integral part of the Firebase platform, allowing developers to deploy and serve web applications and static content with ease. Firebase Hosting provides a reliable and scalable hosting infrastructure, with features such as SSL encryption, custom domain support, and automatic content delivery network (CDN) integration, ensuring fast and reliable performance for users worldwide.

In addition to its core services, Firebase offers powerful analytics and performance monitoring tools that provide insights into user behavior, app performance, and engagement metrics. With Firebase Analytics, developers can track user interactions, measure app usage, and gain valuable insights to optimize their applications for success.

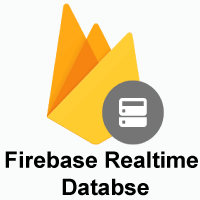
Furthermore, Firebase's extensive documentation, community support, and integration with other Google services make it a preferred choice for developers looking to accelerate their app development processes. Whether you're a seasoned developer or just starting out, Firebase provides the tools and resources you need to build high-quality, feature-rich applications that delight users and drive business growth.

Firebase is a comprehensive platform offered by Google that empowers developers to build, manage, and grow their applications more efficiently. At its core, Firebase provides a suite of tools and services designed to streamline various aspects of app development, including authentication, real-time database management, cloud storage, hosting, and analytics. One of Firebase's key strengths lies in its ease of integration with both web and mobile applications, offering developers a unified platform to handle backend tasks without the need for extensive infrastructure setup. With its robust set of features and scalable infrastructure, Firebase enables developers to focus more on crafting exceptional user experiences and less on managing the underlying technical complexities. Additionally, Firebase offers extensive documentation, community support, and integration with other Google services, making it a preferred choice for developers looking to accelerate their app development processes.

**Authentication:** Firebase Authentication simplifies user authentication in Android apps by providing easy-to-use SDKs for email/password authentication, phone number authentication, social media login (Google, Facebook, Twitter), and more. With Firebase Authentication, developers can quickly integrate secure user authentication flows into their Android apps without worrying about backend infrastructure.



**Realtime Database and Firestore**: Firebase Realtime Database and Firestore are NoSQL databases that enable developers to store and sync app data in real time. These databases are particularly well-suited for Android apps that require offline support and real-time data synchronization across devices. Firebase's offline capabilities ensure that users can access and modify data even when they're offline, with changes automatically synchronized when they regain connectivity.



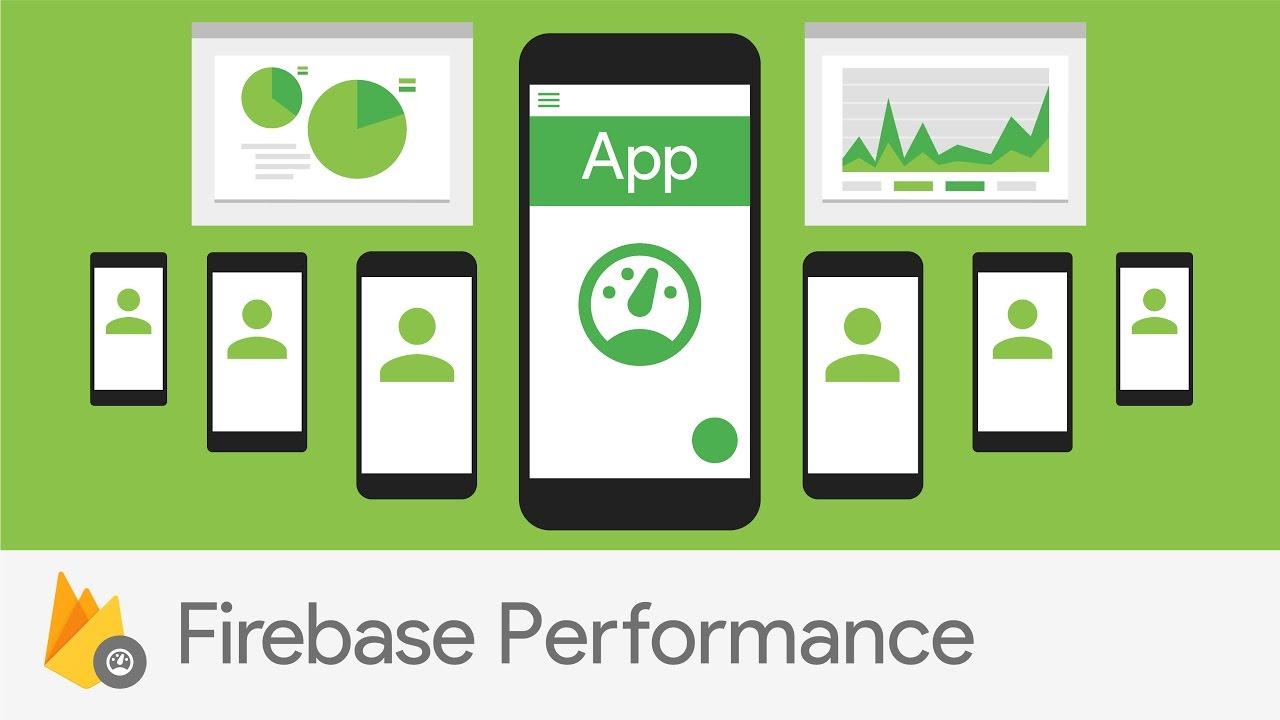
**Cloud Storage**: Firebase Storage offers scalable cloud storage for storing and serving user-generated content such as images, videos, and files. Android developers can easily integrate Firebase Storage into their apps to upload, download, and manage files, with built-in security features and support for direct access URLs.



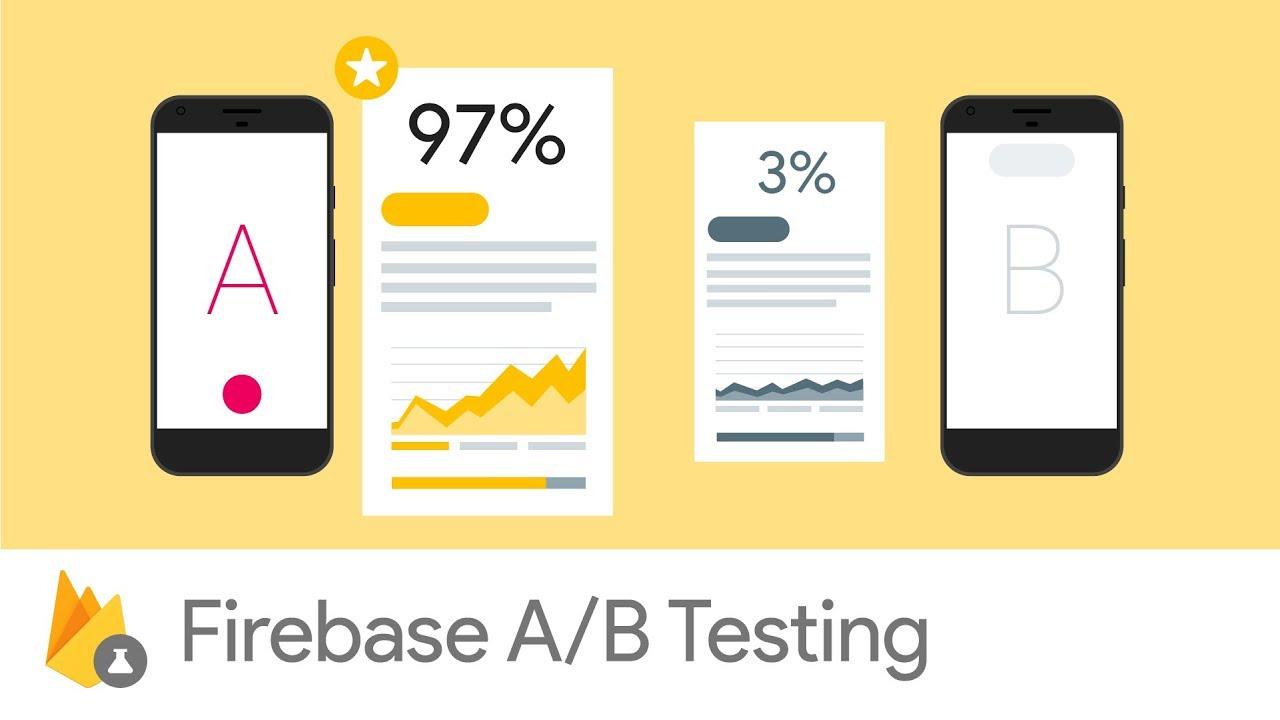
**Cloud Messaging (FCM)**: Firebase Cloud Messaging (FCM) is a powerful messaging solution that enables developers to send notifications and messages to Android devices, iOS devices, and web browsers. FCM allows for targeted messaging, topic-based messaging, and even integration with other Firebase services such as Analytics and Remote Config.



**Crashlytics and Performance Monitoring**: Firebase Crashlytics provides real-time crash reporting and analysis for Android apps, helping developers identify and fix issues quickly to improve app stability and user experience. Firebase Performance Monitoring offers insights into app performance metrics such as app startup time, network latency, and UI rendering, allowing developers to optimize app performance and deliver a smooth user experience.



**Remote Config and A/B Testing:** Firebase Remote Config enables developers to remotely configure app parameters and settings without requiring app updates. This allows for dynamic content changes, feature toggles, and targeted experiments to optimize app performance and user engagement. Firebase A/B Testing complements Remote Config by providing tools for running experiments, measuring user behavior, and optimizing app experiences based on real-world data.



**Machine Learning Kit (ML Kit):** Firebase ML Kit simplifies the integration of machine learning models into Android apps, offering ready-to-use APIs for common tasks such as text recognition, face detection, barcode scanning, and image labeling. ML Kit's on-device and cloud-based APIs enable developers to add powerful machine learning features to their apps with minimal effort.

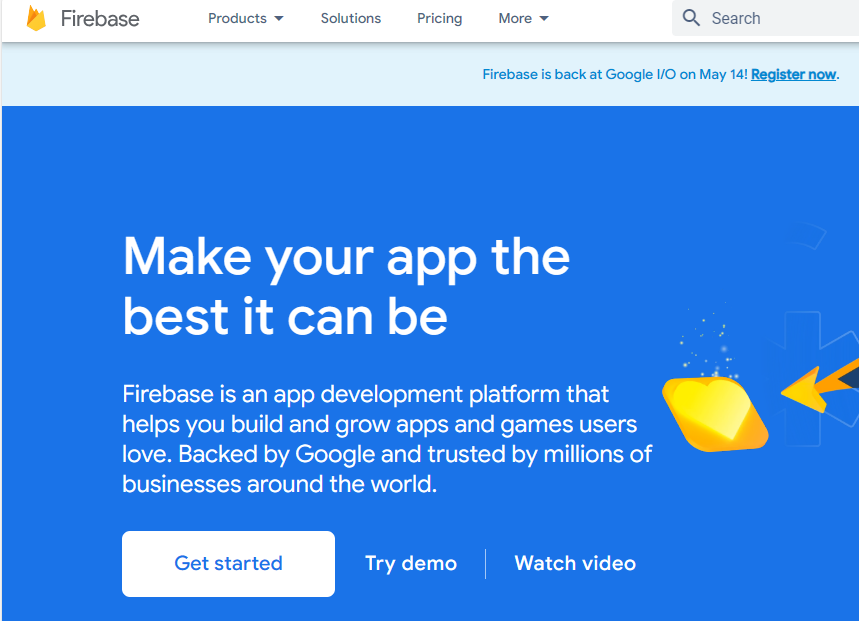


**App Distribution:** Firebase App Distribution simplifies beta testing and app distribution for Android apps, allowing developers to distribute pre-release versions of their apps to testers and stakeholders with ease. Developers can manage distribution lists, collect feedback, and monitor app installs through the Firebase console.

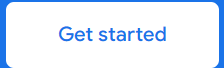


**2.  Add Firebase to project**

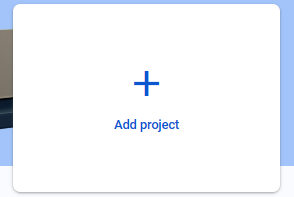
**Create a Firebase Project**:Go to the Firebase Console (https://console.firebase.google.com/) and sign in with your Google account. Click on "Add project" and follow the prompts to create a new Firebase project. Once the project is created, you'll be redirected to the project dashboard.



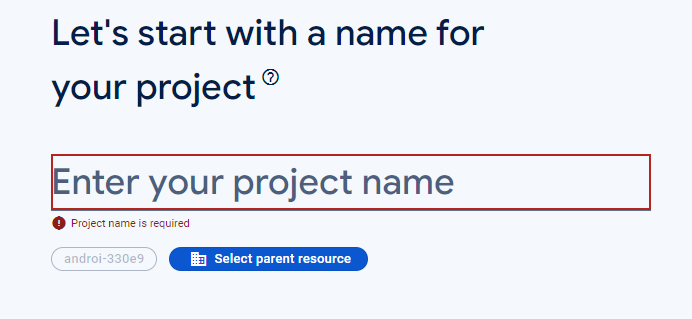
* Click "Get Start" to get started:



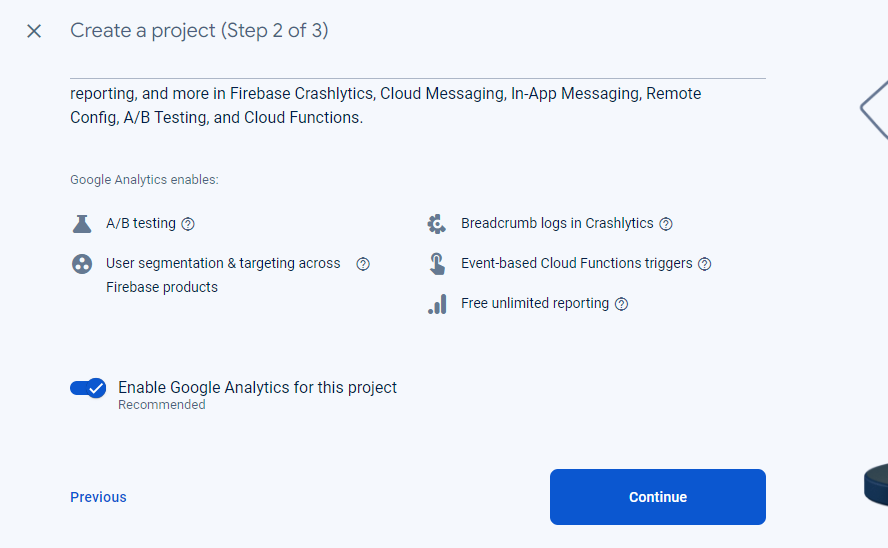
* Then, select "Add Project" to continue:



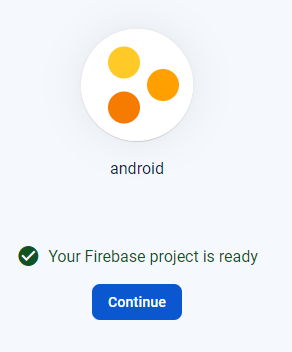
* Enter the name of the project:



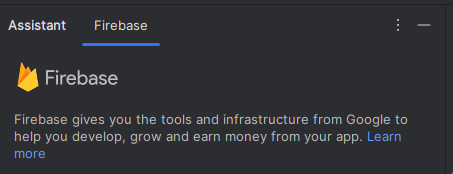
* Select continue:



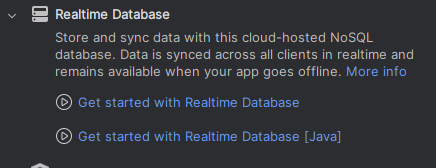
* Complete project creation

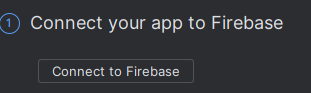


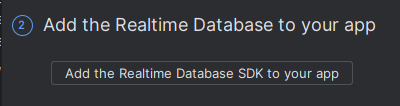
* Then go back to "Android Studio" and use the selection Tool – Firebase



* Select "Realtime Database" - Get started with Realtime Database (Java):



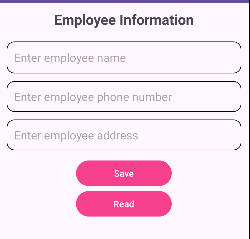




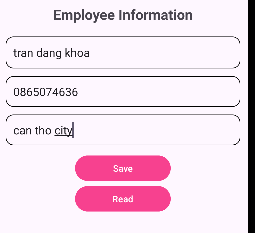
**Lab 12: Progress Exercise: Implementation, Demo & presentation**

Building simple apps such as English vocabulary management, Electric/Water Management, Car/Motorbike maintenance management, Cafeshop management,… (using SQLite or Firebase)

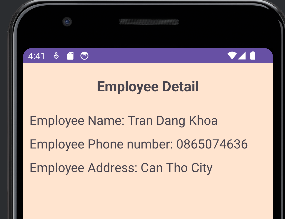
When starting the application, the interface will appear:



Enter the information:



After that, you can review the entered information:



**Lab 13: Progress Exercise: Implementation, Demo & presentation**

Building simple apps such as English vocabulary management, Electric/Water Management, Car/Motorbike maintenance management, Cafeshop management,… (using SQLite or Firebase)

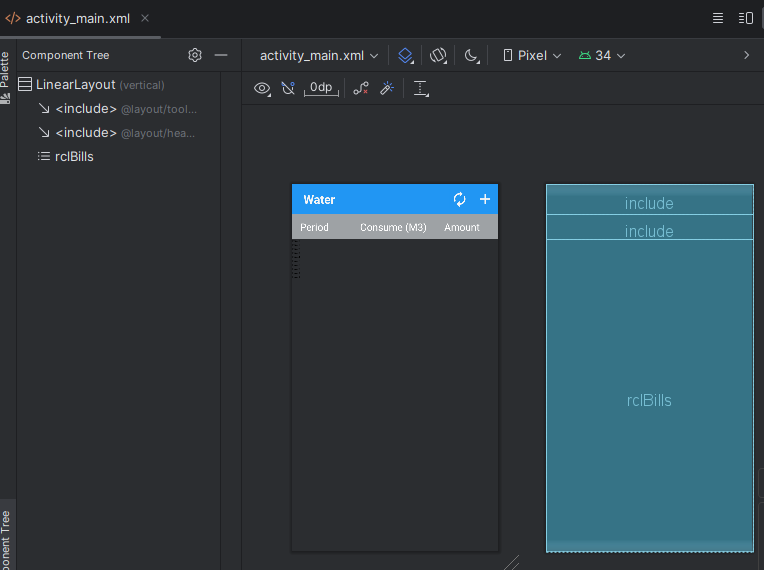
* Mô tả bài toán
* thiết kế CSDL
* thiết kế chương trình, giao diện demo

**Introduction app:**

Water bill management application helps managers easily manage information about water volume and the amount of money that must be paid that month.

**Design the theme:**

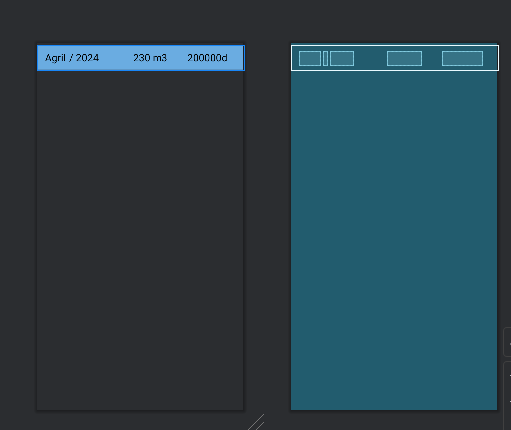
Activity\_main:

****

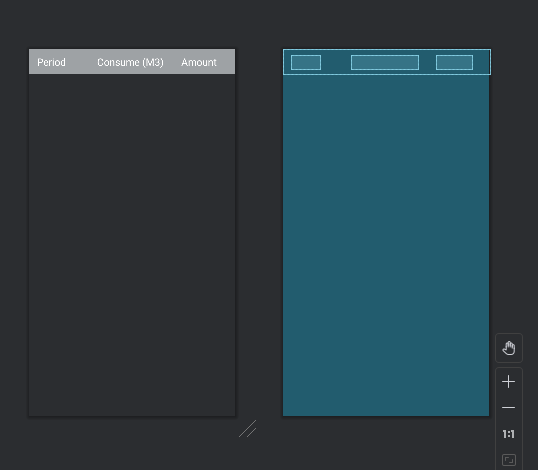
* activity\_add\_bill:

****

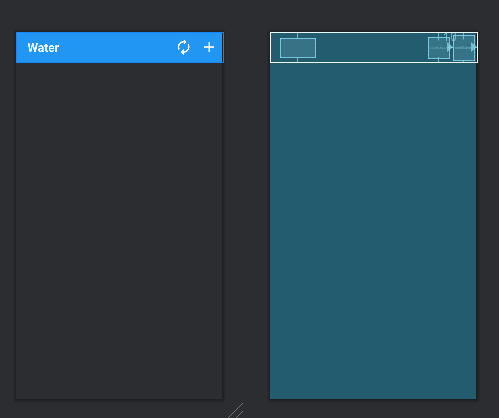
Activity\_bill\_item:



* header\_item:

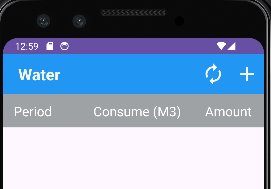


Toolbar\_list:



**Code results:**

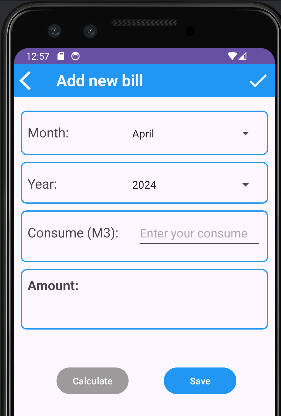
* After starting the app, the following interface will appear:



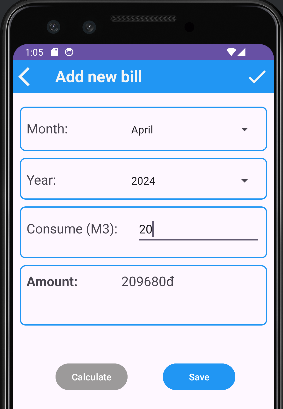
* Can click the button below to add a water bill:



* After clicking on the add bill button, there will be an interface page as follows to enter information:



Enter information about electricity and water into the form such as month and year of water consumed and save then the total amount to be paid:



* Once you have entered the information and saved it, you can click on the button below to view the bills entered above:



* Results of the entered bills:

