DroidInspect : Auto UI Inspector

Software Project Lab 2

Submitted by
Md Nasir Uddin
Roll : BSSE 1310
&
Soumitra Paul
Roll : BSSE 1317

Supervised by
Dr. Mohammad Shoyaib
Professor
Institute of Information Technology
University of Dhaka

Submission Date: 27 March 2024

IIT
University of Dhaka

1.DroidInspect: Auto UI Inspector

DroidInspect is an innovative automated testing tool specifically tailored for Android applications, aimed at overcoming the limitations of manual testing by offering a streamlined, error-resistant, and comprehensive approach. optimizes testing resources, ensures device compatibility, and seamlessly integrates into the development workflow. DroidInspect distinguishes itself from existing tools by its contextual understanding of applications, enhancing efficiency, and utilizing intelligent agents for testing. The motivation behind DroidInspect is to revolutionize Android app testing, saving time, reducing costs, and improving overall quality assurance processes. The database architecture of DroidInspect includes three core tables: App Information, Device Information, and DroidInspect Result. These tables store essential details about applications, devices used for testing, and the outcomes of testing endeavors, respectively. DroidInspect's architecture and testing plans are meticulously designed to ensure effectiveness, including comprehensive requirement specifications, a modular and scalable architecture design, and thorough testing plans covering various phases and scenarios. Overall, DroidInspect aims to set a new standard in Android app testing, offering unparalleled efficiency, reliability, and cost-effectiveness.

Testing plays a crucial role in the Software Development Life Cycle (SDLC), yet it is often neglected due to its tedious nature and resource-intensive requirements. The complexity of the testing process often leads to underestimation by management. Recognizing this challenge, the development of a tool to simplify UI testing becomes imperative. In response, the DroidInspect project aims to address this issue by automating the UI testing process through the generation and execution of test cases. DroidInspect focuses on streamlining blackbox testing for the app's UI, simulating user interactions to comprehensively assess its functionality. This tool is positioned to alleviate the challenges associated with manual testing and enhance the efficiency of the testing phase in software development.

1.1 Purpose

The purposes of this document are:

- Identify and analyze the requirements
- Design the test plan
- Reduce the development effort
- Improve understanding
- Store Information in Database.

1.2 Scope

The scope of this project is defined below:

- The apk file for the android app to be tested will be provided.

● The tool will not require any source code of the app to be tested.

1.3 **Assumptions**

The assumptions of the project are:

● Android device connected to the computer and accessible via USB
● All permissions are granted
● Valid inputs are given with correct label
● Disabled notifications, auto-rotation, flight mode, and enabled mobile data and Wi-Fi
● Biometric and face-recognition inputs will not be considered

2. Motivation

Manual testing in this landscape is time-consuming and error-prone, with apps traversing complex UI paths and exhibiting device-specific bugs. We aim to transform this process by leveraging automation.
Our goal is to streamline testing, enhance efficiency, and boost app quality. Human testers face limitations like slow test case generation and biases, hindering comprehensive coverage. Inadequate testing leads to financial losses, compliance risks, and user dissatisfaction.

In an industry where quality assurance costs 20% - 30% of budgets[2], DroidInspect aims for cost efficiency. Defects in production increase overheads

and reduce ROI[3]. We're here to make Android app testing more efficient, cost-effective, and reliable.
DroidInspect aspires to revolutionize Android app testing, saving time, costs, and enhancing quality. With our automated tool, we envision a future where Android app development is smoother and more dependable.

3. Inception

3.1 Recognizing the Problem:

The team, composed of developers and testers, experiences firsthand the challenges associated with manual testing of Android apps. They find themselves spending significant amounts of time crafting test cases, executing them on various devices, and scrutinizing UI elements for bugs. Despite their efforts, they often encounter oversights or miss critical issues due to the sheer complexity of modern app interfaces.

Delays in testing not only impact project timelines but also pose risks to the overall quality of the apps. The team realizes that there's a pressing need for a more efficient and reliable testing solution that can alleviate these pain points.

3.2 Dreaming Up a Solution:

Motivated by the frustrations of manual testing, the team begins brainstorming ideas for a solution. They envision a specialized tool that can automate the testing process for Android apps, from generating test cases to executing them across different devices and configurations.

This tool would not only save time but also improve the thoroughness of testing by systematically exploring various UI paths and interactions. By automating repetitive tasks, testers could focus their efforts on more creative and high-value aspects of testing, such as edge cases and user experience analysis.

3.3 Planning How It'll Work:

With the concept in mind, the team dives into detailed planning. They outline the specific features and functionalities that the tool should possess, such as:

- Automated test case generation based on app UI elements.
- Reporting mechanisms to track test results.
- Integration with existing testing frameworks or development environments.

Additionally, they define what the tool shouldn't do, such as requiring extensive setup or modifications to the app's source code. Clear goals are set, including reducing testing time, improving bug detection rates, and enhancing overall testing efficiency.

3.4 Starting the Project:

With the team in place and the plan finalized, the project officially kicks off. Development begins in earnest, with regular meetings, progress updates, and collaboration sessions. The team is energized by the prospect of realizing their vision and making a tangible impact on Android app testing practices.

Throughout the project, they remain committed to the core principles of simplicity, effectiveness, and user-centric design. As milestones are achieved and the tool takes shape, anticipation builds for its eventual release and the positive impact it will have on the Android app development community.

4. Elicition

This portion presents the quality function deployment and usage scenario of the tool.

4.1 Quality Function Deployment

Quality Function Deployment (QFD) is a technique that translates the needs of the customer into technical requirements for the software. With respect to this project, the following requirements are identified-

Normal Requirements:

- A Software
- Will take app's apk file and necessary credentials as input
- Getting UI information through the droidInspect tool.
- Performing UI interaction through droidInspect.
- Will write the generated test cases in a human readable way.
- Will generate inputs in input fields.

Expected Requirements:

- Having the option to give the android application APK as input

Exciting Requirements:

- Giving custom user defined inputs to specific UI elements
- Getting inputs from intelligent agents
- Handling infinite cycles

4.2 Usage Scenario:

Scenario: DroidInspect-Automated Android Inspector.

Actors:

1. **User** - The individual using the tool to explore Android applications.
2. **Android System** - The target Android device connected to the desktop.

Preconditions:

- The Automated Android Inspector is properly installed and configured on the desktop.
- The Android device is connected to the desktop via USB or WIFI connection.
- Android Debug Bridge[4] can detect the android device.
- The Android application to be explored is either installed on the device or an APK file is available for installation.
- The user has the necessary permissions and access to use the tool and the Android device.

Main Scenario:

1. User Launches the Tool
   - The user opens the DroidInspact on their desktop.

2. Connects to Android Device
   - The user establishes a connection between the tool and the Android device, ensuring proper communication.

3. Initiates Exploration
   - The user decides whether to:
   - Select an already installed Android application for exploration.
   - Provide an APK file for exploration.

4. Selects Installed App for Exploration
   - If the user chooses to explore an already installed app:

- The user selects the desired Android application from a list of installed apps on the connected device.

5. Provides APK for Exploration (if required)
   - If the user chooses to explore an app via an APK:
   - The tool prompts the user to provide the APK file for installation.

6. APK Installation (if required)
   - The user provides the APK file.
   - The tool installs the APK on the Android device.

7. UI XML Parsing
   - The tool extracts the UI XML structure of the application.

8. Identifying Actionable Elements
   - The tool analyzes the UI XML to identify actionable elements such as buttons, text fields, and menus.

9. Generating Graph
   - The tool creates a graph structure for navigation, with nodes representing different UI states.

10. Exploring UI States
    - The tool starts the exploration process following an exploration strategy, navigating through the UI states based on the generated graph.
    - For each UI state, the tool:
    - Identifies actionable elements.
    - Sends appropriate inputs (e.g., clicks, text input) to interact with the UI.
    - Records the responses and outcomes.

11. Continuous Exploration

- The tool continues to explore the application, branching through different UI states and interacting with actionable elements.

12. Stopping if there is a cycle
    - The tool will halt if there it goes around in cycles along a path
13. Store Information in Database.
    - This tool will store App information , Device Information and Droidbot Result.

Postconditions

- The tool provides the user with a comprehensive exploration report, including recorded interactions and outcomes.
- The exploration session can be saved or terminated as per the user's preference.

Exceptions

- If the Android device disconnects during exploration, the tool will attempt to re-establish the connection or notify the user of the issue.
- If the APK installation fails, the tool will inform the user and request an alternative APK file or resolution.

This revised usage scenario allows the user to choose whether to explore an already installed app or provide an APK for exploration, accommodating both scenarios.

6. Scenario Based Modeling

6.1 Use Case Diagram

A use case diagram is a tool for summarizing information about a system and the users within it. It is typically displayed as a graphic representation of how various system components interact with one another. Use case diagrams will detail the system's events and the order in which they occur, but they do not go into detail on how those events are carried out.

Actors:

Actors are the users who communicate with a system. They may be a person, group, or external system that communicates with your system or application. They must be external data-producing or data-consuming objects. In the use case diagram, stick figures denote the actors employing the use cases. Actors are also divided into two parts:

● **Primary Actor**

Primary actors collaborate to accomplish necessary system functions and produce the system's desired requirements. They often and directly collaborate with the software.

● **Secondary Actor**

A secondary actor is a person, business procedure, or application that gives a use case a certain outcome or information in order to accomplish the use case's ultimate objective. Secondary actors support the whole system for fluent execution of primary actors. And they accomplish this task by producing or consuming information.

Name : DroidInspect: AutoUI Inspector

Level 0:

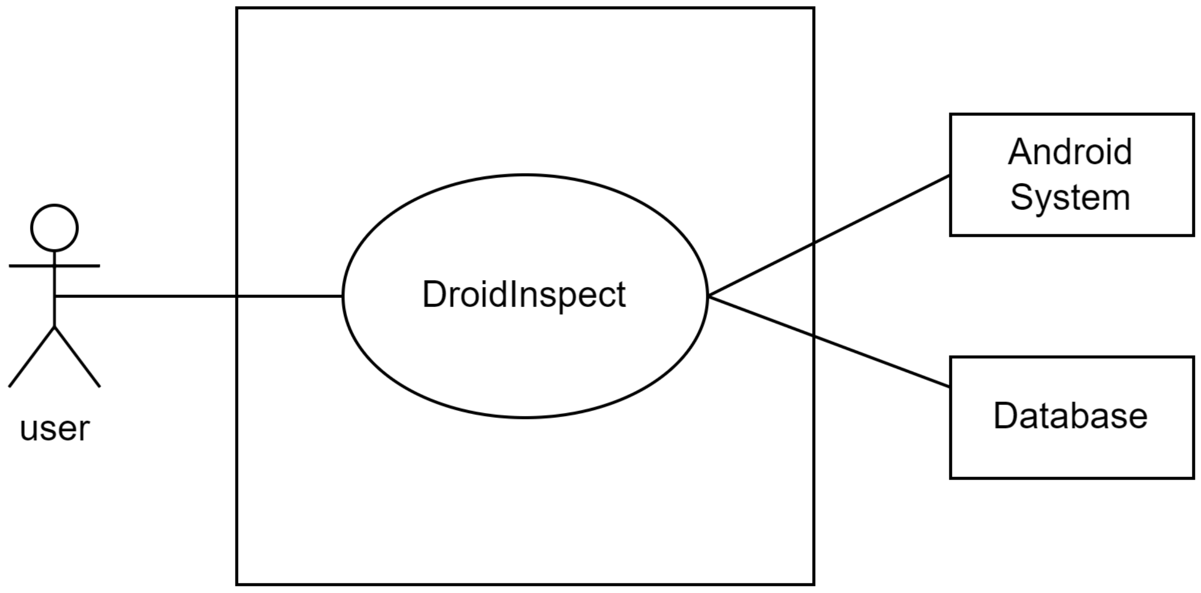
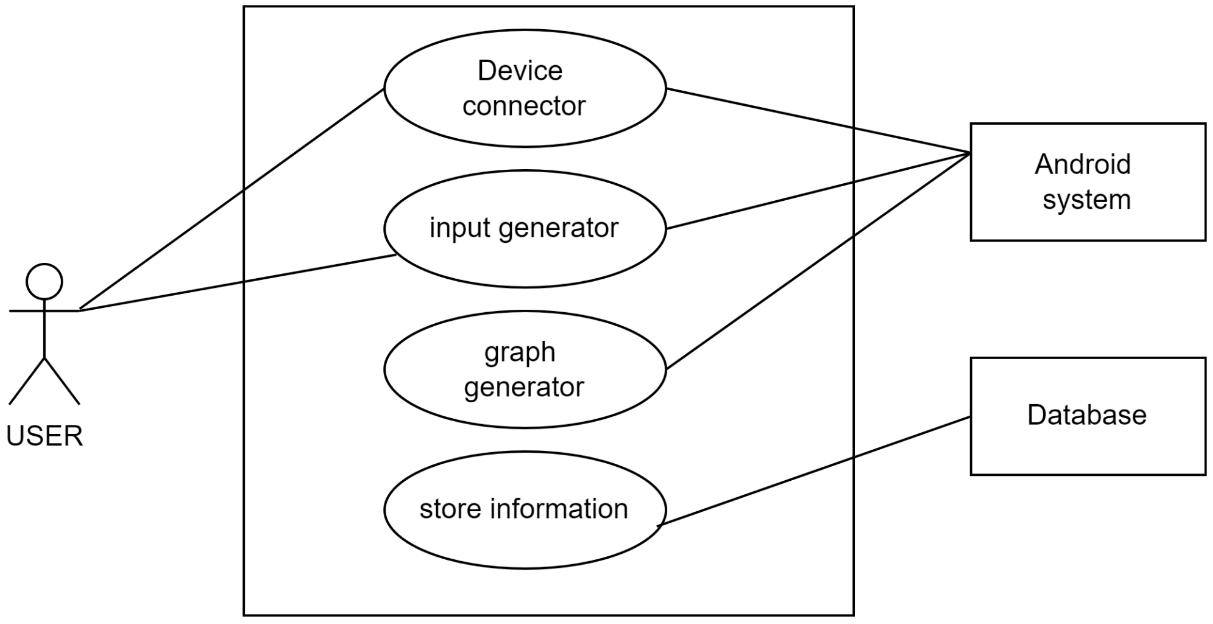


Figure 02 : Level 0 DroidInspect: AutoUI Inspector

Primary actors: User
Secondary actors: Android system,Database
Goal in context: Top level view of DroidInspect.

Name: Modules of DroidInspect.

Level 1:

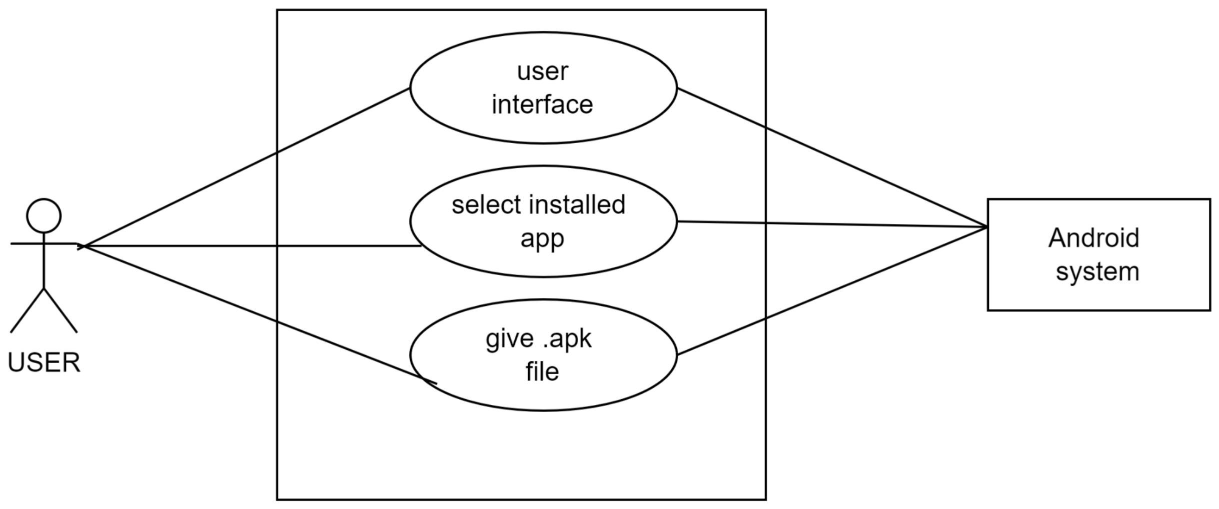


Level 1 Modules of DroidInspect.

Primary actors: User
Secondary actors: Android system, Database.
Goal in context: Module-wise view of DroidInspect.

Name: Device Connector

Level 1.1:

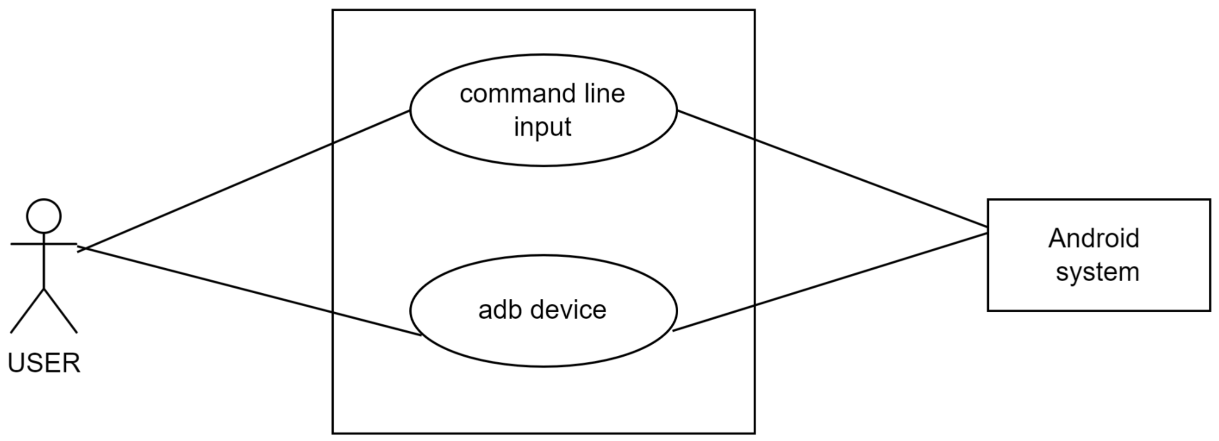


Level 1.1 Device Connector

Primary actors: User
Secondary actors: Android system.

Name: Input Generator

Level 1.2:

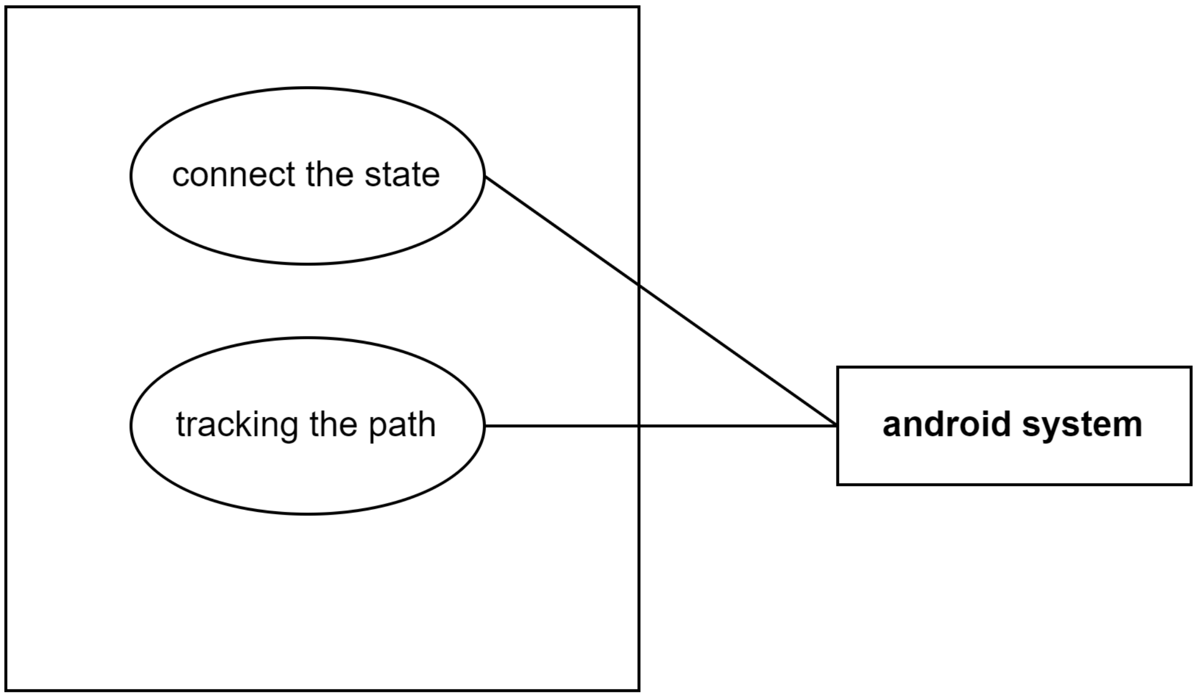


Level 1.2 Input generator

Primary actors: User
Secondary actors: Android system.

Name : Graph Generator

Level 1.3:

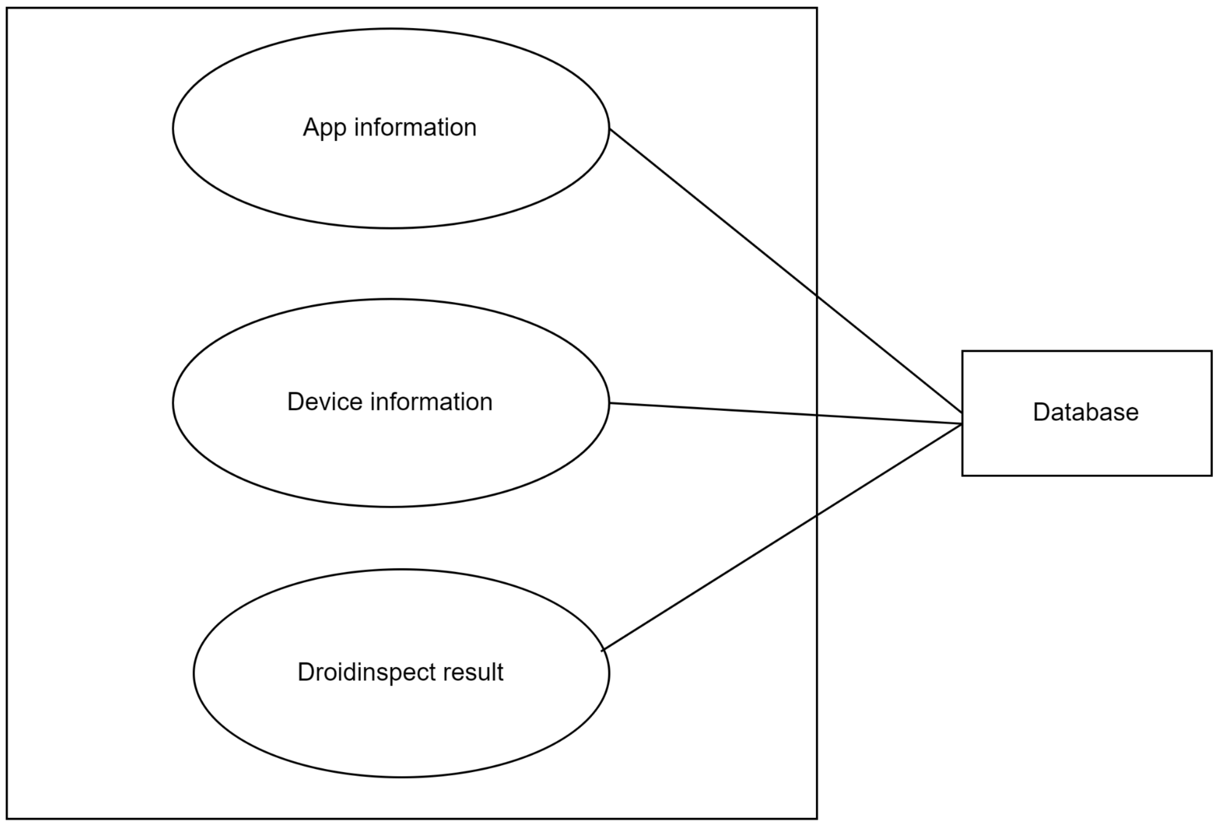


Level 1.1 Graph Generator

Primary actors: None
Secondary actors: Android system.

Name : Store Information

Level 1.4:



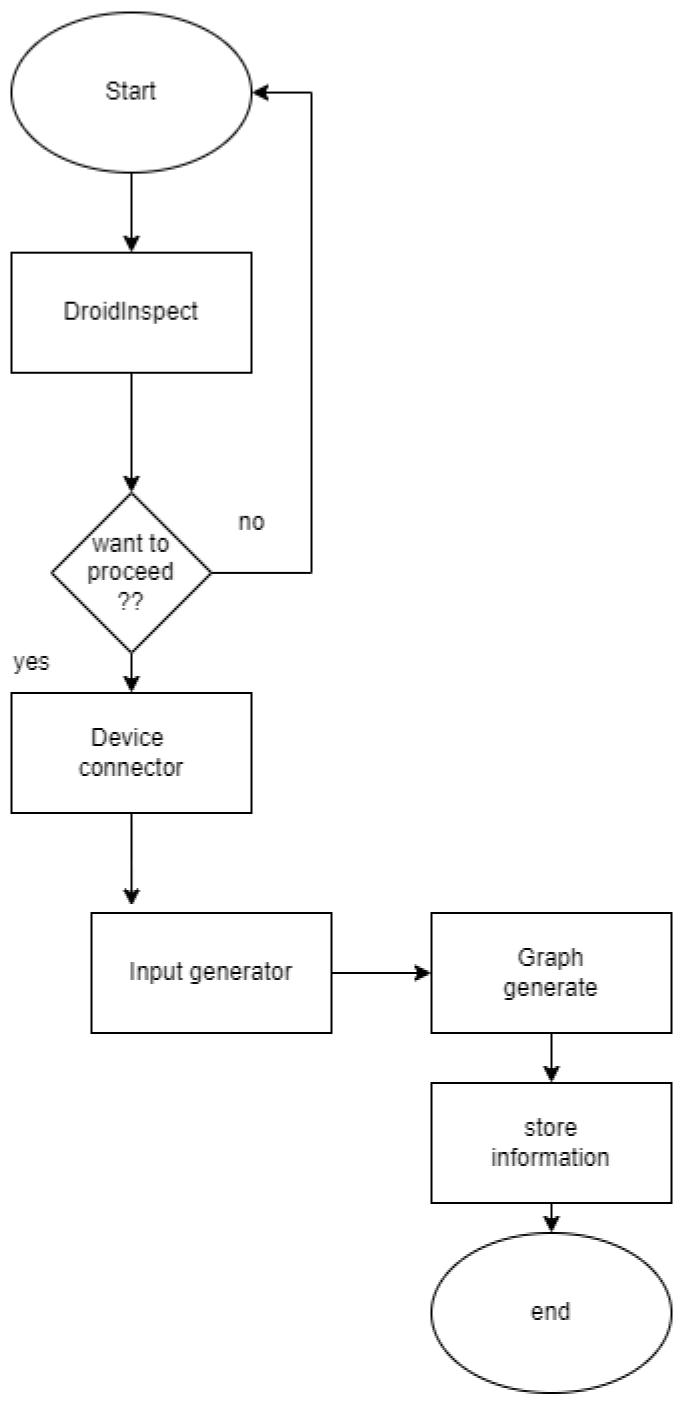
Level 1.4 Store Information

Primary actors: None
Secondary actors: Database.

`

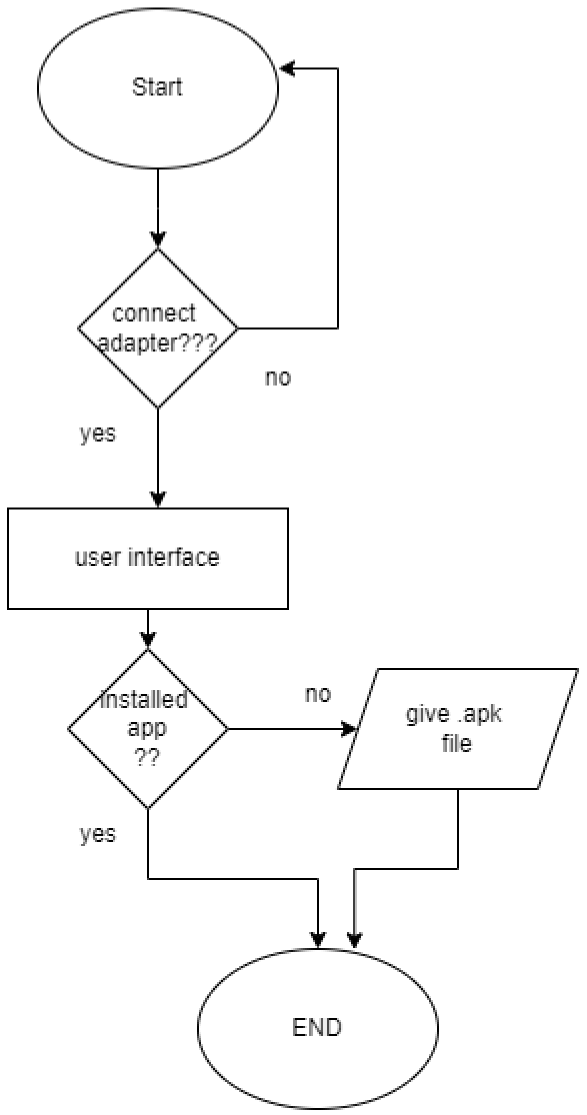
7. ACTIVITY DIAGRAM:

LEVEL 1:
Name: Modules of DroidInspect.
Reference: Use case Diagram Level -1;


Start
DroidInspect
want to proceed ??
no
yes
Device connector
Input generator
Graph generate
store information
end

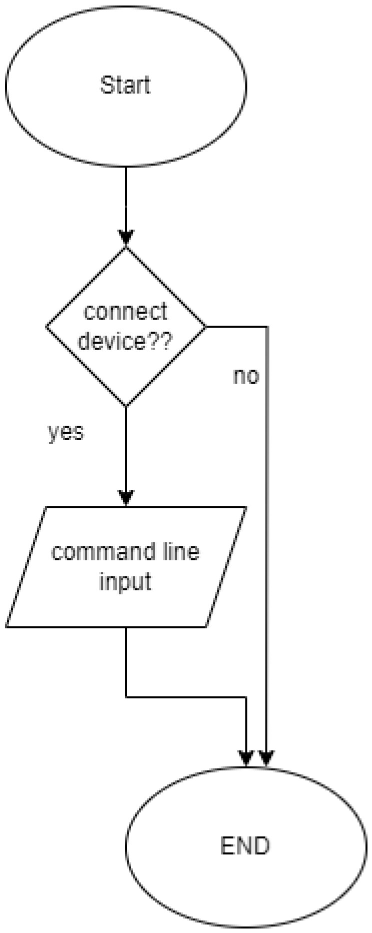
LEVEL 1.1:

Name:Device Connector
Reference: Use case Diagram Level -1.1;

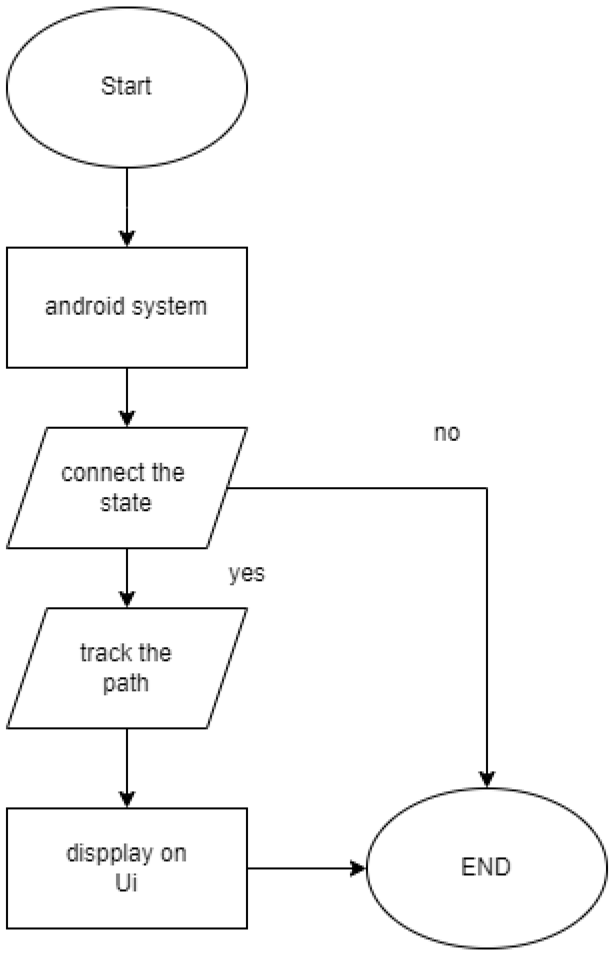


LEVEL 1.2:
Name:Input Generator

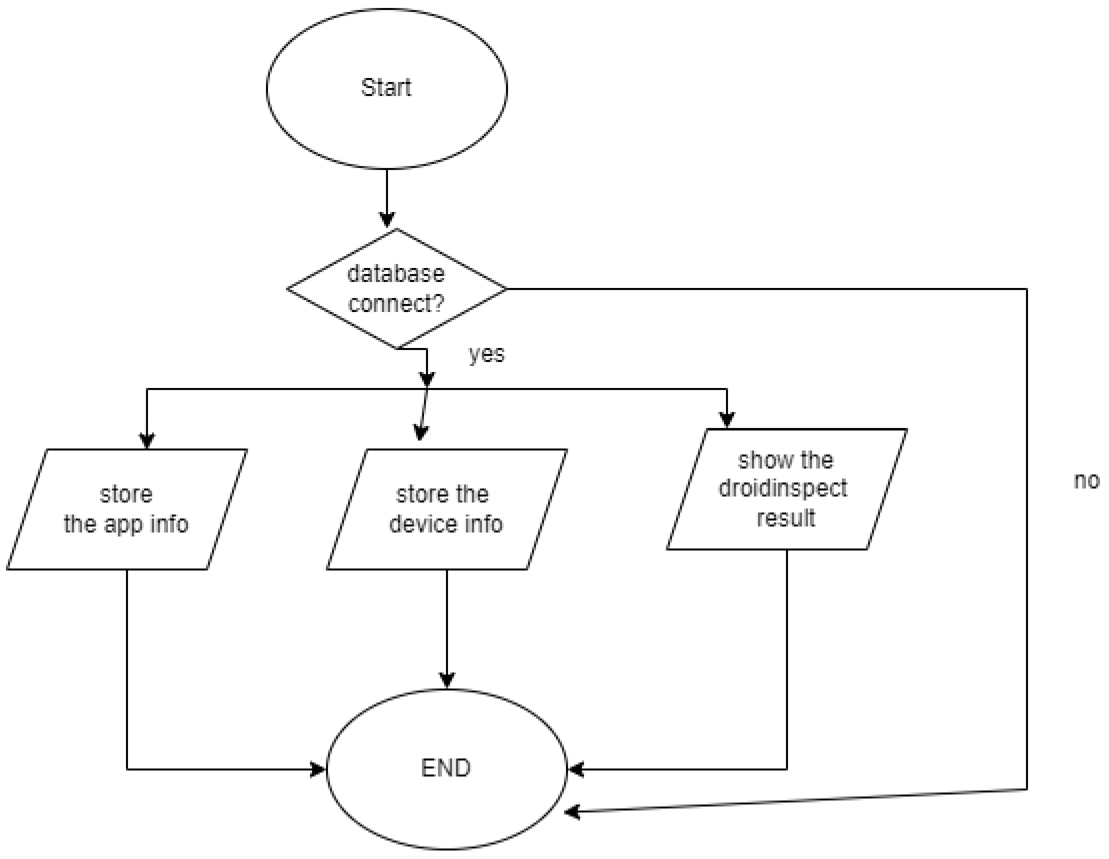
Reference: Use case Diagram Level -1.2;



LEVEL 1.3:
Name: Graph Generator.
Reference: Use case Diagram Level -1.3;



LEVEL 1.4:
Name:Store Information
Reference: Use case Diagram Level -1.4;



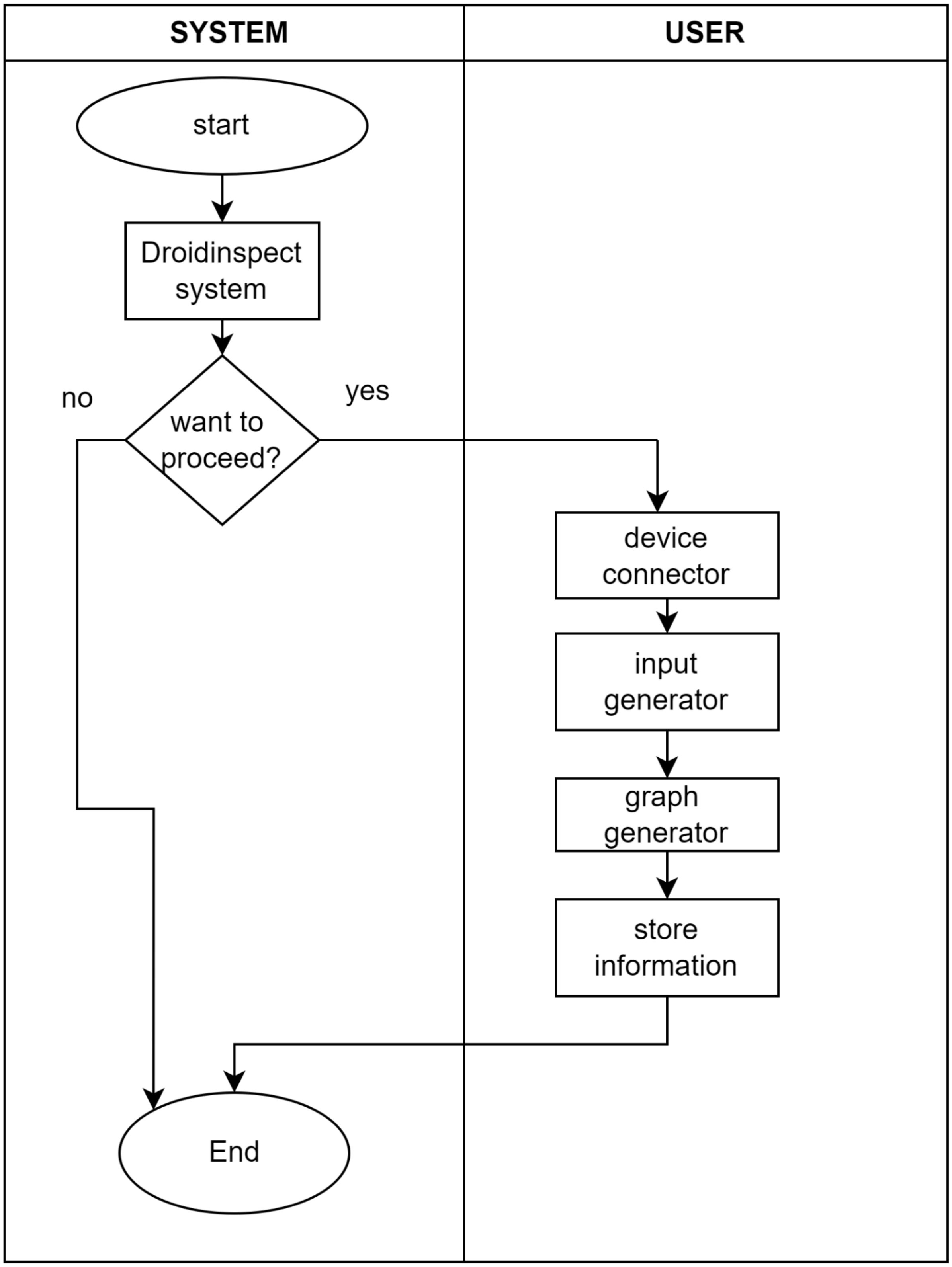
8. Swimlane Diagrams:

A swimlane diagram is a flowchart showing who is responsible for what throughout a flowchart that shows who is responsible for what throughout a certain procedure. Similar to a flowchart, it depicts a process from beginning to end, but it also classifies these phases to indicate which departments or individuals are in charge of each set of actions. Using a pool's lanes as an analogy, it places process stages within the vertical or vertical "swimlanes" of a specific division, team, or worker, as a result ensuring accountability and clarity.

Level: 1

Name: Modules of DroidInspect.

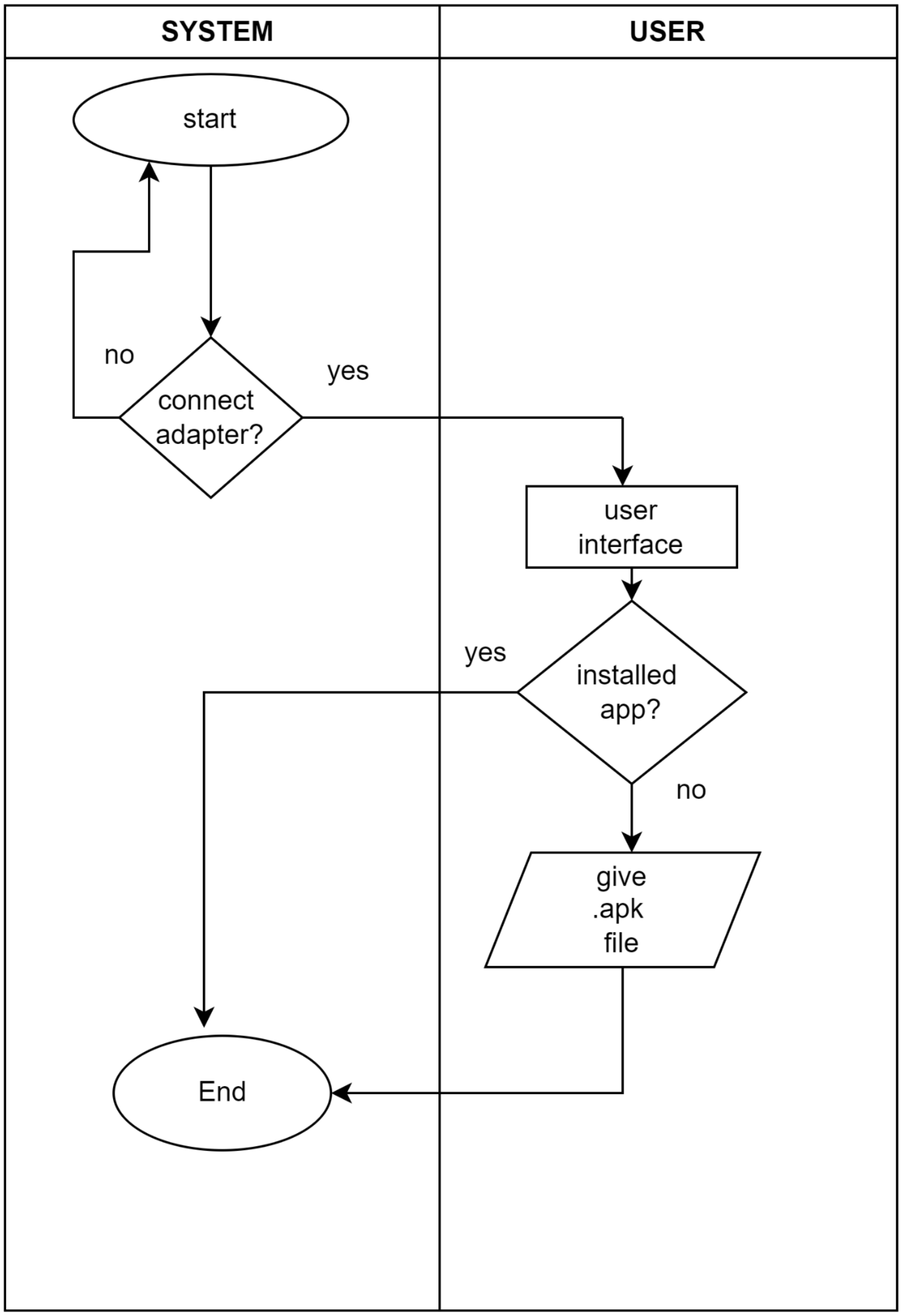
Reference: Use Case Diagram & Activity Diagram Level – 1


SYSTEM
USER
start
Droidinspect system
want to proceed?
no
yes
device connector
input generator
graph generator
store information
End

Level: 1.1

Name: Device Connector

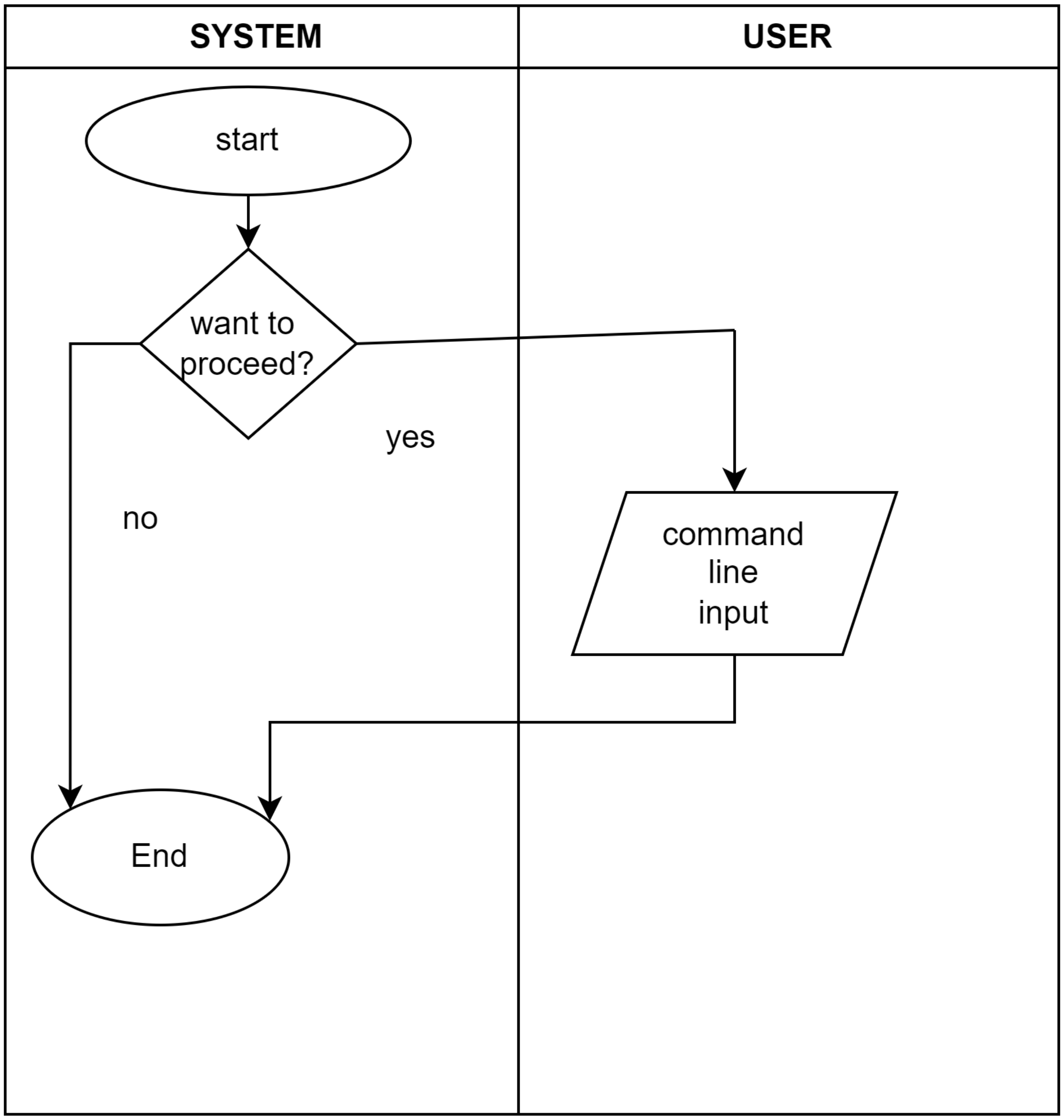
Reference: Use Case Diagram & Activity Diagram Level – 1.1


SYSTEM
USER
start
connect adapter?
no
yes
user interface
installed app?
yes
no
give .apk file
End

Level: 1.2

Name: Input Generator

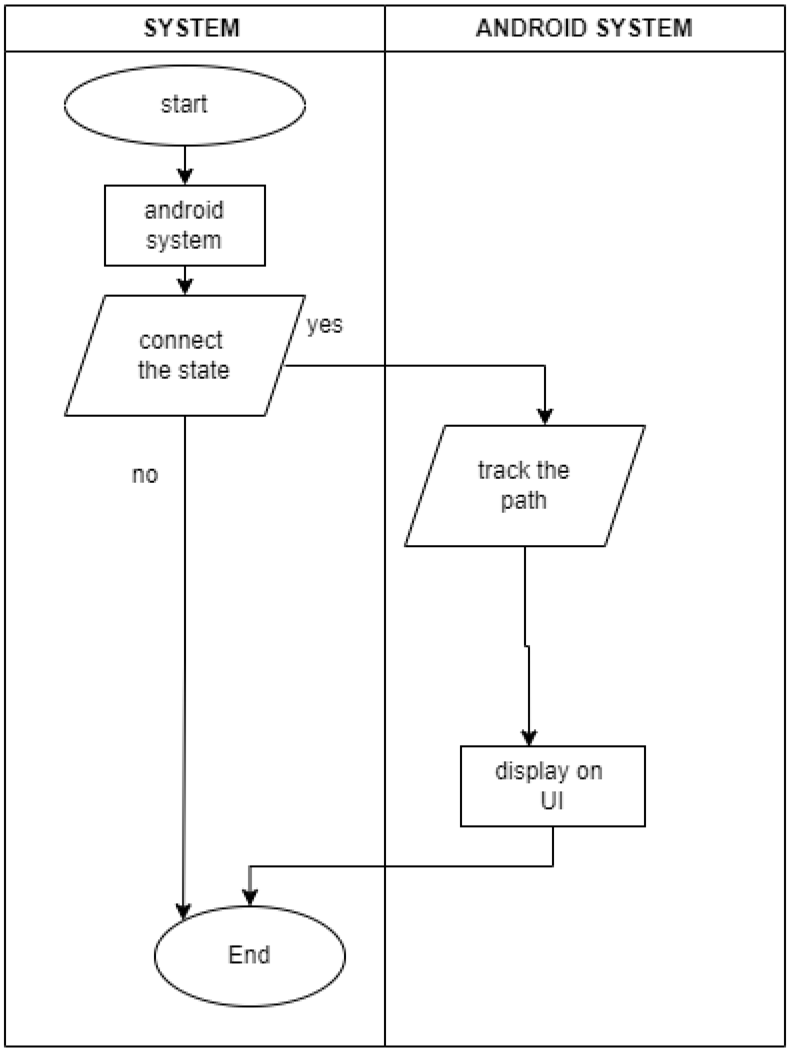
Reference: **Use Case Diagram & Activity Diagram Level – 1.2**


SYSTEM
USER
start
want to proceed?
yes
no
command line input
End

Level: 1.3

Name: Graph Generator

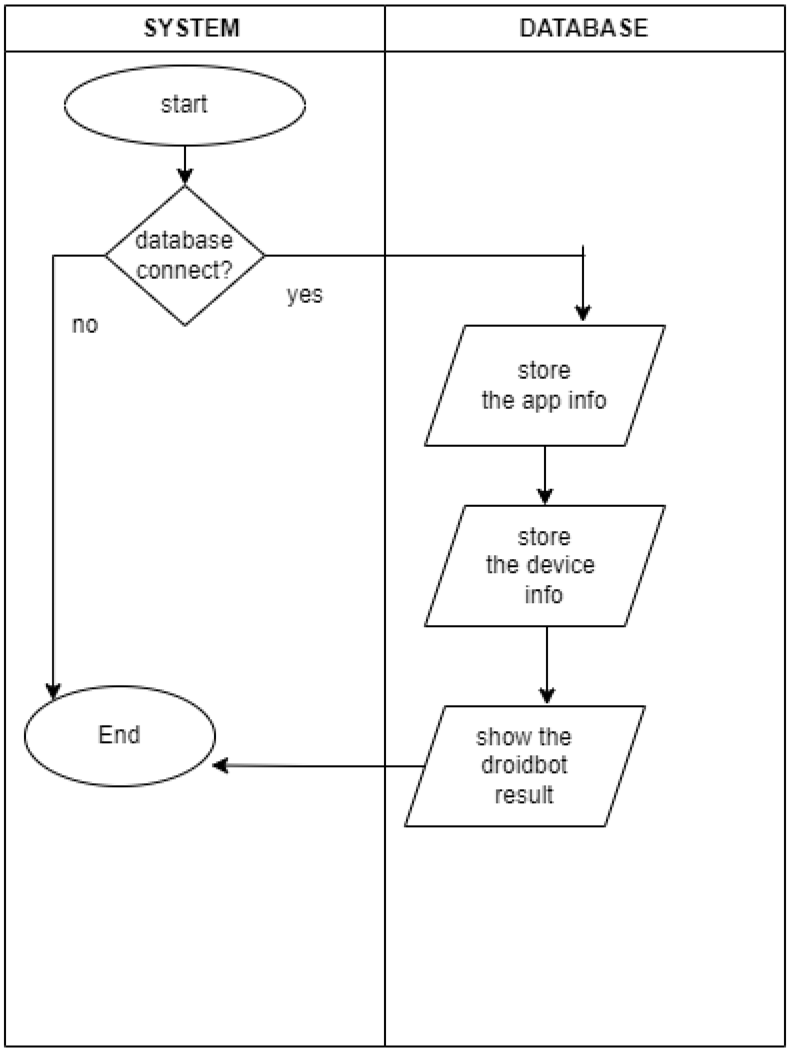
Reference: Use Case Diagram & Activity Diagram Level – 1.3



Level: 1.4

Name:Store Information

Reference: Use Case Diagram & Activity Diagram Level – 1.4



9.DATA MODELING:

List Of Nouns:

LIST OF NOUN	Prb/ Sol	Attributes	
DroidInspect result	s	**test date**, time,utg stage,utg edge	
Testing	s		
Tool	s		
Automation	s		
UI	s		
User	s	userid	
Resources	p		
Device	p		
Compatibility	s		
Workflow	p		
Contextual understanding	p		
Applications	s		
Efficiency	s		

Agents		p		
Motivation		p		
Landscape		p		
Time		p		
Costs		p		
Quality assurance		s		
Industry		s		
Quality		s		
device info		s	device serial,model num,sdk version	
Budgets		s		
Defects		s		
Production		p		
Overheads		p		
ROI (Return on Investment)		s		
Development		s		
Inception		s		
Problem		p		
Team		p		
Developers		p		
Testers		p		
Amounts		p		
Test cases		s		
Devices		p		
Configurations		p		
Solution		s		
Thoroughness		s		
Paths		s		
Interactions		s		
Tasks		s		
app info		s	package name,activities	

Aspects		s		
Edge cases		s		
User experience		s		
Planning		s		
Features		s		
Functionalities		s		
Mechanisms		s		
Reporting		s		
Integration		s		
Frameworks		p		
Environments		p		
Goals		p		
Time		p		
Detection		s		
Rates		s		
Principles		s		
Design		s		
Milestones		s		
Impact		s		
Practices		s		
Portion		s		
Quality Function Deployment		s		
Technique		s		
Needs		p		
Customer		p		
Requirements		p		
Software		s		
Inputs		s		
Fields		s		
Application		s		
Option		s		
Credentials		s		

Exciting Re quirements	p		
Custom user-defined inputs	s		
Intelligent agents	p		
Cycles	s		
utg edge	s		
test date	s		
utg stage	s		
package name	s		
device serail	s		
model num	s		
sdk version	s		

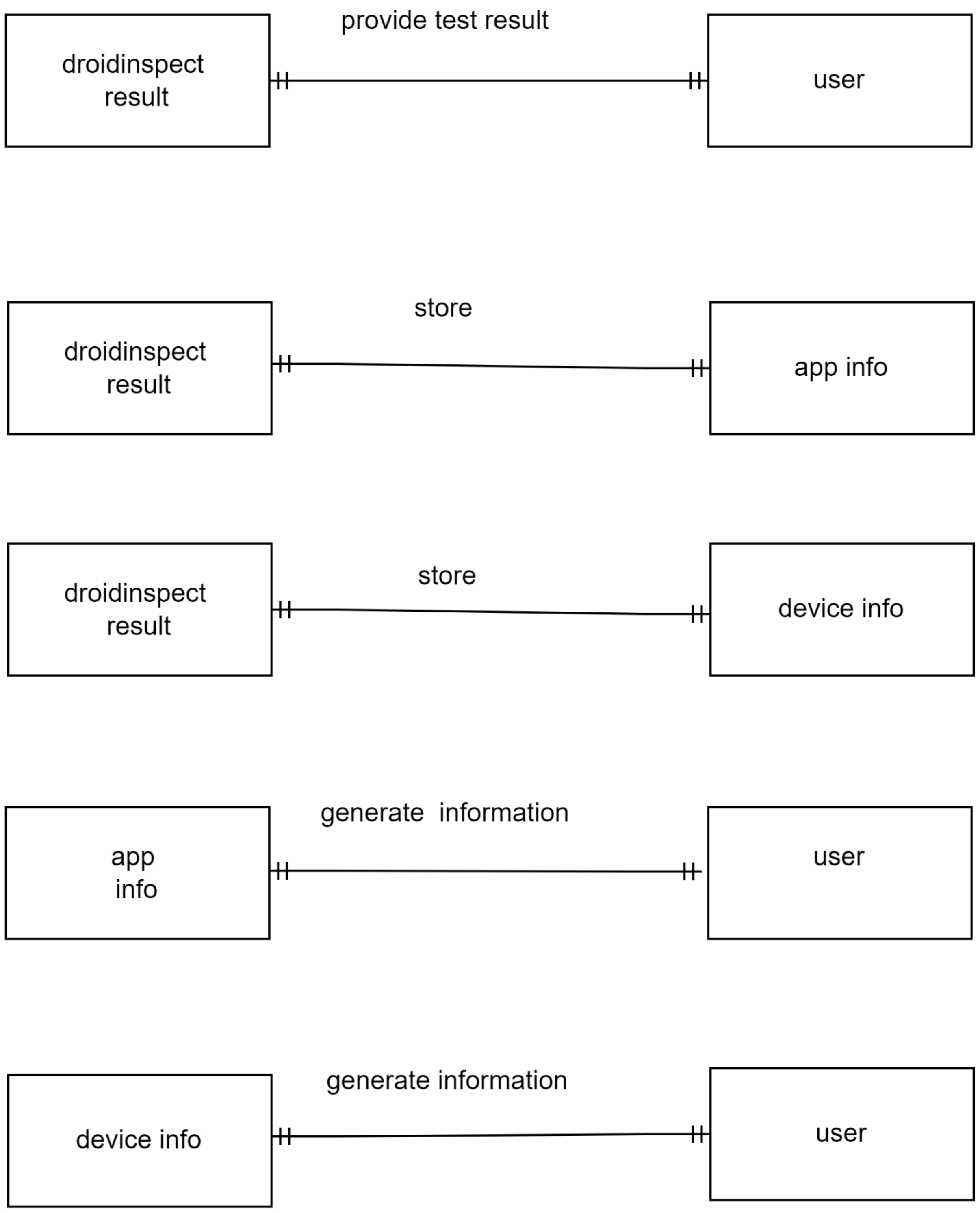
LIST OF DATA OBJECTS:

DATA OBJECTS	Attributes
Droidinspect result	test date, time,utg edge,utg stage
User	user id
App info	package name, activity
Device info	device serial,model num,sdk version

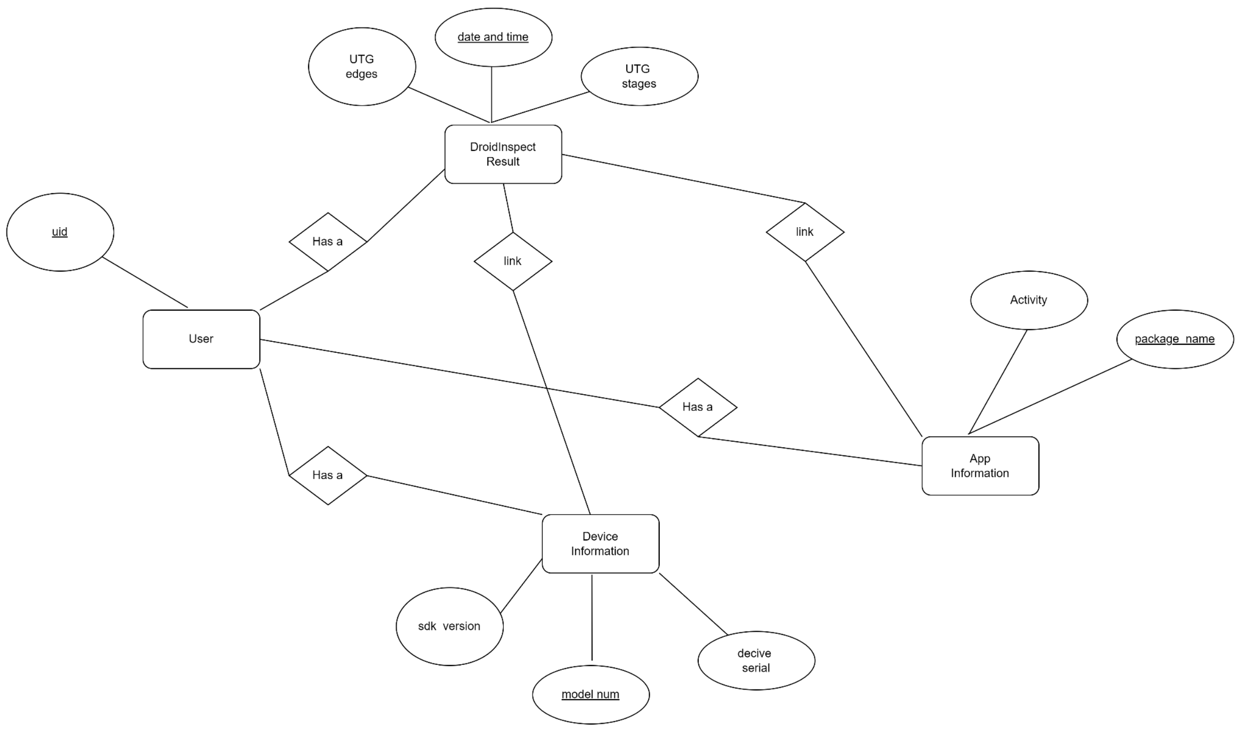
Analysis

Before proceeding to finalize class cards, we need to analyze our list of objects based on their attributes and methods. Depending on their attributes & operations we may need to remove objects and make a superclass / abstract class or interface. And this discussion is now delineated as follows:

9.1 RELATIONAL DIAGRAM:



9.2 ER diagram:



9.3 Schema of ER diagram :

Droidinspect result
Test Date
Test Time
UTG edges
UTG states
Graph

User
User ID
User's Device
connect system
Check activity
get result

App Info
package name
activity
extention
Released date
Operation

Device Info
Device Serial
Model Number
SDK Version
ROM
RAM

10. CLASS BASED MODELING:

Class-based modeling defines the structure of the entire system by identifying the static structure of objects in that system. A class model defines attributes and operations for the objects of each class and also the relationship between the objects, and the collaborations that occur between the classes of the systems. The elements of a class-based model include classes and objects, attributes, operations, class-responsibility-collaborator (CRC) models, collaboration diagrams, and packages.

VERB LIST:

tailored
aimed
overcoming

offering
optimizes
ensures
integrates
distinguishes
enhancing
utilizing
revolutionize
saving
reducing
improving
includes
store
provided
require
connected
accessible
granted
given
disabled
enabled
will
be
recognized
are
design
reduce
improve
Store
be
provided
will

require
connected
accessible
be
granted
given
Disabled
enabled
given
be
will
be
will
do
requiring
being
defined
specified
envisioned
poses
make
is
recognizing
begins
envision
simulating
be
comprehensively
assessing
positioned
alleviate
associated

enhance
improves
focus
exploring
executing
defined
should
possess
track
set
reducing
improving
enhancing
remains
achieved
takes
builds
impacts
plan
be
deployed
translates
identified

General Classification:

In this section we’ll include those classes that are in the solution space. These candidate classes are categorized based on the seven general classifications. The analysis classes manifest themselves in one of the following ways:

1. External entities
2. Things
3. Events
4. Roles
5. Organizational units
6. Places
7. Structures

A candidate class is selected for special classification if it fulfills three or more Characteristics.

		GENERAL CLASSIFICATION	
DroidInspect		2,7	
Testing		2	
Tool		2	
Automation		3	
UI		2	
Streamlining			
Approach			
Resources		2	
Device		2,4,7	
Compatibility		2	
Workflow		3	
Contextual understanding			
Applications			
Efficiency			

Agents		4	
Motivation			
Landscape		2	
Time		2	
Costs		2	
Quality assurance			
Industry		5	
Quality			
Assurance			
Budgets		2	
Defects		2	
Production		2	
Overheads			
ROI (Return on Investment)		3,4	
Development		2	
Inception			
Problem			
Team		5,6	
Developers		4	
Testers		4	
Amounts		2	
Test cases			
time			
Configurations		2	
Solution		2	
Thoroughness			
Paths		2	
Interactions		2	
Tasks		3	
Efforts		3	
Aspects			
Edge cases			

UI elements		2,3,4,7	
Planning		3	
Features		3	
Functionalities		3	
Mechanisms			
Reporting			
Integration			
Frameworks		2	
Environments		2	
Goals		2	
Graph		2,3,7	
Detection			
Rates			
Principles			
Design		3	
Milestones		2	
Impact		2	
Practices			
Portion			
Quality Function Deployment		2	
Technique		2	
Result server		2	
Customer		4	
Requirements		2	
Software			
Input generator		2,7,3	
Fields		2	
Application		2	
Ui nodes		2,3,7	
Credentials			
Exciting Requirements		2,3	
Custom user-defined inputs		2	

Intelligent agents		2	
Cycles		2	
utg edge		2	
test date		2	
utg stage		2	
package name		2	
device serail		2	
model num		2	
sdk version		2	
Database		2,3,7,	

We can see that we got 6 potential classes for further analysis.

Selection Criteria:

In this section we'll include those classes that are selected in general classification.These candidate classes are then selected as classes by six Selection Criteria. The criterias are

1. Retain information
2. Needed services
3. Multiple attributes
4. Common attributes
5. Common operations
6. Essential requirements

A candidate class generally becomes a class when it fulfills around three characteristics.

	Selection criteria	
Device	1,2,3,4,6	
Inputgen erator	2,3,4,6	
Graph	2,3,4,6	
UI elements	1,2,3,4,6	
UInodes	2,3,4,6	
Databas e	2,3,4,6	

Final List of Class :

After analyzing the list of objects we have merged or altered several classes, the final classes remaining are :

selected final class	
device	
input generator	
graph	
Ui elements	
Ui nodes	
database	

CLASS CARDS:

10.1 Class Cards for Device

Device	
Attributes	**Methods**

application	**+startApp()**
	+sendInput()
	+getUIinformation()

Responsibilities	Collaborators
1. send input command 2. get ui information	inputgenerator.

10.2 Class Cards for UI Elements

UI Elements	
Attributes	**Methods**
elementtype **stateid** **actiontype**	**+change Status()**
Responsibilities	Collaborators
1 encapsulates an ui element properties,	inputgenerator graph

10.3 Class Cards for Graph

Graph	
Attributes	**<u>Methods</u>**
paths **state**	**<u>+generategraph()</u>** **<u>+addnode()</u>** **<u>+trackPath()</u>** **<u>+traverse()</u>**
Responsibilities	Collaborators
1.represent the graph 2.traverse the graph	uinode database

10.4 Class Cards for UI Node

UI node	
Attributes	**<u>Methods</u>**
adjancency_list **ui elements**	**<u>+gettransition()</u>** **<u>+getneighbour()</u>**
Responsibilities	Collaborators
1.present the state 2.keep tracking of nodes	graph Ui element

10.5 Class Cards for Input Generator

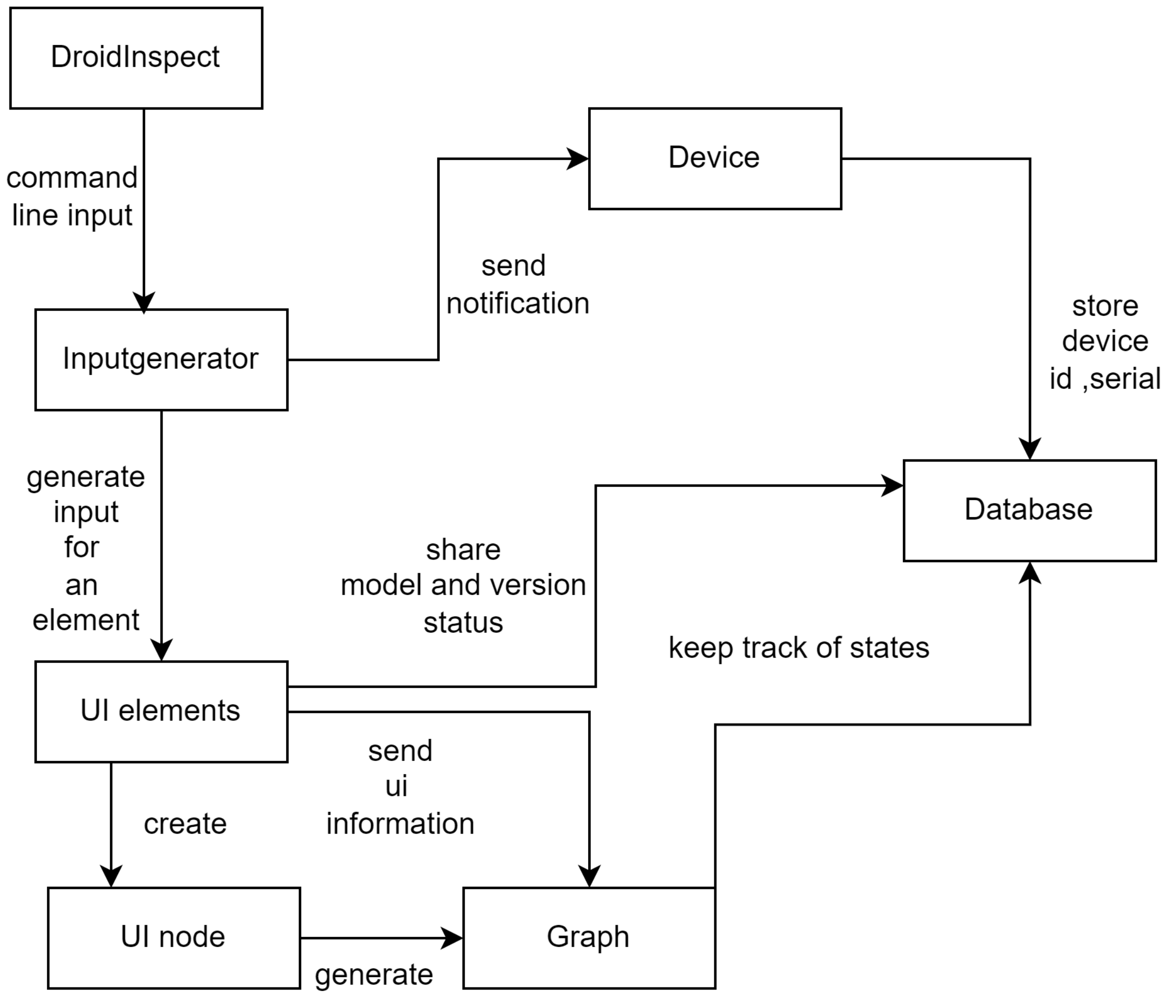
Input generator	
Attributes	**Methods**
castominputlist	**+generatinput()**
Responsibilities	Collaborators
1.store custom input for generating. 2.interface with external models	UI elements Device

10.6 Class Cards for Database

Database	
Attributes	**<u>Methods</u>**
model **test time** **sdk version**	**<u>+generateAppinfo()</u>** **<u>+generateDeviceinfo()</u>**
Responsibilities	Collaborators
1.store the info 2.show it to the user or company	Device Graph UI elements

Class Responsibility and Collaboration Diagram

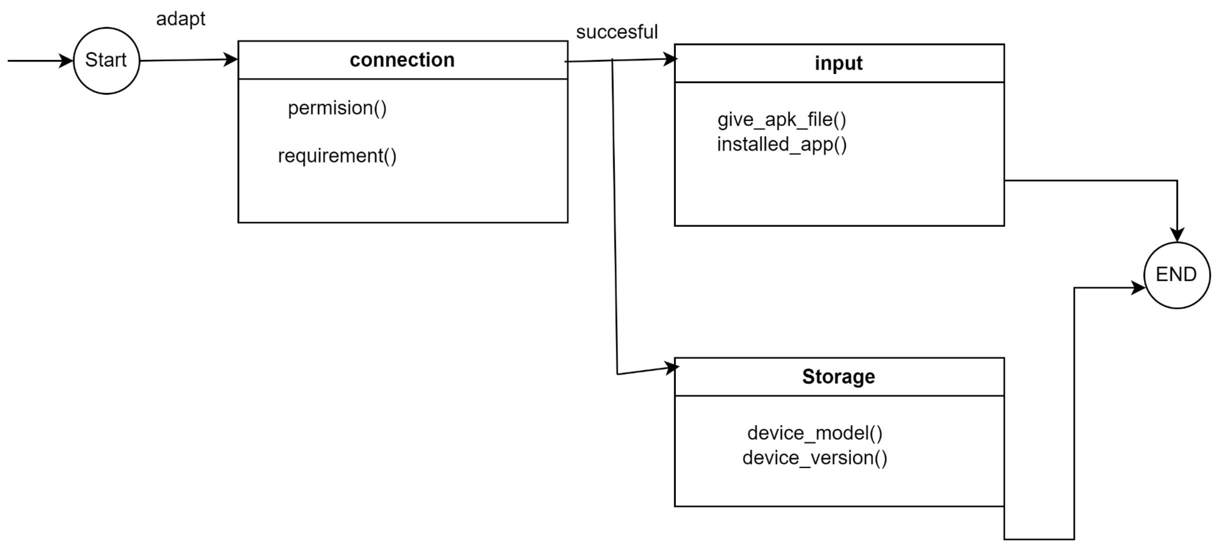
The CRC diagram for DroidInspect :Auto UI inspector is given below -



11. Behavioral Modeling:

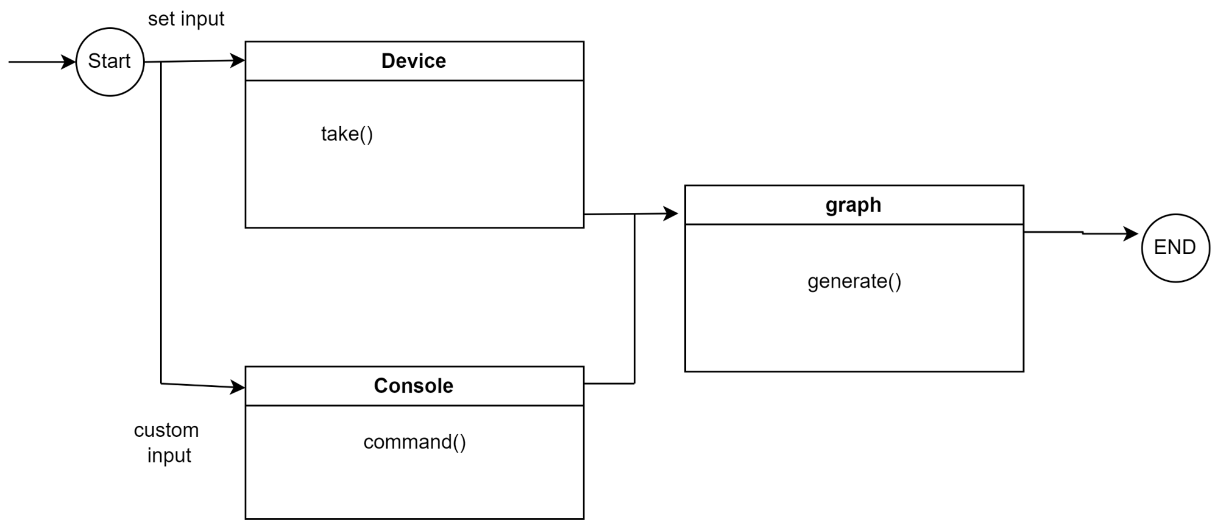
Initiator	Events	Collaborator
Device	Take inputs form user	Input generator
UI elements	help to generae input	Input generator
Input generator	takes input from user	Device
Database	Store information about Device	Device
Input generator	generate input and constract a graph	Graph
Graph	is constite of different thing like	UI elements
UI Nodes	Consist of different	UI elements
Database	contain different types	UI elements

ID -1
Name : Device



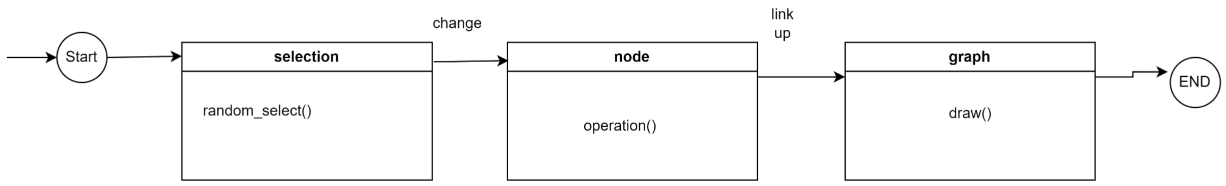
ID -2

Name : Input Generator

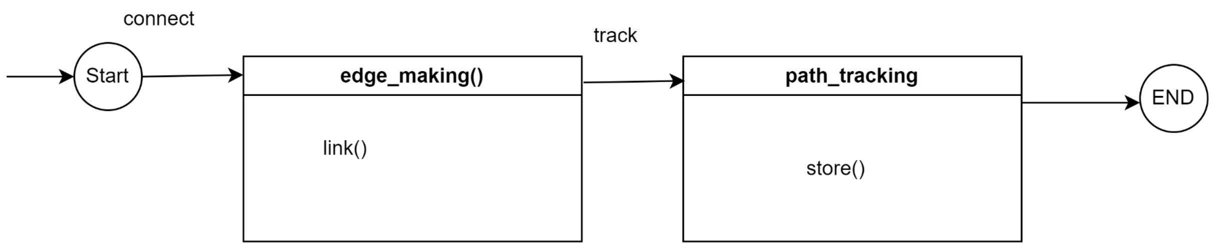


ID -3

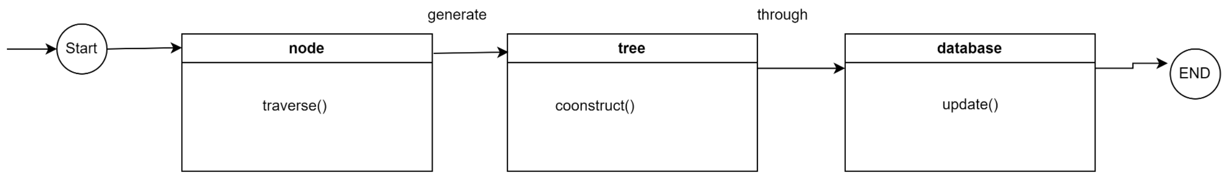
Name : UI elements



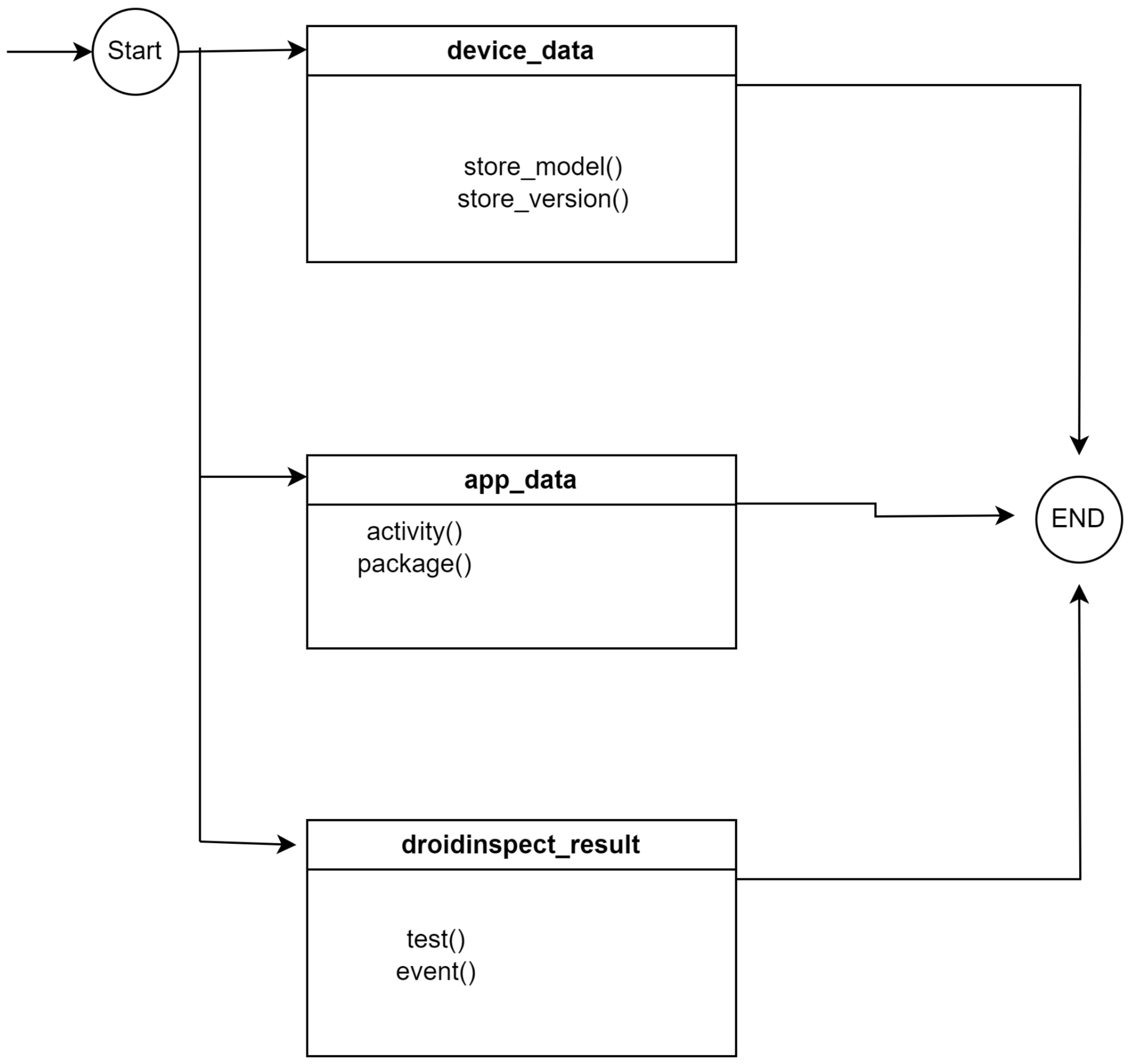
ID -4
Name :UI node



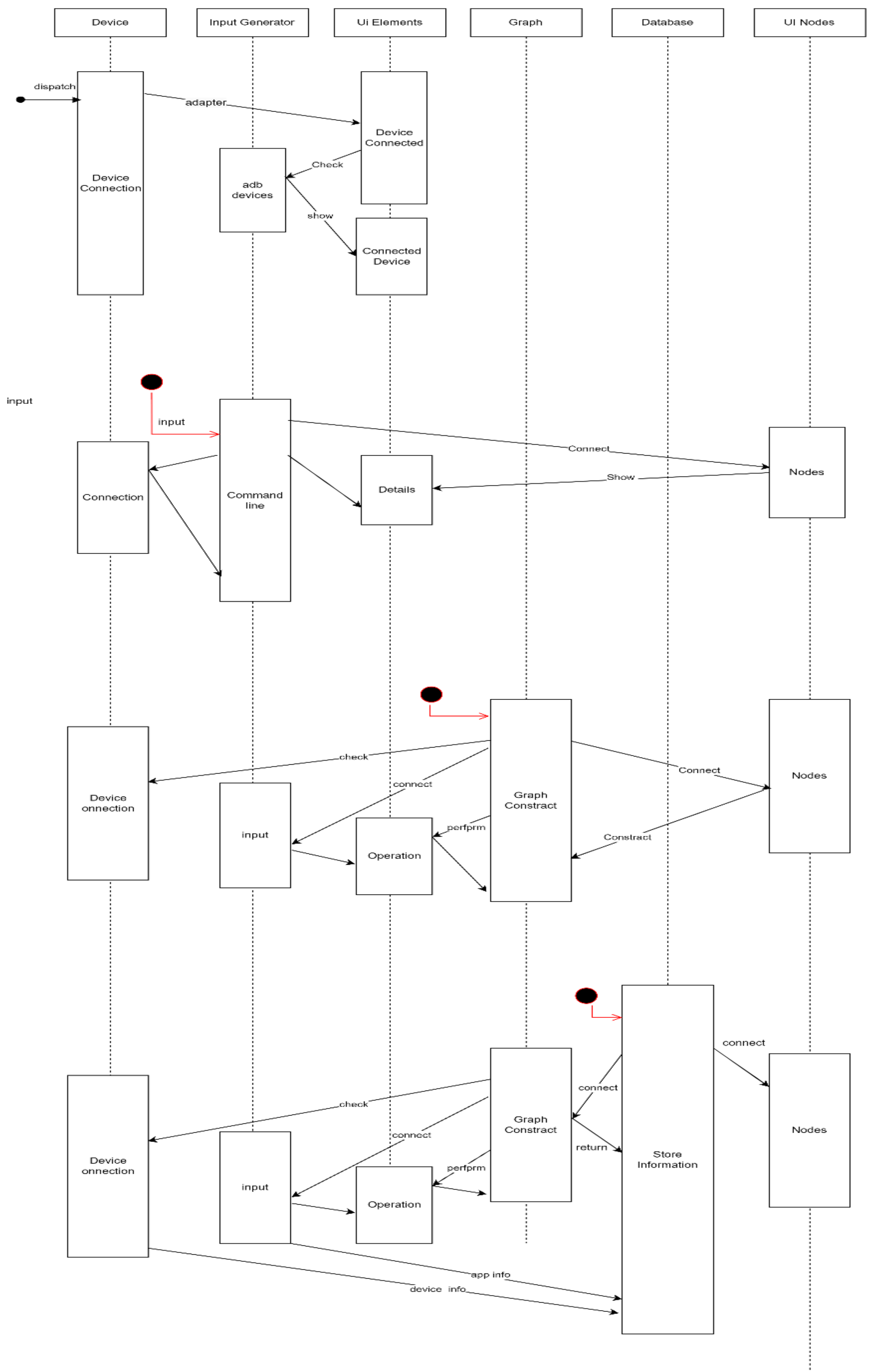
ID -5
Name : Graph

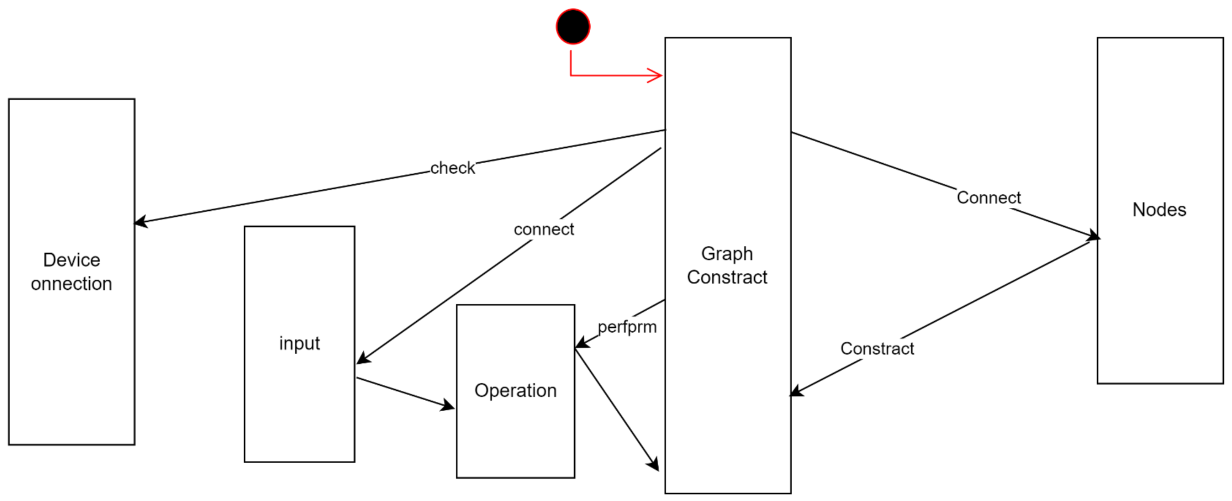


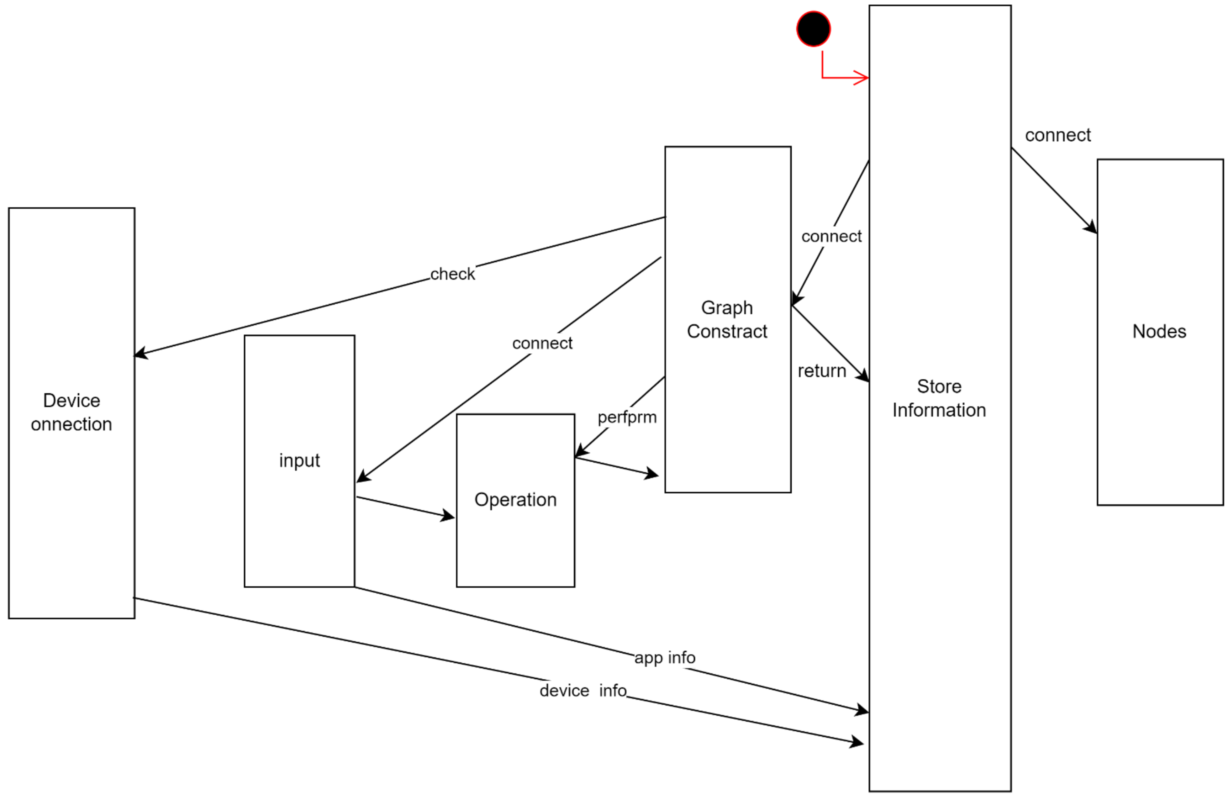
ID -6
Name : Database



Sequence Diagram


Device
Input Generator
Ui Elements
Graph
Database
UI Nodes
dispatch
adapter
Device Connection
Device Connected
Check
adb devices
show
Connected Device
input
input
Connection
Command line
Connect
Details
Show
Nodes
check
Device onnection
connect
input
Graph Constract
Connect
Nodes
perfprm
Operation
Constract
connect
Store Information
connect
return
check
Device onnection
connect
input
Graph Constract
Nodes
perfprm
Operation
app info
device info


Device onnection
input
Operation
Graph Constract
Nodes
check
connect
perfprm
Connect
Constract

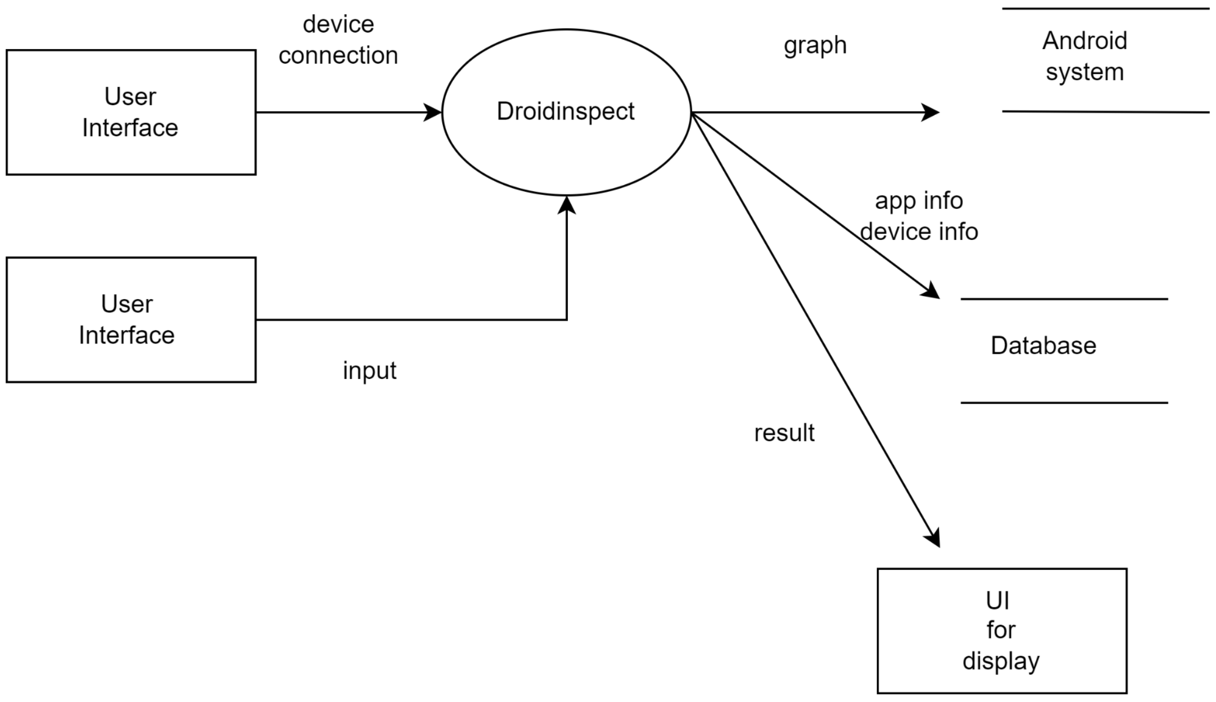

Device onnection
input
Operation
Graph Constract
Store Information
Nodes
check
connect
perfprm
connect
return
connect
app info
device info

Clear View

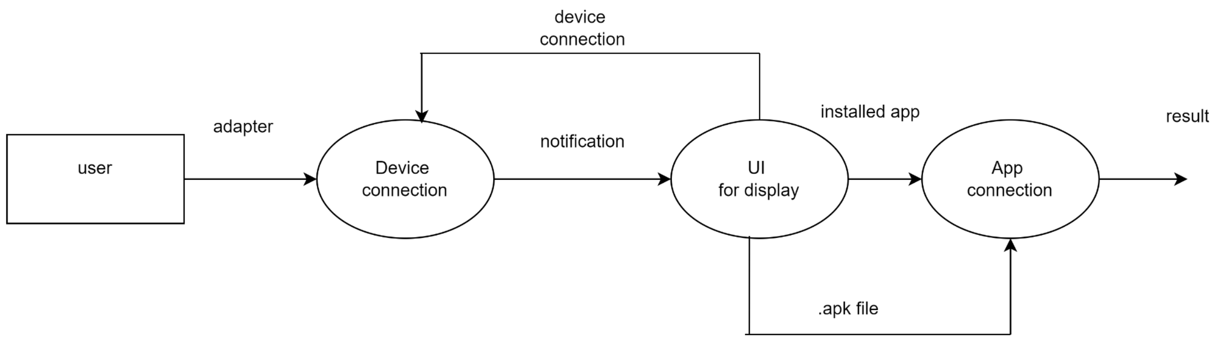
12. Data Flow Diagram (DFD):

A data-flow diagram is a visual representation of how data moves through a system or a process. A data flow diagram (DFD) shows how information moves through any system or process. It displays data inputs, outputs, storage locations, and routes between each destination using predefined symbols such as rectangles, circles, and arrows as well as brief text labels.

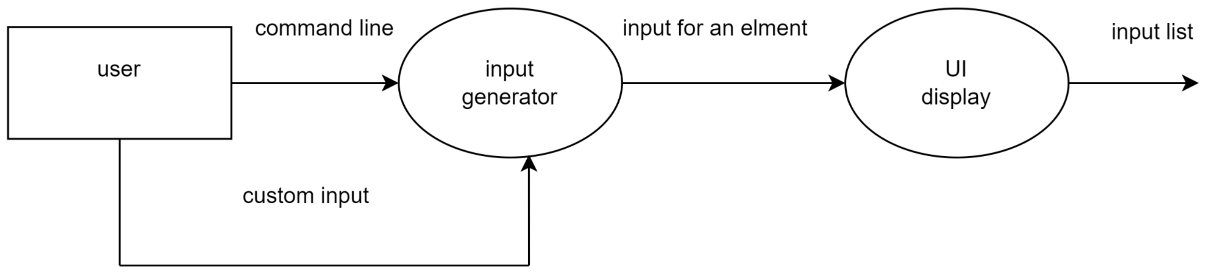
D.F.D Diagram Level 1:



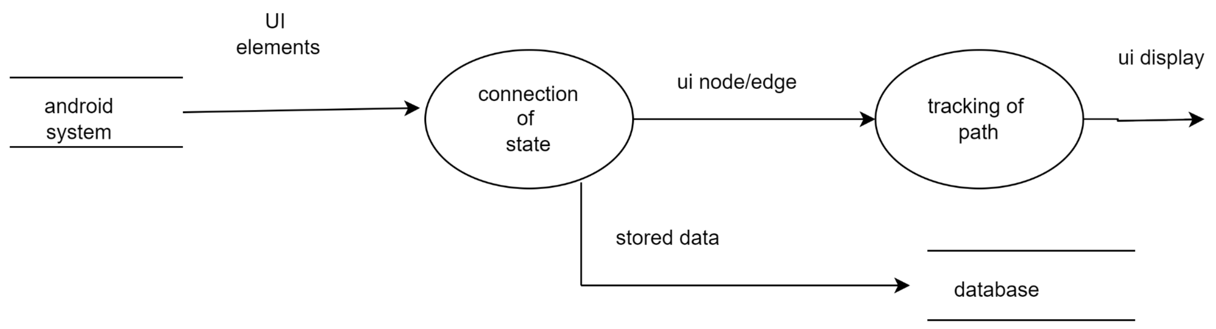
D.F.D Diagram Level 1.1:



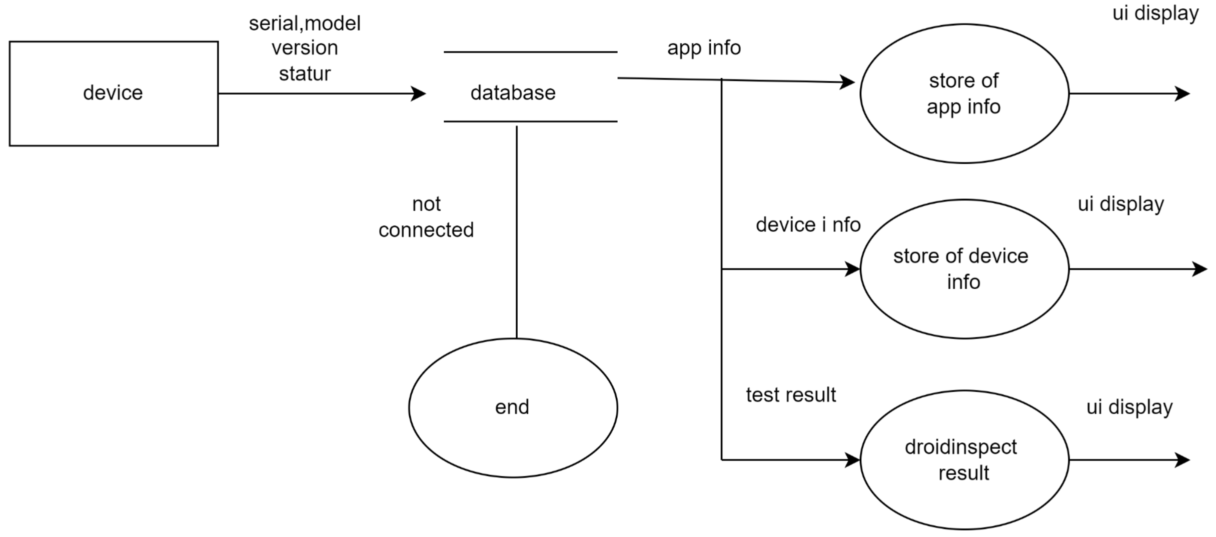
D.F.D Diagram Level 1.2:



D.F.D Diagram Level 1.3:



D.F.D Diagram Level 1.4:


device
serial,model version statur
database
app info
store of app info
ui display
not connected
device i nfo
store of device info
ui display
end
test result
droidinspect result
ui display