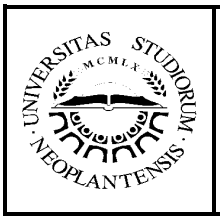


УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА

УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
НОВИ САД
Департман за рачунарство и аутоматику
Одсек за рачунарску технику и рачунарске комуникације

ЗАВРШНИ (BACHELOR) РАД

Кандидат: **Вељко Јовановић**
Број индекса: **РА 137/2020**

Тема рада: **Једно решење окружења за извршење апликација слободне форме на бази Андроид пројекта отвореног кода**

Ментор рада: **проф. др Илија Башичевић**

Нови Сад, јул 2024.

УНИВЕРЗИТЕТ У НОВОМ САДУ ● **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**
21000 НОВИ САД, Трг Доситеја Обрадовића 6

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, **РБР**:			
Идентификациони број, **ИБР**:			
Тип документације, **ТД**:		Монографска документација	
Тип записа, **ТЗ**:		Текстуални штампани материјал	
Врста рада, **ВР**:		Завршни (Bachelor) рад	
Аутор, **АУ**:		Вељко Јовановић	
Ментор, **МН**:		проф. др Илија Башичевић	
Наслов рада, **НР**:		Једно решење окружења за извршење апликација слободне форме на бази Андроид пројекта отвореног кода	
Језик публикације, **ЈП**:		Српски / ћирилица	
Језик извода, **ЈИ**:		Српски	
Земља публиковања, **ЗП**:		Република Србија	
Уже географско подручје, **УГП**:		Војводина	
Година, **ГО**:		2024.	
Издавач, **ИЗ**:		Ауторски репринт	
Место и адреса, **МА**:		Нови Сад; трг Доситеја Обрадовића 6	
Физички опис рада, **ФО**: (поглавља/страна/ цитата/табела/слика/графика/прилога)		7/36/0/3/14/0/0	
Научна област, **НО**:		Електротехника и рачунарство	
Научна дисциплина, **НД**:		Рачунарска техника	
Предметна одредница/Кључне речи, **ПО**:		Андроид, режим слободне форме, ТВ	
УДК			
Чува се, **ЧУ**:		У библиотеци Факултета техничких наука, Нови Сад	
Важна напомена, **ВН**:			
Извод, **ИЗ**:		Овај рад се фокусира на унапређењу корисничког окружења и искуства унутар Андроид пројекта отвореног кода на апликацијама које се извршавају у режиму слободне форме. Првенствено се фокусира на омогућавању корисничким апликацијама да захтевају режим покретања у слободној форми. Кроз рад се провлаче и разна побољшања апликационих прозора као што је премештање прозора у позадину, његово проширење преко целог екрана, усклађивање боја насловне компоненте, закривљени углови и подршка за покретање више апликација истовремено на једној радној површини. Такође даје увид у заштите Андроид спрега и потенцијалне начине за њихов заобилазак, као и усликавање садржаја апликације и приказивање.	
Датум прихватања теме, **ДП**:			
Датум одбране, **ДО**:		10.07.2024. 15:00	
Чланови комисије, **КО**:	Председник:	проф. др Мирослав Поповић	
	Члан:	проф. др Небојша Пјевалица	Потпис ментора
	Члан, ментор:	проф. др Илија Башичевић	

UNIVERSITY OF NOVI SAD ● **FACULTY OF TECHNICAL SCIENCES**
21000 NOVI SAD, Trg Dositeja Obradovića 6

KEY WORDS DOCUMENTATION

Accession number, **ANO**:			
Identification number, **INO**:			
Document type, **DT**:		Monographic publication	
Type of record, **TR**:		Textual printed material	
Contents code, **CC**:		Bachelor Thesis	
Author, **AU**:		Veljko Jovanovic	
Mentor, **MN**:		prof. Ilija Basicevic, PhD	
Title, **TI**:		A realization of free-form application execution on Android Open Source Project	
Language of text, **LT**:		Serbian	
Language of abstract, **LA**:		Serbian	
Country of publication, **CP**:		Republic of Serbia	
Locality of publication, **LP**:		Vojvodina	
Publication year, **PY**:		2024.	
Publisher, **PB**:		Author's reprint	
Publication place, **PP**:		Novi Sad, Dositeja Obradovica sq. 6	
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)		7/36/0/3/14/0/0	
Scientific field, **SF**:		Electrical Engineering	
Scientific discipline, **SD**:		Computer Engineering, Engineering of Computer Based Systems	
Subject/Key words, **S/KW**:		Android, freeform mode, TV	
UC			
Holding data, **HD**:		The Library of Faculty of Technical Sciences, Novi Sad, Serbia	
Note, **N**:			
Abstract, **AB**:		This thesis focuses on enhancing user interface and experience within Android Open Source Project by improving applications that are running in freeform mode. Project covers ways of allowing third-party applications to request a freeform launching mode, sending apps to background, adjusting same design of caption bars across all applications, rounding of window corners and support of running multiple applications in freeform on the same display. Work also explains how Android API restrictions work and possible ways to bypass them, as well as enabling retrieval of task snapshots and their showcase.	
Accepted by the Scientific Board on, **ASB**:			
Defended on, **DE**:		10.07.2024. 15:00	
Defended Board, **DB**:	President:	prof. Miroslav Popovic, PhD	
	Member:	prof. Nebojsa Pjevalica, PhD	Menthor's sign
	Member, Mentor:	prof. Ilija Basicevic, PhD	

	УНИВЕРЗИТЕТ У НОВОМ САДУ ● **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА** 21000 НОВИ САД, Трг Доситеја Обрадовића 6	Број:
	ЗАДАТАК ЗА ЗАВРШНИ РАД	Датум:

(Податке уноси предметни наставник - ментор)

Студијски програм:	Рачунарство и аутоматика		
Студент:	Вељко Јовановић	Број индекса:	РА 137/2020
Степен и врста студија:	Први, основне академске студије		
Област:	Електротехника и рачунарство		
Ментор:	проф. др Илија Башичевић		

НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА ИЗДАЈЕ СЕ ЗАДАТАК ЗА ЗАВРШНИ РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА:

- проблем – тема рада;
- начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна;

НАСЛОВ ЗАВРШНОГ РАДА:

Једно решење окружења за извршење апликација слободне форме на бази Андроид пројекта отвореног кода

ТЕКСТ ЗАДАТКА:

Реализовати окружење за покретање апликација у режиму слободне форме са додатним опцијама неопходним за подршку коришћења овог решења уз спољашње периферијске уређаје као што су рачунарски миш и тастатура. Решење треба да буде на бази Андроид пројекта отвореног кода.

Руководилац студијског програма:	Ментор рада:

Примерак за: o - Студента; o - Ментора

Захвалност

Захваљујем се проф. др Илији Башишевићу на вођењу и стручним саветима током писања дипломског рада. Такође желим да се захвалим својој породици и пријатељима који непрестано верују у мене како током, тако и ван трајања студија.

Посебно желим да се захвалим техничком ментору, Артјому Кузмицком, на подршци, сарадњи, вођству, као и пруженом знању током израде завршног рада.

САДРЖАЈ

Списак слика

Списак табела

Скраћенице

AOSP – Android Open Source Project

JNI – Java Native Interface

SDK – Software Development Kit

API – Application Programming Interface

OS – Operating System

PIP – Picture-in-Picture

JVM – Java Virtual Machine

LIFO – Last In First Out

RAM – Random Access Memory

ADB – Android Debug Bridge

CPU – Central Processing Unit

GPU – Graphics Processing Unit

1. Увод

Циљ овог дипломског рада је омогућавање коришћења и покретања апликација у слободној форми у оквиру Андроид пројекта отвореног кода. Идеја је надоградити режим слободне форме апликације, тако да апликација добије што већи скуп функционалности и одлика које нуде и савремени рачунари, као и њихови оперативни системи. Корисник би само спајањем рачунарског миша и тастатуре претворио свој *Андроид* уређај у радну станицу за завршавање свакодневних послова, као да се у том тренутку налази испред свог лаптоп или *desktop* рачунара.

Решење истовремено примењује програмске језике *Котлин* и *Јава*, као и механизме рефлексије за што потпуније искоришћење Андроид оперативног система са минималним променама унутар SDK ради остваривања претходно наведеног циља.

У раду су дате теоријске основе неопходне за даље разумевање примењиваних решења, њихов концепт у којем се помињу главни проблеми који су присутни, програмско решење у којима су представљени једни од начина за реализацију идеје и решење као визуелни приказ крајњег продукта.

Првенствено циљане платформе су паметни телевизори и STB уређаји са оперативним системом који је базиран на Андроид пројекту отвореног кода. Такви уређаји користе TB варијанту AOSP кода које је њима прилагођено и оптимизовано.

2. Теоријске основе

Андроид је оперативни систем заснован на модификованој верзији *Linux* језгра и других пројеката отвореног кода и првенствено је био намењен за уређаје на додир као што су паметни телефони. [1] У свом језгру, овај оперативни систем је познат и као Андроид пројекат отвореног кода (engl. *Android Open Source Project – AOSP*) који је бесплатан и издаје се под *Apache* лиценцом.

2.1 Андроид ТВ уређаји

Андроид ТВ је адаптација Андроид оперативног система намењена за паметне телевизоре. Подразумевано долази са препознатљивим корисничким окружењем и гласовном претрагом, прегледом садржаја који нуде инсталиране апликације и интегрисаним сервисима компаније *Google*.

2.2 Подршка за више апликативних прозора

Подршка за више апликативних прозора (engl. *Multi-window support*) у *Андроид ОС* омогућава апликацијама да деле исти екран истовремено. Апликације се могу покретати у више режима:

- једна уз другу (engl. *split-screen*)
- једна преко друге (engl. *picture-in-picture*)
- као засебне апликације које могу мењати своје димензије и позицију (engl. *free-form*)

Режим подељеног екрана *(split-screen)*

Режим подељеног екрана омогућава покретање две апликације истовремено дуж целе површине екрана уређаја, тако да се оне налазе једна поред друге или једна изнад

друге. Могуће је контролисати размеру њихових величина у односу на екран помоћу разделника који је смештен између њих.

Режим слика-у-слици *(picture-in-picture PIP)*

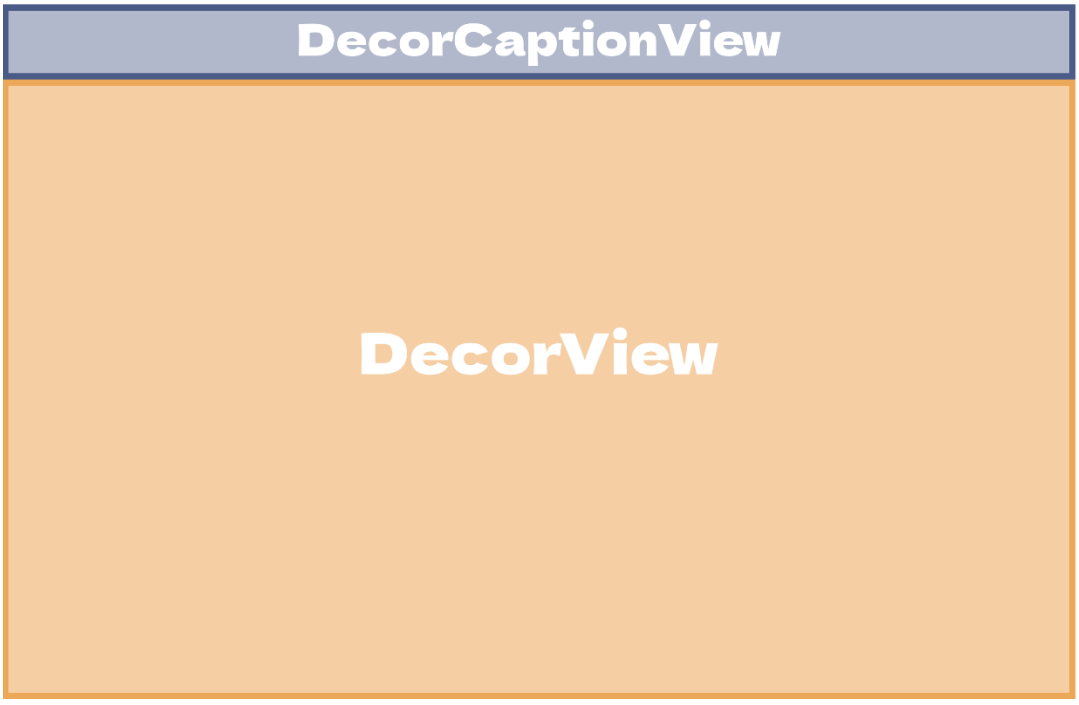
Picture-in-picture је специјални режим који дозвољава систему да покрене емитовање видео садржаја у малом прозору који је прилепљен у углу екрана, а да притом и даље може користити већински остатак екрана за остале апликације и управљање уређајем.

Режим слободне форме *(free-form)*

Free-form режим омогућава кориснику да динамично мења димензије апликативних прозора, као и да истовремено има више од две покренуте апликације на екрану. На следећој слици је приказана општа структура прозора у слободној форми који се састоји од две компоненте:

1. *DecorView*
2. *DecorCaptionView*

Ова два елемента ће бити детаљније објашњена у поглављу – 2.7.



Слика 2.1 Општа структура прозора у слободној форми

2.3 Рестрикције на спреге ван SDK-а

Почевши од Андроид 9 верзије (API ниво 28), платформа уводи рестрикције апликацијама за приступ спрегама које су ван SDK-а. Ове рестрикције се примењују у

случају када апликација референцира спрегу које не потиче из SDK-а или покушава да добави инстанцу објекта кроз рефлексију или ЈНИ. Таква ограничења су уведена како би се побољшало искуство како корисника, тако и програмера, а притом смањио ризик за падом система.

2.3.1 Разлика између спрега у SDK и ван њега

Генерално говорећи, јавне *SDK* спреге су оне које се налазе у документованом, односно индексираном *Андроид* радном оквиру, док управљање спрегама ван *SDK* је имплементација коју *API* сакрива, те су оне подложне променама без упозорења.

Како бисмо избегли пад система или неочекиване исходе, апликације требају да користе само документоване делове класа у *SDK*. Ово такође значи да програмер не би требало да приступа методама и пољима који се не налазе у *SDK* када са класом интерагује користећи механизме као што је рефлексија.

2.3.2 Екстерне листе SDK-а

При свакој верзији Андроида, уводе се додатна ограничења за екстерне *SDK* спреге. Како би се умањио утицај екстерних *SDK* ограничења на рад и развијање апликација, она бивају подељена у листе које дефинишу колико уско је њихова употреба ограничена, а све то у зависности од циљног *API* нивоа. Следећа табела показује и описује сваку листу понаособ [6]:

Листа	Кодно име	Опис
Blocklist	• `blocked`	Листа спрега које програмер не може користити независно од циљног API нивоа. Уколико апликација покуша да приступи овим спрегама, систем ће бацити грешку.
Conditionally blocked	• `max-target-x`	Почевши од Андроид 9 верзије (API ниво 28), сваки API ниво има екстерне SDK спреге којима је приступ ограничен када апликација циља тај API ниво. Ове листе су означене са max-target-x, у којима се уместо x наводи број API нивоа који апликација може да циља,

		пре него што буде спречена да приступа том делу API. Ако апликација покуша да користи спрегу, иако јој је забрањено, систем се понаша као у случају са блокирајућом листом
Unsupported	• неподржане	Спреге које нису део SDK и које су јавне и доступне за приступање обичним апликацијама. Ове спреге нису подржане и подложне су променама без упозорења. За очекивати је да ове спреге буду безусловно блокиране у предстојећим верзијама *Андроида*.
SDK	• Јавне и део SDK	Спреге које се могу слободно користити и подржане су као део официјалне документације Андроид радног пројекта
Тест API	• *Test-api*	Спреге које се користе за интерно системско тестирање. Почевши од Андроид 11 верзије (API ниво 30), тест API је део блокирајуће листе. Он је неподржан и подложан променама у било ком тренутку без упозорења

Табела 1 - Екстерне листе SDK-а

2.4 Рад са дозволама у апликацијама

Стандарне дозволе *(normal)*

Ове дозволе су подразумеване уколико се не наведу. Не уносе велики ризик и дају приступ изолованим деловима *API* са минималним ризиком по систем, корисника и друге апликације. Систем аутоматски додељује ове дозволе при инсталацији оним апликацијама које их захтевају без тражења додатне интеракције од корисника за одобрење. Кориснику је на увид доступна листа дозвола којима ће апликација имати приступ непосредно пред наставак инсталације.

Опасне дозволе *(dangerous)*

Ове дозволе уносе велики ризик због чега су и добиле назив опасне (engl. *dangerous*). Оне дају апликацијама приступ приватним подацима корисника или могу преузети контролу над уређајем, што може негативно утицати на корисника. Услед тога што ове дозволе представљају потенцијални ризик, систем може одлучити да их не додели аутоматски. На пример, опасне дозволе које захтева једна апликација могу бити приказане кориснику и од њега тражити додатну интеракцију за одобрење пре него што им апликација добије приступ.

Потписане дозволе *(signature)*

Дозволе које систем додељује само ако је апликација која их захтева потписана са истим сертификатом као и апликација која је декларисала ту дозволу. Ако се сертификати подударају, систем аутоматски додељује дозволу без додатног обавештавања или интеракције корисника.

Дозволе познатих потписа *(knownSigner)*

Дозволе које додељује систем само уколико је апликација потписана сертификатом који се налази на листи познатих (engl. *allowed*) сертификата. Уколико се сертификат апликације која захтева дозволу налази на листи претходно одобрених сертификата, систем аутоматски додељује и ове дозволе без обавештавања или интеракције корисника.

Потписане или системске дозволе *(signatureOrSystem)*

Некада познате и као "*signature | priviledged"*, ове дозволе систем додељује апликацијама само уколико се оне налазе у директоријуму предвиђеном за њих на системској слици Андроида или су потписане истим сертификатом као и апликација у којој је та дозвола декларисана. Ове дозволе треба избегавати услед тога што су потписане дозволе (engl. *signature permissions*) довољне за већину потреба независно од тога где су инсталиране.

Потписане или системске дозволе се користе у одређеним ситуацијама када више произвођача (engl. *vendor*) има апликацију уграђену у системску слику и неопходно им је да деле специфична својства услед тога што су изграђене заједно.

2.5 Јава & Котлин

2.5.1 Котлин и Јава рефлексије

Рефлексије су скуп одлика језика и библиотека које могу дати увид у структуру програма (апликације) у току извршења. На *JVM* платформама, Котлин преводилац користи алат за рефлексију као посебан артфикат `kotlin-reflect.jar`. Ова библиотека је одвојена како би се умањила величина *runtime* библиотека за апликације које не користе погодности рефлексије.

За коришћење ове библиотеке у *Gradle* или *Maven* пројектима, неопходно је додати *`kotlin-reflect`* зависност у њихове конфигурације:

***Gradle* конфигурација:**

```
dependencies {
    implementation(kotlin("reflect"))
}
```

***Maven* конфигурација:**

```
<dependencies>
  <dependency>
      <groupId>org.jetbrains.kotlin</groupId>
      <artifactId>kotlin-reflect</artifactId>
  </dependency>
</dependencies>
```

2.5.2 Референце на класе

Најједноставнија употреба рефлексије је добављање референце на *Котлин* или *Јава* класу у току извршења. Добављање референце на статички познату класу у *Котлин* програмском језику се остварује помоћу синтаксе:

```
val c = MyClass::class
```

Повратна вредност, односно референца на класу, је *KClass* типа.
Унутар JVM: Референца на *Котлин* класу није исто што и референца на *Java* класу. За добављање референце на *Java* класу, користи се *.java* својство над инстанцом *Kclass*.

2.5.3 Референце на поља

Помоћу рефлексије можемо користити `getField()` да добијемо било које јавно поље у класи или било којој њеној суперкласи. Уколико поље није јавно, може се добавити помоћу `getDeclaredField()` над *Јава* класом.

Пример позивања јавне методе помоћу рефлексије:

```
import java.lang.reflect.*;

   public class PrimerKlase {
      public double d;

      public static void main(String args[])
      {
         try {
            Class klasa = Class.forName("PrimerKlase");
            Field polje = klasa.getField("d");
            PrimerKlase polje2objekat = new PrimerKlase();
            System.out.println("d = " + polje2objekat.d);
            polje.setDouble(polje2objekat, 12.34);
            System.out.println("d = " + polje2objekat.d);
         }
         catch (Throwable e) {
            System.err.println(e);
         }
      }
   }
```

Разлика између `getFeild()` и `getDeclaredField()` је такође у приступу. Не подразумева се да је свим пољима која су добијена методом `getDeclaredField()` могуће приступити без додатних провера. У најчешћем случају, неопходно је позвати `setAccessible(true)` методу над *Field* објектом пре добављања саме инстанце, односно вредности поља. Овај начин приступа пољима која нису јавног типа путем рефлексије такође важи и за методе које нису јавне, а које ће бити поменуте у наредном поглављу

2.5.4 Референце на методе

Помоћу рефлексије можемо искористити `getMethod()` над објектом Јава класе да пронађемо било коју јавну методу у њој или било којој њеној суперкласи. Уколико метода није јавна, може се добавити помоћу `getDeclaredMethod()`.

Пример позивања јавне методе помоћу рефлексије:

```
import java.lang.reflect.Method;

public class Klasa {
    public int saberi(int a, int b)
    {
        return a + b;
    }

    public static void main(String args[])
    {
        try {
            Class klasa = Class.forName("Klasa");
            Object objekatKlase = new Klasa();

            Method saberiMetoda = klasa.getMethod("saberi", int.class, int.class);

            int povratnaVrednost = (int) saberiMetoda.invoke(objekatKlase, 5, 10);
            System.out.println(povratnaVrednost);

        } catch (Throwable e) {
            e.printStackTrace();
        }
    }
}
```

2.6 Радни задаци и њихов позадински стек

Радни задатак (engl. *task*) је колекција свих активности са којима корисник интерагује када покушава да контролише апликацију. Ове активности су организоване у позадински стек (engl. *back stack*) у редоследу у којем су и покретане. Као пример можемо узети апликацију за дописивање; када корисник отвори преписку са жељеним корисником, он ефективно покреће нову активност која се додаје на позадински стек. Притиском на *Back* дугме, до тада коришћена активност се завршава и бива избачена са позадинског стека, приметно враћајући *корак* уназад.

2.6.1 Животни циклус радног задатка и позадинског стека

Почетни екран уређаја (engl. *home screen*) је стартно место за већину радних задатака. При корисниковом додиру на иконицу или пречицу апликације у покретачу апликација, односно почетном екрану, радни задатак апликације прелази у први план (engl. *foreground*). Уколико не постоји радни задатак за иницирану апликацију, у том случају се прави нови, док се главна активност (engl. *main activity)* отвара као примарна активност (engl. *root activity*) у том стеку.

У случају када тренутна активност покреће другу, нова активност се додаје на врх стека и добија фокус система. Претходна и даље остаје у стеку, али је у заустављеном режиму. Док је заустављена, систем задржава њено стање и изглед, тако да при преласку назад на ту активност, као што је случај притиска на *Back* дугме, њено стање прелази у „*настављено”* (engl. *resumed*).

Активности се никада не преуређују, већ бивају додате на или скинуте са стека када су покренуте од стране тренутне активности или одбачене коришћењем *Back* опције. Стога, позадински стек функционише као *LIFO* структура.

2.6.2 Вишепрозорно окружење и стек

Када су апликације покренуте истовремено у једном од вишепрозорних режима поменутих у поглављу – 2.2, систем засебно управља радним задацима за сваки од прозора. Сваки прозор може имати више радних задатака.

2.7 Структура активности и њиховог прозора

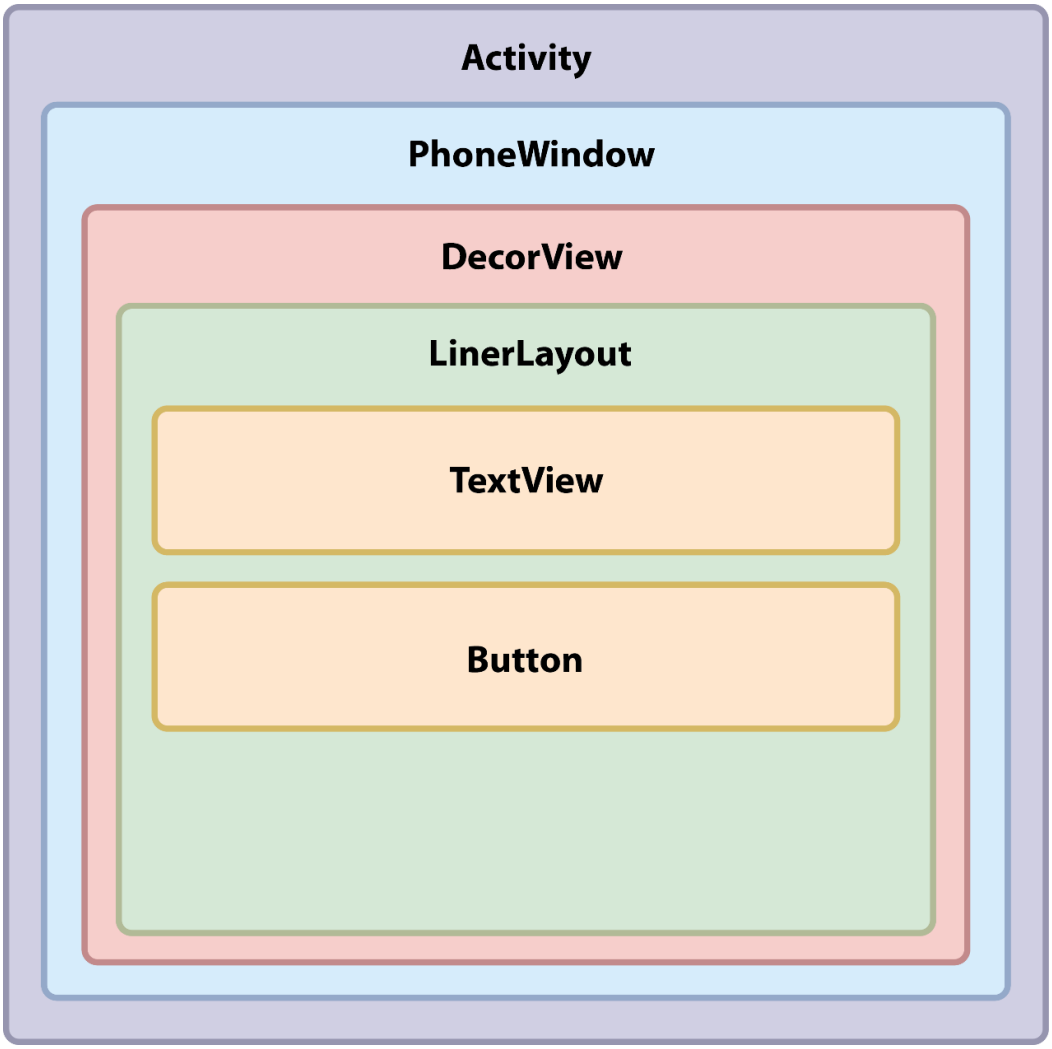
При инстанцирању активности (engl. *Activity*), *ActivityThread* класа позива `attach` методу унутар *Activity* класе и она се извршава пре `onCreate()`. У њој се прави објекат *PhoneWindow* класе.

```
@UnsupportedAppUsage
final void attach(Context context, ... ) {
  // ...
  mWindow = new PhoneWindow(this, window, activityConfigCallback);
  // ...
 }
```

PhoneWindow класа има 2 важна објекта: *DecorView* и *ViewGroup*. *DecorView* је унутрашња класа *PhoneWindow* и она је главни контејнер унутар *View* хијерархије за *Activity*. *DecorView* наслеђује *FrameLayout* класу.

На пример: *DecorView* класа садржи позадину прозора која може бити нацртана, а позивањем `getWindow().setBackgroundDrawable(Drawable drawable)` методе из *Activity* класе мењамо позадину њеног прозора.

Window класа има једну хијерархију *View* објеката који су привезани за њу и који дефинишу понашање тог прозора. Поменута хијерархија је приказана на следећој слици:



Слика 2.2 Хијерархијска структура активности једне апликације

У зависности од режима у којем је прозор, *DecorView* класа може приказивати и *DecorCaptionView* елемент, тако да у случају режима слободне форме, *DecorCaptionView* ће бити видљив, а у осталим режимима неће. Објекат *DecorCaptionView* класе се налази унутар *DecorView*. У њему се садржи део за *наслов* апликације, као и опције за контролу прозора.

```
...
// This is the caption view for the window, containing the caption and //
window control buttons. The visibility of this decor depends on the //
workspace and the window type. If the window type does not require // such a
view, this member might be null.
    private DecorCaptionView mDecorCaptionView;
...
```

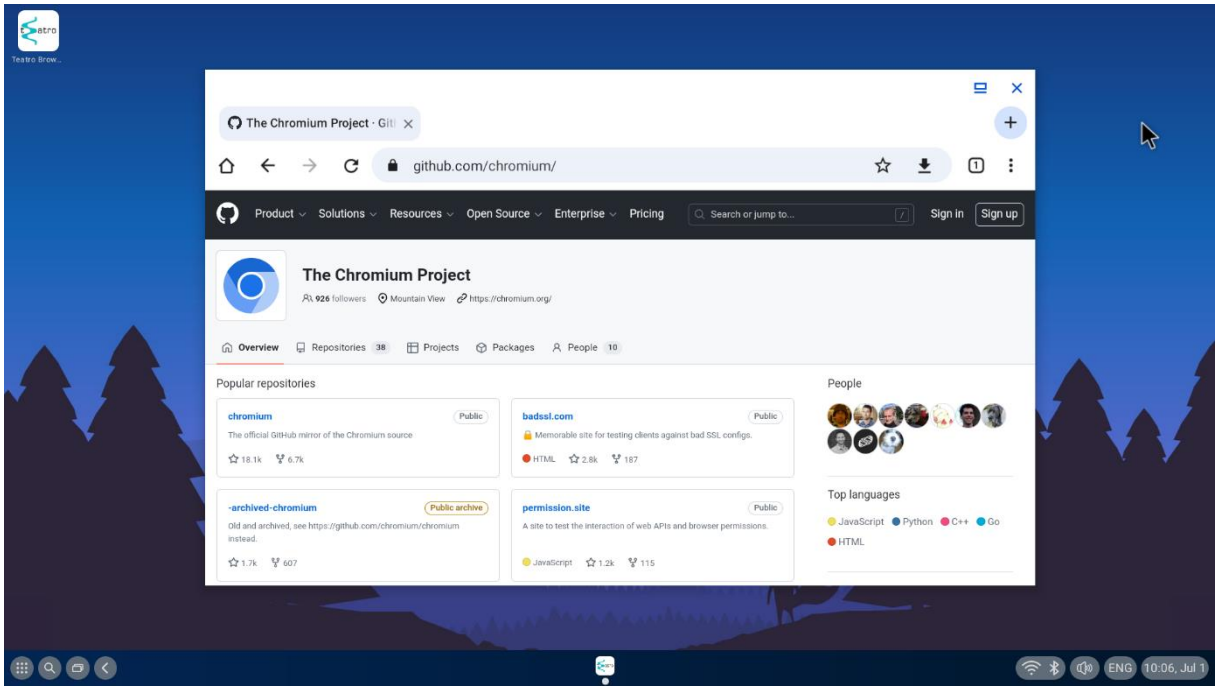
3. Концепт решења

У претходном поглављу су представљене теоријске основе неопходне за разумевање решења, као и алата, библиотека и помагала који су коришћени за његову израду. Рад се фокусира на омогућавању *Андроид ОС* да кориснику пружи што сличније радно окружење и искуство, као што нуде *desktop* рачунари са *Windows*, *MacOS* и *Linux* оперативним системима. Под тим подразумевамо да се све апликације при иницијалном покретању отварају у режиму слободне форме (engl. *free-form*), остављајући довољно места на радној површини тако да је могуће покренути више таквих апликација истовремено и неометано се пребацивати међу њима стварајући тзв. *multi-tasking environment*.

AOSP у свом коду садржи механизме за покретање апликација у режиму слободне форме, али их и даље скрива од програмера користећи заштите и ограђене приступе спрегама (2.3.2). Програмери стандардних корисничких апликација немају приступ већини метода којима би омогућили и могли утицати на *Андроид ОС*, те решење прибегава претходно поменутим механизмима потписивања, рефлексије и модификацијама изворног кода *Андроида*.

Апликација на *Андроид* уређајима која је одговорна за визуелно и интерактивно покретање, контролу, умрежавање и надгледање инсталираних апликација назива се – *покретач* (engl. *launcher*). Све апликације се подразумевано покрећу у режиму прозора највеће могуће величине (engl. *fullscreen*) што не оставља простор за приказивање других апликација које би корисник потенцијално користио истовремено. Решење овог проблема је представљено у поглављу – 4.3.

На следећој слици је приказан пример апликације покренуте у режиму слободне форме без модификација:



Слика 3.1 Изгледа прозора у режиму слободне форме без модификација

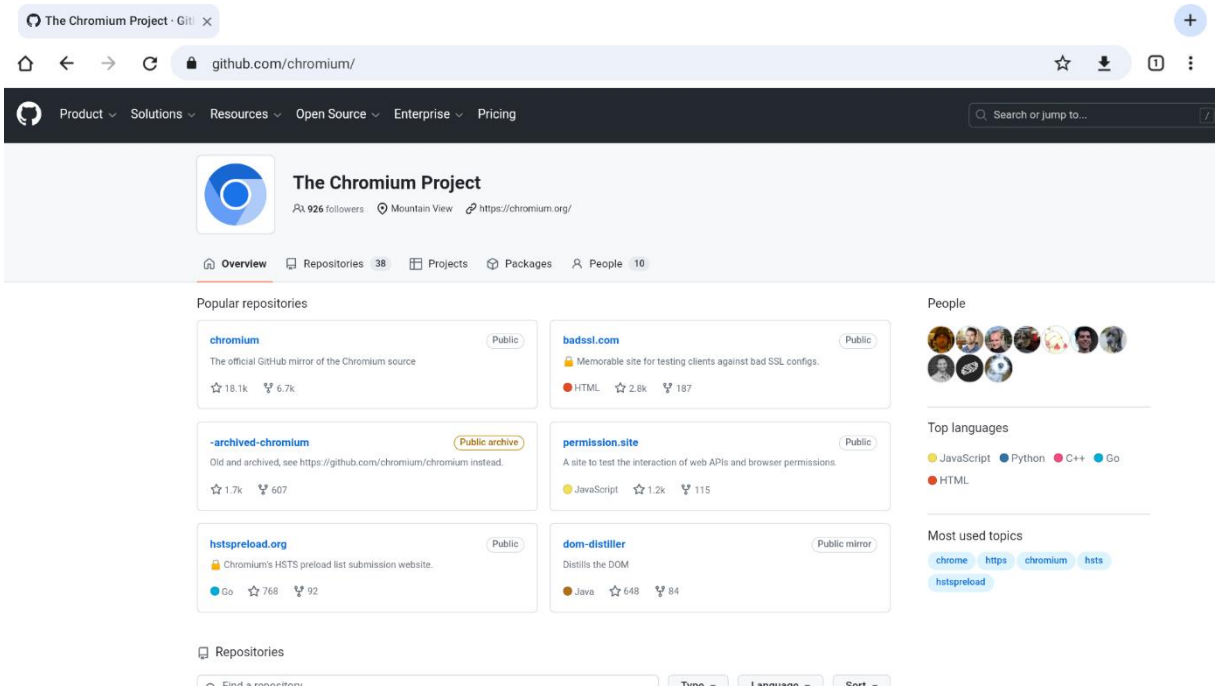
3.1 Недостаци прозора у режиму слободне форме

3.1.1 Режим прозора највеће могуће величине

Дугме за пребацивање прозора из режима слободне форме у режим највеће могуће величине, осим што мења димензије прозора, такође уклања и *DecorCaptionView* компоненту на којој су се дугмад првенствено и налазила. *DecorView* компонента је и даље присутна, али је операција за овакву промену режима прозора неповратна. Не постоји додатно дугме или комбинација на тастатури за повратак у режим слободне форме, те корисник може само да изађе из апликације у потпуности или поново покрене систем.

Решење које је примењено у техничкој изради овог рада је базирано на промени функционалности наведеног дугмета тако да при његовом притиску – апликација памти тренутне димензије у привременој промењивој, задржава режим слободне форме, а потом мења стварне димензије свог прозора на највеће могуће остављајући *DecorCaptionView* компоненту видљивом. Апликација у таквом стању је такође позната и као *maximized* апликација. У случају притиска на наведено дугме док је она у *maximized*

режиму, низ операција које се извршавају је исти, али овај пут се примењују оригиналне димензије из привремене променљиве и апликација се ефективно враћа у претходно стање.



Слика 3.2 Пример апликације у режиму највеће могуће величине без наслова

3.1.2 Режим умањеног прозора

Режим умањеног прозора, такође познат и као *minimized* режим, поставља радни задатак у ком је апликација на позадински стек, тиме је склањајући са почетног екрана при чему она наставља да ради у позадини. Ова опција није присутна у *DecorCaptionView* компоненти, те је она такође додата при изради овог решења непосредно поред опције за *maximized* режим.

Напомена: Кориснику мора бити обезбеђен додатни начин и за враћање ове апликације у предњи план како она не би и даље заузимала ресурсе система у позадини, без могућности приступања.

3.1.3 Дугме за повратак назад – *Back*

Техничко решење овог дипломског рада подразумева да ће се оно користити уз помоћ додатних периферијских уређаја, прецизније рачунарског миша и тастатуре. Опција за извршавање акције „враћања уназад“, односно *Back*, није могуће урадити помоћу ових уређаја, док у поређењу са даљинским управљачем јесте, јер он има физички *Back* тастер. Постоје 2 могућа решења овог проблема:

1. Глобално *Back* дугме унутар покретача апликација
2. *Back* дугме за сваку апликацију засебно

Глобално *Back* дугме унутар покретача апликација би извршавало акцију враћања уназад над апликацијом која тренутно има системски фокус - што може бити збуњујуће и неконзистентно у случају када је на почетном екрану приказано више од једне апликације. Тада је неопходно да корисник има визуелни индикатор који јасно показује која апликација има фокус како би знао над којом апликацијом ће се акција извршити. Уколико би желео да примени акцију враћања уназад над другом апликацијом која није у фокусу, прво мора да ју постави у фокус и тек онда притисне *Back* дугме.

Засебно *Back* дугме за сваку апликацију решава горе поменути проблем и у овом решењу је додато у *DecorCaptionView* компоненту и стога више није неопходно ручно преузимати фокус на апликацију од интереса.

3.2 Покретање апликације у режиму слободне форме

Најчешће коришћени начин за покретање апликација је уз `Context#startActivity` методу и њену варијанту која прихвата *Intent* као обавезан и *Bundle* објекат као опциони аргумент који је серијализован од инстанце *ActivityOptions* класе. Уколико се ова метода позива ван постојеће активности, мора се додати покретачка заставица `Intent#FLAG_ACTIVITY_NEW_TASK`, јер не постоји радни задатак у који би главна активност покренуте апликације могла да се смести, па сходно томе захтевамо прављење новог радног задатка како би свака од апликација у вишепрозорном режиму могла да се обрађује независно једна од друге.

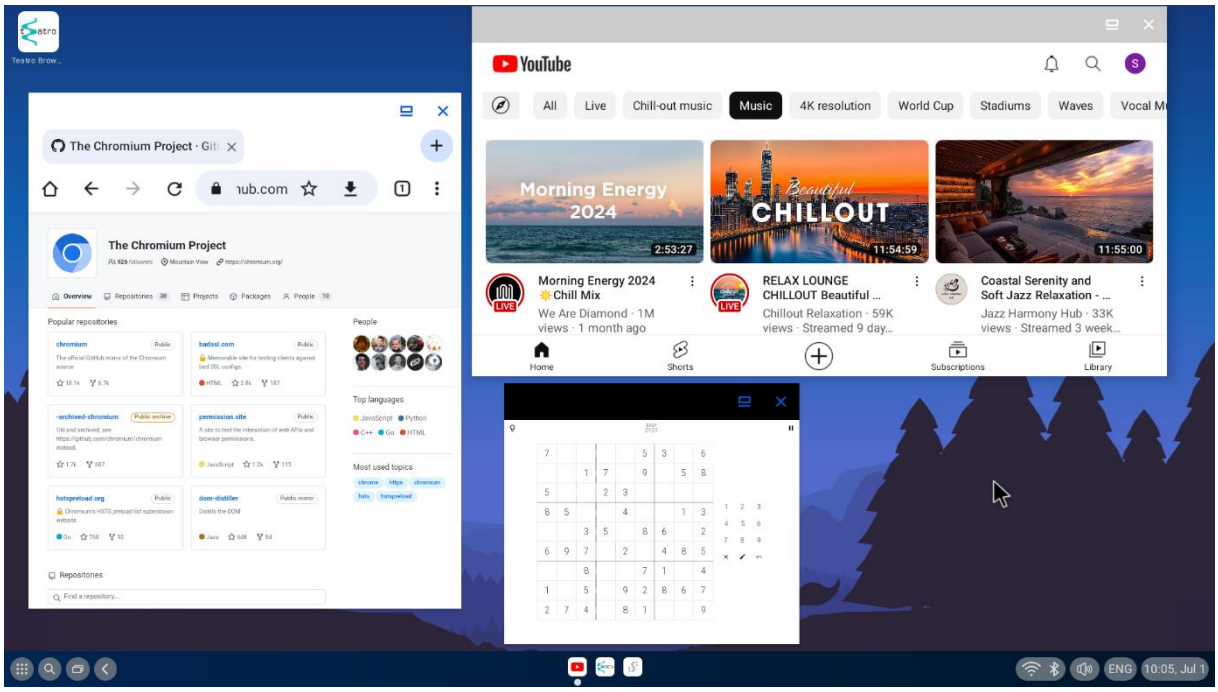
Додатно, све апликације ће подразумевано бити покренуте у режиму највеће могуће величине, а не у режиму слободне форме као што желимо. Могуће је променити подразумевани прозорни режим покретања апликације унутар `ActivityOptions` класе помоћу коришћења механизма рефлексије, а чија примена ће бити представљена касније у поглављу 4.3.

3.3 Неусаглашеност изгледа *DecorCaptionView* елемента

Корисничке и системске апликације подразумевано имају могућност контролисања изгледа сопственог *DecorCaptionView* елемента. Уколико апликације експлицитно не дефинишу његову боју, као и боју елемената на њему, систем ће сам позивати методе за аутоматски одабир палете на основу осталих боја које апликација користи. Ово често није случај и долази до проблема када метода која врши одабир позадинске боје

DecorCaptionView елемента се поклопи са бојом дугмића који су на њему, те их је у том случају готово немогуће приметити.

Решење примењено у овом раду искључује аутоматски одабир боје и уводи подразумеване боје за све апликације, као и API приступ корисничким апликацијама за промену подразумеване боје на жељену. На следећој слици су дате три различите апликације које имају неусаглашене *DecorCaptionView* компоненте.



Слика 3.3 Неусаглашеност *DecorCaptionView* компоненте на више апликација

3.4 Недостатак API за усликавање садржаја апликационих прозора

Већина Андроид уређаја нуди могућност приказивања скорашњих апликација (engl. *recent apps*) из посебног менија за увид и брзи приступ онима које су недавно коришћене или онима које су и даље покренуте у позадини. Апликације је кроз овај мени могуће довести у први план, односно *foreground* или их скроз затворити. Представљене су најчешће са малом сличицом, односно усликаним екраном на којем је приказан садржај (engl. *preview*), називом апликације, њеним логоом и пар опција за контролу.

Проблем са ТВ верзијом Андроид пројекта отвореног кода је што добављање снимка екрана прозора конкретне апликације враћа *null* повратну вредност, уместо конкретну инстанцу *Bitmap* објекта. У програмском решењу је приказана имплементација и неопходне измене за омогућавање усликавања екрана на ТВ базираним уређајима.

4. Програмско решење

4.1 Конфигурација и опције за подршку прозора у слободној форми

Подршка за више прозора је подразумевано укључена од *Андроид* 7.0 верзије, али се може и експлицитно искључити изменом `config_supportsMultiWindow` индикатора на `false` у `config.xml` датотеци уређаја од интереса. Уређаји са малим капацитетом РАМ меморије (engl. *low-RAM devices*), чија повратна вредност `ActivityManager.isLowRam()` методе враћа `true`, искључују подршку за више прозора, игноришући претходну конфигурацију.

Додатно, за омогућавање подршке прозора у слободној форми, неопходно је укључити `PackageManager#FEATURE_FREEFORM_WINDOW_MANAGEMENT` својство на путањи: `/android/frameworks/base/core/java/android/content/pm/PackageManager.java` , као и поставити `config_freeformWindowManagement` на `true`. Андроид и даље неће дозволити покретање апликације у режиму слободне форме док се не укључи и глобално својство `enable_freeform_support`. Постоји више начина за укључивање ове подршке, једни од њих су:

1. Коришћењем ADB алата и извршавањем команде:
   `adb shell settings put global enable_freeform_support true`
2. Кроз нативну Андроид апликацију за подешавања (engl. *Settings*) у секцији *Developer Options* под називом *Enable freeform windows*. Плоча на којој су вршене модификације не поседује *Settings* апликацију, већ као замену користи *TVSettings* апликацију прилагођену уређају који је коришћен за израду техничког дела

дипломског рада и она не нуди могућност контролисања ове опције, те је неопходно примењивати први начин.

4.2 Заобилазак рестрикција за спреге ван SDK

Корисничке апликације немају дозволе као и системске апликације, те су позиви које оне праве, као и приступи пољима унутар *SDK* – ограничени. Сваки позив методе или приступ пољу које је заштићено у *SDK* – доводи до бацања изузетка. На пример, *JVM* прати одакле и ко позива методу гледајући стек позива и зауставља се када је пронађена функција која се не подудара са одређеним филтером, као што може бити филтер за име пакета апликације. Исечак кода који омогућава заобилазак овог типе рестрикције је дат испод:

```
    try {
        Method forName = Class.class.getDeclaredMethod("forName", String.class);
        Method getDeclaredMethod = Class.class.getDeclaredMethod("getDeclaredMethod", String.class,
Class[].class);

        Class<?> vmRuntimeClass = (Class<?>) forName.invoke(null, "dalvik.system.VMRuntime");
        Method getRuntime = (Method) getDeclaredMethod.invoke(vmRuntimeClass, "getRuntime", null);
        Method setHiddenApiExemptions = (Method) getDeclaredMethod.invoke(vmRuntimeClass,
"setHiddenApiExemptions", new Class[]{String[].class});

        Object vmRuntime = getRuntime.invoke(null);
        setHiddenApiExemptions.invoke(vmRuntime, new Object[]{new String[]{"L"}});
    } catch (Throwable ignored) {}
```

4.3 Покретање апликације у режиму слободне форме

Метода за промену подразумеваног режима покретања апликације је сакривена у SDK, па је коришћена рефлексија како бисмо могли да поставимо апликацију у режим слободне форме чим се она покрене. Покретач апликација писан у *Котлин* програмском језику на следећи начин покреће апликацију у слободној форми дужине и ширине по 500px у горњем левом углу екрана:

```
val intent = ...
val FREEFORM_WINDOWING_MODE = 5
val activityOptions: ActivityOptions = ActivityOptions.makeBasic()

try {
    val method =
        ActivityOptions::class.java.getMethod(
            "setLaunchWindowingMode",
            Int::class.javaPrimitiveType
        )

    method.invoke(activityOptions, FREEFORM_WINDOWING_MODE)

    activityOptions.setLaunchBounds(0, 0, 500, 500)

} catch (ex: Exception) {}

intent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK)
launcherActivity.startActivity(intent, activityOptions.toBundle())
```

4.4 Имплементација дугмета за слање прозора у позадину

За додавање новог дугмета на врху прозора који је у режиму слободне форме, неопходно је предузети следеће кораке:

- Додати неопходне *XML* датотеке и векторе који дефинишу начин исцртавања и изглед унутрашње сличице дугмета за умањивање унутар:

  `android/frameworks/base/core/res/res/drawable/…`

- Регистровати *Button* унутар *XML* датотеке која дефинише изглед и распоред корисничких елемената унутар *DecorCaptionView* објекта:

```
<Button
      android:id="@+id/minimize_window"
      android:layout_width="32dp"
      android:layout_height="32dp"
      android:layout_margin="5dp"
      android:padding="4dp"
      android:layout_gravity="center_vertical|end"
      android:contentDescription="@string/..."
      android:background="@drawable/..." />
```

- Додати поље за ново дугме у виду *View* класе унутар *DecorCaptionView.java*

```
private View mCaption;
private View mContent;
private View mMaximize;
private View mMinimize;
private View mClose;
```

- Добавити инстанцу дугмета на основу претходно дефинисаног *android:id* поља
- Направити *Rect* објекат који чува димензије правоугаоника у ком се налази дугме за умањивање и регистровати га на неопходним местима у *DecorCaptionView* за детекцију притиска на њега

```
...
if (mMinimizeRect.contains(x, y - mRootScrollY)) {
   mClickTarget = mMinimize;
}
...
```

- Додати недостајуће ресурсе у осталим деловима радног слоја *Андроида*, а које је неопходно наводити како би правилно регистровали нови кориснички елемент у *SDK* делу *Андроида*
- Написати део кода који се извршава при притиску на дугме за умањивање, а унутар *DecorCaptionView* класе:

```
@Override
public boolean onSingleTapUp(MotionEvent e) {
      ...
      if (mClickTarget == mMinimize) {
            Context context = getContext();
            if (context instanceof Activity) {
                  Activity activity = (Activity) context;
                  return activity.moveTaskToBack(true);
            }
      }
      ...
}
```

4.5 Имплементација *maximize* дугмета за контролу прозора

Дугме за увећавање прозора на највећу могућу величину не додајемо у потпуности као у случају дугмета за слање прозора у позадину, већ прерађујемо функционалност претходно постојећег дугмета за пребацивање у *fullscreen* режим.

Унутар *DecorCaptionView* класе уклањамо функционалност за пребацивање између режима слободне форме и режима највеће могуће величине и као замену користимо *Intent* систем комуникације да обавестимо покретач апликација о догађају притиска на махимизе дугме, препуштајући покретачу да ослушкује овај догађај и сходно њему мења величину прозора.

```
public static final String ACTION_CAPTION_BUTTON = "android.intent.action.CAPTION_BUTTON";
public static final String ACTION_TYPE = "type";

...

@Override
public boolean onSingleTapUp(MotionEvent e) {

	Context context = getContext();
	Activity activity = null;

	if (context instanceof Activity) {
		activity = (Activity) context;
	}

	Intent clickedIntent = new Intent(ACTION_CAPTION_BUTTON);
	clickedIntent.putExtra("package_name", context.getPackageName());

	if (activity != null) {
		clickedIntent.putExtra("task_id", activity.getTaskId());
	}
	...
	if (mClickTarget == mMaximize) {
		toggleFreeformWindowingMode();
		clickedIntent.putExtra(ACTION_TYPE, "maximize");
	}
	...
}
```

У *Intent* објекат пакујемо и идентификатор радног задатка под кључем `task_id`, како би покретач апликација касније могао да зна која апликација захтева промену величине прозора.

Покретач апликација ослушкује пристигле *Intent* објекте и у зависности од њих, помоћу рефлексије приступа *ActivityTaskManager* класи и мења величину прозора на следећи начин:

```
const val RESIZE_TASK_METHOD_NAME = "resizeTask"

fun resizeWindow(context: Context, taskId: Int, newWindowRect: Rect?) {

    val taskManager = context.getSystemService(ACTIVITY_TASK_SERVICE)
    val method = taskManager::class.java.getMethod(
        RESIZE_TASK_METHOD_NAME, Integer::class.javaPrimitiveType, Rect::class.java
    )

    method.invoke(taskManager, taskId, newWindowRect)
}
```

У случају да је апликација већ у *maximized* режиму, очекивано понашање је да поновним притиском на исто дугме, апликација враћа димензије које је имала пре преласка у то стање. Следи исечак кода у *Котлин* програмском језику који показује како би изгледала ова имплементација:

```
val previousBounds = mutableMapOf<Int, Rect>()
...
override fun onReceive(context: Context?, intent: Intent?) {
  ...

  val taskId = intent.getIntExtra("task_id", UNKNOWN_TASK_ID)

  if (taskId == UNKNOWN_TASK_ID)
      return

  lateinit var resizeTo: Rect

  if (Util.isWindowMaximized(context, taskId)) {
      resizeTo = previousBounds.getOrDefault(taskId, Freeform.getFreeformRect())
  } else {
      previousBounds.put(taskId, Util.getWindowBounds(context, taskId))
      resizeTo = Util.getMaximizedBounds(context)
  }

  Util.resizeWindow(context, taskId, resizeTo)
  ...
}
```

4.5.1 Покретање *maximized* режима на дупли клик

Вођени искуствима других оперативних система, имплементирана је и опција за улазак у *maximized* режим и помоћу дуплог клика на *DecorCaptionView* објекат прозора. Овај приступ такође користи *Intent* систем комуникације да након детекције дуплог клика обавести покретач апликација о наведеном догађају како би он могао да сходно њему промени димензије прозора у одговарајуће. Слање овог догађаја се одвија у методи `notifyCaptionDoubleclicked()`.

Неопходно је разликовати једноструки од двоструког клика, као и проверавати временска ограничења између два узастопна клика. Поређењем корисничког искуства и претходног руковања другим оперативним системима, дупли клик се сматра валидним ако је временски интервал између два узастпона клика био мањи или једнак од 700ms. У наставку је дат исечак *Јава* кода и модификација унутар *DecorCaptionView* класе:

- Нова класа-ослушкивач за догађаје који су типа дуплог клика:

```
private class DoubleTapListener implements GestureDetector.OnDoubleTapListener {
  private boolean mIsNotPrimaryButtonClicked = false;
  ...

  @Override
  public boolean onDoubleTapEvent(MotionEvent e) {
    mClickCount = 0;
    mDuration = 0;
    switch (e.getAction()) {
      case MotionEvent.ACTION_DOWN:
        mIsNotPrimaryButtonClicked = e.getButtonState() != MotionEvent.BUTTON_PRIMARY;
        break;
      case MotionEvent.ACTION_UP:
        if (mClickTarget == null && !mIsNotPrimaryButtonClicked) {
           notifyCaptionDoubleClicked();
        }
        break;
    }
    return false;
  }
}
```

- Модификација обрађивача догађаја за једноструке кликове `onSingleTapUp`

```
mClickCount++;
if (mClickCount == 1) {
      mStartTime = System.currentTimeMillis();
}
if (mClickCount == 2) {
      long time = System.currentTimeMillis() - mStartTime;
      mDuration = mDuration + time;
      if (mDuration <= MAX_DURATION) {
            if (mClickTarget == null) {
                  notifyCaptionDoubleClicked();
            }
            mClickCount = 0;
      } else {
            mClickCount = 1;
            mStartTime = System.currentTimeMillis();
      }
      mDuration = 0;
}
```

- Регистрација ослушкивача за догађаје дуплог клика унутар `init` методе:

```
mGestureDetector.setOnDoubleTapListener(new DoubleTapListener());
```

- Позивање догађаја при притиску на једноструки клик унутар `onTouch` методе:

```
mGestureDetector.onTouchEvent(e);
```

4.6 Имплементација *Back* дугмета по прозору

За имплементацију овог дугмета можемо пратити исте кораке као и за *minimize* дугме:

- Додати неопходне *XML* датотеке и векторе који дефинишу начин исцртавања и изглед унутрашње сличице дугмета за умањивање унутар:

  `android/frameworks/base/core/res/es/drawable/…`

- Регистровати *BUTTON* унутар *XML* датотеке која дефинише изглед и распоред корисничких елемената унутар *DecorCaptionView* објекта:

```
<Button
    android:id="@+id/back_window"
    android:layout_width="32dp"
    android:layout_height="32dp"
    android:layout_margin="5dp"
    android:padding="4dp"
    android:layout_gravity="center_vertical|start"
    android:contentDescription="@string/..."
    android:background="@drawable/..." />
```

- Додати поље за ново дугме у виду *View* класе унутар *DecorCaptionView.java*

```
private View mCaption;
private View mContent;
private View mMaximize;
private View mMinimize;
private View mBack;
private View mClose;
```

- Добавити инстанцу дугмета на основу претходно дефинисаног *android:id* поља
- Направити *Rect* објекат који чува димензије правоугаоника у ком се налази дугме за враћање уназад и регистровати га на неопходним местима у *DecorCaptionView* за детекцију притиска на њега

```
...
if (mBackRect.contains(x, y - mRootScrollY)) {
   mClickTarget = mBack;
}
...
```

- Додати недостајуће ресурсе у осталим деловима радног слоја *Андроида*
- Написати део кода који се извршава при притиску на *Back* опцију:

```
@Override
public boolean onSingleTapUp(MotionEvent e) {
      ...
      if (mClickTarget == mBack) {
            Context context = getContext();
            if (context instanceof Activity) {
                  Activity activity = (Activity) context;
                  return activity.onBackPressed();
            }
      return true;
      }
}
```

4.7 Имплементација пречице за режим највеће могуће величине

Извршили смо пренамену опције за режим највеће могуће величине тако да иако проширује апликациони прозор и његову величину у максималне димензије, она и даље задржава слободну форму, без уклањања *DecorCaptionView* елемента. У неким случајевима, често када корисник гледа неки видео садржај или жели бити фокусиран на само једну апликацију, *DecorCaptionView* елемент је сувишан. У изради решења је понуђена опција да се корисник помоћу *F11* тастера на тастатури пребацује између режима слободне форме и режима највеће могуће величине. Пример исечка кода које је коришћено као решење је дато у наставку:

- Модификација `dispatchKeyEvent` методе унутар *DecorView* класе:

```
if (keyCode == KeyEvent.KEYCODE_F11 && isDown) {
  final Window.WindowControllerCallback callback =
    mWindow.getWindowControllerCallback();
  final int windowingMode =
      getResources().getConfiguration().windowConfiguration.getWindowingMode();
  if ((windowingMode == WINDOWING_MODE_FREEFORM || windowingMode ==
WINDOWING_MODE_FULLSCREEN)&& callback != null) {
  try {
      callback.toggleFreeformWindowingMode();
    } catch(RemoteException e) {
      return true;
    }

  }
  ...
}
```

Метода `toggleFreeformWindowingMode`, када се позове, пребацује апликациони прозор у режим слободне форме уколико је она претходно била у *fullscreen* режиму и обрнуто. За побољшање корисничког искуства, најбоље је обавестити корисника о начину изласка из *fullscreen* режима, на пример приказивањем тзв. *Toast* поруке, јер након што корисник уђе у овај режим, нису му доступне друге опције покретача за ванредни излазак или контролу других апликација.

4.8 Проширење API за промену подразумеване боје *DecorCaptionView* елемента и опција на њему

- Проширење *ActivityTaskManager* класе за складиштење позадинске боје *DecorCaptionView* елемента и опција на њему:

```
private static String WINDOW_CAPTION_COLOR = "#00306E";
private static String WINDOW_CAPTION_CONTENT_COLOR = "#FFFFFF";
...
@NonNull
public static String getWindowCaptionContentColor() {
  return WINDOW_CAPTION_CONTENT_COLOR;
}

public static void setWindowCaptionColor(@NonNull String color) {
  WINDOW_CAPTION_COLOR = color;
}

public static void setWindowCaptionContentColor(@NonNull String color) {
  WINDOW_CAPTION_CONTENT_COLOR = color;
}
```

- Инстанцирање ослушкивача за случај када корисничка апликација захтева промену подразумеване боје *DecorCaptionView* елемента и њено ажурирање унутар *DecorView* класе:

```
private BroadcastReceiver captionReceiver = new BroadcastReceiver() {
  @Override
  public void onReceive(Context context, Intent intent)
  {
    ActivityTaskManager.setWindowCaptionColor(intent.getStringExtra("..."));
    ActivityTaskManager.setWindowCaptionContentColor(intent.getStringExtra("..."));
    updateColorViews(null /* insets */, false);
  }
};
```

- Модификације `updateColorViews` методе ради ажурирања подразумеване боје *DecorCaptionView* елемента и опција на њему:

```
...
if (mHasCaption) {
  final int captionColor = calculateStatusBarColor();
  mDecorCaptionView.getCaption().setBackgroundColor(captionColor);

  int color = Color.parseColor(ActivityTaskManager.getWindowCaptionColor());
  mDecorCaptionView.getCaption().setBackgroundColor(color);

  updateDecorCaptionShade();
}
...
```

4.9 Имплементација заобљених углова апликационих прозора

- Инстанцирање *ViewOutlineProvider* објекта и његова конфигурација за каснију примену над *DecorView* елементом:

```
private float mWindowCornerRadius = 8;
private ViewOutlineProvider mWindowOutline = new ViewOutlineProvider() {
  @Override
  public void getOutline(View view, Outline outline) {
    outline.setRoundRect(0, 0, view.getWidth(), view.getHeight(), mWindowCornerRadius);
  }
};
```

- Дефиниција методе за ажурирање и примену *ViewOutlineProvider* објекта над постојећим *View* објектом прозора:

```
private void updateWindowCorner() {
  if (mDecorCaptionView == null) {
    setClipToOutline(false);
    setOutlineProvider(null);
  } else {
    setOutlineProvider(mWindowOutline);
    setClipToOutline(true);
  }
}
```

- Позивање претходно дефинисане `updateWindowCorner` методе како би се углови прозора ажурирали и остали перзистентни при осталим транзицијама које се одигравају над апликационом прозору. Методу је неопходно позвати унутар метода:
  - `setWindowBackground`
  - `updateBackgroundDrawable`
  - `setWindow`
  - `updateDecorCaptionStatus`
  - `onResourceLoaded`

4.10 Имплементација API за снимке екрана покренутих апликација

Узевши у обзир да ТВ варијација *AOSP* решења не усликава аутоматски прозоре покренутих апликација, неопходно је надоградити класу *TaskSnapshotController* са методом чијим ће позивом она то радити по потреби:

```
boolean takeSnapshotCandidate(Task task) {
  if (shouldDisableSnapshots()) {
    return true;
  }

  if (task == null || !task.isVisible()) {
    return false;
  }

  mTmpTasks.clear();
  mTmpTasks.add(task);
  snapshotTasks(mTmpTasks);

  return true;
}
```

- Променити услов који претходно није дозвољавао систему да хвата снимке екрана услед тога што је циљна платформа намењена ТВ уређајима:
  - Пре:

```
private boolean shouldDisableSnapshots() {
  return mIsRunningOnWear || mIsRunningOnTv || mIsRunningOnIoT;
}
```

  - После:

```
private boolean shouldDisableSnapshots() {
  return mIsRunningOnWear || mIsRunningOnIoT;
}
```

- Имплементирати методу унутар *Task* класе која позива методу за усликавање прозора апликације:

```
public void takeTaskSnapshotForce() {
  mAtmService.mWindowManager.mTaskSnapshotController.takeSnapshotCandidate(this);
}
```

- Позвати је унутар `getSnapshot` методе:

```
ActivityRecord topActivity = getTopNonFinishingActivity();

if (topActivity != null && topActivity.inFreeformWindowingMode() &&
topActivity.getState() == ActivityStack.ActivityState.RESUMED) {
  takeTaskSnapshotForce();
}
```

5. Резултати

Модификација *DecorCaptionView* компоненте

На следећим сликама је постепено издвојено унапређивање *DecorCaptionView* компоненте и опција на њој:

Без додатних опција

Присутне су опције за затварање прозора и његово пребацивање у режим највеће могуће величине:

Слика 5.1 Изглед *DecorCaptionView* компоненте без додатних опција

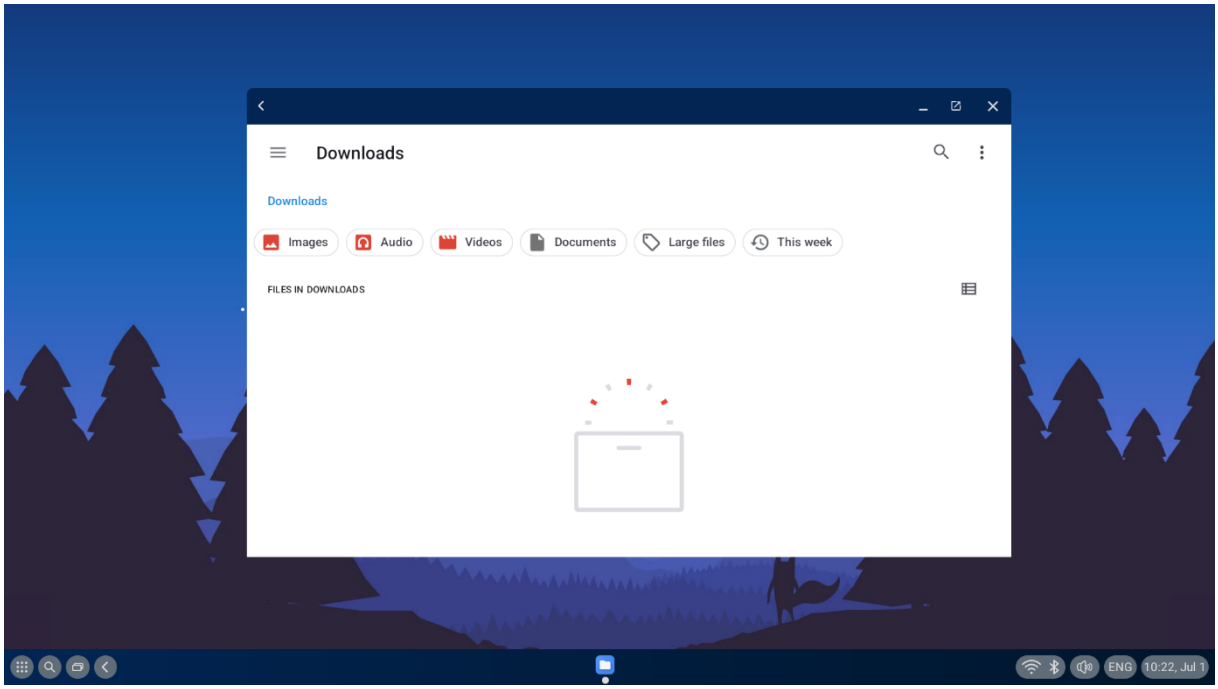
Са додатком *minimize* опције

Слика 5.2 Изглед *DecorCaptionView* компоненте са *minimize* опцијом

Са додатком *back* опције

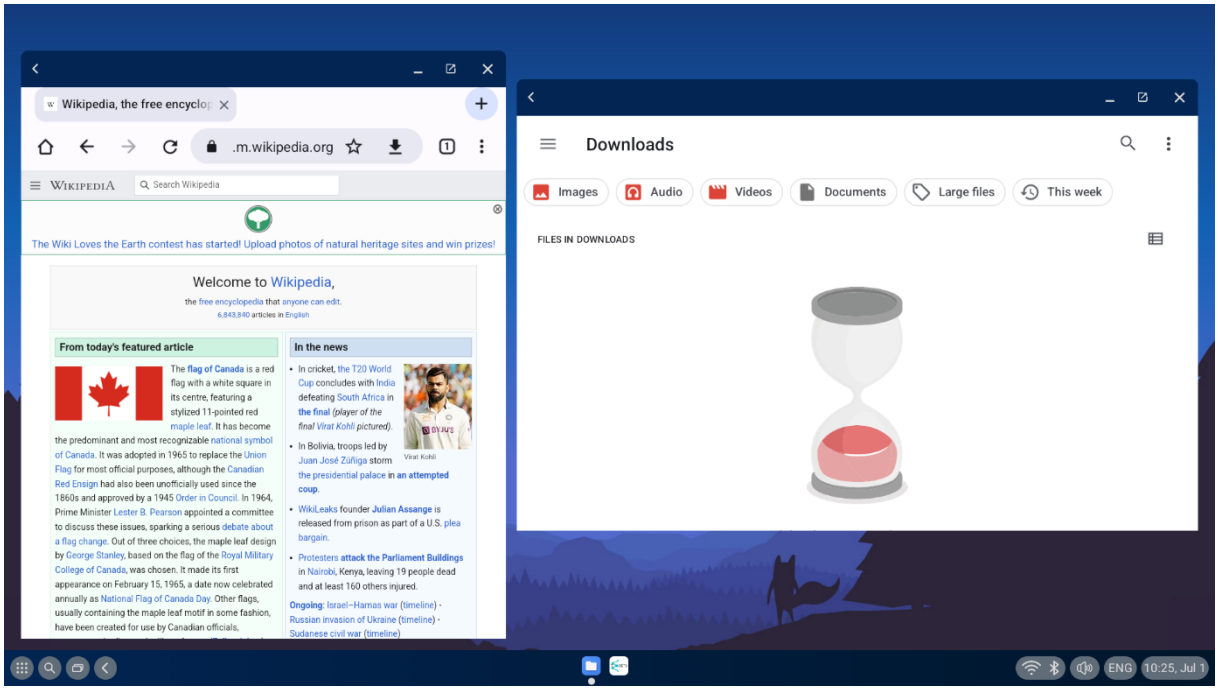
Слика 5.3 Изглед *DecorCaptionView* компоненте са *back* опцијом

Изглед апликације са заобљеним угловима и садржајем



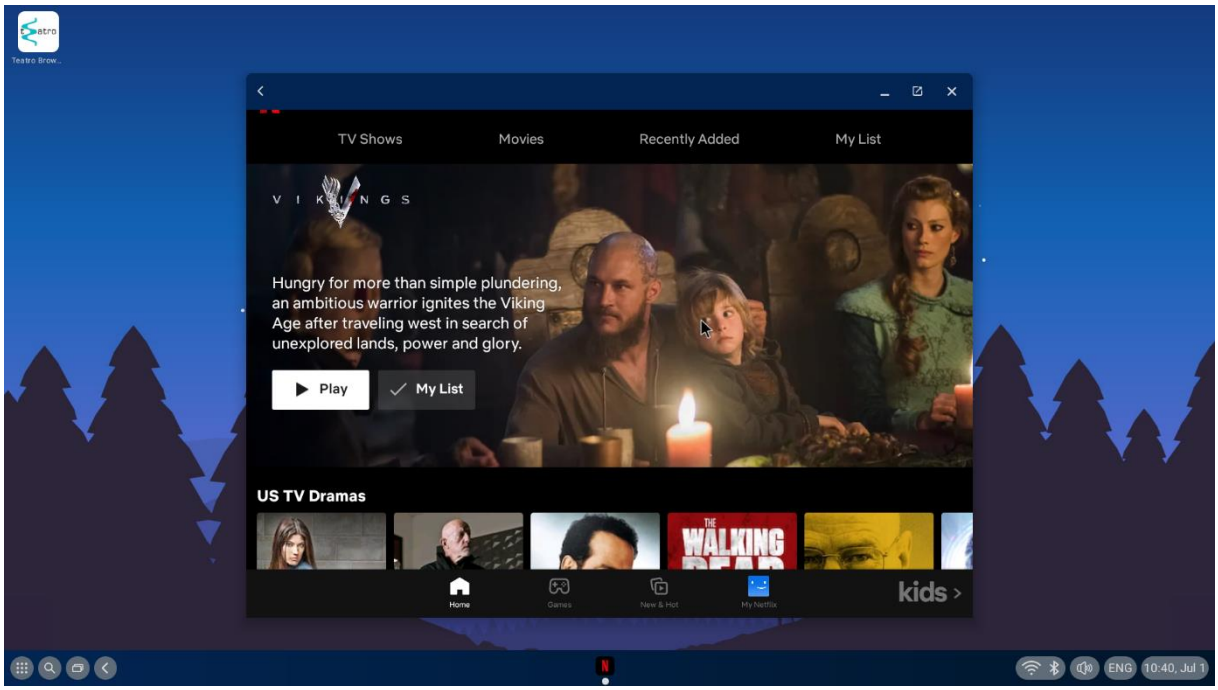
Слика 5.4 Изглед апликације у режиму слободне форме са заобљеним угловима

Усаглашеност боја *DecorCaptionView* компоненте међу свим покренутим апликацијама



Слика 5.5 Различите апликације са истим бојама *DecorCaptionView* компоненте

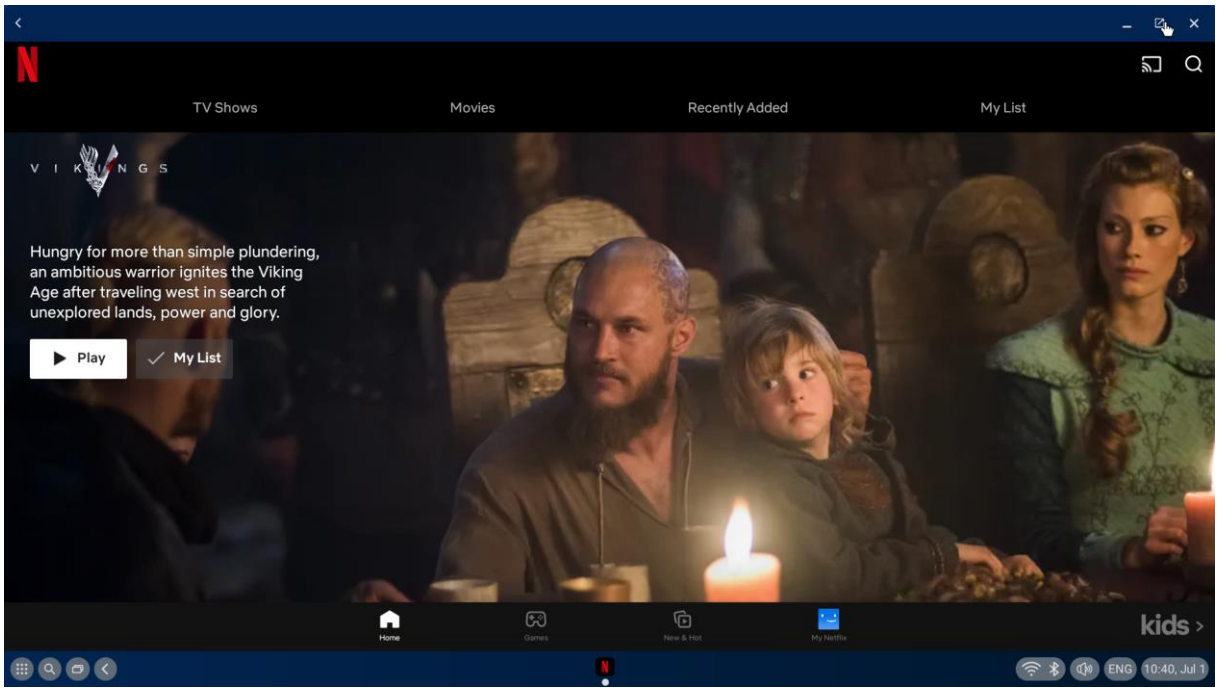
Апликација у слободној форми са садржајем у режиму слободне форме пре пребацивања у *maximized* режим



Слика 5.6 Изглед корисничке апликације пре транзиције у *maximized* режим

Апликација у слободној форми са садржајем у *maximized* режиму

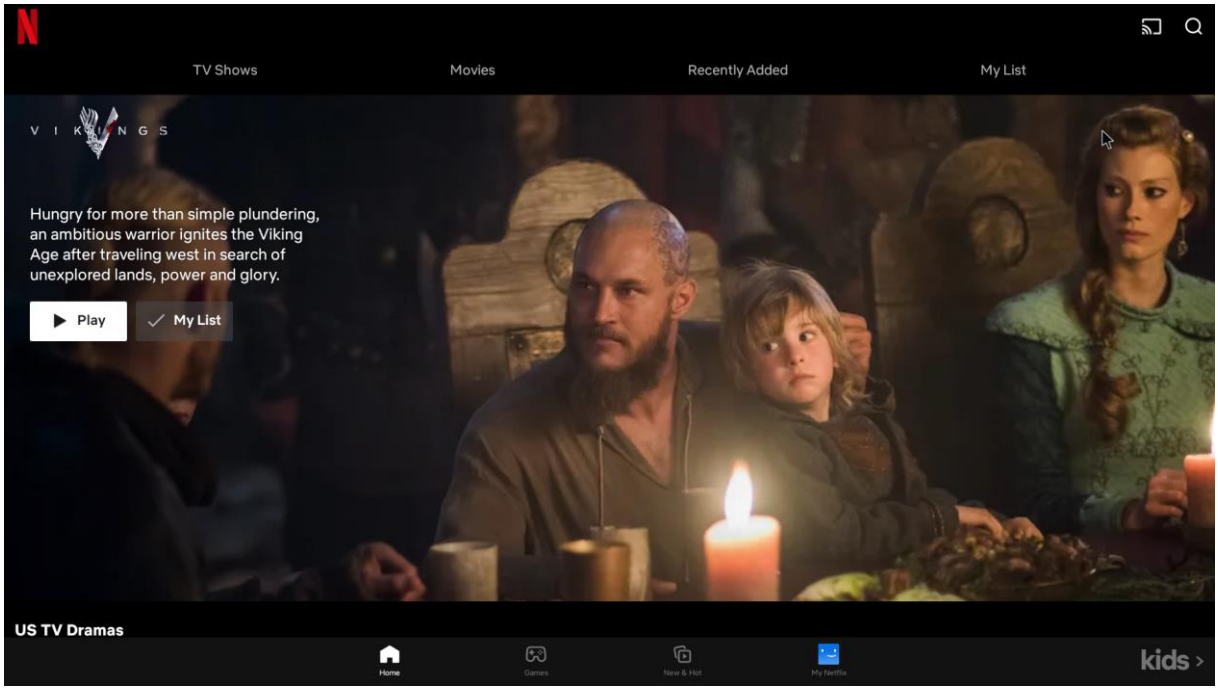
На слици у наставку се може приметити како апликација заузима максимални доступни простор, притом остављајући довољно места за *DecorCaptionView* компоненту, као и палету радних задатака (engl. *taskbar*)



Слика 5.7 Изглед корисничке апликације након транзиције у *maximized* режиму

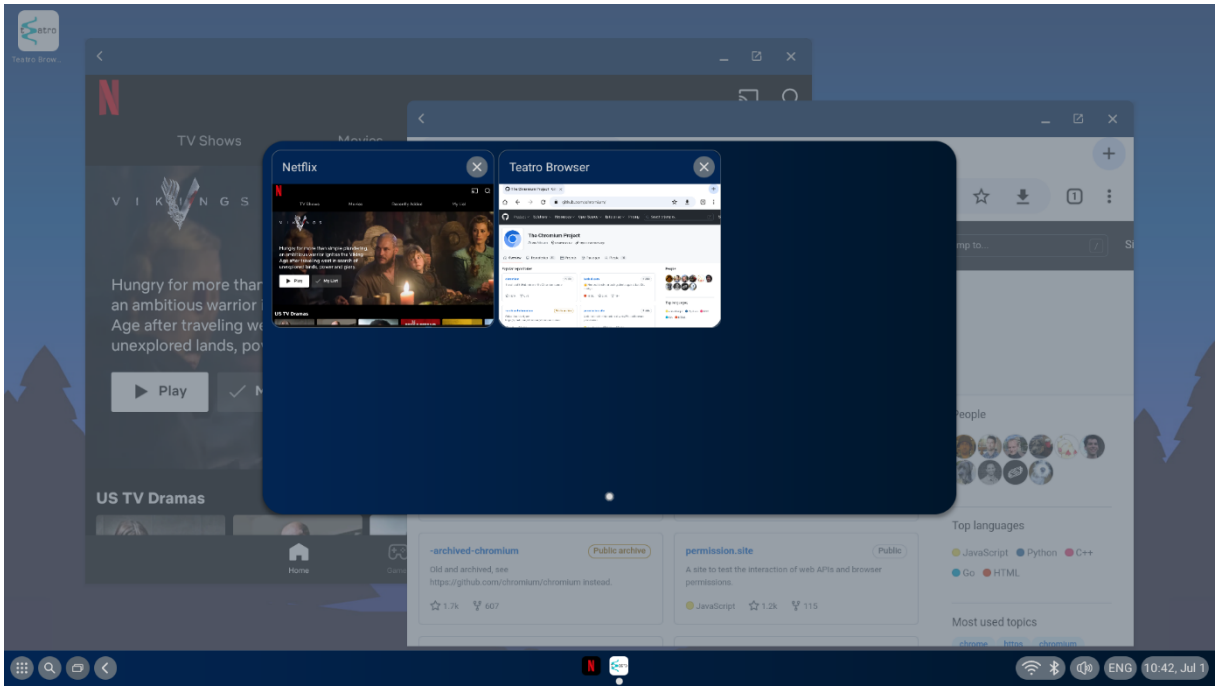
Апликација у режиму највеће могуће величине без *DecorCaptionView* компоненте

Апликација сада заузима простор целог екрана на којој се приказује без додатних елемената у виду *DecorCaptionView* компоненте или корисничких елемената који потичу од покретача апликација



Слика 5.8 Изглед корисничке апликације у *fullscreen* режиму

Визуелни приказ усликаних прозора покренутих апликација



Слика 5.9 Мени покренутих задатака са усликаним садржајима прозора

5.1 Техничка спецификација циљне платформе са *Андроид ОС*

При изради техничког дела коришћене су *AMLogic* STB плоче, односно конкретни модели са главним чипом S905Y4, као и S905X4, такође редом познатији и под кодним именима: *Oppen* и *Ohm*.

Оперативни систем	Андроид 11.0
CPU	Четворојезгарни ARM Cortex-A35
GPU	ARM Mali-G31 MP2
Капацитет меморије	EMMC: 16GB, DDR4: 2GB
Меморија	32-bit, DDR3, DDR4, DDR3L, LPDDR3/4
Видео декодирање	AVI, X.265, VP9, AVS-P кодеци до 4K резолуције при 60 кадрова у секунди
Видео кодирање	S905X4 подржава X.265 и JPEG кодирање до 1080p при 60 кадрова у секунди, док S905Y4 користи софтверске кодере
HDMI-TX	Подржавају 4K резолуцију при 60Hz
AV излаз	CVBS

Табела 2 - Техничке спецификације циљне платформе коришћене за израду

5.2 Прилагођење решења и подршка других платформи и верзија

Поред претходно наведеног решења, такође је успешно прилагођено и покренуто на наредним платформама и верзијама *Андроида*:

Платформа	**Андроид верзија**
AMLogic	14.0
Haier	14.0
Realtek-TV	11.0
Realtek-STB	11.0

Табела 3 - Остале подржане платформе и верзије Андроида

6. Закључак

У овом раду је успешно приказана имплементација за покретање апликација у режиму слободне форме са додатним опцијама неопходним за подршку коришћења овог решења уз спољашње периферијске уређаје као што су рачунарски миш и тастатура. Дат је увид у заштите и препреке које *Андроид* оперативни систем ставља између системског и корисничког дела, као и механизми које можемо користити за њихово заобилажење ради остваривања додатних могућности које нису предвиђене за регуларну употребу. Израда дипломског рада је првенствено вршена на верзији *Андроида* 11.0, али разни експериментални покушаји и знатижеља су довели до успешног прилагођавања овог решења и на верзију *Андроида* 14.0 (*Upside Down Cake)*.

Прativши напредак *Андроида* након верзије 11.0, види се знатни помак и жеља за увођењем редовне подршке апликација у режиму слободне форме. Иако постоје уређаји који више не добијају редовна ажурирања од својих произвођача на новије верзије или из одређених разлога не прате даљи развој, они ће моћи веома лако да надограде тренутне верзије са минималним скупом закрпи и притом унапредити своје TV или STB уређаје за вишенаменску употребу. Један уређај би имао могућност пребацивања између 2 режима:

- Забавни (engl. *entertainment*)

  Режим у ком корисници конзумирају *online* или *offline video* и *audio* садржај као што тренутно нуде њихови TV уређаји у својим домовима

- Продуктивни (engl. *productivity*)

  Режим у ком је могуће покренути више апликација истовремено из скупова продуктивних (*Word*, *Excel*, ...), конференцијских (*Teams*, *Zoom*, *Google Meet*), претраживачких (*Web browser*) и сл.

7. Литература

[1] „*Android operating system”*, мај 2024.
Веза: https://en.wikipedia.org/wiki/Android_(operating_system)

[2] „*Android TV”*, мај 2024.
Веза: https://developer.android.com/training/tv

[3] „*Android – Multi-window support”*, јун 2024.
Веза: https://developer.android.com/guide/topics/large-screens/multi-window-support

[4] „*Android 15 – Improved desktop mode experience”*, јун 2024.
Веза: https://www.androidauthority.com/android-15-desktop-mode-demo-3430991/, jun 2024.

[5] „*Multi-window environment”*, јун 2024.
Веза: https://source.android.com/docs/core/display/multi-window

[6] „*Restrictions on NON-SDK interfaces”*, јун 2024.
Веза: https://developer.android.com/guide/app-compatibility/restrictions-non-sdk-interfaces

[7] „*Improving Stability by Reducing Usage of non-SDK Interfaces”*, јун 2024.
Веза: https://android-developers.googleblog.com/2018/02/improving-stability-by-reducing-usage.html

[8] „*Android permission element”*, јун 2024.
Веза: https://developer.android.com/guide/topics/manifest/permission-element

[9] „*JVM dependencies”*, јун 2024.
Веза: https://kotlinlang.org/docs/reflection.html#jvm-dependency

[10] *Nikolay Elenkov - Android Security Internals An In-Depth Guide to Android's Security Architecture, 2014*

[11] „*Java reflections”*, јун 2024

Веза: https://www.oracle.com/technical-resources/articles/java/javareflection.html

[12] „*Java method reflections”*, јун 2024.

Веза: https://www.baeldung.com/java-method-reflection

[13] „*Tasks and back-stack”*, јун 2024.

Веза: https://developer.android.com/guide/components/activities/tasks-and-back-stack