**Дніпровський ліцей інформаційних технологій**

**при Дніпропетровському національному університеті**

**імені Олеся Гончара**

**Випускна робота**

**на тему:**

**«Демонстрація екрану Android-пристрою на екрані іншого Android-пристрої»**

**Виконав:**

**ліцеїст 11-Г класу Мунтян Даниїл**

**Науковий керівник:**

**Ентін Й. А. \_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Дніпро**

**2017**

**ЗМІСТ**

[Вступ 2](#_Toc499899227)

[Теоретична частина 3](#_Toc499899228)

[Веб-сервіс GitHub 3](#_Toc499899229)

[Обмін даними 4](#_Toc499899230)

[Проблема з доступом до IP-адреси та її рішення. 7](#_Toc499899231)

[Порівняння TCP та UDP протоколів 9](#_Toc499899232)

[TCP-порти 13](#_Toc499899233)

[Реалізація Android клієнту 15](#_Toc499899234)

[Android Studio 15](#_Toc499899235)

[Багатопотоковість на клієнті 15](#_Toc499899236)

[Реалізація TCP-клієнту на Java Android. Ініціалізація 19](#_Toc499899237)

[Огляд класу InetAddress 20](#_Toc499899238)

[Класи ObjectInputStream та ObjectOutputStream. 20](#_Toc499899239)

[Ініціалізація екземпляру класу Socket 21](#_Toc499899240)

[Реалізація трансляції екрану 22](#_Toc499899241)

[Алгоритм Нейгла 22](#_Toc499899242)

[Опис програми. Напрямки використання 23](#_Toc499899243)

[Кнопка SHARE 24](#_Toc499899244)

[Кнопка WATCH 25](#_Toc499899245)

[Кнопка SETTINGS 25](#_Toc499899246)

[Кнопка HELP 26](#_Toc499899247)

[Опис роботи 27](#_Toc499899248)

[Структура проекту 30](#_Toc499899249)

[Вимоги програмного забезпечення 31](#_Toc499899250)

[Висновки 31](#_Toc499899251)

[Джерела інформації 32](#_Toc499899252)

[Додаток 33](#_Toc499899253)

Вступ

**Актуальність:** практично в кожної людини є смартфон, і кожен день ми користуємось нашими девайсами, користуємось інтернетом. Android-пристрої широко розповсюджені. Тому демонстрація екрану свого пристрою на далеку відстані по інтернету може бути корисною або необхідною, до того ж, технологія клієнт-сервер дає змогу підключити до трансляції не тільки один Android-пристрій, а й декілька.

**Метою** **роботи** є створення корисного додатку на Android-пристрій, який дозволить нам ділитися екраном свого смартфону. Також ми переслідуємо особисті цілі: вивчення нової мови програмування Java для Android систем та освоєння клієнт-серверної архітектури програми, використання веб-сервісу GitHub для спільної роботи над проектом клієнту та серверу, освоєння середовища розробки Android Studio та IntelliJ Idea, освоєння багатопотоковості – для обробки декількох клієнтів одночасно.

Головні задачі випускної роботи.

1. Створення серверної частини програми, обробити об’єкт, який було відправлено з клієнту(ів).
2. Створення клієнтської частини (на Android), яка відправляє запити на сервер та отримує інформацію, яку надає користувачу на екран.
3. Створення багатопоточної системи на сервері – бо сервер повинен обробляти відразу декількох клієнтів, які до нього підключились.
4. Створення інтерфейсу програми, який був би зручним для користувача.
5. Освоєння веб-сервісу GitHub задля спільної роботи над одним й тим ж проектом.

Теоретична частина

Веб-сервіс GitHub

Від початку роботи з нашим проектом ми шукали можливість роботи над ним разом. Одним із найзручніших способів такої командної роботи є веб-сервіс GitHub.

GitHub — один з найбільших веб-сервісів для спільної розробки програмного забезпечення. Існують безкоштовні та платні тарифні плани користування сайтом. Базується на системі керування версіями Git і розроблений на Ruby on Rails і Erlang компанією GitHub, Inc (раніше Logical Awesome).

Сервіс безкоштовний для проектів з відкритим вихідним кодом, з наданням користувачам усіх своїх можливостей (включаючи SSL), а для окремих індивідуальних проектів пропонуються різні платні тарифні плани. Сайт має елементи соціальної мережі — крім розміщення коду розробники можуть спілкуватися, коментувати редагування один одного, стежити за новинами знайомих. Можна об'єднувати різні репозиторії, в тому числі такі, що мають різних власників.

Для проектів є персональні сторінки, Вікі та система відстежування помилок. На сайті можна переглядати файли проектів, реалізовано підсвічування синтаксису для більшості мов програмування. Код проектів можна отримати не тільки через Git, а й завантажити звичайний архівний файл.

Ми створили групу «[Android Runners](https://github.com/Android-Runners/)» на GitHub та зберігали там наші проекти - як серверної частини так і клієнтської. Для цього ми користувалися «гілками» (Branch).

Обмін даними

Для передачі даних між пристроями (у нашому випадку – cервером на комп’ютері та Android-смартфоном) ми використовували один із протоколів передавання даних.

Протокол передавання даних — набір угод інтерфейсу логічного рівня, які визначають обмін даними між різними програмами. Ці угоди задають однаковий спосіб передачі повідомлень і обробки помилок при взаємодії програмного забезпечення рознесеного на просторі апаратної платформи, з'єднаної інтерфейсом.

Стандартизований протокол передачі даних також дозволяє розробляти інтерфейси (вже на фізичному рівні), не прив'язані до конкретної апаратної платформи і виробнику (наприклад, USB, Bluetooth).

Мережний протокол — набір правил, що дозволяє здійснювати з'єднання і обмін даними між двома і більше включеними в мережі пристроями.

Різні протоколи, найчастіше, описують лише різні сторони одного типу зв'язку. Взяті разом, вони утворюють стек протоколів. Назви «протокол» і «стек протоколів» також вказують на програмне забезпечення, яким реалізується протокол.

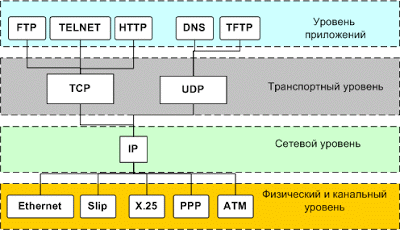
Найпоширенішою системою класифікації мережних протоколів є модель OSI, відповідно до якої протоколи поділяються на сім рівнів за своїм призначенням — від фізичного(формування і розпізнавання електричних або інших сигналів) до прикладного (інтерфейс програмування додатків для передачі інформації додатками).

Стек протоколів TCP / IP — це два протоколи нижнього рівня, що є основою зв'язку в мережі Інтернет. Протокол TCP (Transmission Control Protocol) розбиває передану інформацію на порції і нумерує всі порції. За допомогою протоколу IP (Internet Protocol) всі частини передаються одержувачу. Далі за допомогою протоколу TCP перевіряється, чи всі частини отримані. При отриманні всіх порцій TCP розміщує їх в потрібному порядку і збирає в єдине ціле.

Найвідоміші протоколи, використовувані в мережі Інтернет.

* HTTP (Hyper Text Transfer Protocol) — протокол передачі гіпертексту. Використовується при пересиланні Web-сторінок між комп’ютерами.
* FTP (File Transfer Protocol) — протокол передачі файлів зі спеціального файлового сервера на комп'ютер користувача. FTP дає можливість абоненту обмінюватися двійковими і текстовими файлами з будь-яким комп'ютером мережі.
* POP (Post Office Protocol) — стандартний протокол поштового з'єднання.
* SMTP (Simple Mail Transfer Protocol) — протокол, який задає набір правил для передачі пошти.
* Uucp (Unix to Unix Copy Protocol) — це нині застарілий, але все ще застосовується протокол передачі даних, у тому числі для електронної пошти.
* Telnet — протокол віддаленого доступу. Надає можливість абоненту працювати на будь-якій ЕОМ мережі Інтернет, як на власній, тобто запускати програми, змінювати режим роботи і т. д.
* DTN — протокол для забезпечення наддалекого космічного зв'язку.

Нижче представлено стек протоколів TCP/IP.



Обмін даними в нашому проекті реалізовано через класи Socket та ServerSocket на Java.

Сокети (англ. socket - заглиблення, гніздо, роз'єм) — назва програмного інтерфейсу для забезпечення обміну даними між процесами. Процеси при такому обміні можуть виконуватися як на одній ЕОМ, так і на різних ЕОМ, пов'язаних між собою мережею. Сокет - абстрактний об'єкт, що представляє кінцеву точку з'єднання.

Слід розрізняти клієнтські і серверні сокети. Клієнтські сокети грубо можна порівняти з кінцевими апаратами телефонної мережі, а серверні - з комутаторами. Клієнтський додаток (наприклад, браузер) використовує лише клієнтські сокети, а серверний (наприклад, веб-сервер, якому браузер посилає запити) - як клієнтські, так і серверні сокети.

Інтерфейс сокетів вперше з'явився в BSD Unix. Програмний інтерфейс сокетів описаний в стандарті POSIX.1 І в тій чи іншій мірі підтримується усіма сучасними операційними системами.

Кожен процес може створити слухаючий сокет (серверний сокет) і прив'язати його до якогось порту операційної системи (в UNIX непривілейовані процеси не можуть використовувати порти менше 1024). Процес, що слухає, зазвичай знаходиться в циклі очікування, тобто прокидається при появі нового з'єднання. При цьому зберігається можливість перевірити наявність з'єднань у цей час, встановити тайм-аут для операції тощо.

ОС сімейства UNIX можуть підтримувати багато типів адрес, але обов'язковими є INET-адреса і UNIX-адреса. Якщо прив'язати сокет до UNIX-адреси, то буде створено спеціальний файл (файл сокета) за заданим шляхом, через який зможуть повідомлятися будь-які локальні процеси шляхом читання / запису з нього (див. Доменний сокет Unix). Сокети типу INET доступні з мережі і вимагають виділення номера порту.

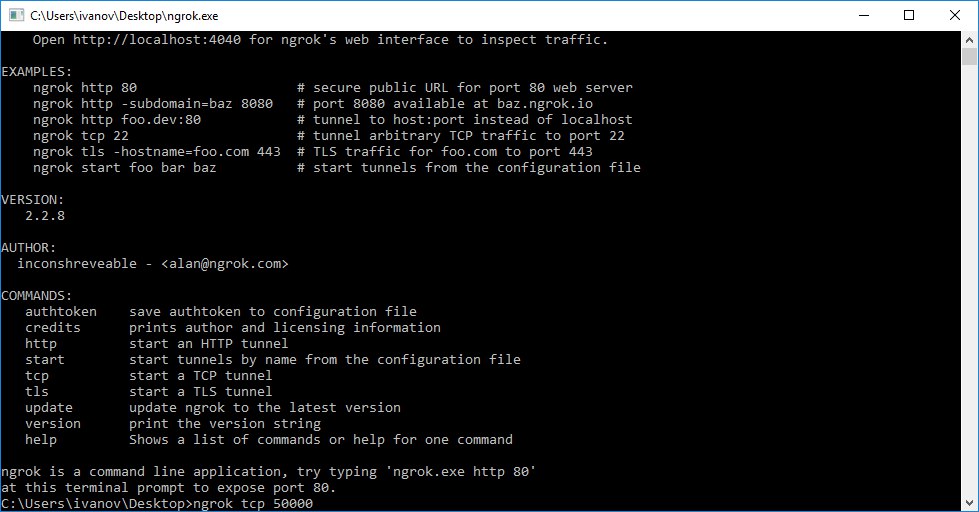
Проблема з доступом до IP-адреси та її рішення.

Міжмережевий екран, або файрвол (англ. Firewall), буквально «вогняна стіна» — пристрій або набір пристроїв, сконфігурованих, щоб допускати, відмовляти, шифрувати, пропускати через проксі весь комп'ютерний трафік між областями різної безпеки згідно з набором правил та інших критеріїв. Це була одна з найбільших проблем, яку ми вирішили, бо раніше не зустрічалися з цим. Це питання можна вирішити двома варіантами: або прокинути порт на роутері, до якого підключено комп’ютер-сервер.

У мові програмування Java є клас Socket, який ми використовували на сервері, та клас ServerSocket, для реалізації клієнтської частини.

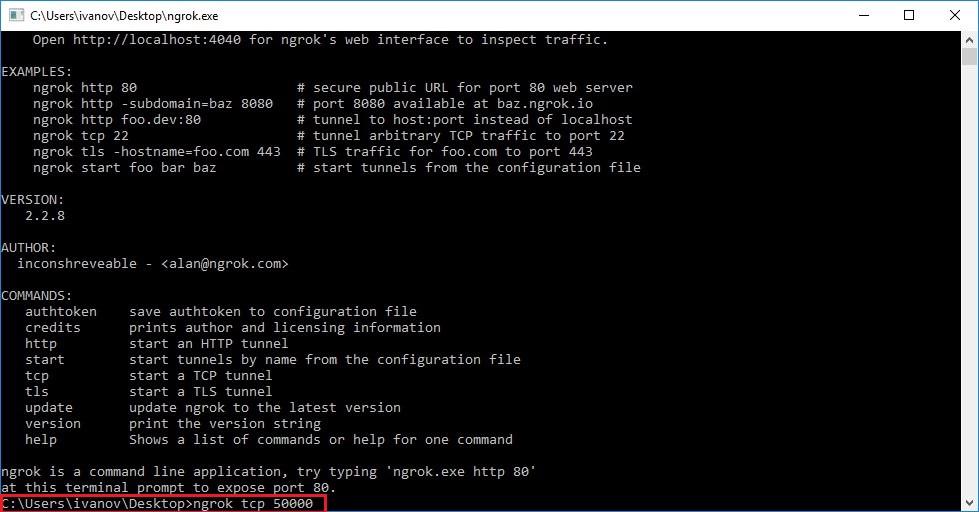
Клієнт та сервер не обов’язково з’єднанні локальною мережею (наприклад під одним WI-FI) , але щоб дані дійшли до серверу, клієнт повинен звернутися до IP комп’ютера, на якому стоїть сервер у зовнішній мережі. Але провайдери надають тільки динамічний IP, який постійно змінюється. Та щоб комп’ютер можна було з’єднати з клієнтами, ми використовували **тунель ngrok**, який дозволяв з’єднуватись до сервера безпосередньо.

На скріншоті зображено інтерфейс прогами ngrok

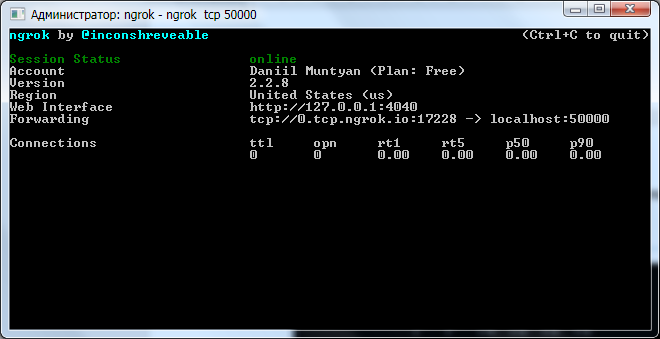


Для того, щоб отримати адресу комп’ютера і ззовні мережі можна було підключитися до нього, використовуючи протокол TCP.

Нижче зображено введення команди у командний рядок консолі.



Після натиснення кнопки «Enter», ми побачимо такий інтерфейс:



У рядку «Web interface» - адреса нашого комп’ютера в мережі. Ми у класі Socket на клієнтській частині вказуємо цю адресу – і дані приходять туди куди нам треба (на сервер).

Порівняння TCP та UDP протоколів

Незважаючи на «привабливість» UDP протоколу, обрано TCP, бо при трансляції екрану важливий порядок приходу кадрів на інший клієнт. Це забезпечує TCP.

TCP – транспортний протокол передачі даних у мережі TCP/IP, який попередньо встановлює з’єднання між хостингами. Тоді при відправці даних вони доходять до цілі у тому порядку, в якому ми їх відправляємо, тобто зберігається хронологія відправки даних, реалізована по принципу структури даних – черга (queue).

UDP протокол – транспортний протокол, який передає пакети без необхідності у зв’язку у IP-мережі.

Різниця між ними у тому, що TCP має більшу надійність, аніж UDP, але при цьому всі ігрові сервери та програми, які вимагають високої швидкості, використовують UDP.

Розглянемо плюси та мінуси використання обох протоколів для передачі.

* 1. TCP гарантує доставку пакетів у незмінному вигляді, незмінній послідовності та без втрат. UDP нічого такого не гарантує.
  2. TCP нумерує пакеті при передачі, UDP - ні.
  3. TCP потребує заздалегідь встановлений зв’язок, UDP з’єднання не потребує, у нього це просто потік даних
  4. UDP забезпечує більш високу швидкість передачі даних
  5. TCP надійніше та здійснює контроль над процесом обміну даних.
  6. UDP переважніше для програм, які відтворюють потокове відео або для серверних ігор
  7. UDP не містить функцій відновлення даних

Щоб встановити зв’язок ззовні до комп’ютера сервера, при використання UDP, не можна використовувати тунель ngrok: там такої команди не існує.

Для вирішення проблеми треба на роутері, до якого підключено комп’ютер-сервер, прокинути порт, за допомогою якого отримаємо можливість звернутися до сервера ззовні. Для цього потрібно «зайти в роутер» через браузер, та у вкладці «Переадресація» обрати «Віртуальні сервери», там додати порт, вказати його, вказати локальний IP комп’ютера-сервера, на який маршрутизатор перекине потік інформації до сервера.

Для реалізації передачі даних ми обрали TCP протокол. Тому далі я детальніше напишу про його роботу.

Transmission Control Protocol, TCP (з англ. «Протокол керування передачею») – разом зі протоколом IP є стрижневим протоколом інтернету, який дав назву моделі TCP/IP. Протокол призначений для управління передачею даних у комп'ютерних мережах, працює на транспортному рівні моделі OSI. Як вже було зазначено, TCP, на відміну від іншого поширеного протоколу транспортного рівня UDP, забезпечує надійне з’єднання та відправку даних від хоста-відправника до хоста-отримувача, для чого встановлюється логічний зв’язок між хостами. Таким чином, TCP належить до класу протоколів із встановленим з’єднанням.

TCP розбиває конкретний потік даних на порції, та додає до кожної з них заголовок з номером послідовності. Отримані таким чином порції даних традиційно називаються TCP-сегментами. Далі кожний сегмент інкапсулюється в IP-пакет і передається через IP-протокол до хоста-отримувача.

Після надходження IP-пакету до хоста-отримувача перевіряється коректність отриманих даних у TCP-сегменті, методом перерахування контрольної суми, та переконується, що попередні сегменти даних також були успішно отримані. Після чого хост-отримувач надсилає запит до хоста-відправника про нову, або повторну передачу порції даних, що одночасно є підтвердженням того, що всі сегменти з номерами послідовності, меншими ніж номер нового запиту, були успішно отримані.

У свою чергу TCP-сегменти деінкапсулюються з IP-пакетів, розміщуються в правильному порядку та з них вилучаються TCP-заголовки. Отриманий таким чином потік даних передається до того протоколу верхнього рівня, з якого первісно надійшли дані на стороні хоста-відправника.

Блок даних (PDU, Protocol data unit) TCP називається сегментом, хоча часто також використовують слово пакет, але таке вживання може вносити плутанину з IP-пакетом.

TCP-сегмент складається із TCP-заголовка і поля Дані (Data), яке називають сегментом даних.

Стандартний розмір TCP-заголовка — 20 байт, але з використанням опцій розмір може зростати до 60 байт. Як правило, опціями хости обмінюються на етапі встановлення з'єднання.

На сервері я використовував деякі поля заголовка як ідентифікацію клієнта, при реєстрації його на сервері.

Поля заголовка.

1. Порт джерела: складається з бітів (від 0 до 15).
2. Порт призначення: 16-31 біти. Порт призначення (Destination port) ідентифікує номер TCP-порту, на який відправляється сегмент.
3. Номер послідовності: 32-63 біти. Номер послідовності (Sequence number) є числом, що відображає номер першого байту в сегменті надісланих даних від хоста-відправника до хоста-отримувача.
4. Номер підтвердження: 64 — 95 біти. Номер підтвердження (Acknowledgment number) фактично є запитом від хоста-отримувача на надіслання нового сегменту даних починаючи зі вказаного номера.
5. Зміщення даних: 96-99 біти. Зміщення даних (Data offset) 4-бітний номер, який визначає розмір TCP-заголовка в 32-бітових словах
6. Зарезервовано: 100—102 біти, зарезервовані для майбутнього використання і повинні містити нулі (000).
7. Розмір вікна: 112—127 біти. Розмір вікна (Window Size) визначає кількість байтів даних, які відправник може надіслати до того, як отримає підтвердження (запит на новий сегмент) від хоста-отримувача.
8. Контрольна сума: 128—143 біти. Контрольна сума (Checksum) розраховується на основі усього TCP-сегменту включно із заголовком та важливих полів IP-пакету.
9. Покажчик важливості: 144—159 біти. Поле береться до уваги тільки в разі встановленого прапорця URG, та містить значення зміщення відносно номеру послідовності сегменту.

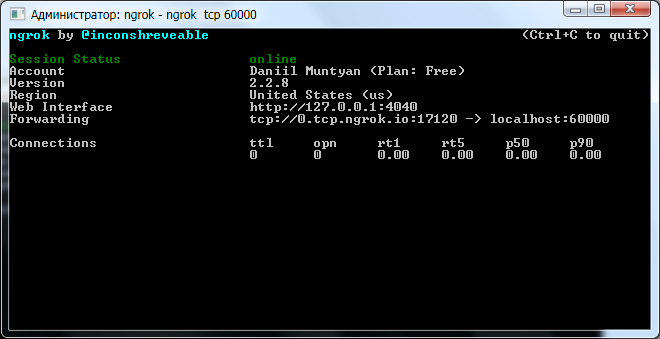
TCP-порти

Поняття порт (port) є ключовим у протоколі TCP. Порт з точки зору операційних систем називається сокетом (socket) процесу. Іншими словами це програмний інтерфейс для забезпечення обміну даними між процесами.

Для розуміння поняття порту з точки зору комп'ютерних мереж легко провести аналогію з роботою звичайної пошти. Коли ви відправляєте листа, то заповнюєте поля Куди: адреса будинку і Кому: людина, яка проживає у цьому будинку. У мережі Інтернет на питання Куди: відповідає IP-протокол, тобто це є IP-адреса хоста, а на питання Кому: відповідає TCP-протокол (або інший протокол транспортного рівня), тобто це номер протоколу прикладного рівня (процес, що відповідає за цей протокол з точки зору операційної системи), який «мешкає» за вказаною IP-адресою хоста. За аналогією зі звичайною поштою ви можете направити 2-а листа 2-м різним людям, які живуть в одному будинку, в Інтернеті ви можете направити TCP-сегменти 2-м різним протоколам прикладного рівня за однією і тією ж самою IP-адресою. Звичайно треба заповнити і зворотню адресу.

В Інтернеті широко вживається форма запису : ip-адреса: порт. Саме так у програмі ngrok виводиться адреса, на яку треба надіслати пакети, щоб вони дійшли до сервера.

На скріншоті нижче продемонстровано написання програмою адреси, на яку треба відсилати пакети: «tcp//0.ngrok.io:17120». Тут портом є «60000».



За надання і використання портів відповідальна IANA[23], хоча на практиці на відміну від використання IP-адрес це правило не завжди дотримується. Наприклад нічого не заважає адміністраторам 2-х хостів домовитися передавати, якийсь тип IP-трафіку по порту 80, який є закріплений за протоколом http і є відкритим на більшості файрволів.

TCP-заголовок має 16-бітові поля порт джерела та порт призначення, які можуть приймати значення від 0 до 65535.

Реалізація Android клієнту

Я, в основному, реалізовував клієнтську частину програми: на Android. Для реалізації інтерфейсу та мобільного додатку я використовував середу програмування Android Studio.

Android Studio

Android Studio — інтегроване середовище розробки (IDE) для платформи Android, представлене 16 травня 2013 року на конференції Google I/O менеджером по продукції корпорації Google — Еллі Паверс (англ. Ellie Powers). 8 грудня 2014 року компанія Google випустила перший стабільний реліз Android Studio 1.0.

Для пошуку помилок у коді та по-крокового налагодження проекту використовувалася консоль розробника та Logcat.

Багатопотоковість на клієнті

Багатопотоковість ([англ.](https://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0" \o "Англійська мова) multi-threading), — властивість [операційної системи](https://uk.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%B0_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0), яка полягає в тому, що [процес](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)), породжений в операційній системі, може складатися з кількох [нитей](https://uk.wikipedia.org/wiki/%D0%9D%D0%B8%D1%82%D1%8C), що виконуються паралельно, або навіть одночасно на багатопроцесорних системах. При виконанні деяких завдань таке розділення може досягти ефективнішого використання ресурсів [комп'ютера](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%27%D1%8E%D1%82%D0%B5%D1%80). Такі ниті виконання ще називають [потоками](https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D1%82%D1%96%D0%BA_(Windows)).

Суттю багатопотоковості є квазібагатозадачність на рівні одного виконуваного процесу, тобто всі ниті виконуються в адресному просторі процесу. Окрім цього, всі ниті процесу мають не тільки спільний адресний простір, але і спільні дескриптори файлів. Процес, що виконується, має як мінімум один (головний) потік.

Багатопотоковість (як доктрину програмування) не слід плутати ані з [багатозадачністю](https://uk.wikipedia.org/wiki/%D0%91%D0%B0%D0%B3%D0%B0%D1%82%D0%BE%D0%B7%D0%B0%D0%B4%D0%B0%D1%87%D0%BD%D1%96%D1%81%D1%82%D1%8C), ані з багатопроцесорністю, незважаючи на те, що операційні системи, що реалізовують багатозадачність, як правило реалізують і багатопотоковість.

Переваги в багатопотоковості такі.

* 1. Спрощення програми в деяких випадках, за рахунок використання загального адресного простору.
  2. Менші відносно процесу часові витрати на створення ниті і взаємодію між нитями.
  3. Підвищення продуктивності процесу за рахунок розпаралелювання процесорних обчислень і операцій вводу/виводу.

В нашому проекті без багатопотоковості не обійтись, бо, по-перше, сервер має обробляти запити декількох клієнтів одночасно, тому апріорі треба використовувати багатопотоковість, а по-друге, у мові програмування Java Android не можна працювати з мережею у головному потоці програми, тому треба створювати паралельні потоки. У проекті їх створено кілька.

Багатопотоковість працює так, що виконує «потроху» кожен потік, тобто програма може перемикатись з одного потоку до іншого з кожною точкою з комою. Потоки розділяють між собою ресурси процесора (пам'ять і відкриті файли). Такий підхід, очевидно, веде до ефективної, але проблематичної комунікації. Кожна програма має лише один потік, що виконується. Той потік, з якого починає виконуватися програма, називається головним або основним. Головний потік здатний створювати додаткові потоки у вигляді об’єктів Runnable. Кожен потік може бути запланований для виконання на окремому ядрі ЦП, використовувати квантування часу на одноядерному процесорі або використовувати квантування часу на декількох процесорах. В двох останніх випадках система буде періодично переключатися між потоками, по черзі даючи виконуватися то одному, то іншому. Така схема називається псевдо-паралелізмом. Немає універсального рішення, яке сказало би як саме потоки Java будуть перетворені в нативні потоки ОС. Це залежить від конкретної реалізації JVM.

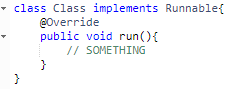
В мові Java потік представляється в вигляді об’єкту-нащадка класу Thread. Цей клас інкапсулює стандартні механізми роботи з потоком.

У програмі реалізовано два типи таких потоків:

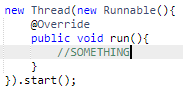
* Потік, який створюється за допомогою окремо створеного класу, який реалізовує інтерфейс Runnable та його метод run(). Для того, щоб почати виконання у паралельному потоку класу, треба створити екземпляр класу Thread. Приклад коду:



Зверху представлено сам виклик початку роботи потока. Знизу на скріншоті продемонстровано сам Runnable-клас.

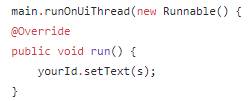


* Потік, який створено безпосередньо у коді, наступним методом:



Тут використано властивість анонімного класу, який ми прямо у параметрах класу Thread і реалізовуємо.

За багатопотоковістю, при реалізації проекту виникало багато питань та проблем, рішення яких є не тривіальним. Прикладом можу навести те, що для Android програмування невід’ємною частиною є віджети, які ініціалізуються у головному потоці програми, але справа в тому, що виводити щось або змінювати параметри у цих віджетах у паралельних потоках заборонено Java’ою. Для рішення такої проблеми треба створювати так званий потік інтерфейсу, призначеного для користувача (UI Thread). Приклад з коду:



Тут main – екземпляр класу MainActivity, головного класу, методи якого викликаються при створенні додатку та в якому відбувається ініціалізація всіх віджетів і де міститься головний потік програми на Android.

Щоб працювати в іншому потокові зі змінними іншого класу, в проекті ці змінні передаються у параметрах конструктора класу, змінні якого вміщені, за допомогою інкапсуляції.

У деяких місцях коду, де потрібно обов’язково виконати тільки ці саме дії, я використовував synchronized-блоки. Все, що написано у цьому блоці, виконується якби відразу, «за раз». Тобто у цих блоках поточний потік забирає на себе всю «увагу» програми, і вона не можна перемикнутись до іншого потоку доти, доки не закінчить виконуватись цей блок. Прикладом використання такого блоку у клієнті є відправка до серверу поточного скріншоту екрану, щоб не робити скріншоти далі, поки ми не відправимо поточний до серверу.

Реалізація TCP-клієнту на Java Android. Ініціалізація

Для реалізації клієнту використовувалася мова програмування Java Android. Я використовував клас Socket, з Java пакету java.net.

Спочатку розглянемо загальне визначення та принципи роботи сокетів взагалі.

Головні принципи роботи. Кожен [процес](https://uk.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D1%81_(%D1%96%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) може створити слухаючий сокет (серверний сокет) і прив'язати його до якогось [порту](https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D1%80%D1%82_%D0%BF%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB%D1%83) операційної системи. Процес, що слухає, зазвичай знаходиться в циклі очікування, тобто прокидається при появі нового з'єднання. При цьому зберігається можливість перевірити наявність з'єднань у цей час, встановити тайм-аут для операції тощо.

Кожен сокет має свою адресу. Зазвичай клієнт явно під'єднується до слухача, після чого будь-яке читання або запис через його [файловий дескриптор](https://uk.wikipedia.org/wiki/%D0%A4%D0%B0%D0%B9%D0%BB%D0%BE%D0%B2%D0%B8%D0%B9_%D0%B4%D0%B5%D1%81%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D1%80) будуть передавати дані між ним і сервером.

При реєстрації на сервері, клієнт ініціалізується та отримує особисте ID, по якому він зможе транслювати свій екран іншим.

Окрім самого класу Socket, у проекті використовуються «прилеглі до нього» - необхідні - класи. Такі як InetAddress, ObjectInputStream, ObjectOutputStream, а також Integer та String – для створення екземпляру класа Socket.

Огляд класу InetAddress

Для використання та роботи з адресами IP у бібліотеках класів Java є клас InetAddress. За його допомогою додаток може визначити адрес IP локального вузла, а також віддаленого вузла, заданого доменним ім’ям. Особливістю цього класу є те, що створення об’єкту класа InetAddress виконується не за допомогою оператора new, а з використанням статичних методів getLocalHost(), getByName() та getAllByName().

У своєму проекті я використовував InetAddress при ініціалізації об’єкту класа Socket. Екземпляр класу InetAddress створено на базі адреси серверу, бо саме на сервер нам треба відправляти дані.

Класи ObjectInputStream та ObjectOutputStream.

У мові програмування Java Android існує пакет java.io, в якому є класи InputStream та OutputStream – суперкласи, відповідно над двома класами ObjectInputStream та ObjectOutputStream. Їх використання є важливою частиною проекта: вони відповідають за відправку та прийняття даних на та з серверу. Створення екземплярів цих класів проходить через socket.getInputStream() та socket.getOutputStream().

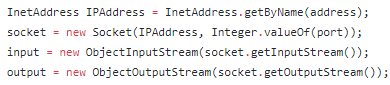
Екземпляр класу ObjectInputStream віповідає за прийняття з серверу даних. Реалізується прийняття через метод readObject(), який повертає екземпляр класу Object.

Екземпляр класу ObjectOutputStream відповідає за відправку на сервер даних. Реалізовується передача через метод writeObject(), аналогічно прийняттю.

Ініціалізація екземпляру класу Socket

Однією з головних частин на клієнтській частині програми є створення Socket’у, без існування якого, функціювання сервер-клієнту неможливе.

На скріншоті нижче демонструється ініціалізація Socket’у та підлеглих до нього класів саме на клієнті.



Тут змінна address має тип String, у ній записана адреса серверу, на який ми будемо надсилати дані та з якого будемо приймати дані.

Другим у параметрі конструктора класу Socket є змінна класу Integer – порт, який «резервується» для встановлення зв’язку між клієнтом та сервером. На даному скріншоті, змінна port має тип String, тому я використовую статичний метод класу Integer, valueOf, який повертає Integer.

Далі іде ініціалізація екземплярів класу ObjectInputStream та ObjectOutputStream, які я описав вище.

Реалізація трансляції екрану

Трансляція проходить по скріншотам. Тобто деяку кількість скріншотів робиться за секунду, при цьому кожен скріншот відправляється на сервер а потім і до того клієнту, якого підключено до трансляції. Детальніше про роботу програми - у розділі «Опис програми».

Алгоритм Нейгла

У роботі нашого проекту використано реалізацію алгоритму Нейгла, шляхом виклику методу .setTcpNoDelay() з екземпляра класу Socket. У API класу Socket, у методі setTcpNoDelay() реалізовано вмикання та вимикання цього алгоритму.

Алгоритм Нейгла названий на честь Джона Нейгла і є засобом підвищення ефективності роботи мереж TCP/IP, що дозволяє зменшити кількість пакетів, які повинні бути відправленні до мережі. Документ Нейгла, Управління перевантаженням мереж IP / TCP (RFC 896) описує те, що він назвав «проблемою невеликих пакетів», яка полягає в тому, що додаток неодноразово посилає дані невеликими порціями, часто розміром в 1 байт. TCP-пакети мають 40 байт заголовка (20 байт TCP, 20 байт IPv4) Внаслідок того передається пакет розміром 41 байт, що несе в собі один байт корисної інформації. Маємо великі накладні витрати. Ця ситуація часто зустрічається в сесії Telnet, де більшість натискань клавіш генерують один байт даних, який негайно передається. Крім того, по повільних каналах зв'язку багато такі пакети можуть знаходитися на шляху в один і той же час, що спричинює перевантаженості мережі.

Алгоритм Нейгла працює шляхом об'єднання кількох невеликих вихідних повідомлень, а потім відправки їх всіх відразу. Зокрема, поки існує відправлений пакет, для якого відправник ще не отримав жодного підтвердження про доставку, відправник повинен тримати в буфері наступні дані для відправки - до тих пір, поки не набереться достатньо даних на повний пакет, який можна відправити одного разу.

Його ми використовували: можливо, що це прискорює роботу проекту, та демонстрацію екрана, бо у проекті відсилаються дані дуже часто.

Опис програми. Напрямки використання

Програма використовується для трансляції на далекій відстані, але вимагає при цьому підключення до інтернету. Також додаток може бути використано задля демонстрації дій навколо – за допомогою камери, тобто при трансляції екрану можна включити камеру та глядачі будуть бачити все навкруги.

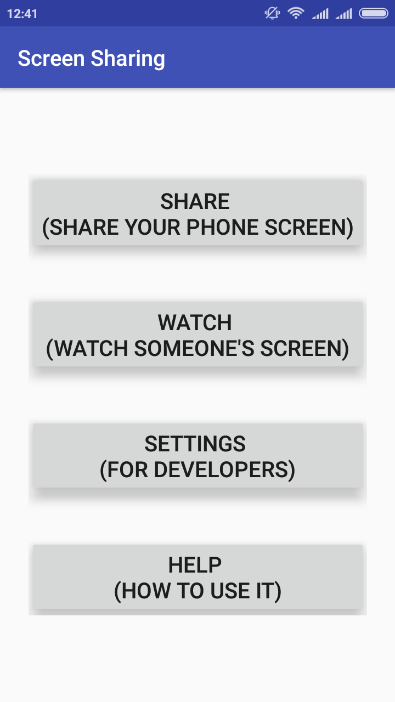
Програма працює таким чином: спочатку відкривається головне меню додатку, бачимо чотири пункти меню.

* SHARE (SHARE YOUR PHONE SCREEN)
* WATCH (WATCH SOMEONE’S SCREEN)
* SETTINGS (FOR DEVELOPERS)
* HELP (HOW TO USE IT)

Кожний з наведених пунктів відповідає назві: перший – демонструвати екран свого смартфону для інших, другий – підключитися до трансляції іншого, третій – налаштування (частіше для розробників), там вказуються адреса та порт серверу, а також якість передачі зображення, четвертий – допомога користувачу у розбору функціоналу додатку.

Ініціалізація клієнту - при натисненні або на кнопку SHARE, або на WATCH.

Інтерфейс головної сторінки:



Кнопка SHARE

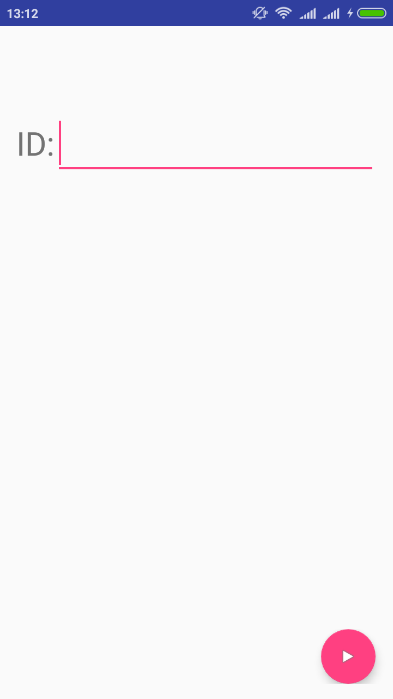
При натисканні на SHARE, маємо наступник інтерфейс:



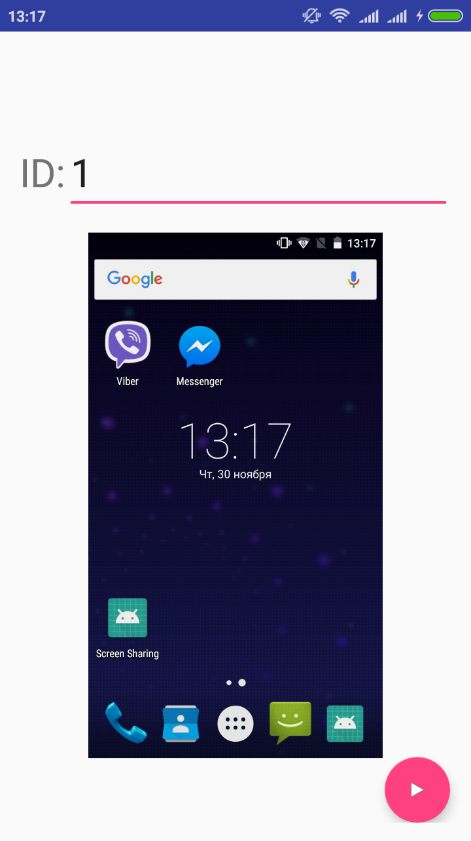
Після початку трансляції (кнопка START), зверху буде написано ID клієнту, який почав трансляцію, для того, щоб інші знали, до якого ID підключатись, щоб подивитись його трансляцію. Кнопка STOP відповідно завершує трансляцію екрана.

Кнопка WATCH

Після натискання на кнопку WATCH, маємо:

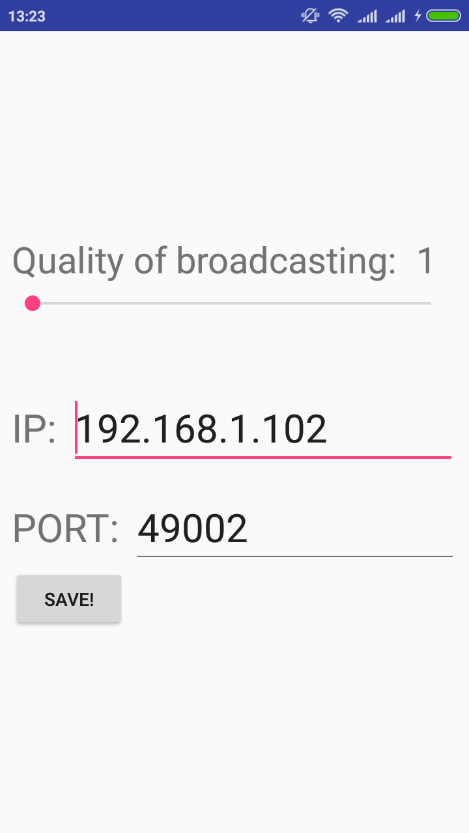


У поле ID треба вводити ID того користувача, який транслює свій екран і до я кого ми хочемо приєднатись. Після натиснення на кнопку внизу екрану, ми побачимо екран транслятора:



Кнопка SETTINGS

При натисканні на SETTINGS відкриється така сторінка:

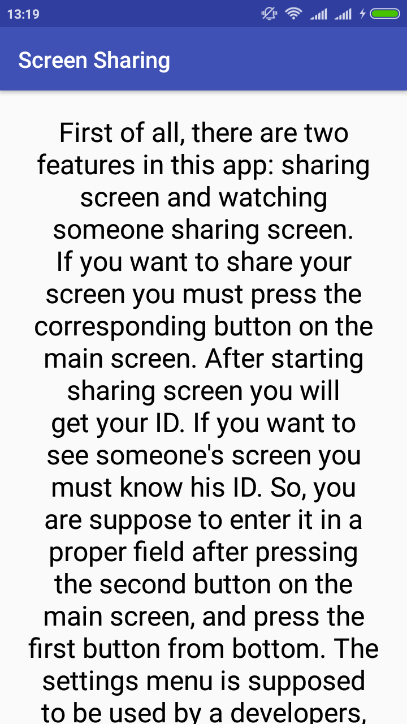


Вона потрібне не стільки для користувача, скільки для розробника, бо у полях ID та PORT треба вводити IP серверу та порт, через який треба підключатись до серверу. Можна встановити якість передачі при трансляції у відсотках (від 1-го до 90-та).

При натисканні на SAVE, у внутрішній пам’яті смартфону створюється папка «ScreenSharing», де в текстовий файл «Config.txt» записуються порт та IP серверу – для того, щоб при зміні IP серверу не змінювати це поле у SETTINGS багато разів, а зробити це лише раз, бо потім програма буде брати ці поля з файлу Config.txt.

Кнопка HELP

Кнопка HELP, створено для роз’яснення користувачу, як працює додаток.



Як бачимо, на цій сторінці міститься текст, написаний англійською, для опису роботи додатку для користувача.

Опис роботи

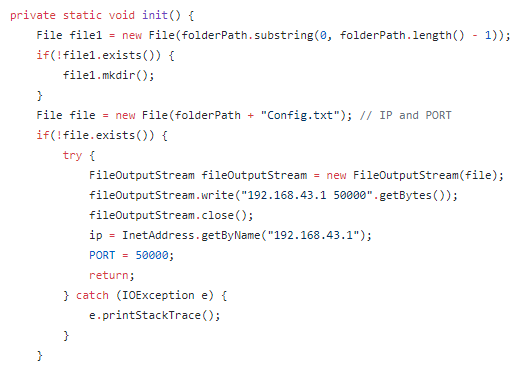
Від кожного клієнта може поступити чотири типи запитів, які ми чекаємо у нескінченному циклі.

1. Якщо клієнт відправив один байт «-1», то це значить, що клієнт почав «роздачу». Далі він буде відправляти масиви байтів, які нам потрібно буде відправляти до клієнтів, які є «глядачами» поточного клієнта.
2. Якщо клієнт відправив один байт «-2», то це значить, що клієнт закінчив «роздачу». Далі сервер повинен сповістити всіх «глядачів», що «роздача» завершена, відправивши їм один байт – «-1».
3. Якщо клієнт відправив байт «-3», то це означає, що він підключається до трансляції якогось клієнта. Тоді на сервер також приходить номер клієнта, до якого підключається поточний клієнт. Якщо клієнт, до якого намагається підключитися поточний клієнт не «роздає», то сервер поверне до поточного клієнта байт «-3». Інакше сервер буде відправляти байти, які приходять від клієнта, до якого підключився поточний клієнт, до поточного клієнта.
4. Якщо клієнт відправив байт «-4», то від відключається від трансляції клієнта, номер якого він укаже.

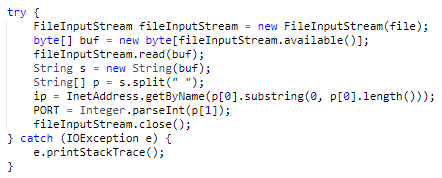
Реалізовано такий алгоритм за допомоги рукописних класів Client, геш-таблиць та Linked List’ів.

Велика проблема при розробці - те, що за визначенням, у Java не можна отримати доступ або взяти значення будь-якої змінної, якщо вона знаходиться в іншому потокові. Тому для зручності я вирішив створити клас PublicStaticObjects, який містить статичні поля, до яких можна буде звернутись з будь-якого потоку та який буде ініціалізувати клієнта на сервері. Код цього класу описано у Додатку. У ньому є поля типу ObjectInputStream та ObjectOutputStream. Далі у будь-якому місці проекту, якщо треба відправити на сервер якісь дані, я пишу: PublicStaticObjects.getOutputStream.writeObject(something). Таке рішення проблеми є дуже зручним, бо не треба зберігати екземпляри класів Socket, ObjectInputStream, ObjectOutputStream у кожному класі проекту, а достатньо тільки взяти їх з класу PublicStaticObjects.

При роботі з файлом Config.txt, я використовував реалізовані у Java класи FileInputStream, FileOutputStream. Збереження даних зі сторінки SETTINGS відбувається наступним чином:



Зчитування з файлу та заповнення полів ip та PORT реалізовуються наступним чином:



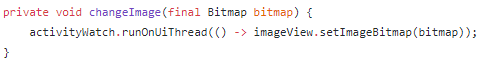
При трансляції екрану, програма повинна працювати «у фоні», тобто повинна бути можливість виходити з додатку, заходити у інші програми. Для цього ми використовуємо класи MediaProjection та MediaProjectionManager.

Метод onImageAvailable(ImageReader imageReader) викликається зразу після очищення черги в ImageReader. У цьому методі я беру поточний скріншот екрану транслятора та відправляю його на сервер. Наступний скріншот буде зроблено, як тільки відправиться попередній, і так до тих пір, доки транслятор не натисне кнопку STOP – тоді спрацює метод stop() у екземпляра класу MediaProjection та трансляція зупиниться.

Для того, щоб отримувати скріншоти транслятора з серверу, у паралельному потоці запускаю клас Receiver, який у нескінченному циклі чекає на дані від серверу:



При їх отриманні він конвертує їх у Bitmap та виводить у відповідний віджет – ImageView:



Структура проекту

Вимоги програмного забезпечення

Вимоги для сервера:

* Операційні системи: Microsoft Windows 10/8/7/Vista/2003/XP (включаючи 64-bit), GNOME або KDE та їхні дочірні (наприклад, Ubuntu).
* RAM: 1 GB ОЗП мінімум, 2 GB ОЗП рекомендується.
* JDK версії 1.8 та вище.
* Встановлена IntelliJ Idea.

Вимоги до клієнта:

* Операційні системи: Android 5.0 та вище (API вище 21)
* RAM: 500 MB ОЗП мінімум, 2 GB ОЗП рекомендується.
* При використанні додатку **обов’язково** треба дати дозвіл програмі використовувати пам’ять смартфону.

Висновки

Під час виконання роботи я зіткнувся з великою кількістю проблем. Починав писати роботу, не знаючи нічого в технології клієнт-серверних додатків, не маючи гадки, що таке IP, що таке порт, і для чого потрібні протоколи передачі даних.

Подолання таких труднощів крок за кроком, іноді методом проб та помилок, іноді за допомогою керівника, ми написали клієнт-серверний додаток.

Робота над цим проектом значно збільшила мої знання з мови програмування Java та Java Android. Особливо важко довелось працювати в Java Android, тому що я зіткнувся з проблемами поділу User Interface Thread, Background Thread, Foreground Thread, які в реальному програмуванні в реальних проектах доводиться вирішувати постійно.

В результаті створено програмний проект, який може створювати скріншот екрану Android-пристрою, перетворювати його в масив байтів й відправляти на сервер. Сервер перевідправляє ці масиви іншим клієнтам. А вони будуть перетворювати їх на зображення і виводити на екран, створюючи видимість відео.

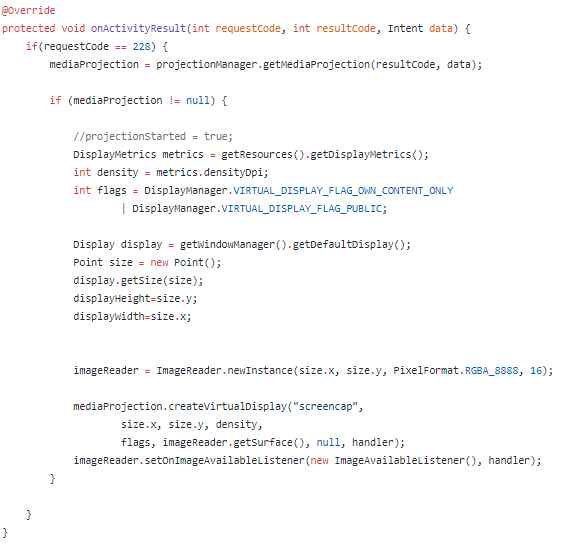
Джерела інформації

* <https://www.wikipedia.org/>
* <https://habrahabr.ru/>
* <http://cs.lmu.edu/~ray/notes/javanetexamples/>
* <http://codeforces.com/>
* <https://stackoverflow.com/>
* <https://docs.oracle.com/en/>
* <https://ru.stackoverflow.com/>
* <https://github.com>

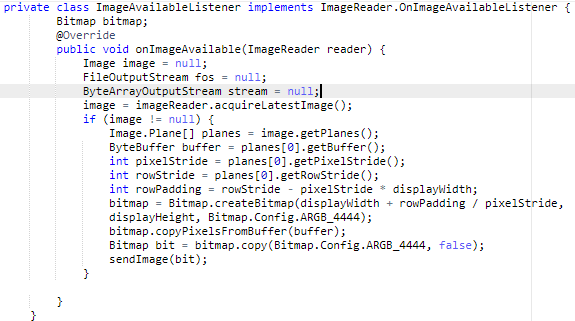
Додаток

Трансляція та робота додатку «у фоні» починається викликом методу startActivityForResult():

Після його виконання, викликається метод onActivityResult, який я перевизначив:



Останній рядок визначає «слухача», тобто клас, який реалізований у паралельному потоці та який «слухає» або чекає на очищення черги у imageReader та відправляє новий скріншот на сервер:



Метод sendImage у паралельному потоці передає поточний скріншот до сервера. В ньому перевизначений метод run() синхронізован, бо інакше ObjectOutputStream, який я створив для передачі даних до серверу переповниться та припинить свою роботу.

