МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 121 «Інженерія програмного забезпечення»

Освітня програма: «Розробка програмного забезпечення»

Група: 4РП-05

Дипломний проект

здобувача освіти денної форми навчання
РП.05.03.000.ДП

ГАТМАНА ДАНІЛА
ОЛЕКСАНДРОВИЧА

м. Одеса
2022 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: **121 «Інженерія програмного забезпечення»**
Освітня програма: «**Розробка програмного забезпечення»**
Група: **4РП-05**

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проекту (роботи) на тему:

Розробка мобільного додатку для мережі дієтичного харчування

Проектний матеріал складається з пояснювальної записки на _____ сторінках та графічного (презентаційного) матеріалу на _____ аркушах (слайдах).

Дипломник _____ (Гатман Д.О.)
Керівник _____ (Іванова Л.В.)

Консультанти:
з економічної частини _____ (Копайгородська Т.Г.)
з охорони праці_____ (Чорновол Н.І.)
з дотримання вимог ЄСКД _____ (Петрашова В.І.)
старший консультант _____ (Скорнякова О.В.)

До захисту допущений
Голова циклової комісії _____ (Скорнякова О.В.)
Завідувач відділення _____ (Суліма Ю.Ю.)

Захист «____»_____ 2022 р. Протокол ЕК №_____
Оцінка ЕК _____

Секретар ЕК _____

Анотація

Тема дипломного проекту «Розробка мобільного додатку для мережі дієтичного харчування».

У цій роботі описані принципи розробки сучасного мобільного додатку та процес підбору інструментів та технологій для розробки мобільних додатків під операційну систему Andorid.

У аналітичній частині приведене технічне завдання до проекту, а також аналіз додатків з сфери послуг. В технологічній частині приведені інструменти для розробки мобільних додатків та порівняння з аналогами. У програмному розділі описані види життєвих циклів додатків та їх переваги, основні принципи побудови архітектури додатку та його тестування. Останніми розділами стали охорона праці під час розробки та конструювання додатку та економічне обгрунтування.

Ключові слова: мобільний додаток, девайс, розробка, Android Studio, Java, JUnit, UI, UX, Android.

Annotation

The theme of the diploma project "Development of a mobile application for the web of diet restaurants".

This paper describes the principles of development of a modern mobile application and the process of selecting tools and technologies for the development of mobile applications for the Andorid operating system.

The analytical part presents the terms of reference for the project, as well as the analysis of applications in the field of services. The technological part provides tools for mobile application development and comparison with analogues. The program section describes the types of application life cycles and their benefits, the basic principles of building application architecture and testing. The last sections were labor protection during the development and design of the application and economic justification.

Keywords: mobile application, device, development, Android Studio, Java, JUnit, UI, UX, Android.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Відділення **комп'ютерних систем** Комісія **КТ та ПІ**

Спеціальність **121 «Інженерія програмного забезпечення»**

Освітня програма **«Розробка програмного забезпечення»**

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР ***Беркань І.В.***

"_____" _____2022 р.

З А В Д А Н Н Я

на дипломний проект (роботу)

Здобувачеві (здобувачці) освіти ***Гатман Даніїл Олександрович***

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) **Розробка мобільного додатку для мережі дієтичного харчування**

затверджена наказом по коледжу від "30"_грудня_2021р. №___306-А2-ОД

2. Термін здачі закінченого проекту (роботи) ____20.06.2022р.

3. Вихідні данні до проекту (роботи)

1. *Технічне завдання*
2. *Програмне забезпечення для створення мобільних додатків*
3. *Вимоги до функціоналу та графічного дизайну;*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які необхідно розробити)

Вступ. Мобільні додатки у бізнесі та суспільстві. Програмне забезпечення для розробки мобільного додатку. Розробка мобільного додатку.. Економічні розрахунки . Охорона праці. Висновок. Список використаних джерел інформації.

5. Перелік графічного (презентаційного) матеріалу (з точним зазначенням обов'язкових креслень, кількості слайдів)

1. *Дизайн-макет мобільного додатку*
2. *Алгоритм роботи додатку*

6. Консультанти по проекту (роботі), із зазначенням розділів проекту, що їх стосується

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Основний	*Іванова Л.В.*	*30.11.2021*	*17.05.2022*
Економіка	*Копайгородська Т.Г.*	*4.05.2022*	*17.05.2022*
Охорона праці	*Чорновол В.І.*	*4.05.2022*	*17.05.2022*
Нормоконтроль	*Петрашова В.І.*	*4.05.2022*	*17.05.2022*
Старший консультант	*Скорнякова О.В.*	*4.05.2022*	*17.05.2022*

7. Дата видачі завдання _____30.11.2021_____

Керівник *Іванова Л.В.*
(підпис)

Завдання прийняв до виконання *Гатман Д. О.*
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/р	Назва етапів дипломного проекту (роботи)	Термін виконання етапів дипломного проекту (роботи)	Відмітка про виконання
1.	*Передмова*	*4.05.2022*	
2.	*Мобільні додатки у бізнесі та суспільстві*	*8.05.2022*	
3.	*Розробка технічного задання для додатку*	*10.05.2022*	
4.	*Розробка та реалізація мобільного додатку*	*15.05.2022*	
5.	*Тестування та публікація ігрового додатку*	*17.05.2022*	
6.	*Економічні розрахунки*	*22.05.2022*	
7.	*Розділ охорони праці*	*26.05.2022*	
8.	*Висновок*	*28.05.2022*	
9.	*Перелік літератури*	*31.05.2022*	
10.	*Оформлення пояснювальної записки*	*2.06.2022*	
11.	*Оформлення графічної частини*	*10.06.2022*	
12.	*Малий захист кваліфікаційної роботи*	*15.06.2022*	
13.	*Захист кваліфікаційної роботи*	*18.06.2022*	

Дипломник *Гатман Д..О.*
(підпис)

Керівник *Іванова Л.В.*

Формат	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			РП.05.03.000.00 ДП	Кваліфікаційна робота		
А4			РП.05.03.000.00 ДП ПЗ	Пояснювальна записка	1	
			РП.05.03.000.00 ДП МП	Мультимедійна презентація		

Ізм.	Арк.	№ докум.	Підпис	Дата	(підпис) РП 05.03.000.00 ДП ПЗ			
Розробив		Гатман Д.О.			Розробка мобільного додатку для мережі дієтичного харчування	Літ.	Аркуш	Аркушів
Перевірив		Іванова Л.В.				Н Д П	7	88
						ВСП «ОТФК ОНТУ» гр.РП-05		
Н. Контр.		Петрашова В.						
Затверд.		Скорнякова О.						

ЗМІСТ

ПЕРЕДМОВА

У суспільстві стрімкими темпами розвиваються інформаційні та мобільні технології, і все більше мобільна техніка та інтернет впливають на наше життя.

З появою інтернету на нього відразу звернув увагу бізнес, так як це місце могло стати гарною нішою для збуту продукції та послуг. Так і сталося, обороти компаній збільшилися в десятки та сотні разів, і це вплинуло на подальший розвиток технологій та революційного створення мобільних технологій.

За допомогою мобільних технологій ми можемо тепер заплатити за паркування прямо з телефону. Вбудовані карти дозволяють не заблукати в будь-якому районі незнайомого міста. Нам доступний розклад транспорту за натисканням однієї лише кнопки. Зрештою, тепер ми можемо дізнаватися та досліджувати цікаві місця, які нас оточують.

Зараз у нас у кишенях лежить більше різноманітної інформації про світ, ніж будь-коли в історії. Фактично для сотень мільйонів людей стало абсолютно природним і звичним відразу шукати у смартфонах та планшетах інформацію про будь-яку діяльність. Наші гаджети вже можуть передбачити, яка інформація може зацікавити, і подають її нам у зручній формі.

Подібне, на перший погляд, поверхневе та легковажне повсякденне використання мобільних технологій змінило людське суспільство. Воно зробило наші зв'язки з іншими людьми набагато ширшим. У мережі інтернет ми знаходимо справжніх друзів, яких ніколи в житті не зустрічали. Ми можемо дізнаватися та обговорювати будь-які актуальні питання та події, що відбуваються у будь-якому куточку світу. Це стимулює виникнення спільнот та обмін думками, які раніше були просто неможливі.

Крім змін у нашій соціальній активності та розширенні доступної інформації, мобільні технології незабаром можуть почати активно впливати на здоров'я. У наше життя стійко увійшли мобільні технології, які кардинальним способом покращують і процеси виробництва, і процеси споживання інформації. Використання мобільних технологій дозволяє бути в курсі всіх подій у світі, докладаючи при цьому мінімум зусиль.

Крім того, мобільні технології дозволяють зменшити вартість продукції для кінцевих споживачів за рахунок оптимізації процесів, скорочення виробничих витрат та невиробничих витрат.

За допомогою мобільних пристроїв ми не прив'язані до робочого місця та легко можемо отримати доступ до інформації в дорозі, в цеху – де завгодно, витративши на це мінімум коштів та зусиль.

Все це лише мала частина того, як мобільні технології трансформували, покращили та спростили життя мільярдів людей. І цей процес паралельно з розвитком самих технологій. Нам, без перебільшення, пощастило жити у мобільну епоху. Ці технології за якісь 15 років стали незамінними помічниками. Дуже велика стала роль гаджетів у безлічі наших справ, щоденної рутині та подіях.

Але нові можливості принесли із собою нові труднощі. У кожної технології є свої переваги та недоліки, з якими доводиться або миритися, або шукати шляхи їх вирішення.

Актуальність теми дипломної роботи полягає в тому, що технології є важливою частиною нашого побуту, що вже не можна уявити собі, що ми не користуємося мобільним телефоном або планшетом, і тим більше інтернетом, ці технології оточують нас усюди, вони відомі маленькій дитині та пенсіонеру. Це розширює можливості повідомлення інформації про товари та послуги широким масам населення, зростання прибутку, та зниження витрат, а також дає конкурентну перевагу для постачальника, та вигоду для споживача.

Метою даної випускної роботи є створення мобільного додатка для мережі дієтичного харчування, для більш ефективного залучення нових клієнтів, підвищення інформованості існуючих клієнтів про роботу мережі, а також закріплення практичних навичок програмування для ОС Android середовищем розробки Android Studio.

РОЗДІЛ 1. МОБІЛЬНІ ДОДАТКИ У БІЗНЕСІ ТА СУСПІЛЬСТВІ

Мені здається, всі розуміють, що мобільне середовище – золота жила для бізнесу. У 2018 році більшість людей використовували смартфони для покупок в Інтернеті та користування мобільними додатками (більше 50%, за даними StatCounter).

1.1.Мобільні додатки в сфері бізнесу

Мобільні програми для бізнесу можна поділити на два види:

1) для внутрішніх потреб підприємства;

2) для маркетингу та збільшення продажів.

Приклади використання додатку для внутрішніх потреб підприємства :

Автоматизація бізнесу. Підвищення продуктивності спільної роботи, наприклад, створення спільного доступу до файлів, підключення месенджерів, соціальних мереж та трекерів для внутрішньої комунікації, а також систем управління проектами та завданнями.

Приклади використання додатку для маркетингу та збільшення продажів.

Як доповнення до онлайн-сервісів компанії: вітрини, каталоги, додатки для покупки квитків, інтернет-банкінг, трекер статусу доставки товарів тощо;

Існують певні нюанси використання таких типів додатків. Програми для автоматизації бізнесу зазвичай встановлюють обов'язково всі учасники ділової команди. Але програми для клієнтів компанії встановлюються за бажанням клієнтів. Яких треба переконати: встановлення програми важливе, зручне, просте та вирішує якісь питання.

Потрібно бути готовим до того, що чимала частина користувачів все-таки не захоче встановлювати додаток, якщо ту ж інформацію можна отримати через мобільний браузер. Від установки програми відмовляються з кількох причин: важко (треба шукати в онлайн-магазині, завантажувати, надавати якісь дозволи), довго, займає місце в пам'яті пристрою та уповільнює його роботу.

Кому потрібний мобільний додаток для вирішення бізнес-завдань?

Є сфери бізнесу, яким мобільний додаток просто необхідний. Наприклад:

Служба доставки їжі.

Клієнту зручно один раз вказати свою адресу та дані карти, а потім, при здійсненні наступних замовлень, просто обирати страви. А для служби доставки це чудова можливість прив'язати до себе клієнтів, збільшити середній чек та кількість звернень за рахунок персональних знижок та спецпропозицій для користувачів додатку.

Рисунок 1.1— Додаток служби доставки їжі Glovo

Кафе та ресторани.

Створення мобільного додатка для кафе та ресторанів – це чудова можливість «прив'язати» клієнта до закладу за рахунок бонусної програми та збільшити прибуток за рахунок доставки страв. Також можна прискорити роботу закладу за рахунок попереднього замовлення через програму. І, звичайно, залучити нових клієнтів.

Магазини.

Мобільні додатки для магазинів дозволяють клієнтам швидко замовляти товар з каталогів, отримувати доступ до ексклюзивного контенту та дізнаватися першими про новинки та знижки. Для магазинів це чудова можливість збільшити відсоток спонтанних покупок, залучити нових клієнтів до свого бренду. Одним із найуспішніших прикладів є інтернет-магазин Rozetka.

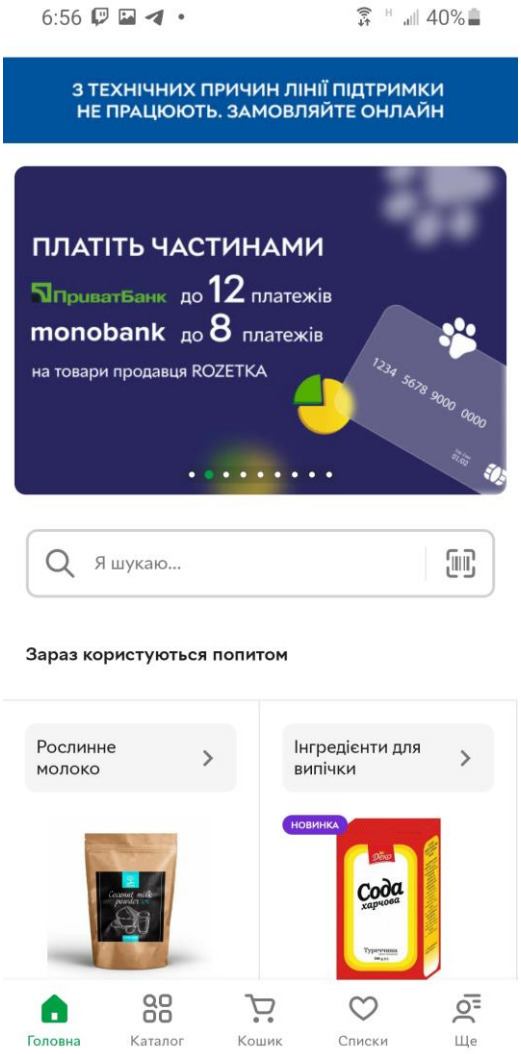


Рисунок 1.2 — Додаток онлайн-магазину Rozetka

Туристичні компанії.

У сфері туризму мобільний додаток просто необхідний. Він дає можливість клієнтам швидко забронювати квиток, готель, та ще й на цьому заощадити. Це збільшує продаж квитків у кілька разів, тому що важко переоцінити зручність покупки квитка з дому, на відміну від поїздок у касу та простоювання черг. Крім

того, можна підписатися на повідомлення про зниження ціни на потрібний вам напрям

Таксі.

Мобільний додаток відмінно працює у сфері замовлення таксі. Відмінним прикладом буде всім відомий UBER, який успішно працює у багатьох країнах світу. Оформити замовлення можна за кілька секунд, вказавши свої побажання та уподобання. Поїздку можна оплатити карткою через програму. Крім того, покупець по карті може відстежити, звідки їде машина і коли вона буде на місці.

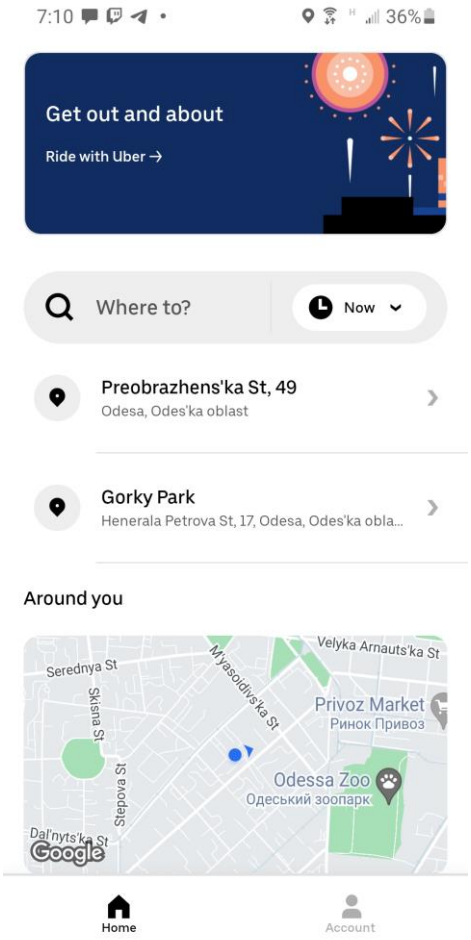


Рисунок 1.3 — Додаток служби таксі Uber

Спортивні клуби

Клуби можуть надати користувачам безкоштовний контент у вигляді відео-тренувань та корисних порад щодо харчування. У додатку можна дізнатися розклад тренувань, записатися до тренера, заощадити на абонементі за рахунок бонусної програми. Це допоможе клубу виділитися на фоні конкурентів, залучити нових клієнтів та скоротити кількість тих, хто забув про тренування.

Переваги мобільних додатків :

Збільшення продажів

Збільшення продажів за допомогою push-повідомлень про акції, знижки та бонуси. Це мотивує клієнтів повертатися та робити покупки саме у вас. Причому повідомлення можна надсилати у необмеженій кількості також і тим, хто знаходиться неподалік вашого магазину, ресторану тощо.

Маркетинговий інструмент

Це відмінний маркетинговий інструмент, оскільки інформація про новинки та хіти стимулює додаткові продажі. Крім того, вивчаючи історію переглядів клієнта, можна надсилати йому лише рекламу, яка його зацікавила. Це забезпечує індивідуальний підхід до клієнта.

Збір інформації про клієнтів

Мобільний додаток дозволяє зібрати інформацію про клієнтів. Що найбільше купують, скільки грошей витрачають, як часто роблять покупки. Проаналізувати цю інформацію та оптимізувати бізнес під потреби клієнтів

Вирішення завдань управління, логістики та контролю

Мобільні додатки здатні підвищити ефективність роботи компанії, відстежувати шлях товару та багато іншого.

Поліпшення якості сервісу

Наприклад, Інтернет-банкінг дозволяє клієнтам моментально купувати квитки, бронювати столики у ресторанах, замовляти їжу, викликати таксі.

Збільшення лояльності

Інтегрувавши програму лояльності в додаток, можна заощадити на покупці пластикових карток, а також збільшити кількість шанувальників вашого бренду.

Залучення нових клієнтів

Якщо ваша програма дійсно корисна, її рекомендуватимуть друзям та знайомим. Це також збільшить упізнаваність вашого бренду.

Економія на рекламній кампанії

Якщо підтримувати інтерес до програми, можна заощадити на рекламних кампаніях і збільшити ефективність програми.

1.2. Типи мобільних додатків та їх переваги

Нативні програми

Під нативним ми маємо на увазі мобільний додаток, який створюється для певної платформи і безпосередньо встановлюється на пристрій користувача (займаючи певний обсяг пам'яті). Такі програми користувач завантажує через магазин програм тієї чи іншої платформи, такої як Play Store для Google та Apple App Store для iOS.

З нативними програмами компанії можуть виготовити програму відповідно до індивідуальних запитів, щоб користувачеві було зручно ним користуватися, на додаток до веб-сайту або іншого каналу, яким він уже звик користуватися. Ця цілісність є істотною перевагою нативних додатків.

Деякі інші важливі переваги нативних програм:

- Позначення геолокації дозволяє компаніям підлаштовувати свої програми лояльності чи акції. Споживачі можуть отримати повідомлення, коли вони знаходяться біля фізичних магазинів, або мають можливість отримати регіональну знижку.

- Дані дій (або бездіяльності) користувача можуть бути легко зібрані та проаналізовані, таким чином полегшуючи оцінку ефективності всієї програми або її окремих функцій.

- Нативні програми, як правило, працюють і «відчуваються» краще. Веб-програми іноді створюються для імітації нативних, але вони обмежуються швидкістю інтернету та можливостями дизайну.

І можливі недоліки:

- Нативні програми часто дорожчі в розробці, особливо для компаній, яким потрібні програми на кросплатформових ОС

- Нативні програми повинні бути схвалені кожним магазином програм, а процес привернення уваги до нього користувачів може бути складним (якщо це не додаток для внутрішнього користування в компанії)

Веб-програми

Як ви вже могли здогадатися, ці програми працюють через веб-браузер на пристрої користувача. Ці програми, по суті, є індивідуалізованими веб-сайтами, які зроблені таким чином, щоб виглядати та використовуватися як нативні програми, але насправді вони не знаходяться на пристрої користувача. Їх можна порівняти з хмарним сховищем у порівнянні з даними, що зберігаються на жорсткому диску комп'ютера.

Ось деякі ключові переваги веб-додатків:

- Програми на веб-основі легко підтримуються і вони можуть функціонувати на платформі з будь-якою ОС

- Розробники можуть пропонувати додатки без необхідності їх затвердження будь-якими магазинами додатків

- Швидша розробка циклів з використанням CSS, HTML та JavaScript

І кілька мінусів:

- У веб-програм немає доступу до пристрою користувача. Незважаючи на те, що іноді було б зручно, це обмежує багато функцій, які використовуються в нативних додатках для більш персоналізованого використання

- Користувачі повинні використовувати їх через мережу, що значно знижує контроль безпеки

- Пошук програми може бути важким, тому що її нема в звичному магазині додатків

Гібридні програми

Гібридні програми є чимось середнім між нативними та веб-додатками. Фактично вони створюються так, щоб виглядати та використовуватися як нативні додатки. Їх також встановлюють на телефон користувача та їх можна знайти у магазинах програм. Відмінність полягає в тому, що вони обов'язково повинні розміщуватися в рамках нативної програми і створені для роботи через WebView, і таким чином вони можуть отримувати доступ до інформації на пристрої користувача для більших можливостей.

Додаткові переваги гібридних додатків:

- Гібридні додатки мають найбільшу функціональність і персоналізацію для користувача

- Розробники не обмежені однією платформою, натомість вони можуть створити гібридну програму, яка буде працювати з декількома платформами (у разі роботи як нативна програма)

- Гібриди — гарна опція для розробників, які створюють візуально насичені програми, наприклад, ігри (які не будуть добре працювати у вигляді веб-додатків)

У будь-якому випадку, є деякі недоліки, про які варто подумати при виборі гібридної програми:

- Занадто складні програми найкраще робити нативними

- Розробка вимагає додаткових часу та зусиль (у порівнянні з веб-додатками), щоб така програма виглядала та відчувалася користувачем як нативна

- Магазини програм можуть відхиляти гібридні програми, які працюють недостатньо плавно

Вибір відповідної моделі мобільного додатка – це дуже важливий етап у його розробці, на який впливають кілька факторів, таких як технічна оцінка розробників; потреба у доступі до інформації на пристрої; вплив швидкості інтернету на програму; одне-або багатоплатформне додаток.

РОЗДІЛ 2. РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ

2.1 Інструменти для розробки додатку та їх переваги

Існує безліч інструментів для розробки мобільних додатків, які допоможуть у розробці мобільного додатку та зроблять цей процес меньш трудомістким. Ось деяки з них :

Xamarin у **Visual Studio**

Visual Studio це IDE від Microsoft, що підтримує ряд мов програмування, включаючи C#, VB.net, JavaScript і багато іншого. За допомогою фреймворку Xamarin, який входить до Visual Studio, можна створювати кросплатформні програми за допомогою C#, а потім тестувати їх на кількох пристроях, підключених хмарно. Це хороший і безкоштовний вибір, якщо ви плануєте випустити програму і для Android, і для IOS, але не дуже бажаєте писати свій код двічі. Також він є відмінним варіантом для тих, хто вже знайомий з C# та Visual Studio. Мінусом є те, що Xamarin незручний у використанні Java бібліотек і, як і з будь-якою іншою альтернативою Android Studio, ви втрачаєте підтримку Google та розширені вбудовані функції.

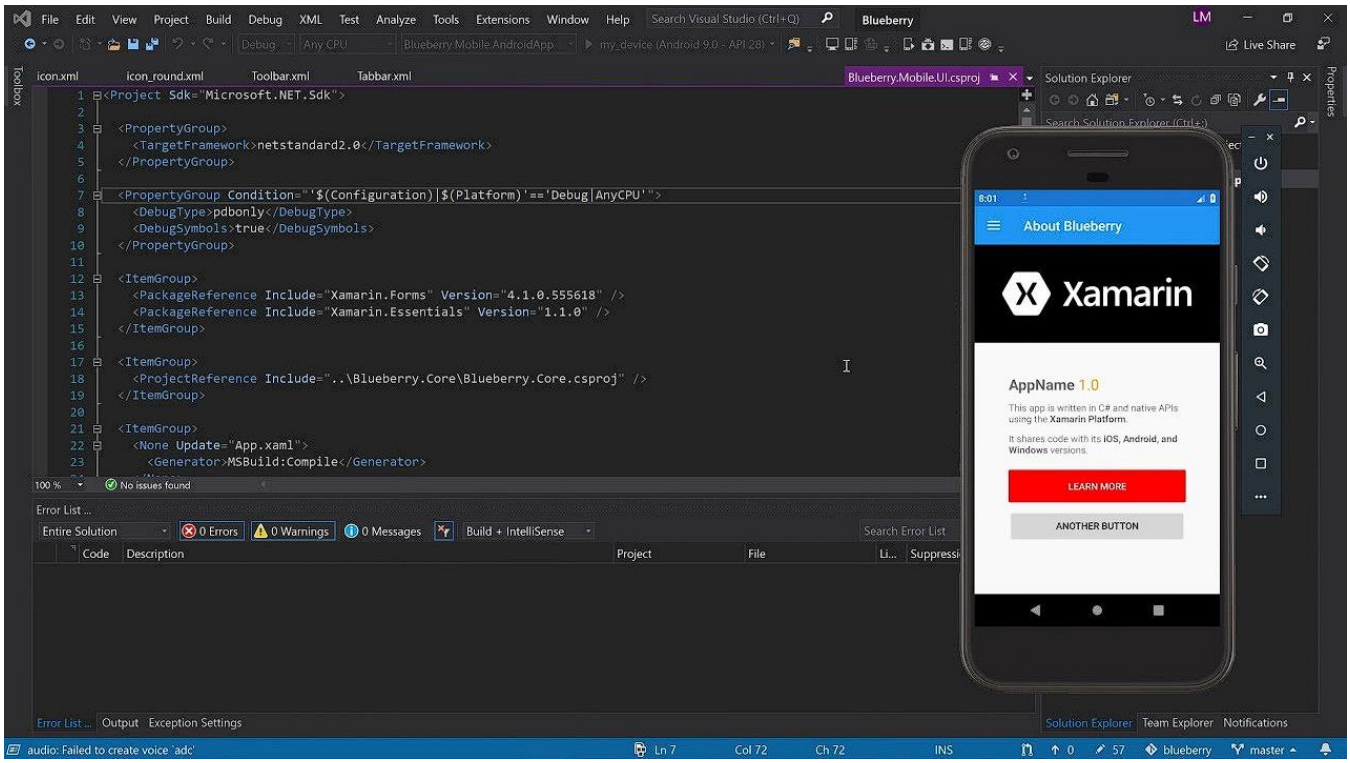


Рисунок 2.1 — Інтерфейс Xamarin в середовищі Visual Studio

Android Studio

Android Studio це інтегроване середовище розробки виробництва Google, за допомогою якого розробникам стають доступні інструменти для створення програм на платформі Android OS. Android Studio можна встановити на Windows, Mac та Linux. Android Studio створювалася за підтримки IntelliJ IDEA.

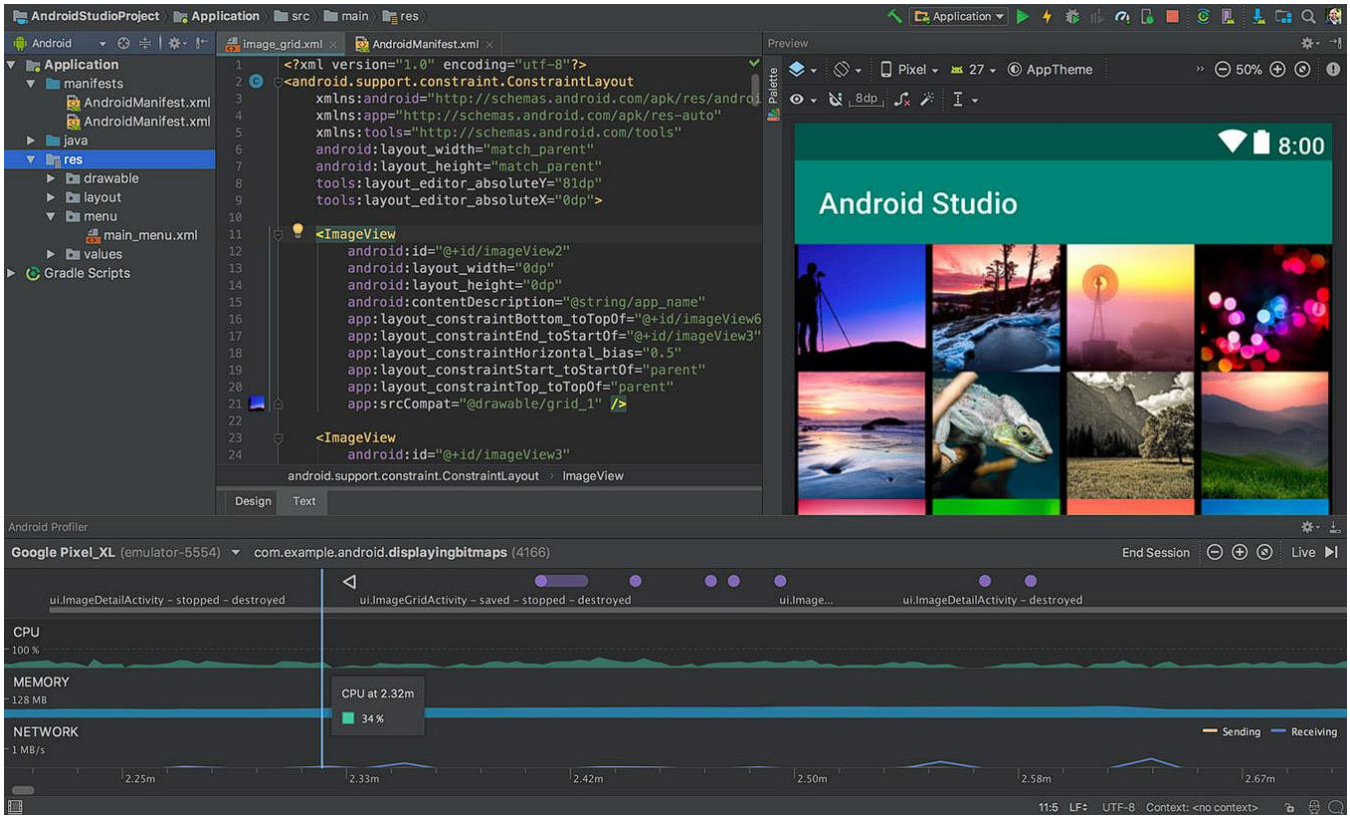


Рисунок 2.2 — Інтерфейс Android Studio

IDE можна завантажити та користуватися безкоштовно. У ній є макети для створення UI, з чого зазвичай починається робота над додатком. Studio містить інструменти для розробки рішень для смартфонів і планшетів, а також нові технологічні рішення для Android TV, Android Wear, Android Auto, Glass і додаткові контекстуальні модулі.

Середовище Android Studio призначене як для невеликих команд розробників мобільних додатків (навіть у кількості однієї людини), або великих міжнародних організацій з GIT або іншими подібними системами управління версіями. Досвідчені розробники зможуть вибрати інструменти, які найбільше підходять для масштабних проектів. Рішення для Android розробляються в Android Studio за допомогою Java або Kotlin. В основі робочого процесу Android Studio закладено концепт безперервної інтеграції, що дозволяє відразу виявляти наявні проблеми.

Тривала перевірка коду забезпечує можливість ефективного зворотного зв'язку з розробниками. Така опція дозволяє швидше опублікувати версію мобільного додатка в Google Play App Store. Для цього є також підтримка інструментів LINT, Pro-Guard і App Signing.

За допомогою засобів оцінки продуктивності визначається стан файлу з пакетом прикладних програм. Візуалізація графіку дає змогу дізнатися, чи програма відповідає орієнтиру Google в 16 мілісекунд. За допомогою інструмента для візуалізації пам'яті розробник дізнається, чи його додаток буде використовувати занадто багато оперативної пам'яті та коли відбудеться її очищення. Інструменти для аналізу батареї показують, яке навантаження посідає на пристрій.

Android Studio сумісна з платформою Google App Engine для швидкої інтеграції у хмари нових API та функцій. У середовищі розробки ми знайдемо різні API, такі як Google Play, Android Pay та Health. Є підтримка всіх платформ Android, починаючи з Android 1.6. Є варіанти Android, які суттєво відрізняються від версії Google Android. Найпопулярніша з них – це Amazon Fire OS. В Android Studio можна створювати APK і для цієї ОС.

Особливості :

Нові функції відображаються з кожною новою версією Android Studio. На даний момент доступні такі функції:

Розширений редактор макетів: WYSIWYG, здатність працювати з UI компонентами за допомогою Drag-and-Drop, функція попереднього перегляду макета на декількох конфігураціях екрана.

Складання додатків, засноване на Gradle.

Різні види збірок та генерація декількох .apk файлів

Рефакторинг коду

Статичний аналізатор коду (Lint), що дозволяє знаходити проблеми продуктивності, несумісності версій та інше.

Вбудований ProGuard та утиліта для підписування додатків.

Шаблони основних макетів та компонентів Android.

Підтримка розробки додатків для Android Wear та Android TV.

Вбудована підтримка Google Cloud Platform, яка включає інтеграцію з сервісами Google Cloud Messaging і App Engine.

Android Studio 2.1 підтримує Android Preview SDK, а це означає, що ми зможемо розпочати роботу зі створення програми для нової програмної платформи.

Нова версія Android Studio 2.1 здатна працювати з оновленим компілятором Jack, а також отримала покращену підтримку Java 8 та вдосконалену функцію Instant Run.

Звернувши увагу на всі переваги Android Studio я вирішив, що вестиму розробку використовуючи саме це середовище розробки.

2.2. Java – як мова програмування для розробки під ОС Android.

У програмуванні для Android використовуються об'єктно-орієнтовані технології, тому я наведу огляд об'єктних технологій. В умовах постійно зростаючого попиту на нові потужні програмні продукти досить важко поєднувати такі вимоги, як швидкість розробки, правильність роботи та економічність. Об'єкти - це повторно використовувані програмні компоненти. Як об'єкти можуть використовуватися дата, час, відео, людина, автомобіль та інші предмети матеріального світу. Практично кожен іменник може бути адекватно представлений програмним об'єктом у поняттях атрибутів (наприклад, ім'я, колір та розмір) та поведінки (наприклад, обчислення, переміщення та передача даних). Розробники бачать, що використання модульної структури та об'єктноорієнтованого проектування при розробці додатків підвищує продуктивність роботи.

Щоб краще зрозуміти суть об'єктів та їх вмісту, скористаємося простою аналогією. Уявіть собі, що ми за кермом автомобіля, та натискаєте педаль газу, щоб набрати швидкість. Що має статися перед тим, як ми отримуємо таку можливість? Перш ніж ми поведемо автомобіль, хтось має його спроектувати. Виготовлення будь-якого автомобіля починається з інженерних креслень, які докладно описують пристрій автомобіля. Зокрема, на цих кресленнях показано пристрій педалі акселератора. За цією педаллю ховаються складні механізми, які

безпосередньо прискорюють автомобіль, подібно до того, як педаль гальма приховує механізми, що гальмують автомобіль, а кермо приховує механізми повороту. Завдяки цьому люди, які не мають поняття про внутрішній пристрій автомобіля, можуть легко ним керувати. Перш ніж ми сядемо за кермо машини, її потрібно збудувати на основі інженерних креслень. Втілений у металі автомобіль має реальну педаль газу, за допомогою якої він може прискорюватися, але це не все — він не може це робити самостійно, а лише після того, як водій натисне на педаль.

Скористаємось прикладом з автомобілем для демонстрації деяких ключових концепцій об'єктно-орієнтованого програмування. Для виконання операції у програмі потрібен метод, у якому «ховаються» інструкції програми, які безпосередньо виконують операцію. Метод приховує ці інструкції від користувача подібно до того, як педаль газу автомобіля приховує від водія механізми, що спричиняють прискорення автомобіля. Програмна одиниця, що називається класом, включає методи, які виконують завдання класу. Наприклад, клас, що представляє банківський рахунок, може включати три методи, один з яких поповнює рахунок, другий знімає кошти з рахунку, а третій визначає поточний баланс. З концептуальної точки зору клас подібний до інженерного креслення, в якому зображено пристрій педалі газу, рульового колеса та інших механізмів.

Водій не сяде за кермо автомобіля, поки його не збудують за кресленням, і побудова об'єкта класу необхідна для виконання задач, що визначаються методами класу. Цей процес називається створенням екземпляра. Отриманий об'єкт зветься екземпляром класу. На основі одних і тих же креслень можна створити багато автомобілів, а на основі одного класу можна створити багато об'єктів. Використання існуючих класів для створення нових класів заощаджує час та сили розробника. Повторне використання також полегшує створення більш надійних та ефективних систем, оскільки раніше створені класи та компоненти зазвичай проходять ретельне тестування, налагодження та оптимізацію. Подібно до того, як концепція використання взаємозамінних елементів лягла в основу промислової революції, повторно використовувані класи відіграють ключову роль у програмній

революції, що була ініційована впровадженням об'єктних технологій. Коли ми ведемо машину, натискання педалі газу відправляє автомобілю повідомлення із запитом на виконання певного завдання (прискорення автомобіля). Подібним чином надсилаються повідомлення об'єкту. Кожне повідомлення є викликом методу, який «повідомляє» об'єкту необхідність виконання деякої задачи. Наприклад, програма може викликати метод deposit об'єкта банківського рахунку, щоб поповнити банківський рахунок.

Будь-який автомобіль крім можливості виконувати певні операції також має атрибути, такі як колір, кількість дверей, запас палива в баку, показання спідометра та одометра. За аналогією з операціями атрибути автомобіля представляються на інженерних діаграмах (як атрибути автомобіля можуть виступати одометр і вказівник рівня бензину). Кожен автомобіль містить власний набір атрибутів. Наприклад, кожен автомобіль «знає» про те, скільки бензину залишилося в його баку, але йому нічого не відомо про запаси пального в баках інших автомобілів.

Об'єкт, як і автомобіль, має власний набір атрибутів, які він «переносить» із собою під час використання цього об'єкта у програмах. Ці атрибути визначаються як частина класу об'єкта.

Наприклад, об'єкт bankaccount має атрибут балансу, що представляє кількість коштів на банківському рахунку. Кожен об'єкт bankaccount «знає» про кількість коштів на власному рахунку, але нічого не знає про розміри інших банківських рахунків. Атрибути визначаються за допомогою інших змінних екземпляра класу.
Класи інкапсулюють атрибути та методи в об'єкти (атрибути та методи об'єкта між собою тісно пов'язані). Об'єкти можуть обмінюватися інформацією між собою, але зазвичай вони не знають про деталі реалізації інших об'єктів, що приховані всередині самих об'єктів. Подібне приховування інформації життєво важливо в практиці хороших програмних архітектур.

За допомогою успадкування можна швидко та просто створити новий клас об'єктів. При цьому новий клас успадковує характеристики існуючого класу, які можуть частково змінюватися. Також у новий клас додаються унікальні характеристики, властиві лише цього класу.

Якщо згадати аналогію з автомобілем, "трансформер" є об'єктом більш узагальненого класу "автомобіль", у якого може підніматися або опускатись дах.

Мови програмування, подібні Java називаються об'єктно-орієнтованими. Програмування на таких мовами, зване об'єктно-орієнтованим програмуванням (ООП), реалізує об'єктно-орієнтовані проекти у вигляді працездатних систем

Одна очевидна перевага Java над іншими мовами - те, що API Android дуже схоже на API мови Java, і Android підтримує якщо не всі доступні в J2SE SDK класи, то принаймні найважливіші. Ще одна перевага: ви можете використовувати для розробки під Android ті ж інструменти, що і Java. Наприклад, IDE Eclipse, адже Google надає для Eclipse плагін для розробки програм Android.

У будь-якому випадку розберемося в особливостях внутрішнього функціонування Android.

Як працює Android?

Як я вже згадував, в операційній системі Android для розробки програм використовується Java. Ви можете написати код програми для Android за допомогою Java API, який надається, який потім буде скомпільований у файли класів. На цьому схожість закінчується. Android не використовує віртуальну машину Java (JVM) для виконання файлів класів, натомість, в ньому використовується віртуальна машина Dalvik, яка не є істинною JVM і не працює з Java-байткодом. Для виконання на віртуальних машинах Dalvik файли класів компілюються у формат DEX (Dalvik EXecutable – виконувані файли Dalvik). Після перетворення DEX, файли класів разом з іншими ресурсами об'єднуються в пакети Android (APK) для поширення та інсталяції на різних пристроях.

Головне, що слід знати: в основі базової бібліотеки класів віртуальної машини Dalvik лежить безліч проекту Apache Harmony, внаслідок чого вона не підтримує все API J2SE.Тепер давайте розберемося, як програми Android виконуються на пристроях.

За замовчуванням, операційна система Android надає кожному додатку унікальний ідентифікатор користувача. Після запуску програм Android, кожна з них виконується у своєму процесі, у своїй власній віртуальній машині.

При необхідності операційна система Android керує запуском та зупиненням процесів додатків. Це означає, що всі програми Android працюють ізольовано один від одного, але, зрозуміло, можуть вимагати доступ до апаратних та інших системних ресурсів.

Під час інсталяції або запуску програми Android вона запитує права, необхідні для доступу до Інтернету, телефонної книги або інших системних ресурсів. Користувач явно надає ці права, інакше в дії буде відмовлено.

Всі ці права доступу описуються у файлі маніфесту програми Android. На відміну від Java, маніфест Android є XML-файлом, в якому перераховані всі компоненти програми та налаштування для них.

Чотири основні компоненти програми Android: активності, сервіси, постачальники контенту та широкомовні приймачі (broadcast receivers). З них найчастіше зустрічаються активності, що відповідають окремій екранній формі програми Android. Наприклад, у грі для операційної системи Android може бути кілька екранів: для входу в систему, рекордів, інструкцій та екран самої гри. Кожен із цих елементів відповідає різним активностям у нашому додатку.

Як і Java, в ОС Android добре те, що вона виконує деякі завдання замість розробника, наприклад, створює об'єкти активностей. За організацію активностей відповідає клас System. Якщо потрібно запустити активність, достатньо викликати метод startActivity() з об'єктом Intent як параметр. У відповідь на цей виклик клас System або створить новий об'єкт активності або повторно використовує старий.

Аналогічно складання сміття в мові Java, що відповідає за надзвичайно важливе завдання повторного використання пам'яті, Android управляє запуском, зупиненням, створенням та знищенням програм. Може здатися, що він дуже їх обмежує, але це не так. Android надає події життєвого циклу, які можна перевизначати для втручання у цей процес.

Ось і все про те, як працює Android. Java-розробнику, безумовно, має сенс вивчити Android, оскільки ця система використовує Java, отже я зможеу скористатися своїми знаннями методик програмування на Java, патернами проектування та рекомендованими практиками для створення хороших програм Android.

Java протягом багатьох років домінує на ринку як провідна мова програмування, і саме він ліг в основу Android на зорі існування мобільної операційної системи. Однак у 2018 році Kotlin було оголошено офіційною мовою програмування для Android. Гнучкий та універсальний, він надає набагато більше можливостей, ніж традиційний Java.

Java займає особливе становище, з яким дуже складно конкурувати будь-якій альтернативній мові програмування. Тому, незважаючи на всі зусилля Google з просування Kotlin, питання про те, якою з двох мов краще, залишається відкритим. І щоб дати на нього відповідь, важливо розуміти плюси та мінуси кожного варіанту.

Java: Java є об'єктно-орієнтованою мовою програмування. Спочатку створений компанією Sun Microsystems, тепер належить Oracle. Це одна з найстаріших мов, можливості якої виходять далеко за межі розробки програм під Android.

Плюси:

Підходить для розробки як нативних, так і кроссплатформенних програм.

Величезна кількість готових бібліотек, у тому числі для проектів, орієнтованих на Android.

Програми, створені за допомогою Java, легші і компактніші в порівнянні з їх аналогами на основі Kotlin, що покращує загальний користувальницький досвід.

Висока швидкість розробки.

Мінуси:

Це досить складна мова у плані синтаксису, що збільшує ймовірність помилок та багів.

ava-розробники стикаються з проблемами при розробці програм на базі Android API через певні обмеження в коді.

Потрібно більше пам'яті порівняно з програмами Kotlin.

Kotlin: Kotlin з'явився, коли Android почав вимагати більш сучасних програмних рішень і тих можливостей, які Java просто не спромоглися охопити.

JetBrains - творці однієї з найпопулярніших IDE під назвою Intelliji IDEA - також є засновниками Kotlin.

Спеціально для Android вони створили нову мову програмування з відкритим кодом на базі JVM (віртуальної машини Java). Це дає розробникам можливість оновлювати застарілі програми Java, використовуючи інструменти Kotlin та переносити старі проекти, раніше виконані на Java, у нове середовище, не переписуючи повністю.

Плюси:

Головним плюсом Kotlin, безперечно, є мінімалізм. Те, що вимагало б 50 рядків коду Java, в Kotlin можна вмістити в 2-3 рядки. Саме тому ця мова стрімко набирає популярності серед розробників у всьому світі. Мінімалістичний код також означає, що при створенні програм за допомогою Kotlin набагато менша можливість допустити і не помітити помилки.

Kotlin дозволяє розробляти чисті API.

За допомогою байт-коду Java бібліотеки Java разом з його фреймворками можна легко використовувати в Kotlin, що робить перехід з однієї мови програмування іншою досить плавним.

Kotlin включає null у свою систему типів, що неможливо серед Java.

Розробникам Kotlin є найширша бібліотека Anko, що включає сотні проектів, доступних на GitHub.

Мінуси:

Kotlin використовує досить складний синтаксис, до якого непросто звикнути.

Kotlin у деяких випадках має більш повільну швидкість компіляції порівняно з Java, хоча у багатьох інших аспектах ця мова перевершує свого попередника.

Спільнота Kotlin досить молода, тому кількість доступних ресурсів на навчання досить обмежена, що може утруднити пошук рішень складних проблем. Проте з розвитком мови кількість тематичних ресурсів також зростає.

В Android Studio є такі функції, як автозаповнення та компіляція, які працюють повільніше у додатках на основі Kotlin, ніж у аналогах, створених з використанням Java.

2.3. Вибір моделі життєвого циклу додатку

Життєвий цикл та існуючі стратегії розробки програмного продукту

Програмний продукт – це комплекс взаємозалежних програм, який націлен на вирішення певної проблеми (завдання) масового попиту, підготовленийй до реалізації. Мобільні додатки – це теж програмні продукти. Вони можуть створюватися як індивідуально – розробка на замовлення, так і для масового поширення серед користувачів.

В основі розробки завжди лежить певна технологія виконання проектних робіт із застосуванням сучасних інструментальних засобів програмування. Суть її полягає у підборі існуючих або розробці абсолютно нових алгоритмів, а також написання програм, що залежать від характеру обробки інформації та інструментальних засобів. Звідси випливає, що для створення будь-якого програмного продукту витрачаються значні ресурси – трудові, матеріальні, фінансові. На додаток до цього є ще момент супроводу – це той етап, коли програмний продукт вже потрапив до його безпосередніх користувачів. Виправлення виявлених помилок, створення нових версій програм, доповнення функціоналу – це етап супроводу.

Поняття життєвого циклу

Під терміном життєвого циклу програмного продукту прийнято розуміти сукупність процесів та етапів розвитку організмів живої природи, технічних систем, продуктів виробництва від моментів зародження чи появи потреби їх створення та використання до припинення функціонування чи застосування. Іншими словами, це ряд подій, що відбуваються з системою з моменту створення і до кінця її розробки та впровадження.

Модель життєвого циклу програмного продукту залежить від специфіки, масштабу та складності проекту, а також тих умов, у яких система створюється та функціонує.

Типова модель процесів життєвого циклу складної системи починається з формування ідеї системи чи потреб у ній. Така модель охоплює проектування, розробку, застосування та супровід системи, і закінчується зняттям системи з експлуатації.

Основні стратегії розробки програмних продуктів

Розглянемо існуючі моделі життєвого циклу. І першою з них буде водоспадна або каскадна модель.

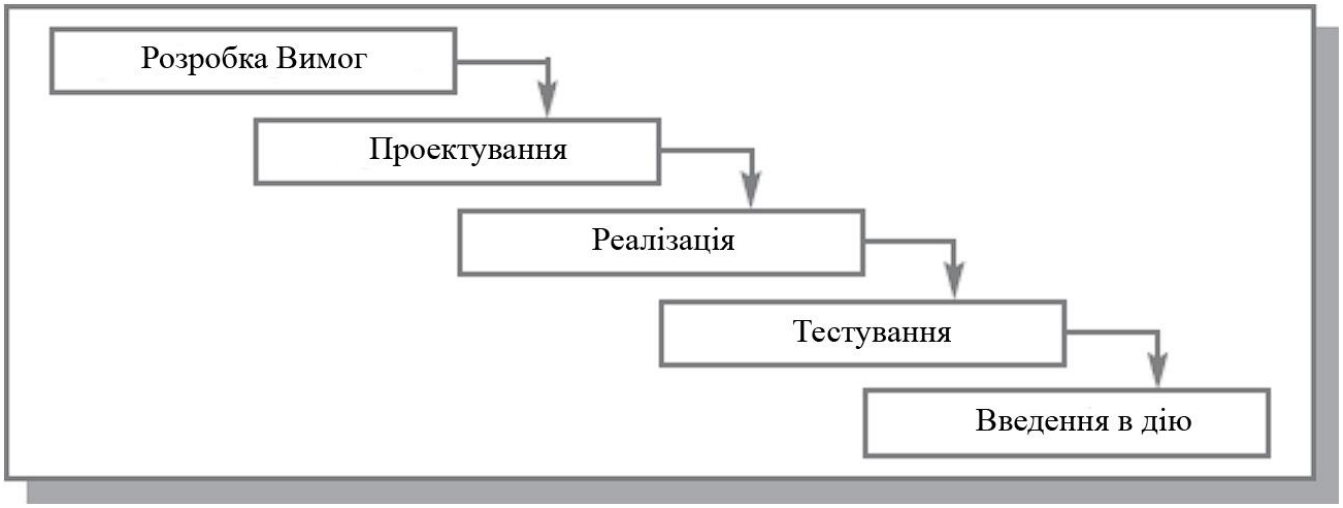


Рисунок 2.1 — Водоспадна модель життєвого циклу.

Вона була запропонована в 1970 році Уінстоном Ройсом, і передбачає послідовне виконання всіх етапів проекту в строго фіксованому порядку. Перехід на наступний етап означає повне завершення робіт на попередньому етапі. Вимоги, визначені на стадії формування вимог, суворо документуються у вигляді технічного завдання та фіксуються на весь час розробки проекту. Кожна стадія завершується випуском повного комплекту документації, достатньої для того, щоб технологія могла бути продовжена іншою командою розробників. Можна виділити такі етапи проекту відповідно до цієї моделі:

Формування вимог;

Проектування;

Реалізація;

Тестування;

Впровадження;

Експлуатація та супровід.

Також можна виділити ряд переваг і недоліків такої моделі. До переваг можна віднести:

Повна узгоджена документація на кожному етапі;

Легко визначити терміни та витрати на проект.

До недоліків каскадної моделі відноситься те, що перехід від однієї фази проекту до іншої передбачає повну коректність результату попередньої фази. Це означає, що неточність будь-якої вимоги або некоректна інтерпретація в результаті призводить до того, що доводиться повертатися до ранньої фази проекту. Такий «відкат» порушує командний графік та спричиняє зростання витрат.

Таким чином, ця модель може застосовуватись тільки для створення невеликих систем.

Ще одна модель життєвого циклу – поетапна модель із проміжним контролем. І тут розробка програмного продукту ведеться ітераціями з циклами зворотнього зв'язку між етапами.

Подібні міжетапні коригування дозволяють враховувати реально існуючий взаємовплив результатів розробки на різних етапах. Час життя кожного з етапів розтягується протягом усього періоду розробки.

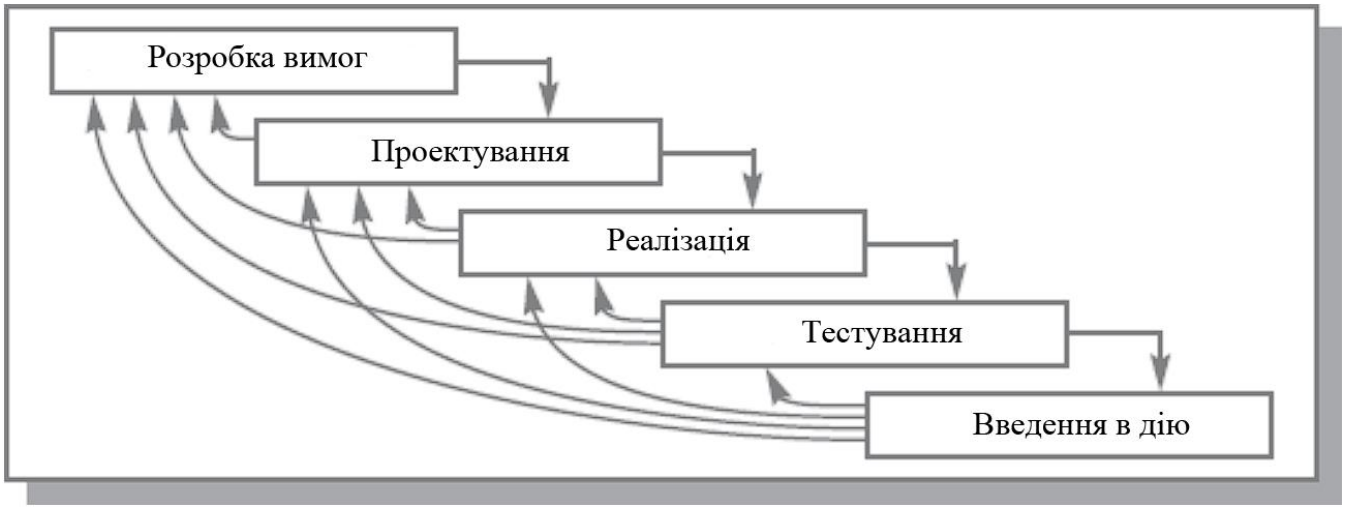


Рисунок 2.2 — Поетапна модель із проміжним контролем.

Альтернативою послідовної моделі є модель ітеративної та інкрементальної розробки. Вона передбачає розбиття життєвого циклу проекту на послідовність ітерацій, кожна з яких нагадує «міні-проект», включаючи всі процеси розробки щодо створення менших фрагментів функціональності, порівняно з проектами в цілому.

Кожна ітерація має на меті отримання працюючої версії програмної системи, що включає функціональність, визначену інтегрованим змістом усіх попередніх і

поточної ітерації. Результатом фінальної ітерації є вся потрібна функціональність продукту. Тобто в основі такої моделі лежить еволюційність.

У цьому випадку шанси на успішне створення складного програмного продукту будуть максимальні, адже спочатку кожен такий крок містить у собі певний успіх, а також можливість повернутися до попереднього успішного етапу у разі невдачі. Саме так розробник може отримати з навколишнього світу зворотний зв'язок та виправити можливі помилки у проекті. Але все ж таки такий підхід матиме й свої мінуси:

Дуже довгий час відсутнє цілісне розуміння можливостей та обмежень проекту;

При ітераціях доводиться відкидати частину виконаної раніше роботи.

І все-таки, у світі варіанти ітераційного підходу реалізовані у більшості сучасних методологій розробки. Розглянемо ще одну можливу модель життєвого циклу – спіральну модель .

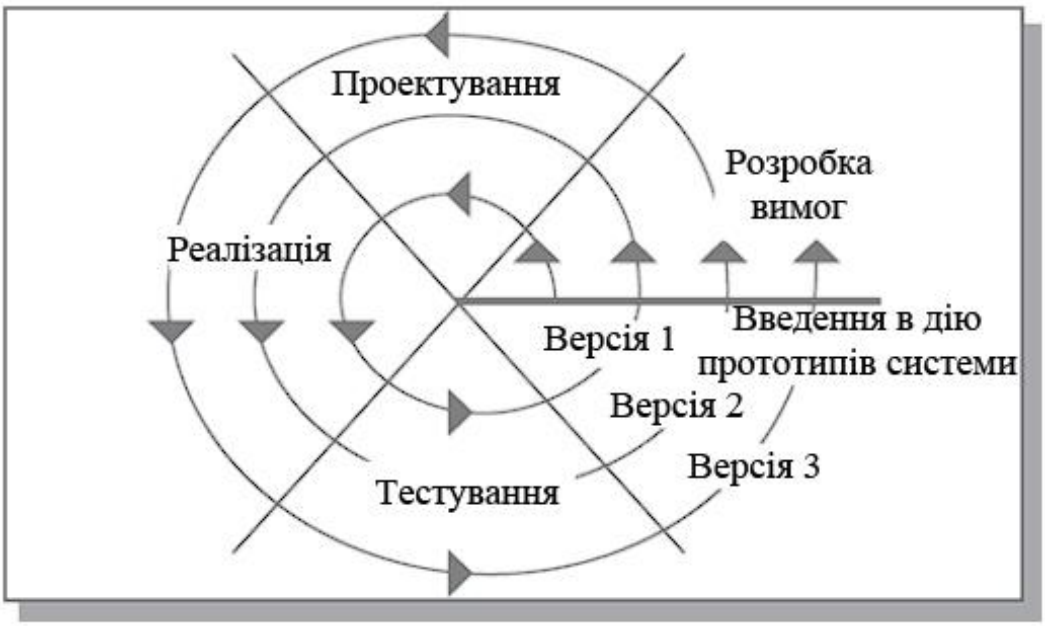


Рисунок 2.3 — Спіральна модель.

Ця модель була розроблена в середині 1980-х Баррі Боемом. Основується вона на класичному циклі Демінга PDCA (plan-do-check-act) - це повторюваний процес прийняття рішення, що є найпростішим алгоритмом дій керівника з управління процесом і досягнення його цілей. Починається такий цикл із планування – встановлення цілей та процесів, необхідних для досягнення цілей,

планування робіт з досягнення цілей процесу та задоволення можливого користувача, а також планування виділення та розподілу необхідних ресурсів. Наступний етап – виконання запланованих робіт. Після нього йде перевірка – збирання інформації та контроль результату на основі ключових показників ефективності, що вийшов під час виконання роботи, виявлення та аналіз відхилень, встановлення причин відхилень. Завершальний етап такого циклу – вплив, тобто вжиття заходів щодо усунення причин відхилень від запланованого результату, зміна у плануванні та розподілі ресурсів. При використанні такої моделі життєвого циклу програмного продукту створюється кілька ітерацій, у разі це витки спіралі, методом прототипування . Кожна така ітерація відповідає створенню фрагмента чи версії програмного продукту. Далі уточнюються цілі та характеристики проекту, оцінюється якість отриманих результатів та планується робота наступної ітерації.

Під час роботи з кожною ітерацією проводиться оцінка:

Ризик перевищення термінів та вартості програмного продукту;

Необхідність виконання ще однієї ітерації;

Ступінь повноти та точності розуміння вимог до системи;

Доцільність припинення проекту.

Відмінною особливістю спіральної моделі життєвого циклу програмного продукту є особлива увага, що приділяється ризикам, що впливають на організацію та контрольним точкам. Найбільш поширені ризики щодо формулювання Боема:

Дефіцит спеціалістів;

Нереалістичні терміни та бюджет;

Реалізація невідповідної функціональності;

Розробка неправильного інтерфейсу користувача;

Непотрібна оптимізація та відточування деталей;

Безперервний потік змін;

Нестача інформації про зовнішні компоненти, що визначають оточення системи або залучені в інтеграцію;

Недоліки у роботах, що виконуються зовнішніми ресурсами (стосовно програмного продукту);

Недостатня продуктивність підсумкової системи;

Розрив у кваліфікації спеціалістів різних областей.

Кожна із стратегій має безліч реалізацій на конкретних моделях. Всі ці стратегії та моделі будуть ефективні тільки при застосуванні до конкретного типу проектів, саме тому є класифікатори проектів з розробки, що дозволяють обґрунтовано вибирати модель життєвого циклу розробки програмного продукту для кожного конкретного випадку.

Моделі повного життєвого циклу

Багато формалізованих підходів, зокрема спіральна та каскадна, передбачають послідовне проходження програмного продукту за певними етапами життєвого циклу :

Постановка задачі;

Аналіз ризиків;

Аналіз вимог;

Побудова проектних специфікацій;

Проектування;

Реалізація;

Тестування;

Введення в експлуатацію та супровід;

Виведення з експлуатації.

Як відомо, такі моделі виходять дуже великі, навіть громіздкі, що створює проблеми в умовах змін вимог. Найчастіше зниження продуктивності колективу розробників спричиняє зміну графіка робіт та збільшення бюджету проекту. Сама ідеологія попереднього алгоритмічного проектування спочатку передбачає відсутність гнучкості, розробники що неспроможні оперативно реагувати на зміни вимог замовника. На жаль, багато колективів розробників вважають, що процес створення програмного забезпечення поки що є недостатньо всеосяжним і формалізованим, тим самим вони сприяють неконтрольованому зростанню різноманітних стандартів проектування, діаграм, нотацій та специфікацій. Так, наприклад, стандарт UML 2.0 (Unified Modeling Language) передбачає можливість

використання 14 типів діаграм, з яких практично зазвичай використовується не більше половини.

У сучасному світі під час розробки досить складного програмного продукту при використанні об'єктно-орієнтованого підходу передбачають насамперед наявність гнучкої та масштабованої архітектури, здатної оперативно реагувати на зміни потреб замовника. Адже, як відомо, замовник не завжди здатний чітко сформулювати вимоги до системи на етапі проектування. Звідси випливає той факт, що замовник не може скласти вимоги до інтерфейсу користувача, адже ці вимоги повинні залежати вже від кінцевих користувачів конкретно взятого програмного продукту і змінюватися в процесі під індивідуальну зручність і побажання користувачів. Звідси ж і випливає, що замовник спочатку не знатиме вимог і для кінцевого інтерфейсу.

Зворотний зв'язок на цьому етапі дуже важливий, адже його недостатність на стадії програмування веде до неконтрольованого зростання витрат на супровід.

Більшість моделей повного життєвого циклу програмного продукту, зрештою, досить надмірні, неефективні і занадто дорогі. Зрештою, документація та діаграми ніколи не зможуть усунути архітектурні недоліки дизайну системи, а також врахувати постійне накопичення та розвиток досвіду розробника.

Як стало зрозуміло з перерахованого вище, моделі повного життєвого циклу не ідеальні. Усі наявні недоліки було перераховано. І у зв'язку з цим були знайдені методики, які допоможуть позбавитися цих мінусів, і нижче буде опис таких способів розробки програмного забезпечення.

Гнучкі методики розробки програмного продукту.

До гнучких методик відносять такі відомі методики, як Crystal, Scrum, XP. Що ж це таке і чим вони кращі?

Гнучкі методики передбачають швидке постачання початкової версії працюючої програми замовнику для тестування, ігноруючи на початковому етапі багато проектних вимог і маючи на увазі подальше розширення функціональності програми.

Основною метою є забезпечення безперервності розвитку проекту з можливістю оперативного реагування на зміни вимог бізнесу.

Важкі моделі повного життєвого циклу програмного продукту занадто незграбні, щоб передбачити повний перегляд замовником свого «бачення» на етапі завершення проекту. Гнучкі методики пропонують колективам розробників ПЗ сконцентруватися на якості продукту, що випускається, на шкоду документації та іншим артефактам проектно-процесної діяльності.

Програмний продукт випускається короткими ітераціями, про які я згадував вище. Кожна з ітерацій закінчується певною версією, яка надходить в експлуатацію, а також використовується замовником для уточнення вимог та попереднього тестування. Розробники та замовники формують терміни та бюджет кожної ітерації, оцінюючи попередні досягнуті результати та необхідний приріст функціоналу від версії до версії. При цьому попередній збір детальної інформації про вимоги до системи, прийнятий у моделях повного життєвого циклу програмного продукту, буде марною тратою часу і сил, оскільки кінцева реальність може виявитися зовсім іншою. Конкретні деталі вимог згодом можуть і мають змінюватися у взаємодії з відповідальним представником замовника.

У зв'язку з цим у практичній діяльності доцільно оперувати поняттями життєвого циклу окремої складальної версії або ітерації, що дозволить сконцентрувати бачення замовників та розробників на функціональному, вужчому аспекті діяльності команди розробників. Таким чином, у гнучкій розробці етапи проектування, програмування та супроводу безперервні, тісно пов'язані та невіддільні один від одного, а також входять до складу окремої короткої ітерації розробки.

У результаті, у ході вивчення джерел з конкретної теми, було з'ясовано і описано визначення життєвого циклу програмного продукту, власне, що являє собою програмний продукт, які існують моделі життєвого циклу, і які класифікації та методи з ними пов'язані, а також як правильно вибрати стратегію розробки для конкретного проекту, на що потрібно звернути увагу, приймаючи таке рішення, і як це може позначитися на роботі команди надалі та успішності програмного продукту.

2.4. Розробка дизайн-концепції та макету додатку

Візуальний інформаційний дизайн

Інформаційні дизайнери працюють над візуалізацією даних, вмісту та засобів навігації. Зусилля інформаційного дизайнера спрямовані на те, щоб подати дані у формі, що сприяє їх правильному розумінню. Результат досягається через управління візуальною ієрархією за допомогою таких засобів, як колір, форма, розташування та масштаб. Поширеними об'єктами інформаційного дизайну є

всілякі графіки, діаграми та інші способи відображення кількісної інформації.

Щоб створювати привабливі та зручні для користувача інтерфейси, дизайнер інтерфейсів повинен володіти базовими візуальними навичками - розумінням кольору, друкарні, форми та композиції - і знати, як їх можна ефективно застосовувати для передачі поведінки та подання інформації, для створення настрою та стимулювання фізіологічних реакцій. Дизайнеру інтерфейсу також потрібне глибоке розуміння принципів взаємодії та ідіом інтерфейсу.

Будівельні блоки візуального дизайну інтерфейсів

Дизайн інтерфейсів зводиться до питання, як оформити і розмістити візуальні елементи таким чином, щоб виразно відобразити поведінку та подати інформацію. Кожен елемент візуальної композиції має ряд властивостей, і поєднання цих властивостей надає елементу сенс. Користувач отримує можливість розібратися в інтерфейсі завдяки різним способам застосування цих властивостей до кожного з

елементів інтерфейсу. У тих випадках, коли два об'єкти мають спільні властивості, користувач припустить, що ці об'єкти пов'язані чи схожі. Коли користувачі бачать, що властивості відрізняються, вони вважають, що об'єкти не пов'язані. Створюючи інтерфейс користувача, треба проаналізувати перелічені нижче візуальні властивості кожного елемента чи групи елементів. Щоб створити корисний і привабливий інтерфейс користувача, слід ретельно попрацювати з кожною з цих властивостей.

Форма

Форма – головна ознака сутності об'єкта для людини. Ми дізнаємось об'єкти за контурами. Якщо ми побачимо на картинці синій ананас, то ми його відразу пізнаємо, тому що ми пам'ятаємо його форму. І лише потім ми здивуємося дивним кольором. При цьому розрізнення форм вимагає більшої концентрації уваги, ніж аналіз кольору чи розміру. Тому форма - не найкраща властивість для створення контрасту, якщо потрібно залучити увагу користувача.

Розмір

Більші елементи привертають більше уваги, особливо якщо вони значно перевершують розмірами оточуючі елементи. Люди автоматично впорядковують об'єкти за розміром та схильні оцінювати їх по розміру; якщо у нас є текст у чотирьох розмірах, передбачається, що відносна важливість тексту зростає разом з розміром і що жирний текст важливіший, ніж текст із нормальним накресленням.

Колір

Колірні відмінності швидко привертають увагу. В деяких професійних областях кольори мають конкретні значення, і цим можна користуватися. Так, для бухгалтера червоний колір – негативні результати, а чорний – позитивні.

Кольори набувають сенсу і завдяки соціальним контекстам, у яких відбувається наше дорослішання. Наприклад, білий колір на Заході асоціюється з чистотою та світом, а в Азії та арабських країнах – з похороном та смертю. При цьому колір спочатку не має властивості впорядкованості і не виражається кількісно, тому далеко не ідеальний для передачі такої інформації. Крім того, не слід робити колір єдиним способом передачі інформації, оскільки колірна сліпота трапляється досить часто. Треба застосовувати колір із розумом. Щоб створити ефективну візуальну систему, що дозволяє користувачеві виявляти подібності та відмінності об'єктів. Слід використовувати обмежений набір кольорів - ефект веселки перевантажує сприйняття користувача та обмежує можливості щодо передачі йому інформації.

Вибір палітри кольорів для програми необхідно проводити дуже обережно. За різними даними, тією чи іншою формою колірної сліпоти страждають до 10% чоловіків, і використання, наприклад, червоного та зеленого кольорів для вказівки контрасту ускладнює роботу з додатком для цих людей.

Яскравість

Поняття темного і світлого набувають сенсу переважно в яскравість фону. На темному тлі темний текст майже не видно, тоді як на світлому він різко виділятиметься. Контрастність люди сприймають легко і швидко, тому значення яскравості може стати хорошим інструментом привернення уваги до тих елементів, які потрібно підкреслити. Значення яскравості - також упорядкована змінна, наприклад, темніші (з нижчою яскравістю) кольори на карті легко інтерпретуються: вони позначають великі глибини чи великі значення інших параметрів.

Напрямок

Напрямок корисний, коли потрібно передавати інформацію про орієнтації (вгору або вниз, вперед або назад). Треба пам’ятати, що сприйняття напрямку може бути утруднено у разі деяких форм і при малих розмірах об'єктів, тому її краще використовувати як вторинну ознаку. Так, якщо потрібно показати, що ринок акцій пішов униз, можна використовувати спрямовану вниз стрілку червоного кольору.

Текстура

Зрозуміло, зображені на екрані елементи не мають справжньої текстури, але здатні створювати її видимість. Текстура рідко буває корисна для передачі відмінностей або залучення уваги, оскільки вимагає значну концентрацію на деталях. Проте текстура може бути важливою підказкою. Засічки та опуклості на елементах інтерфейсу зазвичай вказують, що елемент можна перетягувати, а фаски або тіні біля кнопки підсилюють відчуття, що її можна натиснути.

Розташування

Розташування - це змінна, впорядкована та виражається кількісно, отже, корисна в передачі ієрархії. Розташування також може бути засобом створення просторових відносин між об'єктами на екрані та об'єктами реального світу (наприклад, небо в верхній половині, земля в нижній).

Розташування елементів мобільного додатка дуже сильно впливає на зручність використання та залежить від того, як користувач буде тримати пристрій.

Елементи управління та дизайн навігації

Елементи управління – це доступні для маніпулювання самодостатні екранні об'єкти, за допомогою яких люди взаємодіють із цифровими продуктами. Елементи - це базові будівельні блоки графічного інтерфейсу користувача.

Розглядаючи елементи управління з урахуванням цілей користувача, їх можна розбити на чотири основні категорії:

- командні елементи керування, які застосовуються для виконання функцій;
- елементи вибору, які дозволяють вибирати дані або налаштування;
- елементи введення, які застосовуються для введення даних;
- елементи відображення, які використовуються для наочного безпосереднього маніпулювання.

Деякі елементи управління поєднують у собі властивості більш ніж однієї категорії.

Командні елементи керування

Командні елементи управління виконують дії, причому роблять це негайно. Головним і насправді єдиним командним елементом є кнопка, яка має безліч варіантів відображення. Елементи меню є також командними ідіомами.

Кнопки

Кнопки зазвичай легко пізнаються завдяки їх псевдотривимірності. Дія виконується одразу після натискання на кнопку. Часто особливим чином виділяється кнопка за замовчуванням, відповідна найчастіше використовуваної дії.

Кнопки здаються опуклими, а натиснута кнопка змінює колір.

Кнопка - зручний та привабливий з візуального погляду елемент управління. Весь її вигляд підказує, що на неї можна натиснути, і це характеризує її очікуване призначення. Рекомендується змінювати зовнішній вигляд натиснутої кнопки, оскільки це полегшує розуміння роботи програми користувачем.

Кнопки-значки

Кнопки, розміщені на панелі інструментів, зазвичай стають квадратними, втрачають текстовий напис та обзаводяться піктограмою - поясненням як графічного значка.

Вважається, що кнопки-значки дуже зручні: вони постійно на увазі взаємодіяти з ними простіше, ніж з елементами меню. Оскільки вони постійно помітні, то легко запам'ятовуються. У більшості користувачів не виникає питань щодо очікуваного призначення кнопки. Проблема в тому, що зображення на кнопці іноді буває незрозумілим. Наприклад, піктограма із зображенням дискети, традиційно означає збереження, часто незрозуміла молодим користувачам, які ніколи не працювали із реальними дискетами.

Гіперпосилання

Текстові гіперпосилання - поширений спосіб навігації в мережі та веб-програм, але при програмуванні для мобільних пристроїв їх слід уникати. Справа в тому, що потрапити по посиланню пальцем з першого разу часто важко і користувачів дратує необхідність повторення цих дій.

Елементи керування вибором

Елементи вибору дозволяють користувачеві вибрати із групи допустимих об'єктів той, з яким буде скоєно дію. Елементи вибору також застосовуються для дій з налаштування. Поширеними елементами вибору є прапорці та списки. Раніше використання елементів керування вибором не призводило до негайного виконання дій - була потрібна ще й активація.

Командний елемент.

Наразі можливі обидва варіанти. Якщо бажано дати користувачу можливість кілька разів здійснити вибір перед виконанням дії слід створити явний командний елемент керування (кнопку). Якщо ж користувачеві корисно одразу бачити результат своїх дій, і ці дії легко скасувати, розумно зробити так, щоб елемент вибору грав також роль командного елемента.

Прапорці.

Призначення прапорця є очевидним. Натиснувши на прапорець, користувач негайно побачить галочку, що з'явилася. Прапорець простий, наочний і витончений, однак ґрунтується на тексті. Якісний текст може виключити можливість неоднозначного розуміння прапорця. Однак цей же пояснювальний текст змушує користувача сповільнюватися для прочитання, а також займає значний екранний простір.

Зазвичай прапорці мають квадратну форму. Треба пам’ятати, що користувачі розпізнають візуальні об'єкти за формою, та квадратна форма прапорців – важливий стандарт.

Вимикачі.

Існує можливість зробити прапорець наочнішим, застосувавши в ньому як основу кнопку-значок, яка може фіксуватися в натиснутому стані. Такий елемент називається вимикачем

Стан вимикача залишається незмінним до наступного натискання. Вимикачі економно витрачають екранний простір: вони займають менше місця, тому що їхнє призначення описується не за допомогою тексту, а за допомогою візуальних засобів. Зрозуміло, це означає, що їм притаманний той самий недолік стандартних клавіш-значків - неоднозначність піктограм.

Тригери.

Кнопки-тригери – це різновид елементів керування. Вони покликані заощаджувати екранний простір, на жаль, ціною значної дезорієнтації користувача.

Класичний приклад - розміщення на одній кнопці функцій відтворення та паузи для музичного плеєра. Підводним каменем такого підходу є те, що елемент управління можна помилково вважати індикатором стану програвача ("на паузі" або "йде відтворення"). Елемент управління може бути або індикатором стану, або кнопкою перемикання станів, але не тим і іншим водночас.

Радіокнопки.

Радіокнопки зовні схожі на прапорці, але є взаємовиключними, тобто вибір одного з варіантів автоматично анулює попередній вибір. У кожний момент часу може бути вибрано лише одну кнопку. Радіокнопки завжди об'єднуються в групи з

двох або більше радіокнопок, причому в кожній групі одну радіокнопку завжди вибрано. Радіокнопки завжди круглі з тієї ж причини, з якої прапорці завжди мають квадратну форму: саме такими вони були початково.

Радіокнопки займають навіть більше місця, ніж прапорці, однак у деяких випадках така витрата екранного простору виправдана. Кнопка-значок перетворила радіокнопки так само, як прапорці, замінивши їх у основному інтерфейсі програми. Якщо два або більше вимикачі об'єднані схемою взаємного виключення - так, щоб у кожний момент міг бути включений лише один з них, - вони поводяться точно так, як радіокнопки.

Так утворюються радіокнопки із значками. Елементи управління кольором в

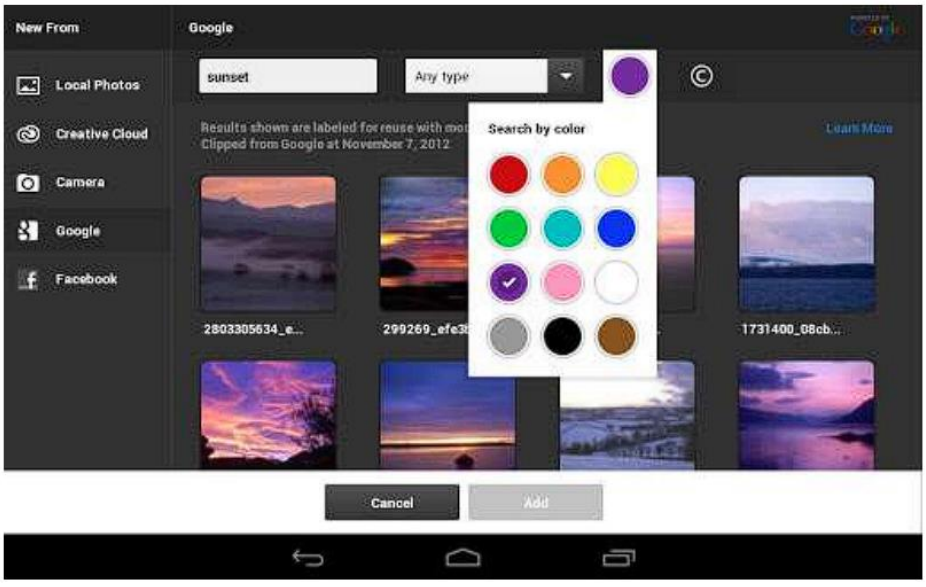


Рисунок 2.4 — Приклад з Adobe Photoshop

Adobe Photoshop - хороший приклад радіокнопок з значками.

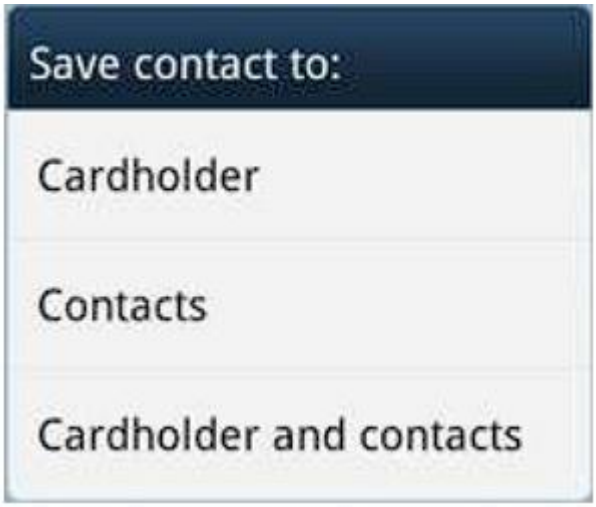


Рисунок 2.5 — Стандартний список в Android

Списки. Елементи керування типу "список" дозволяють здійснювати вибір з кінцевої множини текстових рядків, кожен з яких представляє команду, об'єкт чи ознаку. Подібно до радіокнопок, списки – потужний інструмент, що спрощує взаємодію за рахунок усунення можливості на неправильний вибір. Списки – це невеликі текстові області зі смугою прокручування, що автоматично підключається при необхідності.Користувач може вибрати один рядок тексту, натиснувши на нього.

Список, що розкривається - варіант, що зустрічається повсюдно стандартного списку. Він показує лише вибраний елемент в один рядок, але Якщо натиснути на стрілку, відкриваються інші варіанти вибору. Елемент управління подання у вигляді списку надає можливість супроводжувати кожен рядок тексту з піктограмою. Така можливість дуже корисна - існує безліч ситуацій, коли можна

спростити роботу користувача, маючи графічні ідентифікатори поряд із рядками важливих варіантів вибору.

Комбо-списки та комбо-кнопки.

Комбо-елементи є поєднанням елементів. Комбокнопка – різновид радіокнопки зі значком. Зазвичай вона виглядає як кнопка-значок з невеликою стрілкою, але якщо натиснути на стрілку та утримувати її в натиснутому стані, розгортається меню. Комбо-список є поєднанням списку і поля редагування.

Варіант із списком, що розкривається, значно економить екранний простір. Комбо-список добре підходить для тих випадків, коли необхідно організувати вибір єдиного об'єкта.

Елементи введення.

Елементи введення дають користувачеві можливість не лише вибирати існуючі відомості, але й запроваджувати нову інформацію. Найпростіший елемент – поле редагування тексту (поле введення). До цієї категорії потрапляють також такі елементи управління, як лічильники та повзунки.

Обмежувальні елементи введення.

Будь-який елемент управління, що обмежує набір значень, доступних для введення користувачем є обмежуючим елементом введення. Так, наприклад, повзунок зі шкалою значень від 0 до 100 є обмежуючим елементом введення. Незалежно від дій користувача не може бути введено число, що виходить за діапазон певних значень. Простіше кажучи, обмежуючі елементи введення використовуватися скрізь, де необхідно обмежити безліч допустимих значень.

Обмежуючий елемент введення повинен чітко інформувати користувача про допустимі межі. Текстове поле, яке відкидає введення користувача після того, як він виконав введення, не може вважатися обмежуючим елементом управління. Якщо користувач має висловити вибір числовим значенням у певних межах, слід надати йому елемент управління, що повідомляє про ці межі і запобігає введенню

неприпустимих значень. Але для введення точних значень краще підходять лічильники.

Лічильники.

Лічильник складається з невеликого поля введення та двох прикріплених до нього кнопок. Завдяки лічильникам грань між обмежуючими та необмежуючими елементами введення даних стає розмитою. Маленькі кнопки зі стрілками дозволяють користувачеві змінювати значення у полі редагування невеликими кроками. Ці кроки можуть виконуватися до певної межі: значення не може

перевищити максимум, встановлений програмно, або стати меншою за встановлений мінімум. Якщо користувач захоче ввести певну кількість, він може зробити це за рахунок прямого введення числа у полі редагування.

Рукоятки та повзунки.

Рукоятки та повзунки дуже ефективно витрачають екранний простір, і обидва ці елементи управління чудово справляються з завданням забезпечення візуального зворотного зв'язку за налаштуваннями Повзунки та рукоятки застосовуються в основному як обмежуючі елементи керування введення. Наприклад, повзунки - чудовий засіб для дій, пов'язаних із масштабуванням.

Необмежуючі елементи введення

Мабуть, головний елемент, що не обмежує введення, - поле введення тексту.

Цей найпростіший елемент керування дозволяє користувачам набирати будь-які алфавітно-цифрові рядки. Як правило, поля введення – це невеликі області, всередині яких можна набрати одне-два слова, але вони можуть бути реалізовані у вигляді досить складних текстових редакторів. Коли користувачеві запропоновано текстове поле, що не обмежує введення, яке при цьому приймає лише рядки певного формату, ймовірно, є необхідність допомогти користувачам вводити "допустимі" рядки. Є безліч стандартних форматів, що вводяться - дати, телефонні номери, поштові індекси, номери соціального страхування. Ключ до успішного проектування елемента введення з перевіркою даних - у добре розвиненому зворотному зв'язку з користувачем.

Елементи керування відображенням. Елементи керування відображенням використовуються для керування візуальне подання інформації на екрані. Типовими прикладами елементами відображення є роздільники та смуги прокручування. Сюди ж входять роздільники сторінок, лінійки, напрямні, сітки та рамки.

Текстові елементи. Ймовірно, найпростіший елемент управління відображенням – елемент виводу текстової інформації, який відображає текстове повідомлення в деякій позиції на екрані. Він надає текстові мітки для інших елементів управління та виводить дані, які не можуть або не повинні бути змінені користувачем. Єдина серйозна проблема цього елемента полягає в тому, що він часто використовується там, де мають бути присутні елементи введення (і навпаки).

Смуги прокручування.

Смуги прокручування служать важливою метою - вони дозволяють осмисленим чином поміщати великі обсяги інформації всередині рамок вікон і панелей. На жаль, вони витрачають екранний простір та ними складно маніпулювати. Однак чудова перевага смуги прокручування полягає у створенні контексту поточного положення у вікні. Бігунок смуги Прокручування вказує поточне положення і нерідко масштаб "території", доступною для прокручування.

Розділювачі.

Розділювачі – зручний інструмент для поділу головного вікна програми на кілька пов'язаних між собою панелей, у кожній із яких можна переглядати, змінювати або переносити ту чи іншу інформацію. Рухливі роздільники завжди повинні повідомляти про свою рухливості у вигляді зміни форми курсора. Однак слід виявляти обережність, вибираючи, які саме роздільники мають стати рухливими. У загальному випадку роздільник не повинен переміщатися таким чином, щоб вміст панелі ставав непридатним до використання.

Висувні панелі.

Висувні панелі – це панелі програми, які можна відкривати та закривати в одну дію. Висувні панелі - чудове місце для елементів управління та функцій, які використовуються спільно з основною робочою областю програми, але не настільки часто. Висувні панелі зручніші, ніж діалогові вікна, так як не закривають головне вікно

2.5. Архітектура додатку

Потрібно сказати, що архітектура Android-програм зазвичай не буває надто складною, тому, можливо, схему з Clean Architecture можна спростити без втрати якості. І цю схему у будь-якому разі потрібно якось адаптувати для конкретного використання Android.

Така адаптація була викладена у статті Фернандо Цехаса “Architecting Android… The clean way?”, яку я детально розберу.

Найбільш критичною з точки зору тестування програми є бізнес-логіка або бізнес-правила, що визначають суть роботи програми. І вони мають бути насамперед незалежні від інших елементів та протестовані.

Щоб досягти незалежності та можливості тестування, пропонується розбити додаток на 3 ключові шари:

Шар даних (Data Layer)

Шар бізнес-логіки (Domain Layer)

Шар представлення (Presentation Layer)

При цьому щоб забезпечити максимальну незалежність цих шарів,у кожному з них використовується своя модель даних, яка конвертується при взаємодії між шарами.

Схема цих шарів виглядає так:

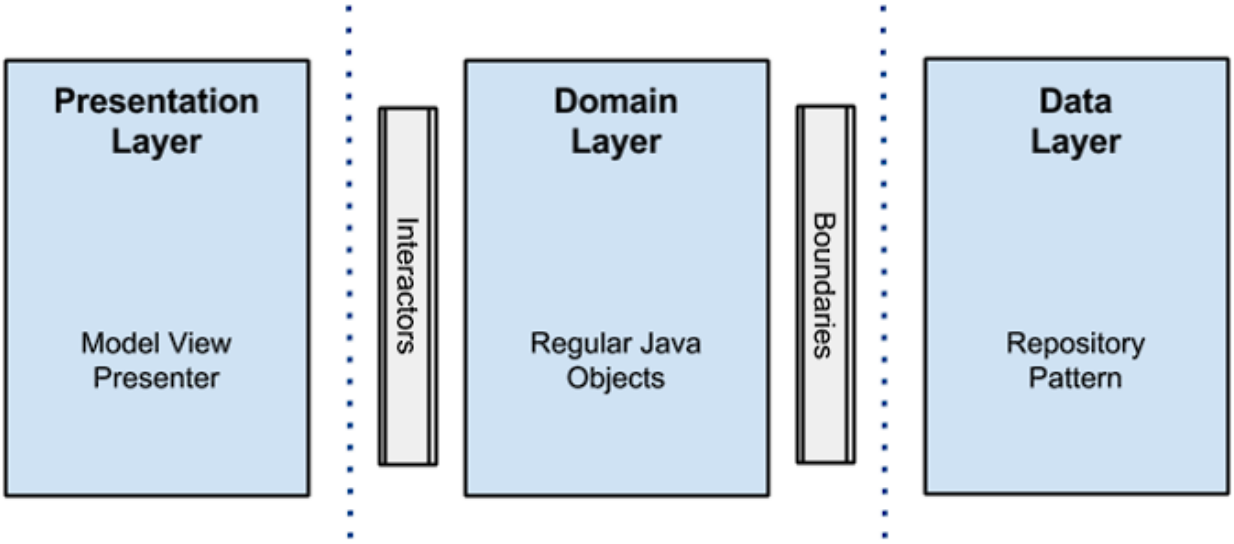


Рисунок 2.6 — Схема шарів в додатку.

Основна ідея всіх архітектурних патернів зберігається – ми розбиваємо код на модулі на кілька шарів, серед яких обов'язково є шар, що містить логіку програми, та шар, який відповідає за представлення даних та роботу з UI.

У запропонованому варіанті від Цехасу до цих двох основних шарів додається шар для роботи з даними. Це дозволяє спростити клієнт-серверну взаємодію та позбавляє бізнес-логіку програми від необхідності працювати з отриманням даних із різних джерел.

Розглянемо зазначені шари докладніше.

Шар даних

Цей шар відповідає насамперед отримання даних з різних джерел та його кешування. Він реалізується за рахунок патерну Repository, і його загальну схему можна представити так:

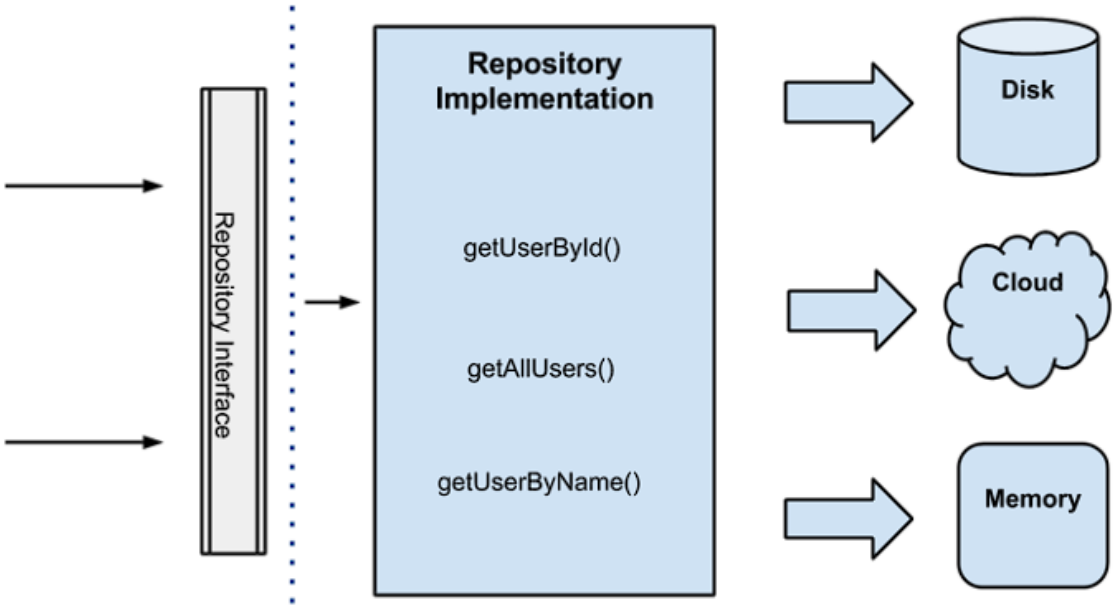


Рисунок 2.6 — Схема шару даних

Наприклад, коли необхідно отримати якогось користувача з певним ім’ям, реалізація репозиторію перевіряє наявність цього користувача в локальному сховищі, за відсутності збереженого користувача вона відправляє запит на сервер і отримує відповідь, яка зберігається в локальне сховище і повертається. Або, наприклад, репозиторій може завжди звертатися до сервера, а вже у разі помилки повертати збережений результат.

Існує кілька переваг використання такого підходу. По-перше, інші шари, які запитують дані, не знають про те, звідки ці дані надходять. Більше того, їм не

потрібно цього знати, оскільки це ускладнює логіку роботи і модуль перебирає зайву відповідальність. По-друге, шар даних у разі виступає єдиним джерелом інформації, що, як обговорювалося раніше, дуже зручно.

Шар бізнес-логіки

На шарі бізнес-логіки міститься, як не дивно, вся бізнес-логіка програми. Цей шар є об'єднанням шарів сценаріїв взаємодії та бізнес-логіки в оригінальній “чистій” архітектурі. Саме до цього шару звертається шар представлення для виконання запитів та отримання даних.

Фернандо Цехас пропонує реалізовувати шар бізнес-логіки як Java-модуля, який не містить ніяких залежностей від Android-класів. І це хороший підхід, тому що для реалізації бізнес-логіки нам потрібні тільки класи моделей та стандартні засоби Java. Більш того, такий підхід дозволить легко тестувати цей шар за допомогою звичайних тестів JUnit, що дуже зручно. У такому разі іноді не буде можливості виконати будь-який метод або використовувати деякі класи інших шарів. Тому для взаємодії із цим шаром використовуються інтерфейси.

Шар уявлення

І, зрозуміло, додаток - це насамперед взаємодія з користувачем. Тому нам потрібен спеціальний шар, який відповідатиме за логіку відображення даних на екрані, за взаємодію з користувачем та інші процеси, пов'язані з UI. Цей шар не повинен містити логіку, не пов'язану з UI.

Саме цей шар прив'язується до екранів та допомагає організувати взаємодію з шаром бізнес-логіки та роботу з даними. Цей шар може бути реалізований з використанням будь-якого паттерну, наприклад, MVC, MVP, MVVM та інших.

Звичайно, найчастіше використовується патерн MVP. Це досить простий патерн, який дозволить мені розділити екран на UI-частину (View), на логіку роботи з UI (Presenter) та об'єкти для взаємодії з UI (Model). У загальному вигляді цей патерн виглядає так:

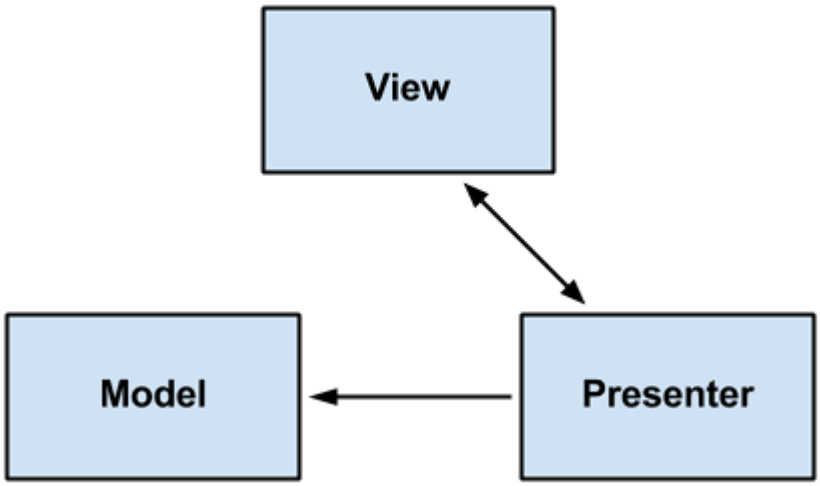


Рисунок 2.7 — Схема патерну MVP

І, зрозуміло, при розділенні додатка на такі частини не можна забувати про головне правило з "чистої" архітектури - правило залежностей:

Тепер зрозуміло, як розбити програму на модулі і зробити ці модулі відносно незалежними один від одного, але потрібно прояснити ще три питання, які у постійно виникають при побудові архітектури клієнт-серверної програми.

По-перше, це питання про тестування. Так, якщо ми будуємо додаток у вигляді незалежних модулів, їх легко тестувати. Але як саме здійснюється це тестування на практиці?

Очевидно, що найважливіший шар з погляду тестування – це бізнес-логіка. Оскільки я використовую його у вигляді звичайного Java-модуля, то тестувати його дуже просто. Для цього достатньо стандартного фреймворку JUnit і, можливо, Mockito. Тут немає проблем, специфічних для Android.

Шар даних також тестуються за допомогою Unit-тестів на JUnit, але це вже потребує певних зусиль, оскільки цей модуль містить залежність від класів Android. Наприклад, для тестування цього шару можна використовувати фреймворк Robolectric.

І останній шар, шар уявлення, не містить критичної логіки, зате містить багато UI-елементів, які також мають бути протестовані. Для цього можна скористатися спеціальними фреймворками, які дозволяють проводити інтеграційне та UI-тестування додатків, наприклад Espresso.

Друге питання пов'язане з обробкою помилок. Помилки в такій схемі можуть виникати на будь-якому рівні, але найбільше цікавлять помилки сервера, які виникають на рівні даних. Тут є кілька можливих рішень:

У разі помилки при отриманні даних можна дістати дані локального сховища. Досить часто така політика цілком виправдана, але при цьому все одно варто якось повідомляти користувача про помилку, що відбулася.

Передавати помилку через механізм зворотного дзвінка. При отриманні помилки у шарі даних шар бізнес-логіки можно виконати її первинну обробку та передати її шару уявлення, що й відобразить помилку користувачеві. Цей підхід можна використовувати у поєднанні з першим варіантом.

І останнє відкрите питання – це питання управління життєвим циклом. Оскільки шар даних ніяк не відноситься до проблем перестворення Activity та інших проблем життєвого циклу, а шар бізнес-логіки взагалі нічого не знає про систему Android, то очевидно, що обробкою життєвого циклу повинен займатися рівень уявлення.

2.6. Розробка технічного завдання

Складання технічного завдання та вибір методології

Тож з чого все починається? Спочатку клієнт – майбутній замовник, приходить з ідеєю мобільного додатку. Для замовника важливо розуміти, що саме він хоче досягти розробкою програми, а розробники вже допоможуть визначити, які обмеження для цієї розробки.

На сьогоднішній день дуже важливо сформулювати технічне завдання. Це дозволяє дізнатися, що конкретно хоче замовник, які цілі та завдання майбутнього додатка, хто є кінцевими користувачами та багато іншого, що дуже важливо на етапі проектування, та й надалі.

Тут вже буде зрозуміло, яку саме модель – каскадну чи гнучку, потрібно буде використати. Адже ці підходи дуже різні, як я описав вище, тому важливо вже зараз визначитися, вибирається каскадна модель, де продукт розробляється відразу, або гнучка модель, де додаток розробляється ітераціями, кожна з яких поєднує всі перелічені стадії розробки.

У результаті, саме на даному етапі формується опис базових функцій мобільного додатка, стає зрозумілим, для якої платформи створюватиметься програмний продукт: iOS, Android або крос-платформа, і, звичайно, вибирається методологія: Agile або Waterfall.

Планування та оцінка

На цьому етапі складається план робіт з оцінки необхідних ресурсів, як матеріальних, і тимчасових. Дуже важлива роль менеджера проекту в цей момент, оскільки саме він координуватиме роботу команди та спілкуватиметься із замовником.

Також проводиться оцінка технічного завдання. Це необхідно, щоб обговорити із замовником усі незрозумілі місця, не описані сценарії в отриманому завданні. Експрес-оцінка може займати всього кілька годин або день. Вона допомагає визначити зразкове уявлення про трудовитрати. Для детальнішої оцінки потрібно набагато більше часу – від кількох днів до тижня. Після такої оцінки вже точно можна визначити, як, коли і яка програма вийде в результаті роботи. На даному етапі так само важлива участь бізнес-аналітика. Саме він може спростити роботу замовника та розробників та отримати єдине уявлення про додаток, все точно розрахувати.

Є момент, який також потрібно враховувати – замовник може змінити вимоги у процесі розробки продукту. Це означає, що кількість завдань може значно збільшитись, що вплине як на перенесення дати релізу, так і на бюджет проекту.

У результаті, після цього етапу розробники вже точно знають певні завдання, які вони повинні виконати, щоб досягти бажаної мети у розробці заданого програмного продукту, а також замовнику зрозумілий бюджет проекту

Аналітика

Даний етап не завжди входить у процес розробки мобільного додатка, так як клієнти, а саме замовник, може самостійно виконати роботу по цьому етапу, але все ж таки краще цей етап залишити розробникам, так як це буде вже професійна робота, отже успіх планованого додатка буде набагато вище.

Після проведення цього етапу на виході формуються:

Специфікація функціональних вимог;

Специфікація дисфункційних вимог;

Основа графічного інтерфейсу;

План проекту;

Детальний бюджет.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ МОБІЛЬНОГО ДОДАТКУ

3.1. Визначення задач та вимог до додатку

Для того щоб нам чітко усвідомлювати в якому напрямку рухатись далі, нам необхідно скласти список вимог, яким має відповідати наш додаток.

Технічні вимоги:

1. Відповідати розробленому дизайну
2. Працювати без серйозних помилок
3. Працювати з Android 7.0
4. Працювати без візуальних затримок

Функціональні вимоги:

Користувач заходить у додаток, без будь-якої авторизації, як тільки програма завантажитися має з'явитися розділ Магазин.

Для переходу в інші розділи повинні бути відповідні кнопки, з інтуїтивно зрозумілими картинками та назвами розділів.

У розділі магазину має бути можливість прокручування товарів, у блоці кожного товару повинна бути відповідна ціна та кнопка, яка при натисканні буде добавляти його у корзину. Також повинні бути кнопки, які будуть сортувати товари за категорією та кнопка корзини.

Кнопка корзини при натисканні повинна переносити нас на екран зі списком всіх товарів, доданих у кошик та їх загальною вартістю. Також у користувача має бути можливість видаляти товари зі списку.

На екрані Профіль при його відкритті повинен з'явитися список усіх колись зроблених заказів.

Згідно з вище перерахованими вимогами, які ми визначаємо для нашої мобільної програми можна провести аналіз який допоможе нам ефективно, швидко і без зайвих фінансових та людських витрат здійснити комплексну, самодостатню розробку програми.

Додаток можна буде легко розширювати, підтримувати, утримувати легко програмний код, що читається, буде зрозумілий людям, які будуть його розширювати та підтримувати надалі.

3.2 Середовище створення дизайну Figma

Перед тим, як ми почнемо розробляти сам додаток треба зробити прототип дизайну для нього. В цьому нам допоможе сервіс Figma. Він пропонує безліч інструментів для створення найважких та привабливих дизайнів з можливістю потім експортувати якісь частини дизайну у вигляді векторної графіки, чи навіть увесь дизайн цілком.

Для початку переходимо на сайт www.figma.com , де жмемо конпку «Try Figma for free», потім нам потрібно пройти процедуру реєстрації, після чого мі зможемо скачати та установити додаток Figma на свій комп'ютер. Додаток зустрічає нас привабливим інтерфейсом з можливістю вести бесплатно лише один проект, але для нас це в самий раз. Створюємо проект та новий дизайн-файл.

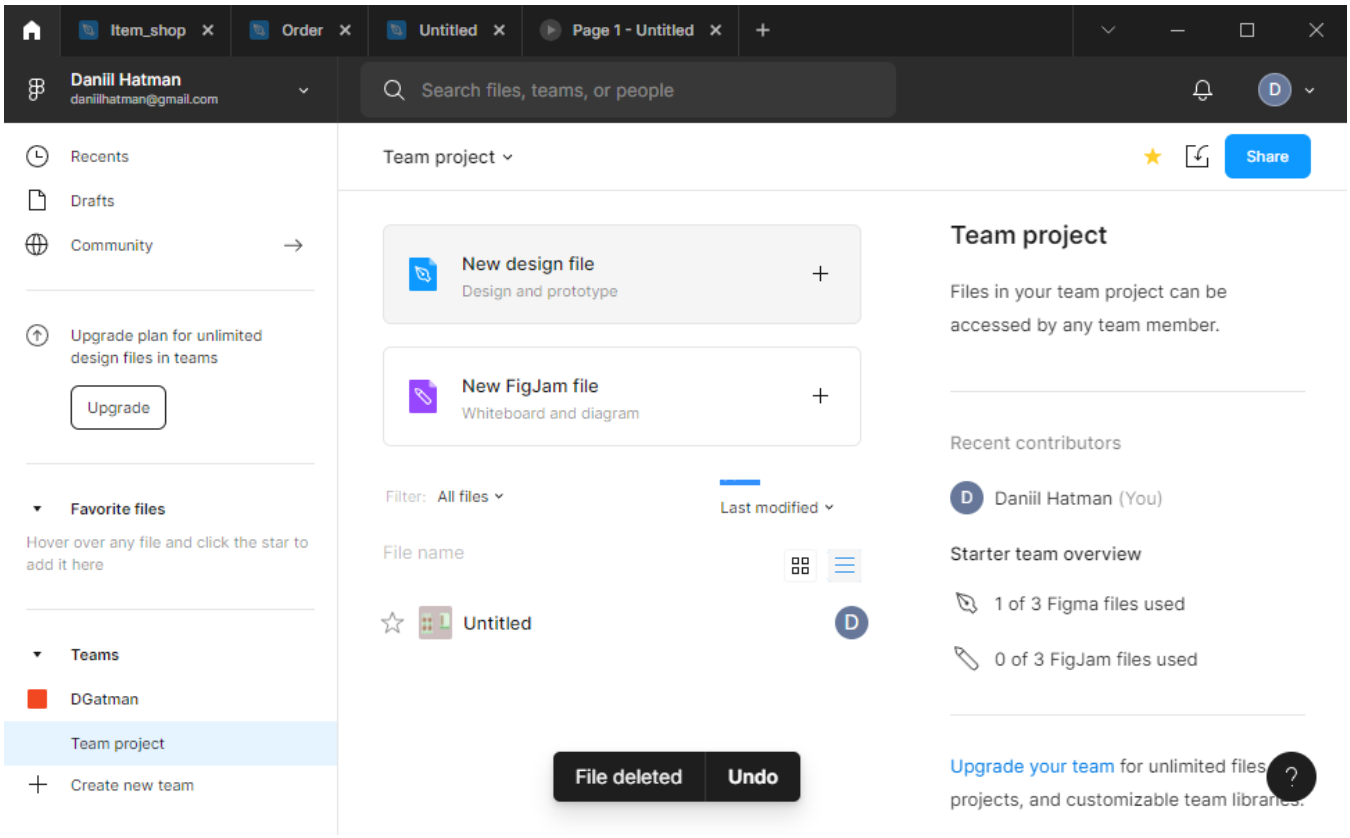


Рисунок 3.1 — Стартове меню Figma

Після створення дизайн-файлу ми бачемо пусте середовище. Нам потрібно створити наш перший екран, для цього у верхньому меню натискаєио на «Frame», та у меню справа обираємо потрібний нам розмір макету, у нашому випадку це буде «Android Large»

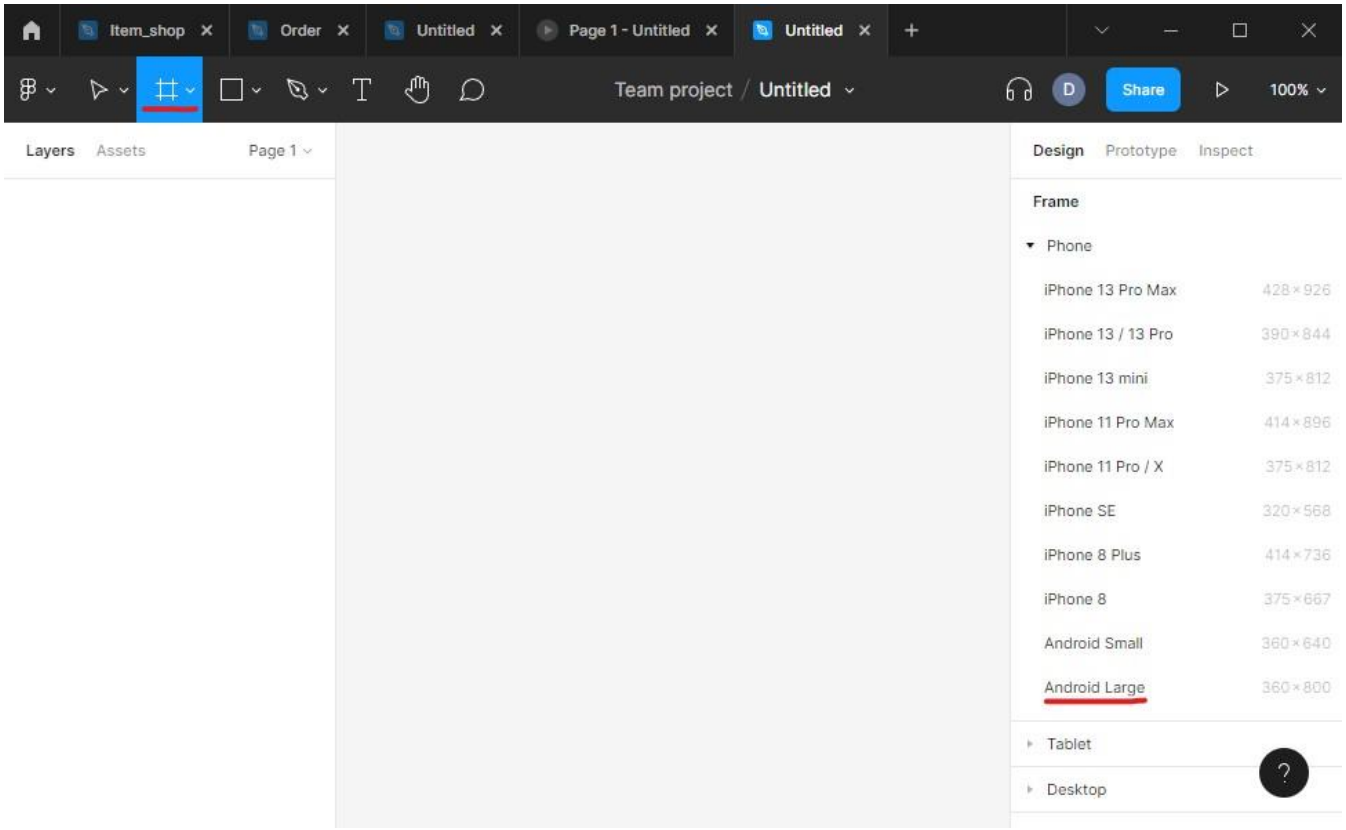


Рисунок 3.2— Створення проекту

Далі я створив екран магазину, на якому я розташував кнопки, які будуть відповідати за фільтрацію товарів у магазині, кнопку переходу до екрану корзини та стрічку зверху,яка буду відібражати розділ,у якому ві зараз знаходитеся. Знизу я зробив панель з кнопками магазину та профілю. Посередині я залишив місце для самого списку товарів.

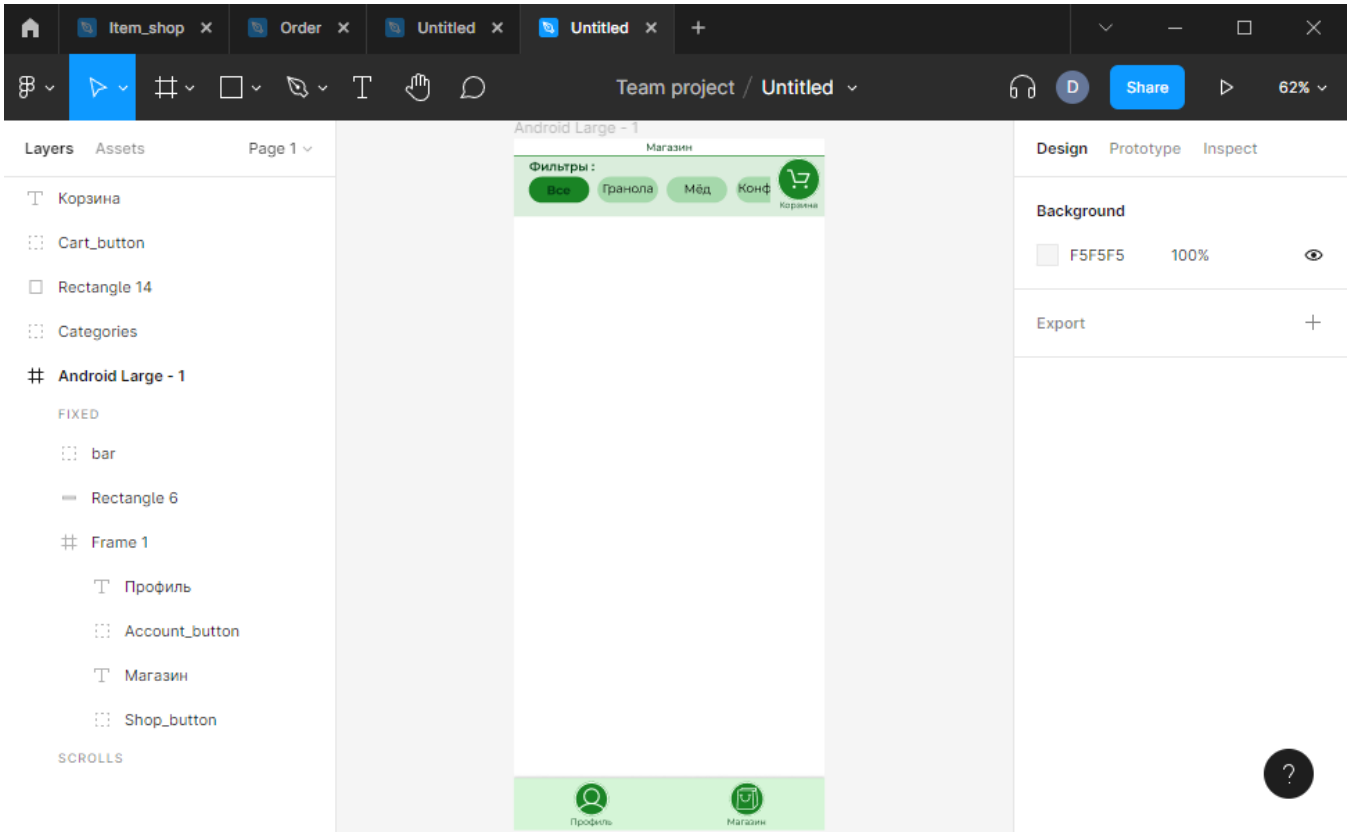


Рисунок 3.3 — Дизайн екрану магазину

Залишилося зробити дизайн товару в магазині, екран корзини та екран профилю.

Рисунок 3.4 — Дизайн товару

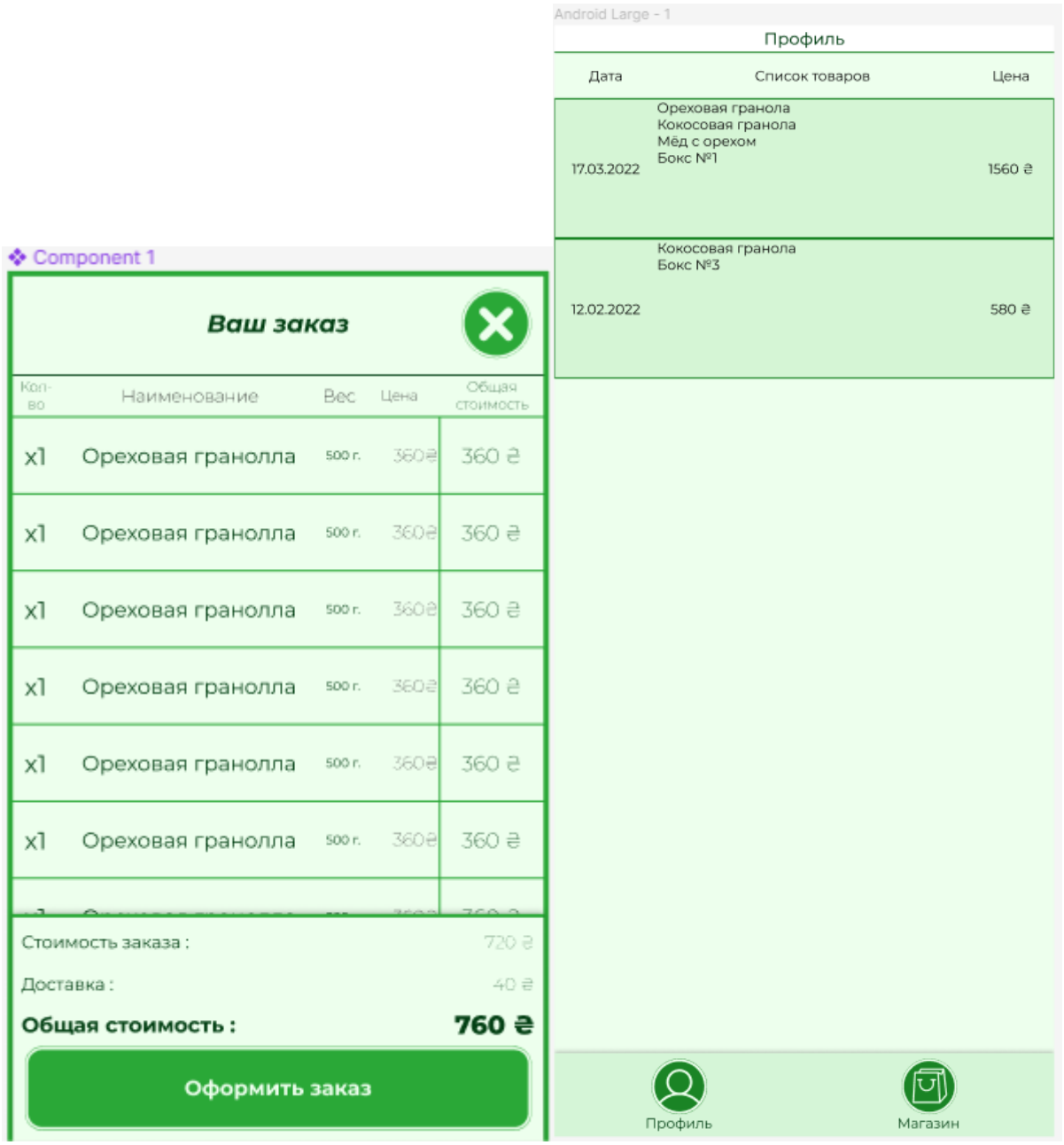


Рисунок 3.5 — Дизайн корзини та профілю

Тепер ми можемо почати переносити ці дизайни в проект в Android Studio.

3.3. Середовище розробки Android Studio

Для початку розробки нам потрібно встановити комплекс інструментів для разробки JDK (Java Development Kit).

Він включає в себе компілятор Java (javac) стандартні бібліотекі класів, приклади, документацію та виконавчу систему Java (Java Runtime Environment).

Далі нам потрібно встановити саме середовище розробки Android Studio. Для цього ми переходимо за посиланням : developer.android.com/studio . Та натискаємо кнопку «Download Android Studio». Після цього читаємо «Terms and Conditions»,

ставимо прапорець у чекбоксі «I have read and agree with the above terms and conditions» та жмемо кнопку «Download Android Studio for Windows». Після того, як завершиться скачування відкриваємо файл та надаємо йому прав адміністратору. Далі жмемо кнопку «Next», обираємо потрібні для встановки компоненти та жмемо «Next» та потім жмемо «Instal».

Після установки запускаємо Android Studio, на нас очікує ще недовгий процес налаштування, де ми обиремо потрібні для встановки компаненти та тему для інтерфейсу середовища (світлу чи темну).

Нас зустрічає досить зрозумілий інтерфейс, де пропонується вибрати макет для майбутнього додатку з вже готових та платформу для якої ми розробляємо продукт.

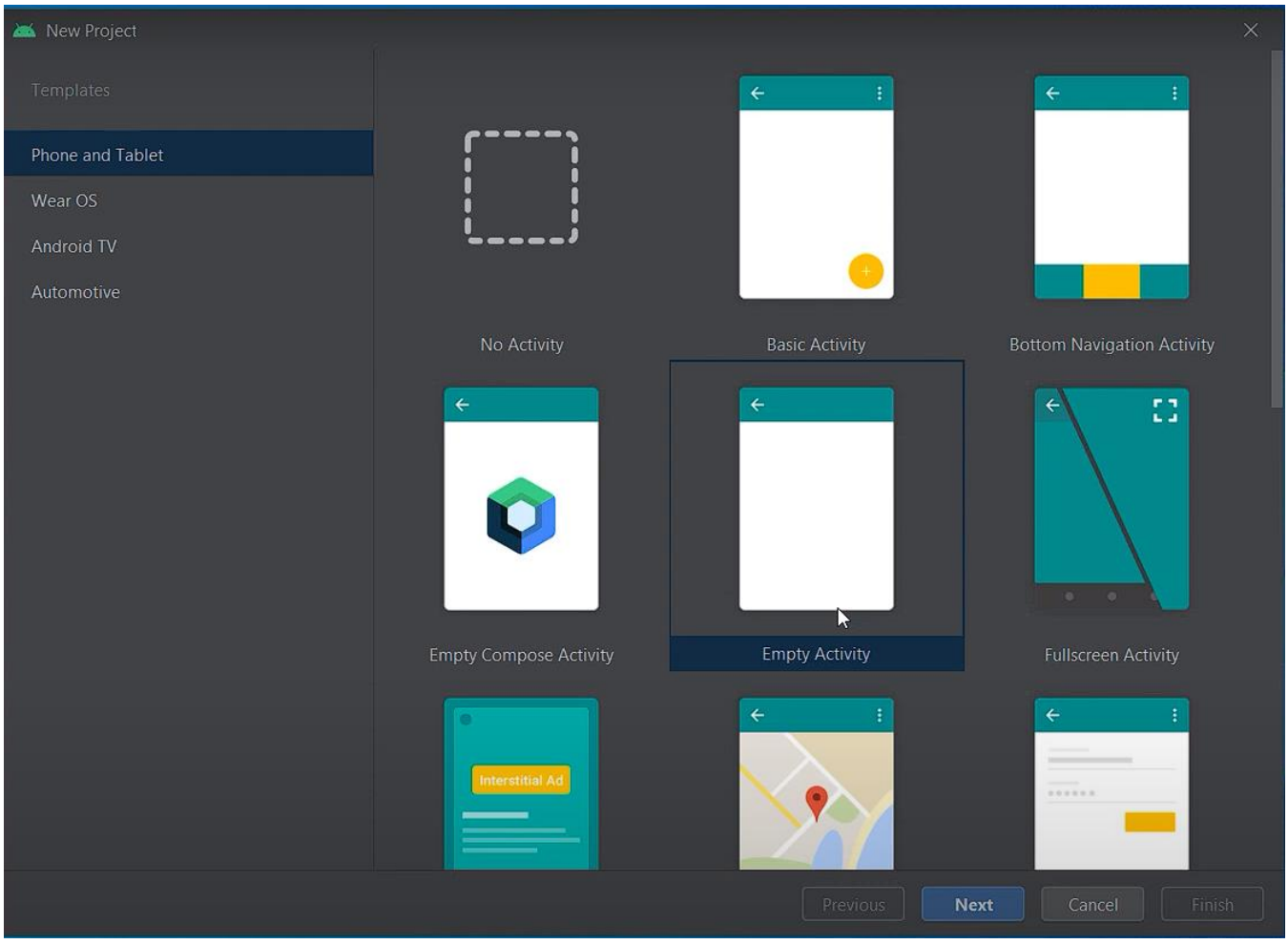


Рисунок 3.6 — Стартове меню Android Studio

Ми обираємо варіант «Empty Activity».

Далі потрібно обрати назву для додатку та його директорії. Після цього ми потраплюємо у інтерфейс розробки. Ми будемо працювати з трьома видами файлів

- це xml-файли, де ми будемо прописувати основний дізайн нашого додатку,java-класи, які відповідають за логіку роботи інтерфейсу, та java-класи, які будуть відповідати за бізнес-логіку та класи об’єктів, що будуть задієні в інтерфейсі.

3.4. Створення візуальної частини додатку

Для початку перейдемо в файл activity_main.xml. Це екран, якій буде запускатися на старті нашого додатку. Сюди ми перенесемо вже разроблений нами дізайн магазину. Для цього мі будемо вікористовувати візуальний інтерфейс Android Studio, а потім будемо корегувати деякі нюанси безпосередньо в самому файлі. По-перше ми експортуємо з Figma у форматі SVG (векторна графіка) елементи інтерфейсу, такі як : кнопки розділів, кнопку фільтру (нам знадобиться лише одна, як макет, на основі якого будуть генеруватися інші), кнопку корзини та перенесемо усі кольори, котрі ми використовувалі в дизайні у файл colors.xml для подальшого вікористовування. Для імпорту ми натискаємо ПКМ по папці drawable та обираємо New->Vector Asset, у відкритому вікні вказуємо шлях до нашого файлу та жмемо Next->Finish. Тепер ми можемо використовувати ці векторні зображення для нашого дизайну.

Для оформлення плитки товару(рис. 3.4) нам треба створити новий файл дизайну. Для цього натискаємо ПКМ по папці layout та обираємо New->Layout Resource File, назвемо його як product_item.xml. Для відтворення дизайну будемо використовувати контейнер CardView. В ньому, за допомогою атрибуту cardCornerRadius ми можемо скруглити кути. В ньому за допомогою контейнерів LinearLayout ми будемо розтошовувати потрібний нам текст(TextView) та зображення(ImageView). В цей дизайн ми будемо підставляти потрібні нам значення, тому кожному полю треба призначити зрозумілий id, щоб потім к ньому звертатися.

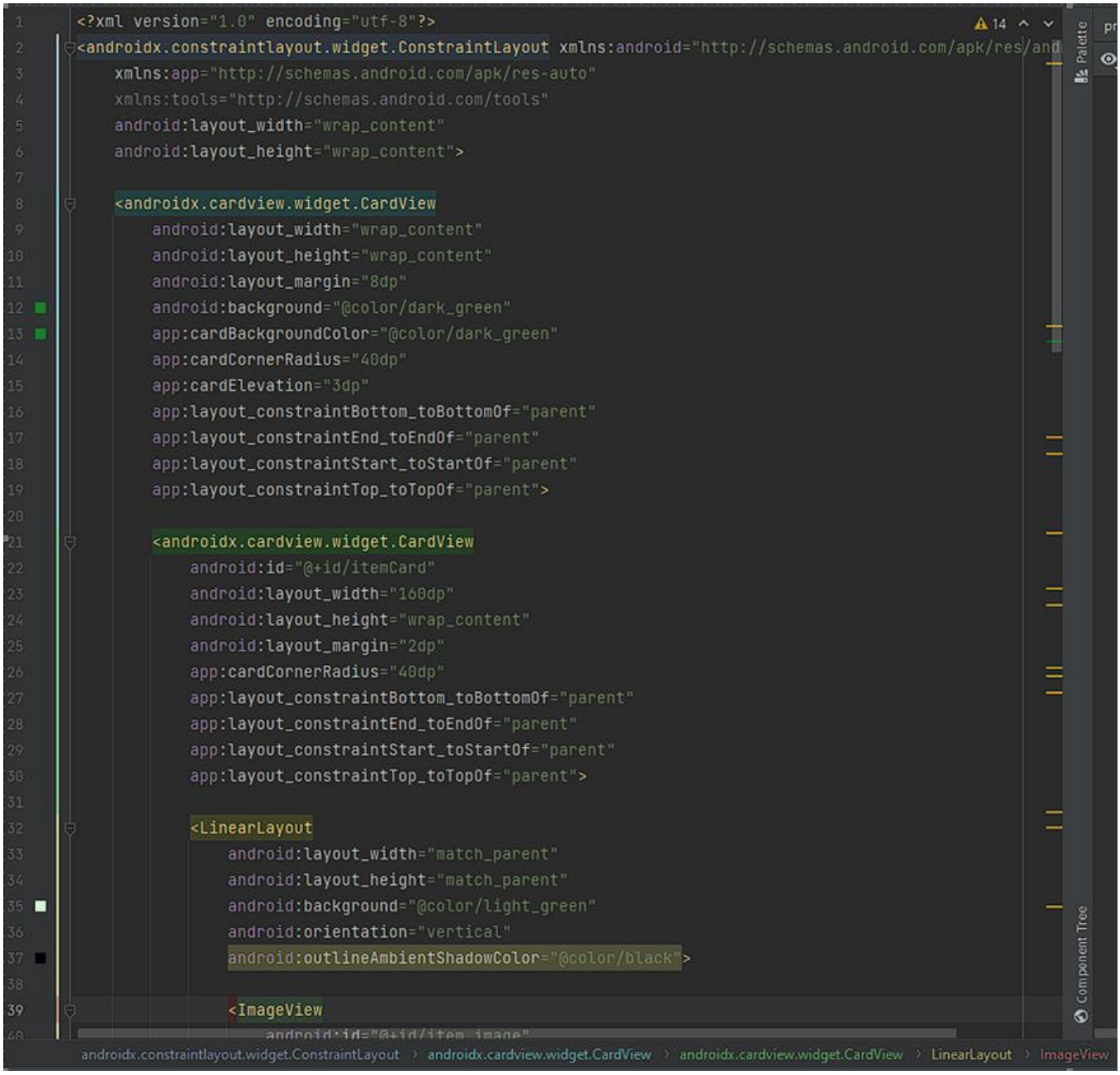


Рисунок 3.7 — Файл product_item.xml

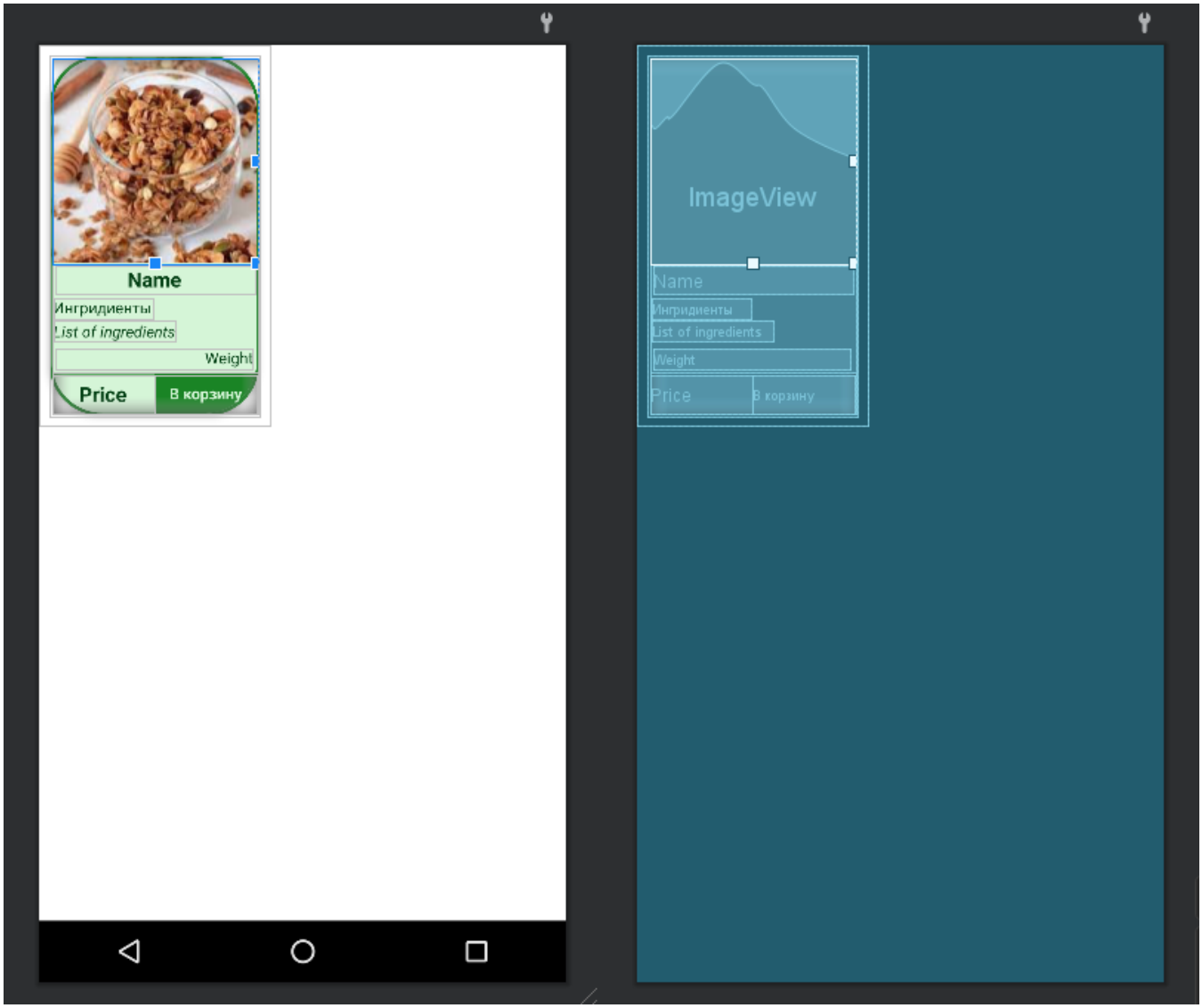


Рисунок 3.8 — Файл product_item.xml у вигляді дизайну

Тим же чином ми створюємо дизайн-макет і для елементів фільтру та елементів заказу. Назвемо його filter_item.xml(В нього буде єдине поле - назва) та order_item.xml(В ньому будуть поля : назва продукту,його кількість,ціна,та загальна вартість).

Також нам потрібно створити новий екран для розділу «Корзина». Для цього ми натискаємо ПКМ по папці layout та обираємо New->Activity->Empty Activity. Назвемо його як activity_cart_page. В ньому треба розташувати кнопку виходу в магазин та поля для подальшого вивода суми.

Коли ми перенесли наш дізайн та відкорегували усі недоліки в місці, де буде розташовуватися список товарів та в місці де буде виводитися список фільтрів розташовуємо такий контейнер,як RecyclerView. Він дозволить нам загружати в інтерфейс об’єкти з списку, який ми будемо в нього передавати. Також

розташовуємо його в розділі «Корзина» де буде виводитися заказ та у розділі «Профіль»,де ми будемо віводити історію заказів клієнта.

3.5. Робота з RecyclerView.

Наш дизайн майже готовий. Залишилося наповнити його товарами, забезпечити виведення фільтрів та елементів заказу в корзині.

Розберемо цей процес на прикладі виводу товарів на сторінці магазину.

Для початку ми створюємо папку «model», в якій ми створимо нашу першу модель – Product. Це java-клас, який буде основою для створення об’єктів, які і наповнять наш магазин. В ньому нам потрібні такі поля як : id, назва, опис, зображення(ми будемо хранити його назву), вага, ціна та id категорії, до якої він відноситься. Також потрібно прописати для нього конструктор, в який ми будемо заповнювати усі його поля, та геттери і сеттери (методи для отримання(getter) чи встановлення(setter) поля об’єкта). Клас виглядає наступним чином :

```
public class Product {
    int id;
    String name,description,img,weight,price;
    int filter;

    public int getFilter() { return filter; }

    public int getId() { return id; }

    public void setId(int id) { this.id = id; }

    public String getWeight() { return weight; }

    public void setWeight(String weight) { this.weight = weight; }

    public String getPrice() { return price; }

    public void setPrice(String price) { this.price = price; }

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    public String getDescription() { return description; }

    public void setDescription(String description) { this.description = description; }

    public String getImg() { return img; }

    public void setImg(String img) { this.img = img; }

    public Product(int id, int filter, String weight, String price, String name, String description, String img) {
        this.id = id;
        this.weight = weight;
        this.price = price;
        this.name = name;
        this.description = description;
        this.img = img;
        this.filter = filter;
    }
}
```

Рисунок 3.9 — Клас-модель Product

Далі нам потрібно написати до цього класу клас-адаптер, який буде перетворювати об’єкти на основі класу Product в справжні елементи інтерфейсу. Для цього ми створюємо папку «adapter», а всередені неї java-клас ProductAdapter.

В ньому ми успадковуємо стандартний клас бібліотеки Android – RecyclerView.Adapter та переопріділяємо його методи onCreateViewHolder, onBindViewHolder getItemCount. Також всередині нашого класу ProductAdapter ми створюємо вкладений клас ProductViewHolder, який успадковуємо від RecyclerView.ViewHolder де ми створюємо поля типів TextView та ImageView, які ми прив'яжемо до елементів створенного нами макету.Для цього і було потрібно вказати зрозумілий id для кожного елементу.

Цей клас буде виглядати наступним чином :

```
package com.dgatman.deliciouswithnaumenko.presentation.adapter;

import ...

public class ProductAdapter extends RecyclerView.Adapter<ProductAdapter.ProductViewHolder> {
    Context context;
    List<Product> productList;

    public ProductAdapter(Context context, List<Product> productList) {
        this.context = context;
        this.productList = productList;
    }

    @NonNull
    @Override
    public ProductViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        View productItems = LayoutInflater.from(context).inflate(R.layout.product_item,parent, attachToRoot: false);
        return new ProductAdapter.ProductViewHolder(productItems);
    }

    @Override
    public void onBindViewHolder(@NonNull ProductViewHolder holder, int position) {
        int imageId= context.getResources().getIdentifier(productList.get(position).getImg(), defType: "drawable",context.getPackageName());
        holder.item_image.setImageResource(imageId);
        holder.name.setText(productList.get(position).getName());
        holder.ingredients.setText(productList.get(position).getDescription());
        holder.weight.setText(productList.get(position).getWeight() + " грамм");
        holder.price.setText(productList.get(position).getPrice() + " ₴");

       holder.buy_button.setOnClickListener(new View.OnClickListener() {
           @Override
           public void onClick(View view) { OrderData.add(productList.get(position)); }
       });
    }

    @Override
    public int getItemCount() { return productList.size(); }

    public static final class ProductViewHolder extends RecyclerView.ViewHolder {
```

Рисунок 3.10 — Клас-адаптер ProductAdapter

```
    public static final class ProductViewHolder extends RecyclerView.ViewHolder {
        TextView name,ingredients,weight,price,buy_button;
        ImageView item_image;

        public ProductViewHolder(@NonNull View itemView) {
            super(itemView);
            name = itemView.findViewById(R.id.name);
            ingredients = itemView.findViewById(R.id.ingredients);
            weight = itemView.findViewById(R.id.weight);
            price = itemView.findViewById(R.id.price);
            item_image = itemView.findViewById(R.id.item_image);
            buy_button = itemView.findViewById(R.id.buy);

        }
    }
}
```

Рисунок 3.11 — Вкладений клас ProductViewHolder

І останнє що нам залишилося зробити – ініціювати в класі MainActivity змінні типу RecyclerView(productRecycler) та ProductAdapter та створити метод, який буде приймати список об’єктів класу Product та перетворювати його на плитку елементів дизайну. Назвемо цей метод setProductRecycler, він буде приймати List<Product> та не буде нічого повертати.

В цьому методі спочатку ми створюємо об’єкт класу RecyclerView.GridLayoutManager layoutManager, в конструктор якого мі передаємо контекст, в якому він буде працювати та кількість стовпців.

Далі ми привласнюємо створенному нами об’єкту RecyclerView ссилку на існуючий елемент інтерфейсу, за допомогою методу findViewById, та вже цьому існуючому об’єкту звертаємося до методу setLayoutManager, в який надаємо об’єкт, створенний в початку цього методу layoutManager. Нарешті створюємо об’єкт типу ProductAdapter та передати у нього список елементів, який ми для тестування, створимо прямо у класі MainActivity. Та нам залишилося лише призначити цей об’єкт адаптером для productRecycler за допомогою методу setAdapter.

```
58      private void setProductRecycler(List<Product> productList) {
59          RecyclerView.LayoutManager layoutManager = new GridLayoutManager( context: this, spanCount: 2);
60          productRecycler = findViewById(R.id.productRecycler);
61          productRecycler.setLayoutManager(layoutManager);
62          productAdapter = new ProductAdapter( context: this,productList);
63          productRecycler.setAdapter(productAdapter);
64
65      }
```

Рисунок 3.12 — Метод setProductRecycler

Рисунок 3.13 — Результат, який ми отримали.

Ми отримуємо згенерований список товарів, який в будь-який момент можна доповнити. Він вертикально пролистується, так поки список не великий, ми можемо завантажувати його цілком, не розділяючи на частини. Якщо він далі розширюватися нам слід потурбуватися про завантаження частинами при повному прогортанні.

3.6. Додавання дій при натисканні кнопок

Для початку ми зробимо клікабельною кнопку переходу до розділу «Корзина». Для цього нам знадобиться дизайн цього екрану, який ми вже зробили

та обробник подій, який буде реагувати на натискання по кнопці «Корзина». Для того щоб нам його створити ми переходимо у файл MainActivity.class та створюємо в ньоми новий публічний метод,який не буде нічого вертати,та буде приймати об’єкт типу View, назвемо його openCartPage. Потім в атрибуті кнопки корзини onClick ми вказуємо цей метод. Відтепер при настисканні кнопки «Корзина» буде відбуватися виконування цього методу. В цьому методі нам потрібно створити об’єкт класу Intent, в конструктор якого ми передаємо з якої та в яку активність він буде нас переносити, в нашому випадку це this,activity_cart_page. Потім викликаємо метод startActivity в який і передаємо наш об’єкт Intent. Тепер тестуємо, як це буде віглядати.

```
90      public void openCartPage(View view){
91          Intent intent = new Intent( packageContext: this, CartPage.class);
92          startActivity(intent);
93      }
```

Рисунок 3.14 — Метод openCartPage

При натисканні кнопки «Корзина» ми отрімуємо такий результат :

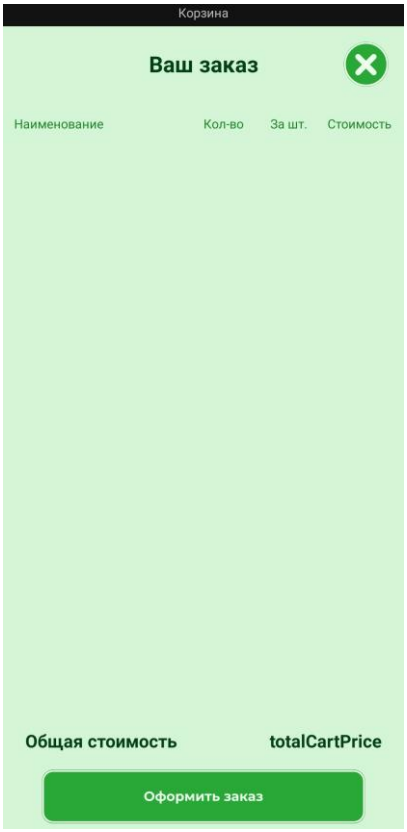
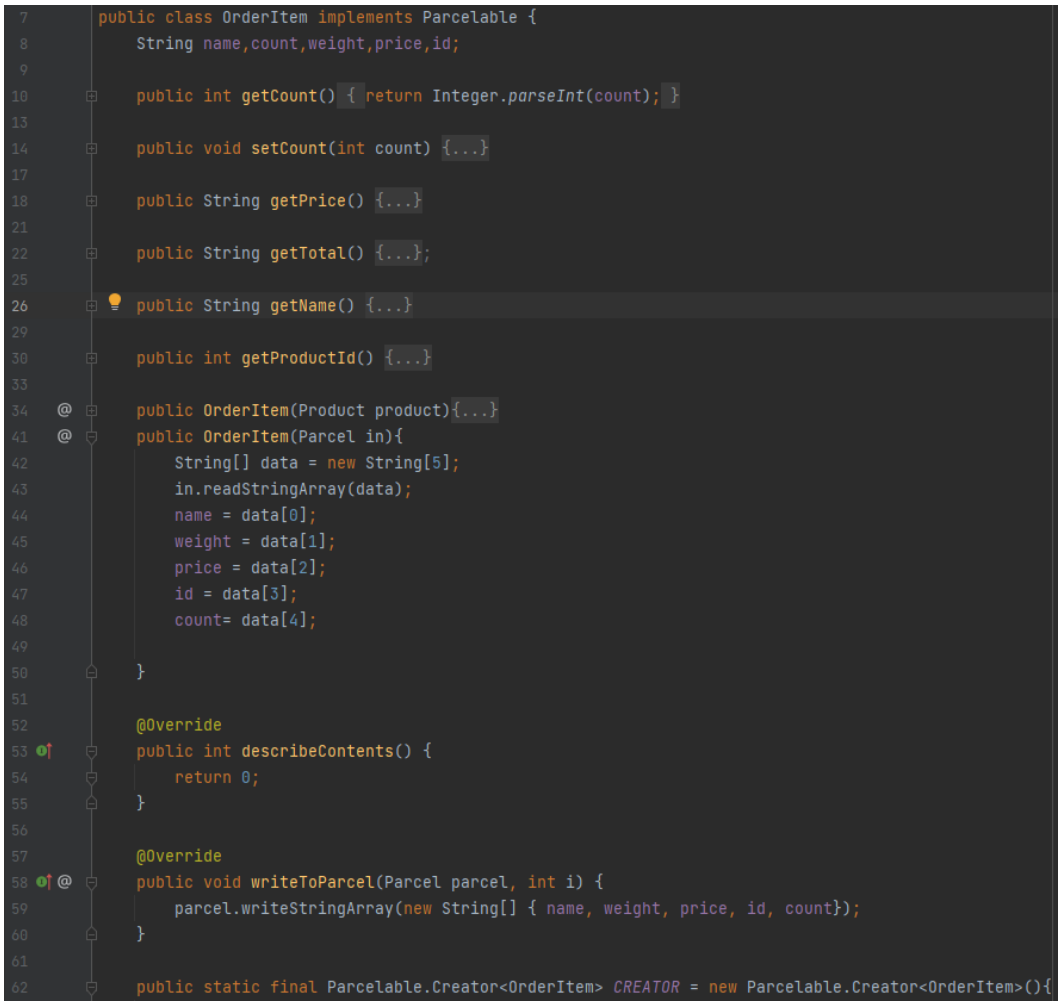


Рисунок 3.15 — Екран, на якій нас переносе при натисканні кнопки «Корзина»

Тепер нам потрібно таким же чином налаштувати кнопку віходу з цього екрану до екрану «Магазин». Тепер ми можемо відкривати та закривати розділ «Корзина», але ще не можемо його заповняти. Для цього потрібно реалізувати обробник подій для кнопки «В корзину» на кожному нашому товарі. Для цього нам потрібно перейти до ProductAdapter та у методі onBindViewHolder додати обробник натискання на об’єкт інтерфейсу TextView з написом «В корзину», який буде викликати метод addToCart з MainAcrivity.class, який, в свою чергу, буде добавляти в список об’єктів класу OrderItem orderList. Але нам потрібно ще передати цей список у activity_cart_page, для цього ми повинні імплементувати в клас OrderItem інтерфейс Parcelable та реалізувати його методи. Наш Клас буде виглядати наступним чином :



```
public class OrderItem implements Parcelable {
    String name,count,weight,price,id;

    public int getCount() { return Integer.parseInt(count); }

    public void setCount(int count) {...}

    public String getPrice() {...}

    public String getTotal() {...};

    public String getName() {...}

    public int getProductId() {...}

    public OrderItem(Product product){...}
    public OrderItem(Parcel in){
        String[] data = new String[5];
        in.readStringArray(data);
        name = data[0];
        weight = data[1];
        price = data[2];
        id = data[3];
        count= data[4];

    }

    @Override
    public int describeContents() {
        return 0;
    }

    @Override
    public void writeToParcel(Parcel parcel, int i) {
        parcel.writeStringArray(new String[] { name, weight, price, id, count});
    }

    public static final Parcelable.Creator<OrderItem> CREATOR = new Parcelable.Creator<OrderItem>(){
```

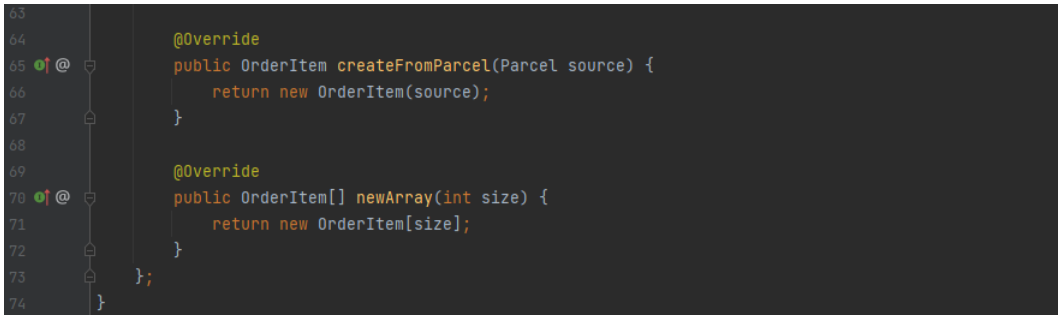


Рисунок 3.16 — Клас OrderItem після імплементації інтерфейсу Parcelable

Відтепер ми можемо передавати список об'єктів цього класу у другу активність за допомогою методу класу Intent putParcelableArrayListExtra. Метод переходу в розділ «Корзина» зараз виглядає наступним чином :

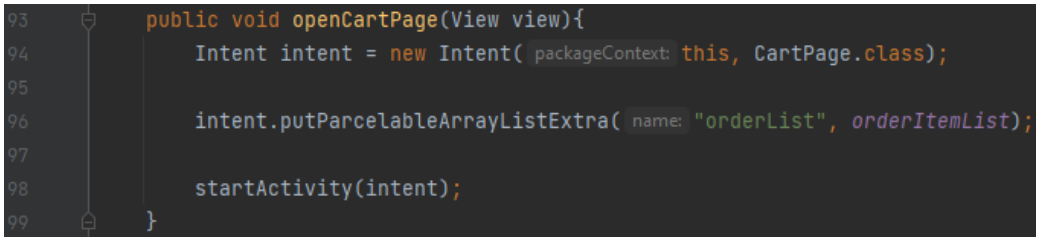


Рисунок 3.17— Метод openCartPage в MainActivity.class

Далі нам потрібно зробити тим же чином клікабельними кнопки профілю, оформлення заказу та фильтрів.

3.7. Експлуатація додатку та тестування

Після завершення усіх кориговок я можу згенерувати apk–файл для встановлення додатку на мій смартфон та мануально протестувати додаток.

Для цього у верхній панелі обираємо Build->Build Bundle(s)/APK(s)-> Build APK(s),файли будуть у папці AndroidStudioProjects\ DeliciousWithNaumenko\ app\ build\ outputs\ apk\ debug\ . Файл типу apk треба скачати на Android-смартфон та запустити, після цього додаток встановиться на ваш смартфон. Після тестування було поміченно, що при додаванні в корзину одного й того ж самого товару вони не складалися :

Рисунок 3.18 — Меню корзини

Щоб позбутися цього недоліку було вирішенно переписати метод додавання у корзину та зробити в ньому перевірку, чи такий товар вже є в корзині, та у цьому випадку лише додавати до його поля count +1.

Також було б не плохо підключити додаток до серверу з базою даних,щоб не хранити всю інформацію локально, але оренда серверу буде коштувати немалих грошей, тому було вирішено залишити так, як це зараз є.

4. ЕКОНОМІЧНА ЧАСТИНА

4.1 Резюме

В даному дипломному проекті розроблено мобільний додаток для мережі дієтичного харчування.

Ефективність кожного програмного продукту визначається його якістю та ефективністю процесу розробки. Якість ПП визначається наступними складовими: з точки зору користувача; з позиції використання ресурсів; виконання вимог до програмного забезпечення.

Оцінка якості програмного продукту з точки зору користувача визначається необхідним на стадії функціонування розміром оперативної пам'яті ЕОТ, витратами машинного часу, пропускною спроможністю каналів передачі даних. Оцінка якості програмного продукту включає визначення трудомісткості і вартості його створення.

4.2. Визначення трудомісткості розробки програмного забезпечення.

Тривалість розробки програмного продукту залежить від його обсягу, трудомісткості розробки, кваліфікації виконавців, а також планових термінів, визначених умовами ринку. Методом структурної аналогії по відповідних каталогах аналогів програмного забезпечення визначається обсяг програмних засобів, у тисячах умовних машинних команд програми аналога

Каталог аналогів

Таблиця 4.1

Найменування ПП	Обсяг функції ПП – V_o, усл. машинних командах.
1. ПП СУБД	2500 – 9800
2. Комплексні системи ведення	950 – 7430
3. ПП введення інформації	1060 – 5750
4. ПП оптимізації розрахунків	1300 – 4200
5. ПП автоматизації засобів по каталогу	680 – 7000
6. ПП автоматизованих розрахунків	1300 – 8600

7. ПП загальної математики і ПП імітаційного моделювання	7800 – 8800
8. ПП організації обчислювального процесу	10200 – 13000

У таблиці 4.1 представлені аналоги програмного забезпечення, функції яких, у більшому або меншому ступені, виконує розроблений програмний продукт. Для нашого варіанта виділено сірим кольором.

Вибравши аналог ПП, що містить Vo в умовних машинних командах, трудомісткості визначати на основі табл.4.2

Таблиця.4.2

Обсяг ПП, тис.умов.машинних команд	Норма часу, люд/год
1.00	229
2.00	244
3.00	262
4.00	283
5.00	306
6.00	330
7.00	357

На підставі отриманого значення, по довіднику, визначається укрупнена норма часу на розробку аналога програмного забезпечення (коректується поправочним коефіцієнтом враховуючої умови розробки ПП, тобто в умовах комп'ютера, Кк=0,7÷0,8): T^a = 330 х 0,8 = 264 (люд/годин).

Трудомісткість програмного продукту визначається по кожному етапу розробки окремо на підставі трудомісткості аналога з урахуванням складності розробки, ступеня новизни і ступеня використання в розробці стандартних модулів на підставі формул:

(4.1)

$$T_{ТП} = T^a p \times L_2 \times K_H \qquad (4.2)$$

$$T_{РП} = T^a p \times L_3 \times K_H \times K_T \qquad (4.3)$$

Для розрахунку необхідні наступні коефіцієнти:

Li – питома вага і-го етапу розробки (див. табл. 4.2.);

Кн – поправочний коефіцієнт, що враховує ступінь новизни (див. табл. 4.3.);

Кт – поправочний коефіцієнт, що враховує ступінь використання в розробці типових програм (див. табл. 4.4.).

Таблиця 4.2.Значення питомих коефіцієнтів трудомісткості стадії в загальній трудомісткості розробки ПП.

Код стадії	Ступінь новизни		
	А	Б	В
ТЗ (L_1)	0,15	0,12	0,12
ТП (L_2)	0,16	0,15	0,11
РП (L_3)	0,55	0,58	0,61

Для нашого варіанта виділено сірим кольором.

Таблиця 4.3.

Значення поправочного коефіцієнта, що враховує ступінь новизни

Код ступеня новизни	Ступінь новизни	Значення $К_н$
А	Принципово нові ПП	1,75 – 1,2
Б	ПП – розвиток визначеного параметричного ряду	1,0 – 0,8
В	ПП маючий аналог	0,7

Для нашого варіанта виділено сірим кольором.

Таблиця 4.4.

Значення коефіцієнта ступеня використання в розробці типових програм

Ступінь охоплення реалізованих функцій розроблювального ПП типовими програмами, %	Значення $К_т$
60 і вище	0,6
40-60	0,7
20-40	0,8
До 20	0,9

Для нашого варіанта виділено сірим кольором.

Тепер розраховуємо трудомісткість по кожному етапу окремо:

Трудомісткість технічного завдання

$$T_{тз}=T^{a}*L_1*K_н= 264*0,12*0,7=22.17 \quad (люд/годин) \quad (4.1)$$

Трудомісткість розробки технічного проекту

$$T_{тп}=T^{a}*L_2*K_н =264*0,11*0,7 =20.33 \quad (люд/годин) \quad (4.2)$$

Трудомісткість розробки робочого проекту

$$T_{рп}=T^{a}*L_3*K_н*K_т=264*0,61*0,7*0,7=78.90 \quad (люд/годин) \quad (4.3)$$

Для подальших розрахунків визначили кількість папера, витраченого на кожен етап: технічне завдання $N_{тз}$= 3 (стр), розробка ТП $N_{тп}$=11(стр), розробка робочого проекту $N_{рп}$=28(стр), пояснювальна записка відповідно $N_{пз}$=17 (стр) Розрахунок зведений у таблицю 4.5

Таблиця 4.5.

Розрахунок трудомісткості ПП

Найменування етапів	Розрахунок, годин.		
1	2	3	4
1.ТЗ	$T_{Ртз}$=22.17	$T_{кк}=0,7*N_{тз}=0.7*3=2.1$	$T_{нк}=0,15*N_{тз}=0.15*3=0.45$
2.Розробка ТП	$T_{Ртп}$=20.33	$T_{кк}=0,7*N_{тп}=0.7*11=7.7$	$T_{нк}=0,15*N_{тп}=0.15*11=1.65$
3.Розробка РП	$T_{Ррп}$=78.90	$T_{кк}=0,7*N_{рп}=0.7*28=19.6$	$T_{нк}=0,15*N_{рп}=0.15*28=4.2$
4.Розробка ПЗ	$T_{пз}=1,5*$ $*N_{пз}=1,5*17=25.5$	$T_{кк}=0,7*N_{пз}=0.7*17=11.9$	$T_{нк}=0,15*N_{пз}=0.15*17=2.55$
Усього, в т.ч.:	ΣT=146.9		
- на розробку	$\Sigma T_р$=121.4		
- контроль керівника		$\Sigma T_{кк}$=41.3	
- нормоконтроль			$\Sigma T_{нк}$=8.85

4.3 Розрахунок ціни програмного продукту.

У цьому розділі для визначення ціни розраховуємо основну заробітну плату виконавців, матеріальні витрати, вартість машино – години і витрати на розробку ПО. Розрахунок основної заробітної плати виконавців приведений у таблиці 4.6. Відповідно до статті 8 «Закону про Державний бюджет України на 2022» встановлено мінімальну заробітну плату у місячному розмірі з 1 січня 2022 року - 6500 гривень; мінімальну погодинну тарифну ставку – 39.26 грн.

Таблиця 4.6 Розрахунок основної заробітної плати виконавців.

Найменування робіт	Трудомісткість робіт, години	Погодинна тарифна ставка, грн.	Розрахунок, грн.
1.Розробка ПП	146.9	39.26	5767.29
2.Контроль керівника	41.3	50.00	2065
3.Нормоконт-роль	8.85	50.00	442.5
Усього	-	-	ΣЗо= 8274.79

Зробимо розрахунок матеріальних витрат на розробку ПП. Розрахунок зведемо в таблицю 4.7

Таблиця 4.7

Розрахунок матеріальних витрат на розробку ПО

Найменування матеріальних витрат	Тип, модель	Кількість	Ціна одиниці, грн.	Вартість, грн.
Папір	Лист А4	59	0.41	24.19
Разом	-	-	-	$В_{м\,і}$ =24.19
Транспортно– заготівельні витрати (10%)				2.42
Усього				$В_м=В_{мі}+В_{тр\_з}$= 24.19+2.42=26.61

На підставі отриманих даних по окремих статтях витрат складена калькуляція планової собівартості в цілому ПП за формою, приведеною в таблиці 4.8.

Таблиця 4.8.

Розрахунок статей витрат планової собівартості

Стаття витрат	Значення, грн.	Формула розрахунку
1. Матеріали	26.61	
2. Основна заробітна плата	8274.79	
3.Додаткова заробітна плата	1241.22	$Зд = 0{,}15 \times Зо$
4.Відрахування до єдиного фонду соціального внеску	3909	$Вє.с.в. = 0{,}22 \times (Зо + Зд)$
5. Накладні витрати	2482.44	$Внак. = 0{,}3 \times Зо$
6. Повна собівартість	15934.06	$С_{пов} = В_м + З_о + З_д + Вє.с.в. + Внак.$

Розмір прибутку, що включається в ціну, визначаємо по наступній формулі:

$$П=(С_п*Р)/100=(15934.06*0.15)/100=23.90 \quad (4.4)$$

Де р – плановий рівень рентабельності (10-15%).

Оптова ціна (кошторисна вартість) визначається по формулі:

$$Ц_о = С_{пов}+П = 15934.06+23.90=15957.96 \quad (4.5)$$

Податок на додану вартість визначаємо по наступній формулі:

$$ПДВ = 0.2 * Ц_о = 0.2*15957.96=3191.59 \quad (4.6)$$

Виходячи з отриманих даних, ціна реалізації розробленого програмного продукту на основі наступної формули, становитиме:

$$Ц_р=Ц_о+ПДВ = 15957.96+3191.59=19149.55 \quad (4.7)$$

5.ОХОРОНА ПРАЦІ

Охорона праці - це система правових, соціально-економічних, організаційнотехнічних, санітарно-гігієнічних і лікувально-профілактичних заходів та засобів, спрямованих на збереження життя, здоров'я і працездатності людини у процесі трудової діяльності.

Одним з резервів підвищення ефективності виробництва є вдосконалення методів забезпечення безпеки праці, тому що травматизм визначає істотну частину непродуктивних втрат робочого часу, а боротьба з травматизмом, крім гуманістичного спрямування, має чітко виражений економічний аспект.

Безпека праці виступає і як один з факторів, які забезпечують високу продуктивність праці. Доведено, що висока продуктивність праці може бути досягнута тільки в умовах, коли забезпечена її безпека.

В розділі охорона праці дипломного проекту вирішується питання безпеки праці програміста при розробці ним ігрового додатку класичного жанру.

5.1. Аналіз та безпека умов праці працівника на робочому місці

Забезпечення безпечних і здорових умов праці в значній мірі залежить від правильної оцінки небезпечних та шкідливих виробничих факторів. Однакові по складності зміни в організмі людини можуть бути викликані різними причинами. Це можуть бути фактори виробничого середовища, надмірне фізичне і розумове навантаження, нервово-емоційна напруга, а також різне сполучення цих причин.

Оператори і програмісти зіштовхуються із впливом таких фізично небезпечних і шкідливих виробничих факторів, як підвищений рівень шуму, підвищена температура зовнішнього середовища, відсутність або недостатня освітленість робочої зони, електричний струм, статична електрика тощо. До основних шкідливих факторів при роботі з комп'ютером відносять: тривале сидяче положення, електромагнітне випромінювання, навантаження на зір

На робочому місці програміста повинні бути створені умови для безпечної та високопродуктивної праці.

5.2. Розробка заходів з охорони праці

Виробничі приміщення. Вимоги ДСанПІН 3.3.2.007-98 визначають обємно-планувальні рішення будівель та приміщень для роботи з ВДТ. Розміщення робочих місць з ВДТ ЕОМ і ПЕОМ у підвальних приміщеннях, на цокольних поверхах заборонено. Площа на одне робоче місце становить не менше 6,0 м2 , а об'єм – не менше ніж 20,0 м3 . У приміщеннях слід щоденно робити вологе прибирання. Вони повинні бути оснащені аптечками першої медичної допомоги. При приміщеннях мають бути обладнанні побутові приміщення для відпочинку.

При кольоровому оформленні виробничих і допоміжних приміщень необхідно враховувати орієнтацію їхніх вікон стосовно частин світу і використовувати гармонійне сполучення кольорів. Для стін і робочих поверхонь використовують мало насичені (основні) кольори, для невеликих помешкань або ділянок, що рідко потрапляють у поле зору працюючих, а також для створення контрастності – кольори середньої насиченості (допоміжні), для маленьких по площі поверхонь – насичені (акценти) – як функціональне фарбування. Стелі у всіх приміщеннях повинні бути білими. Поверхні устаткування в приміщеннях повинні бути матовими або напівматовими, для виключення випадку відблисків світла в очі працюючого, а стіни бути пофарбованими фарбами пастельних тонів.

Освітлення. Приміщення для роботи з ПК повинні мати природне та штучне освітлення, відповідно до ДБН В.2.5-28-2006. У приміщеннях, призначених для роботи з відео терміналами, доцільно, щоб вікна були орієнтовані на північ або північний захід. На вікнах повинні бути штора або жалюзі, що регулюють рівень освітленості і захищають від прямого влучення сонячних променів на робоче місце.

Для штучного освітлення у приміщенні використовуються люмінесцентні лампи типу ЛБ, які в порівнянні з лампами розжарювання мають ряд істотних переваг: за спектральним складом світла вони близькі до природного світла, мають підвищену світлову віддачу (у 2-5 разів вищу, ніж у ламп розжарювання); мають триваліший термін служби – до 10 тис годин

Параметри повітря робочої зони. Для підтримки в приміщеннях нормального, що відповідає гігієнічним вимогам складу повітря, видалення з нього

шкідливих газів, пилу використовують вентиляцію. Механічна вентиляція (кондиціонери, вентилятори і т.д.) залежно від напрямку руху повітряних потоків, може бути витяжною, припливною і припливно-витяжною. При природній вентиляції (за допомогою вікон) повітря надходить у приміщення і видаляється з нього внаслідок різниці температур і тиску.. Механічна вентиляція забезпечується вентиляторами, що забирають повітря зовні і направляє його до будь-якого робочого місця. або устаткування, а також видаляють забруднене повітря.

У виробничих приміщеннях на робочих місцях мають забезпечуватись оптимальні значення параметрів мікроклімату: температури, відносної вологості й рухливості повітря – ГОСТ 12.1.005-88, СН 4088-86.

Параметри мікроклімату	Взимку	влітку
Температура, C^0	22-24	23-25
Відносна вологість, %	40-60	40-60
Швидкість руху повітря, м/с	0,1	0,1-0,2

Шум і вібрація, ЕМВ. При розумовій праці, яка вимагає зосередженості припустимий рівень шуму становить 50дБ. Для зменшення шуму й вібрації в приміщенні устаткування, апарати й прилади встановлюють на спеціальні прокладки, що амортизують. Якщо стіни в приміщенні є джерелами шумоутворення, вони повинні бути облицьовані звуковбирним матеріалом.

У разі надмірного шуму чи вібрації технічного обладнання, роботодавець повинен забезпечити працівників антивібраційними килимками.

Кількісно вплив електромагнітного поля на людину оцінюється величиною поглинутої її тілом електромагнітної енергії, W,Вт, або питомої енергії, що поглинається Wп, Вт/кг. Основні заходи захисту від ЕМВ — це захист часом, захист відстанню, екранування джерел випромінювання, зменшення випромінювання в самому джерелі випромінювання, виділення зон випромінювання, екранування робочих місць, застосування ЗІЗ.

Організація робочих місць з ПК. Роботодавець, який використовує найману працю робітників, повинен забезпечити відповідність їхніх робочих місць комфортним та безпечним умовам. Розмір одного робочого місця має становити не

менше 6 квадратних метрів. При необхідності, суміжні робочі місця співробітників, що працюють з комп'ютером, слід розділити перегородками висотою до 2 метрів.

При визначенні достатнього розміру приміщення і робочого місця на одну особу необхідно додатково враховувати шафи, сейфи, тумби або інші предмети меблів чи обладнання, які знаходяться в кімнаті.

На столі працівника можливо розмістити допоміжні для роботи пристрої (принтери, колонки, сканери), а також місця для зберігання документів, за умови, що це не обмежуватиме видимість екрану і не заважатиме працівнику.

Робочий стілець співробітника має бути підйомно-поворотним, легко регульованим за висотою та забезпечувати належну підтримку та зручне положення спини і хребта особи. Щодня необхідно проводити вологе прибирання приміщення, та очищати робоче місце та безпосередньо монітор комп'ютера від запиленості.

Мають бути чітко встановлені перерви для відпочинку працівників (окрім обідньої), як правило, тривалістю 10-15 хвилин раз на годину або дві, в залежності від складності роботи. У будь-якому випадку, роботодавець повинен передбачити такий розпорядок роботи на підприємстві, щоб час непреревної роботи з комп'ютером був не більше ніж 4 години.

Пожежна безпека. Під пожежною безпекою розуміють систему державних і суспільних заходів, спрямованих на охорону від вогню людей і власності. Пожежна безпека приміщень, що мають електричні мережі, регламентується ГОСТ 12.1.033-81, ГОСТ 12.1.004-85. Робота оператора ЕОМ повинна вестися в приміщенні, що відповідає категорії Д пожежної безпеки (негорючі речовини й матеріали в холодному стані).

Пожежна безпека об'єкта забезпечується:

- Системою запобігання пожежі;
- Системою протипожежного захисту;
- Організаційно-технічними заходами.

Всі приміщення повинні бути забезпечені первинними засобами пожежогасіння: пожежним водопостачанням (пожежні крани ПК), пожежні щити з набором пожежного інструменту, вуглекислотними або порошковими

вогнегасниками. У випадку виникнення пожежі необхідно відключити електроживлення, викликати по телефону 101 пожежну команду, евакуювати людей із приміщення відповідно до плану евакуації і приступити до ліквідації пожежі.

ВИСНОВОК

Мета цієї дипломної роботи полягає у створенні мобільного додатка для мережі дієтичного харчування, для більш ефективного залучення нових клієнтів, підвищення інформованості існуючих клієнтів про роботу підприємства, а також
закріплення практичних навичок програмування для ОС Android середовищем розробки Android Studio.

Для досягнення поставленої мети було сформульовано такі завдання:

1. Здійснити пошук та аналіз способів підвищення потоку клієнтів
за допомогою мобільного додатка
2. Провести аналіз вимог до мобільного додатку
3. Спроектувати інтерфейс користувача
4. Створити мобільний додаток
5. Здійснити розрахунок економічних показників ефективності
створення мобільного додатка

У першому розділу було проаналізовано ринок мобільних додатків та їх користь для бізнесу.

У другому та третьому розділі був проведений аналіз та пред'явлені вимоги до мобільному додатку, проаналізовано функціональні вимоги до інтерфейсу, за якими був створений прототип користувача інтерфейсу. За допомогою якого надалі проводилася розробка мінімальної версії мобільного додатка для ОС Android с застосуванням новітніх технологій у галузі мобільних технологій.

У четвертому розділі дипломної роботи розглядався розрахунок економічних показників ефективності створення мобільного програми, та пошук та аналіз способів підвищення потоку клієнтів з за допомогою мобільного додатка.

Були знайдені та проаналізовані та представлені способи підвищення потоку клієнтів за допомогою мобільного додатка. У п’ятому розділі було проаналізовано умови труда та їх вплив на людину. Під час вирішення завдання було вивчено теоретично методи пред'явлення та аналізу вимог до програмного забезпечення, користувальницького інтерфейс. Було розроблено додаток, який відповідає пред'явленим вимогами дипломною роботою, та функціонує згідно з описаними вимогами. Також були закріплені та отримані нові навички роботи в середі розробки Android Studio. Таким чином, завдання були вирішені в повному обсязі, ціль досягнуто.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ІНФОРМАЦІЇ

1 Закон України Про охорону праці, №235-IV, 22.11.2002

2 ДНАОП 0.03-8.03-97 Гігієнічна класифікація праці за показниками шкідливості та небезпечності факторів виробничої середовища, важкості та напруженості трудового процесу

3 БНіП 2.09.04-87. Адміністративні та побутові будівлі

4 Статистика мобільних операційних систем в Україні – [Електронний ресурс].

5 Інтерфейс Android Studio – [Електронний ресурс]. – Режим доступу: https://developer.android.com/studio

6 Зв'язки між об'єктами у діаграмі послідовностей – [Електронний ресурс]. – Режим доступу: https://www.visual-paradigm.com/guide/uml-unified-modelinglanguage/what-is-sequence-diagram/

7 Тестування мобільних додатків – [Електронний ресурс]. – Режим доступу: https://jetruby.com/ru/blog/testirovanie-mobilnyh-prilozhenii/

8 SMART – Мілошевич Д. Набір інструментів для управління IT-проектами: підручник. - ДМК, 2006. - 674 с.

9 Руснак І.С., Бабюк О.В., Федорова О.Є. Охорона праці під час роботи з комп'ютером. – Чернівці: Рута, 2005. – 128 с.

10 Електронний ресурс – http://developer.android.com.

11 Інформаційні системи. Визначення інформаційної системи. Створення моделі інформаційних систем. [Електронний ресурс] / Морозов Олександр – 2002 – Режим доступу: http://www.unicyb.kiev.ua/~boiko/it/morozoff.html

12 Програмна інженерія. -/Лавріщева К. М.-Підручник. -К.: Академперіодика, 2008. -319 с.

13 Основи інженерії якості програмних систем. -Андон Ф. І., Коваль Г. І., Коротун Т. М., Лаврищева О. М., Суслов В. Ю.-Київ: Академперіодика, 2007. -680 с.

14 The Java Language Specification. Java SE 8 Edition. [Електронний ресурс]/ James Gosling, Bill Joy, Guy Steele, Gilad Bracha, Alex Buckley – Режим доступу: http://docs.oracle.com/javase/specs/jls/se8/html/index.html

15 Канер Кем, Фолк Джек, Нгуен Енг Кек - Тестування програмного забезпечення. Фундаментальні концепції управління бізнес-додатків. - Київ: ДіаСофт, 2001. - 544 с. - ISBN 9667393879

16 Software Perfomance Testing. [Електронний ресурс] Вікіпедія. Режим доступу: http://en.wikipedia.org/wiki/Software_performance_testing

17 Android Debug Bridge [Електронний ресурс] Android Developers. Режим доступу: http://developer.android.com/tools/help/adb.html 10. YouTube / [Електронний ресурс] // Режим доступу: http://www.youtube.com