Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»
Інститут електроенергетики
(інститут)
факультет інформаційних технологій
(факультет)
Кафедра інформаційних технологій та комп'ютерної інженерії
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня бакалавра
(бакалавра, спеціаліста, магістра)

студента Дакаленка Даніїла Олеговича
(ПІБ)
академічної групи 126-20-1
(шифр)

спеціальності 126 Інформаційні системи та технології
(код і назва спеціальності)

за освітньо-професійною програмою

«Інформаційні системи та технології»
(офіційна назва)

на тему Розробка Android-додатку моніторингу витрат користувача
(назва за наказом ректора)

Керівники	Прізвище, ініціали	Оцінка за шкалою		Підпис
		рейтинговою	інституційною	
кваліфікаційної роботи	доц. Сергєєва К.Л.			
розділів:				

Рецензент	доц. Ширін А.Л			

Нормоконтролер	проф. Коротенко Г.М.			

Дніпро
2024

ЗАТВЕРДЖЕНО:
завідувач кафедри
_____інформаційних технологій _____
_____та комп'ютерної інженерії_____
(повна назва)
_____ _Гнатушенко В.В._
(підпис) (прізвище, ініціали)

«_____»_____2024_року

ЗАВДАННЯ
на кваліфікаційну роботу
ступеня ____бакалавра_____
(бакалавра, спеціаліста, магістра)

студенту_____Дакаленку Д.О._____ **академічної групи** _____126-20-1____
(прізвище та ініціали) (шифр)
спеціальності _____126 «Інформаційні системи та технології»_____
за освітньою-професійною програмою _____
(за наявності)
_____«Інформаційні системи та технології»_____

на тему ___Розробка Android-додатку моніторингу витрат користувача
_____

затверджену наказом ректора НТУ «Дніпровська політехніка» від 23.05.2024. .№ 469-с

Розділ	Зміст	Термін виконання
Розділ 1	Аналіз стану області рішення задач	05.02.2024 – 11.03.2024
Розділ 2	Практичний розділ	12.03.2024 – 15.04.2024

Завдання видано_____ _____К.Л. Сергєєва_____
(підпис керівника) (прізвище, ініціали)

Дата видачі __05.02.2024 р._____

Дата подання до екзаменаційної комісії____02.06.2024 р.___

Прийнято до виконання_____ _____Дакаленко Д.О._____
(підпис студента) (прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 93 с., 22 рис., 1 таблиця, 1 додатку, 10 джерел. Об’єкт розробки: мобільний застосунок для управління та контролю за фінансами на платформі Android. Мета роботи: розробка мобільного застосунку, який забезпечить користувачам можливість автоматизації процесів обліку фінансів, ефективного контролю доходів і витрат, а також генерації фінансових звітів. Завдання роботи включають:

1. Дослідження основних принципів розробки мобільних додатків на платформі Android.
2. Аналіз вимог до програмного застосунку для обліку фінансів та визначення основних функціональних можливостей.
3. Проектування архітектури програмного застосунку та розробка користувацького інтерфейсу.
4. Реалізація всіх необхідних функціональностей програмного застосунку, включаючи керування транзакціями, розрахунок загального балансу та перегляд звітів.
5. Проведення тестування та налагодження програмного застосунку для забезпечення його стабільної роботи.
6. Оцінка ефективності та зручності використання додатка через проведення користувацького тестування.

У першому розділі роботи проведено огляд методів та підходів до розробки мобільних застосунків для обліку фінансів, розглянуто актуальність фінансової грамотності та персонального фінансового менеджменту. Проаналізовано існуючі програмні рішення та технології для розробки мобільних додатків на платформі Android.

У другому розділі описано процес налаштування середовища для розробки мобільних застосунків, вибір оптимальних інструментів та технологій. Проаналізовано і описано архітектуру програмного застосунку, а також розроблено користувацький інтерфейс. Описано процес розробки

застосунку, включаючи реалізацію основних функцій, таких як додавання, редагування та видалення транзакцій, генерування звітів, а також інтеграція з різними сервісами. Проведено тестування проекту та підтверджено його функціональність.

Практична цінність цієї кваліфікаційної роботи полягає у створенні сучасного мобільного застосунку, який допомагає користувачам ефективно управляти своїми фінансами, підвищуючи фінансову обізнаність та сприяючи досягненню фінансових цілей.

Ключові слова: мобільний застосунок, управління фінансами, облік доходів і витрат, Android, автоматизація фінансових процесів.

ABSTRACT

Explanatory note: 93 pages, 22 figures, 1 table, 1 appendice, 10 references. Object of development: a mobile application for finance management and control on the Android platform. Objective: to develop a mobile application that provides users with the ability to automate finance accounting processes, effectively manage income and expenses, and generate financial reports. The tasks of the study include:

1. Investigating the fundamental principles for developing mobile applications on the Android platform.
2. Analyzing the requirements for finance accounting software and identifying key functional capabilities.
3. Designing the application architecture and developing the user interface.
4. Implementing all necessary functionalities of the application, including transaction management, overall balance calculation, and report viewing.
5. Conducting testing and debugging of the application to ensure its stable operation.
6. Evaluating the efficiency and user-friendliness of the application through user testing.

The first section provides an overview of methods and approaches for developing mobile applications for finance management, highlights the importance of financial literacy and personal financial management, and analyzes existing software solutions and technologies for developing Android mobile applications.

The second section describes the process of setting up the development environment for mobile applications, selecting optimal tools and technologies. It includes an analysis and description of the application architecture and the development of the user interface; details the application development process, including the implementation of core functions such as adding, editing, and deleting transactions, generating reports, and integrating with various services. The project was tested to confirm its functionality.

The practical significance of this qualification work lies in the creation of a modern mobile application that helps users effectively manage their finances, enhancing financial awareness and aiding in achieving financial goals.

Keywords: mobile application, finance management, income and expense accounting, Android, financial process automation.

ЗМІСТ

ВСТУП

Ефективне управління особистими фінансами відіграє важливу роль у досягненні мінімальної фінансової стабільності та досягненні поставлених цілей. Зі стрімким розвитком мобільних технологій та збільшенням популярності мобільних пристроїв й застосунків до них, розробка застосунків для обліку фінансів стає дедалі актуальною.

Під час аналізу актуальності розробки застосунку для обліку фінансів на платформі Android, можна виділити кілька аспектів, які підкреслюють його важливість і практичність.

По-перше, як вже було згадано, це зростання популярності мобільних пристроїв, бо з кожним роком кількість користувачів мобільних пристроїв, особливо на платформі Android, продовжує тільки зростати.

Мобільні телефони та планшети стали невід'ємною частиною нашого повсякденного життя, тому багато користувачів віддають перевагу використанню мобільних застосунків для управління своїми фінансами через їхню доступність і зручність у будь-якому місці перебування.

По-друге, це нагальна потреба в ефективному управлінні фінансами, оскільки особисті фінанси завжди відіграють важливу роль у нашому сучасному житті, тому вміння ефективно управляти доходами та витратами є ключовим аспектом мінімальної фінансової стабільності й досягнення поставлених фінансових цілей.

Ручний облік фінансових операцій може бути трудомістким та схильним до помилок, оскільки існує людський чинник. В той же час додаток для обліку фінансів на платформі Android надає можливостей для автоматизації процесів під час обліку, значно спрощує введення даних та сприяє швидкому доступу до фінансової інформації. Це у значній мірі економить час та сили користувачів, а також знижує ймовірність помилок до мінімуму.

Додаток для обліку фінансів на платформі Android може допомогти користувачеві аналізувати свої фінанси та генерувати звіти про доходи,

витрати та загальний баланс. Це допомагає користувачам отримувати чітке уявлення про свої фінансові потоки, виявляти тренди та приймати обґрунтовані фінансові рішення, включаючи стратегічні, які розраховані на більш великий проміжок часу.

Усі вищезгадані фактори та чинники вказують на сучасну актуальність розробки застосунку для обліку фінансів на платформі Android. Подібний застосунок допомагає користувачам ефективно управляти своїми фінансами, підвищує фінансову обізнаність та сприяє досягненню фінансових цілей у випадку правильного розподілу власного матеріального становища.

Метою даної роботи є розроблення мобільного застосунку для управління та контролю за фінансами. Він має передбачати можливості для додавання, редагування та видалення транзакцій з вказівкою суми витрат або доходів. Також необхідно передбачити можливість для фіксування суми усіх витрат та доходів й додати вказівку дати для транзакцій, що були здійснені в певний момент часу.

Для досягнення поставленої мети в кваліфікаційній роботі ставляться для вирішення наступні завдання дослідження:

1) дослідити основні принципи для розробки мобільних додатків на платформі Android;

2) проаналізувати вимоги до програмного застосунку для обліку фінансів та визначити основні функціональні можливості;

3) спроектувати архітектуру програмного застосунку, а також розробити користувацький інтерфейс;

4) реалізувати усі необхідні функціональності програмного застосунку, включно з керуванням транзакціями, розрахунком загального балансу та перегляд звітів;

5) провести тестування та налагодження програмного застосунку для забезпечення його стабільної роботи;

6) провести оцінку ефективності та зручності використання додатка через проведення користувацького тестування.

Об’єктом дослідження є процеси роботи мобільного застосунку для управління та контролю за фінансами.

Предметом дослідження є апаратно-програмне забезпечення для розроблення мобільного застосунку для управління та контролю за фінансами.

Практичне значення одержаних результатів полягає у підвищенні якості управління та контролю за фінансами.

Результатами роботи є сучасний мобільний застосунок, який працює на операційній системі Android й дозволяє управляти та контролювати власні фінанси.

РОЗДІЛ 1 АНАЛІЗ СТАНУ ОБЛАСТІ РІШЕННЯ ЗАДАЧІ

1.1 Актуальність дослідження фінансової грамотності та поняття персонального фінансового менеджменту.

Фінансова грамотність у сучасному світі стає важливою темою для досліджень та впровадження нових методів і засобів управління особистими фінансами. Зі зростаючою складністю фінансового середовища та постійними змінами в економіці та на фінансових ринках, розуміння основних принципів фінансів стає необхідним для кожної людини. Однією з ключових переваг вивчення фінансової грамотності є здатність ефективно управляти особистими фінансами. Особи, які володіють відповідними знаннями здатні приймати обґрунтовані рішення щодо бюджетування, інвестування та управління ризиками, що підвищує їх стійкість до фінансових труднощів та забезпечує більш стабільне фінансове майбутнє.

Дослідження фінансової грамотності також висвітлює проблему фінансової неграмотності, яка може спричинити серйозні фінансові труднощі як для окремих осіб так і для суспільства в цілому. Підвищення рівня фінансової освіченості може сприяти зміцненню економічної стабільності та зниженню ризиків виникнення фінансових криз.

Особистий фінансовий менеджмент є системним підходом до управління фінансами індивіда або сім'ї з метою досягнення фінансового благополуччя та визначення стратегій для досягнення конкретних фінансових цілей. Цей процес включає аналіз та планування доходів і витрат, визначення пріоритетів у витратах, ефективне використання фінансових інструментів, інвестування, управління ризиками та планування на майбутнє [5].

Основою особистого фінансового менеджменту є бюджетування, яке дозволяє особі чітко визначити куди спрямовуються її кошти та як ефективно використовувати ресурси. Управління кредитами, інвестуванням та

страхуванням є важливими складовими для забезпечення фінансової стійкості та захисту від негативних чинників.

Особистий фінансовий менеджмент також враховує психологічні аспекти фінансових рішень, такі як управління стресом та емоціями. Врахування цих аспектів допомагає створити ефективну стратегію, яка відповідає конкретним потребам та цілям індивіда.

Використання сучасних технологій, таких як – мобільні додатки та онлайн-інструменти може значно спростити управління особистими фінансами, забезпечуючи доступ до інформації та інструментів для прийняття обґрунтованих рішень.

1.2. Методи аналізу особистих фінансів

Оцінка фінансового стану в особистому житті є надзвичайно важливим елементом для забезпечення фінансової стабільності та досягнення амбітних цілей. Фінансове становище визначається різними факторами, такими як рівень доходів, розподіл і ефективність витрат, а також управління фінансовими ресурсами. Цей аналіз є критичним, оскільки він відображає економічну життєздатність і визначає можливості для фінансового зростання та розвитку.

Процес оцінки фінансового стану дозволяє людині чітко зрозуміти своє поточне становище та розробити стратегії для майбутніх дій. Він враховує такі важливі аспекти, як наявність резервних фондів для екстрених ситуацій, можливість накопичення та інвестування для майбутнього, а також забезпечення фінансової безпеки у випадку непередбачених обставин. Цей процес є потужним інструментом для визначення пріоритетів у витратах і раціонального використання ресурсів. Він допомагає мінімізувати зайві фінансові ризики та максимально використовувати можливості для отримання прибутку.

Оцінка фінансового стану є ключовою для створення основи глибокого розуміння власних фінансових цілей і потенціалу. Це, у свою чергу, сприяє ефективному плануванню, управлінню ресурсами та досягненню стійкої фінансової позиції в житті.

Перш за все, слід розглянути, які конкретні аспекти фінансового управління потребують особливої уваги та контролю.

Сфери особистих фінансів

П'ять сфер особистих фінансів – це дохід, заощадження, витрати, інвестування та захист (рис. 1.1).

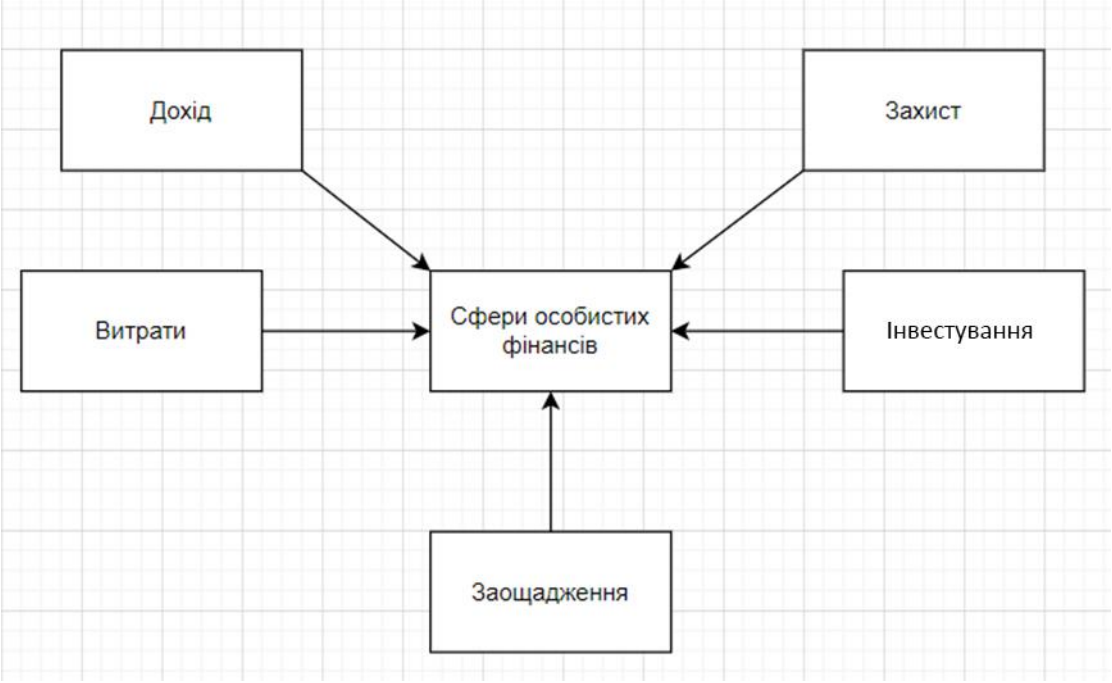

Рис. 1.1. 5 сфер особистих фінансів.

1) Дохід є відправною точкою особистих фінансів, виступаючи як основне джерело грошових надходжень, які можна розподілити на витрати, заощадження, інвестиції та захист. Він включає не лише заробітну плату, але й дивіденди, орендні платежі та інші джерела грошових потоків.

2) Витрати представляють собою відтік грошових коштів і зазвичай забирають основну частину доходу. Вони охоплюють все на що витрачається свій дохід, включаючи оренду, іпотеку, продукти, хобі, харчування, меблі для

дому, ремонт житла, подорожі та розваги. Здатність ефективно управляти витратами є критично важливою, адже нерозумне використання ресурсів може призвести до фінансових проблем, таких як борги з високими відсотковими ставками.

3) Заощадження є тією частиною доходу, що залишається після всіх витрат. Кожен повинен прагнути мати заощадження для покриття непередбачених витрат або надзвичайних ситуацій. Це означає не використовувати весь свій дохід, що може бути складно. Незважаючи на труднощі, важливо мати резервний фонд, який покриватиме від трьох до дванадцяти місячних витрат. Гроші, що просто зберігаються на ощадному рахунку, з часом втрачають свою купівельну спроможність через інфляцію. Тому кошти які не призначені для надзвичайних ситуацій слід інвестувати для збереження або примноження їхньої вартості.

4) Інвестування полягає у придбанні активів, таких як акції та облігації з метою отримання прибутку на вкладені кошти. Воно спрямоване на збільшення багатства понад суму, яку було вкладено. Однак інвестування пов'язане з певними ризиками, оскільки не всі активи зростають у ціні та існує ймовірність зазнати збитків. Для тих, хто не знайомий з інвестуванням, варто присвятити час навчанню або звернутися до професіонала для отримання консультації.

5) Захист фінансових ресурсів включає методи убезпечення від несподіваних подій, таких як хвороби чи нещасні випадки, а також заходи збереження багатства. Це може включати страхування життя і здоров'я, майнове страхування, а також пенсійне планування. Захист є важливою складовою фінансового управління, оскільки він дозволяє забезпечити стабільність та безпеку фінансових ресурсів на довгострокову перспективу.

Серед методів аналізу можна виділити наступні:

- Особиста фінансова звітність

Особиста фінансова звітність є важливим інструментом для оцінки фінансового стану індивідуума в певний момент часу. Цей документ або

електронна таблиця містить загальну інформацію про особу, таку як ім'я та адреса, а також детальну розбивку загальних активів і зобов'язань. Вона слугує потужним засобом для відстеження фінансових цілей і загального стану статків, а також може бути використана при подачі заявок на кредити чи іпотеки.

Розуміння особистої фінансової звітності є ключовим для кожного, хто прагне до фінансової стабільності. Ці звіти можуть бути підготовлені як для компаній, так і для фізичних осіб. Фінансовий звіт фізичної особи, відомий як особиста фінансова звітність, є спрощеною версією корпоративного фінансового звіту. Вони є важливими інструментами для демонстрації фінансового стану суб'єкта, показуючи чисту вартість особи – різницю між її активами і зобов'язаннями.

Ведення оновленої особистої фінансової звітності дозволяє відстежувати зміни у фінансовому становищі з часом. Це безцінний інструмент для тих, хто прагне покращити своє фінансове становище або подати заявку на кредит, уникаючи непотрібних запитів до кредитних звітів і зменшуючи ризик відмови у кредитуванні.

- SWOT-аналіз

SWOT-аналіз є універсальним методом оцінки конкурентної позиції компанії та розробки стратегічного планування. Цей аналіз враховує як внутрішні, так і зовнішні фактори, а також оцінює поточний і майбутній потенціал. Спрямований на створення реалістичної та об'єктивної картини сильних і слабких сторін організації, ініціатив або галузі, що базується на фактах і даних. Він допомагає уникнути попередньо сформованих переконань або "сірих зон", зосереджуючись на реальному контексті. Організації повинні використовувати SWOT-аналіз як орієнтир для стратегічного планування, а не як жорсткий план дій, враховуючи можливість коригування стратегії в міру змін умов.

Кожен SWOT-аналіз включає наступні чотири категорії. Хоча елементи та відкриття в межах цих категорій будуть відрізнятися від компанії до компанії, SWOT-аналіз не буде повним без кожного з цих елементів (рис. 1.2):

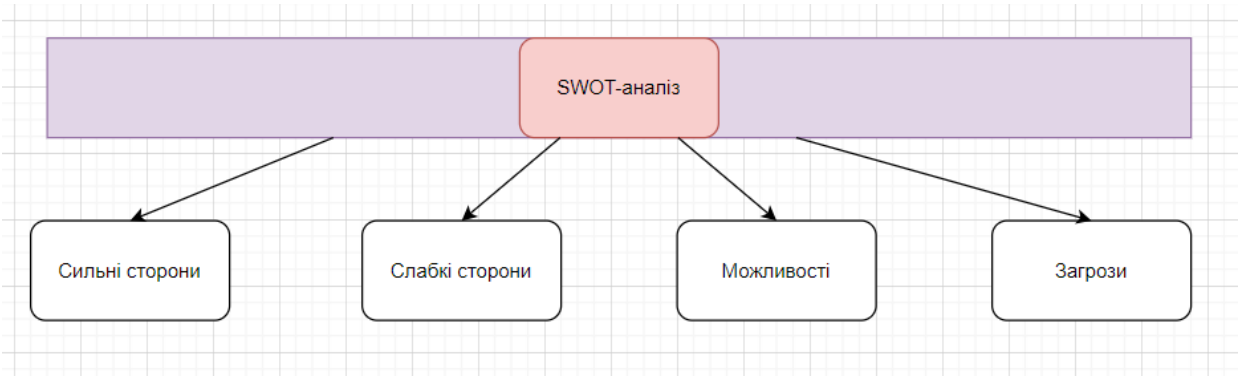

Рис. 1.2. Основні компоненти SWOT аналізу.

- Сильні сторони визначають ті аспекти, в яких організація досягла успіху і які виділяють її серед конкурентів. Це можуть бути відомий бренд, лояльна клієнтська база, стійкий фінансовий баланс, унікальні технології або продукти.
- Слабкі сторони перешкоджають організації працювати на найвищому рівні ефективності. Це ті області, де бізнес потребує вдосконалення для збереження своєї конкурентоспроможності.
- Можливості представляють собою сприятливі зовнішні фактори, які можуть надати організації конкурентну перевагу.
- Загрози є факторами, які можуть завдати шкоди організації.

Аналітики представляють SWOT-аналіз у вигляді квадрата, розділеного на чотири квадранти, кожен з яких присвячений одному з елементів SWOT. Така візуальна схема дає швидкий огляд позиції компанії. Хоча всі пункти під певним заголовком можуть бути не однаково важливими, всі вони повинні представляти ключові уявлення про баланс можливостей і загроз, переваг і недоліків тощо.

SWOT-таблиця часто складається так, що внутрішні фактори розташовуються у верхньому рядку, а зовнішні - у нижньому, крім того,

елементи в лівій частині таблиці є більш позитивними/сприятливими аспектами, в той час як елементи в правій частині є більш проблемними/негативними елементами (рис. 1.3).

Рис. 1.3. Приклад таблиці SWOT

- Матриця прийняття рішень Ейзенхауера

Матриця Ейзенхауера – це метод організації завдань за їх терміновістю та важливістю, що дозволяє ефективно розставляти пріоритети у роботі. Її автором є Дуайт Д. Ейзенхауер, 34-й президент США та п'ятизірковий генерал часів Другої світової війни. У своїй промові 1954 року Ейзенхауер процитував неназваного президента університету, який сказав: "У мене є два типи проблем: термінові та важливі. Термінові не є важливими, а важливі ніколи не бувають терміновими". Ця ідея стала основою для створення Матриці Ейзенхауера.

Відома також як матриця тайм-менеджменту, скринька Ейзенхауера або матриця "термінове - важливе", вона допомагає розподілити завдання на

чотири категорії: ті, що потрібно виконати негайно, ті, що можна відкласти на пізніше, ті, що можна делегувати, та ті, що варто видалити.

Чотири квадранти матриці Ейзенхауера допомагають структурувати довгий список справ, що може здаватися непосильним. Метою цього інструменту є перегляд завдань одне за одним і їх розподіл за квадрантами. Після розподілу завдань у відповідні категорії, ви зможете ефективно планувати свою діяльність і виконувати найважливішу роботу.

Рис. 1.4. Матриця Ейзенхауера

1) Квадрант 1 називається "термінові та важливі". Сюди потрапляють завдання, які мають високу терміновість і важливість. Якщо у вашому списку справ є завдання, яке потребує негайного виконання, має серйозні наслідки та впливає на ваші довгострокові цілі, його слід помістити в цей квадрант. Це можуть бути критичні проекти, невідкладні зустрічі або важливі дедлайни.

2) Квадрант 2 називається "важливі, але не термінові". Сюди входять завдання, які є ключовими для досягнення ваших довгострокових цілей, але не

потребують негайного виконання. Ці завдання варто планувати на майбутнє, оскільки вони сприяють стійкому розвитку і покращенню. Прикладами можуть бути стратегічне планування, навчання новим навичкам, фізичні вправи або робота над особистими проектами.

3) Квадрант 3 – це "термінові, але не важливі". Тут розміщуються завдання, які потребують негайної уваги, але не мають значного впливу на ваші довгострокові цілі. Ці завдання зазвичай є відволікаючими та можуть включати відповіді на електронні листи, телефонні дзвінки або незначні запити. Їх варто делегувати іншим, якщо це можливо, або виконувати лише тоді, коли у вас є вільний час.

4) Квадрант 4 сюди потрапляють завдання, які не є ні терміновими, ні важливими. Ці відволікаючі фактори не приносять жодної реальної користі і лише витрачають ваш час. Приклади включають безцільний перегляд соціальних мереж, надмірне перегляд телевізора або виконання беззмістовних завдань. Їх слід видалити зі списку справ, щоб зосередитися на більш значущих завданнях [5].

Використовуючи матрицю Ейзенхауера, ви зможете ефективніше розподілити свої завдання, зосереджуючись на тих, що дійсно мають значення, і відкидаючи ті, що лише витрачають ваш час. Це дозволить вам досягти ваших довгострокових цілей, покращити продуктивність і зменшити стрес.

1.3. Структура додатків управління фінансами

Додаток для управління фінансами є спеціалізованим інструментом, розробленим для забезпечення користувачів засобами для ефективного контролю над своїми фінансами. Основні функції таких програм включають створення бюджету, відстеження витрат, синхронізацію та моніторинг рахунків, відстеження інвестицій та постановку фінансових цілей. Зокрема, додаток дозволяє користувачам створювати персоналізовані бюджети та встановлювати ліміти витрат для різних категорій, таких як продукти, розваги

та транспорт, а також відстежувати витрати в режимі реального часу, щоб уникнути перевищення встановлених лімітів. Додатково, програма підтримує підключення до банківських рахунків, кредитних карток та інших фінансових рахунків для автоматичного імпорту даних, забезпечуючи миттєве оновлення та надання актуальної інформації про стан рахунків. Це включає сповіщення про великі транзакції або підозрілі активності.

Деякі додатки також пропонують можливість відстежувати динаміку інвестиційних портфелів за допомогою інтерактивних графіків та аналітики в режимі реального часу. Інструменти для аналізу інвестицій включають прогнозування ринку та персоналізовані рекомендації, базовані на історичних даних і ринкових трендах. Щодо постановки фінансових цілей, додаток допомагає користувачам визначати та досягати фінансових цілей, таких як накопичення на придбання автомобіля, будинку або запуск власного бізнесу, надаючи можливість відстежувати прогрес і пропонуючи індивідуальні рекомендації для досягнення цих цілей з урахуванням особистих фінансових показників та звичок.

Користувацький інтерфейс додатку для управління фінансами є вирішальним фактором його зручності та ефективності. Зрозумілий та інтуїтивно зрозумілий інтерфейс сприяє комфортному відстеженню фінансів та досягненню фінансових цілей. Головний екран повинен швидко надавати користувачеві огляд фінансового стану, чітко відображаючи важливі дані, такі як залишки на рахунках, доходи та витрати, а також містити інтерактивні елементи для швидкого доступу до детальнішої інформації. Меню навігації має бути простим і інтуїтивно зрозумілим, забезпечуючи легкий доступ до всіх функцій програми та дозволяючи користувачеві швидко знаходити необхідні інструменти та інформацію з мінімальною кількістю кліків.

Використання інтерактивних діаграм, графіків та інших візуальних елементів сприяє кращому розумінню та аналізу фінансових даних. Візуальні інструменти повинні бути адаптивними дозволяючи користувачеві налаштовувати відображення даних відповідно до своїх потреб. Додаток має

пропонувати можливість налаштування відповідно до індивідуальних потреб користувача, включаючи вибір рахунків для відображення на інформаційній панелі, редагування категорій витрат та налаштування сповіщень. Інтеграція з іншими особистими фінансовими інструментами дозволяє забезпечити централізоване управління фінансами.

Важливою складовою є вбудована система сповіщень, яка допомагає користувачеві завжди бути в курсі змін у фінансових рахунках, наближення до бюджетних лімітів, великих транзакцій та інших важливих подій. Можливість налаштування сповіщень відповідно до особистих потреб дозволяє отримувати лише необхідну інформацію у зручний час, що значно підвищує ефективність користування додатком.

1.4 Аналіз існуючих програмних застосунків для управління та контролю за фінансами

Дивлячись на сучасні існуючі програмні застосунки, які пропонують управляти та контролювати власні фінанси, можна зробити висновок, що серед них переважають ті, що використовують багатоплатформеність у своїх розробках. Тобто популярністю користуються ті компанії, які пропонують своїм користувачам працюючі на різних девайсах застосунки.

Наприклад, застосунок Personal Capital (рис. 1.5) підтримує функціональність на телефонах, планшетах та персональних комп’ютерах.

Рис. 1.5. Приклад застосунку для управління та контролю за фінансами Personal Capital

Personal Capital призначений для управління інвестиціями та фінансами. Він надає можливості для відстежування інвестиційного портфеля, планувати пенсію, контролювати витрати та доходи, а також генерувати звіти [1].

Ще одним схожим програмним застосунком є Quicken (рис. 1.6). Він являється одним із найстаріших додатків для фінансового управління. Quicken надає можливості для відстежування витрат, планування бюджету, керування інвестиціями, створення звітів тощо [2].

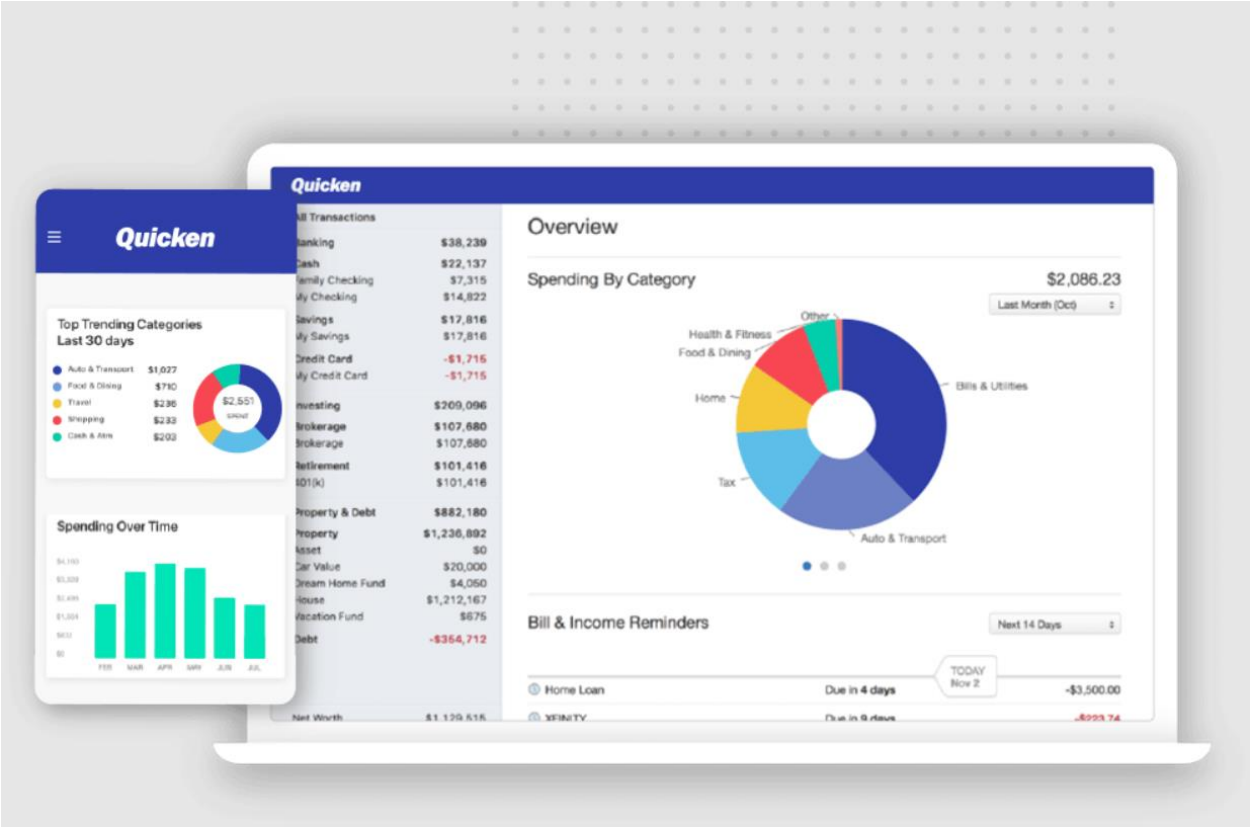

Рис. 1.6. Приклад застосунку для фінансового управління Quicken

PocketGuard (рис 1.7) допомагає відстежувати витрати та доходи, створювати бюджети, а також контролювати свій фінансовий стан. PocketGuard автоматично синхронізується з банківськими рахунками та надає в режимі реального часу інформацію про доступні кошти та поточні витрати [3].

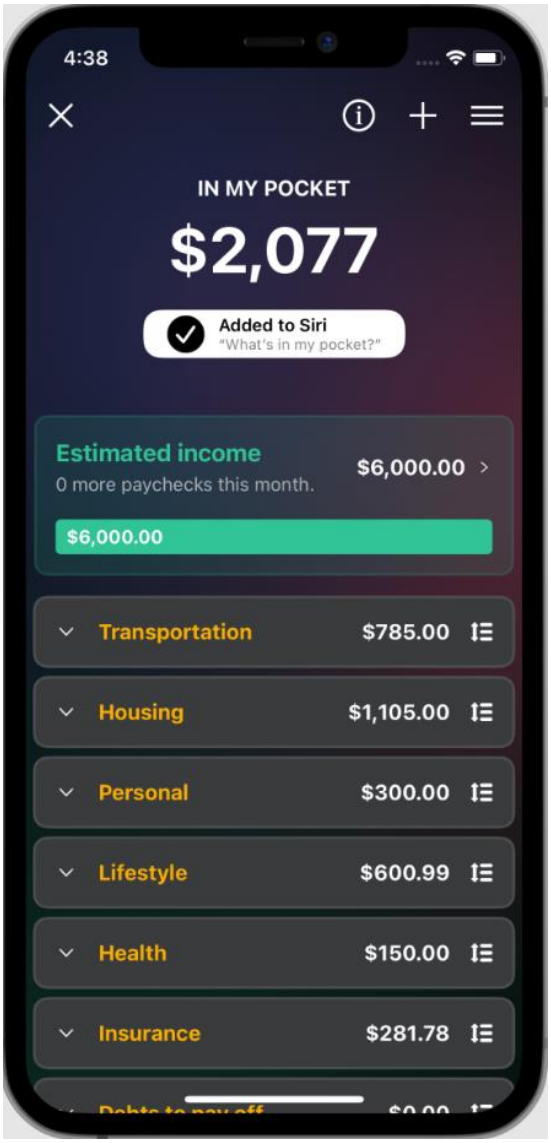

Рис. 1.7. Приклад мобільного застосунку для відслідковування витрат та доходів PocketGuard

І наостанок мобільний застосунок Wally (рис. 1.8) теж надає необхідні можливості для того, аби можна було відстежувати витрати та доходи, створювати бюджети й аналізувати фінансові звички. Додаток також надає графічну звітність та персоналізовані аналітичні дані [4].

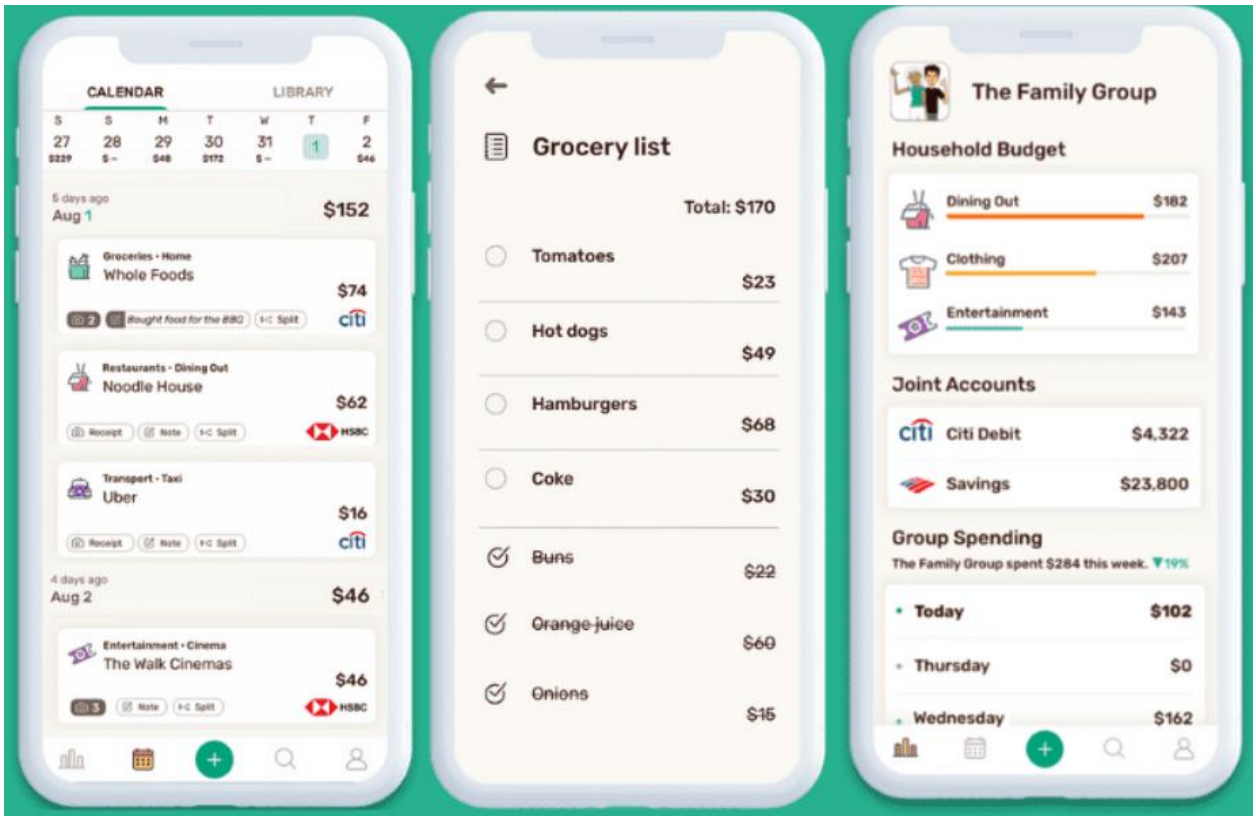

Рис. 1.8. Приклад мобільного застосунку для відстежування витрат та доходів Wally

1.5 Аналіз технологій та засобів розроблення мобільних застосунків для операційної системи Android

Для того, аби створити власний мобільний застосунок під операційну систему Android, всього існує декілька різноманітних засобів, методів та технологій.

Наприклад, Java та Kotlin, які є офіційними мовами програмування для розробки мобільних застосунків під операційну систему Android. Java являється традиційною мовою розробки Android-додатків, а Kotlin – новою, більш сучасною мовою, яка стає все більш популярною серед розробників. Нові застосунки під Android найчастіше використовують саме мову програмування Kotlin, а не Java. Хоча використання Java все ще знаходиться на високому рівні серед розробників.

Офіційне інтегроване середовище розробки (Integrated Development Environment, скорочено IDE) Android Studio (рис. 1.9) від компанії Google дозволяє зручним та простим чином створювати власні Android-додатки з використанням вищезгаданих мов програмування. Android Studio надає великий набір інструментів та функцій, які спрощують розробку, налагодження та тестування додатків для мобільних пристроїв, працюючих з використанням операційної системи Android.

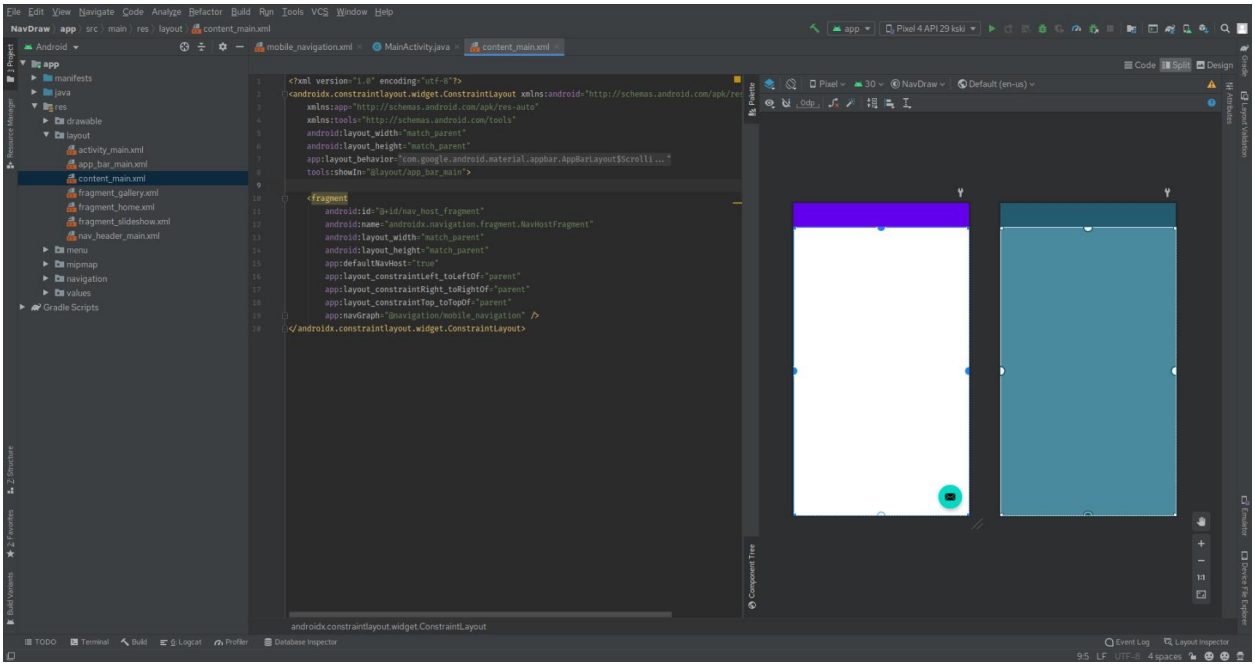

Рис. 1.9. Інтерфейс середовища розробки мобільних застосунків для операційної системи Android Studio

Іншим способом для створення мобільних додатків є використання фреймворків. Наприклад, існують різні фреймворки, які допомагають спростити розробку мобільних додатків під операційну систему Android. Деякі з популярних фреймворків включають Xamarin, React Native, а також Flutter. Перелічені фреймворки надають можливості розробникам використовувати спільний код заради створення застосунків під різні платформи, включаючи Android та iOS.

Android SDK (Software Development Kit) для Android надає великий набір інструментів, бібліотек та ресурсів для того, аби розробляти застосунки під цю операційну систему. Android SDK містить документацію, приклади коду, емулятори та інші інструменти, які допомагають розробникам створювати мобільні додатки під Android.

API та сервіси Google Play теж не відстають від сучасних засобів для розроблення мобільних застосунків. Google надає безліч API (Application Programming Interface) та сервісів, які розробники можуть використовувати у своїх додатках під операційну систему Android. Деякі з них включають Google Maps API, Google Firebase для аутентифікації та хмарних сервісів, Google Pay для платежів тощо.

Використання матеріального дизайну, що є дизайн-мовою, запропонованою компанією Google, є доцільним під час розробки мобільних застосунків. Ця мова забезпечує узгоджений та сучасний користувацький інтерфейс для Android-застосунків. Використання матеріального дизайну дуже допомагає при створенні застосунків, які потребують інтуїтивний та сучасний інтерфейс.

Окрім вищеперерахованих засобів та технологій для створення мобільних додатків під операційну систему Android, існує низка інших популярних інструментів та методів:

1) використання мови програмування C# та Xamarin, платформи для розробки мобільних застосунків, яка використовує мову програмування C#. За її допомогою можна створювати застосунки під операційні системи Android, iOS та інші платформи, використовуючи спільний код;

2) сучасний та потужний ігровий рушій Unity, який також може бути використаний для створення мобільних додатків (рис. 1.10). Його перевага в тому, що він надає інструменти для розробки інтерактивних та графічно багатих додатків під Android;

3) PhoneGap, також відомий як Apache Cordova, представляє собою фреймворк, який дозволяє використовувати сучасні веб-технології (мову

текстової розмітки HTML, каскадну таблицю стилів CSS, мову скриптів JavaScript) для створення мобільних застосунків під різні платформи, включно з операційною системою Android;

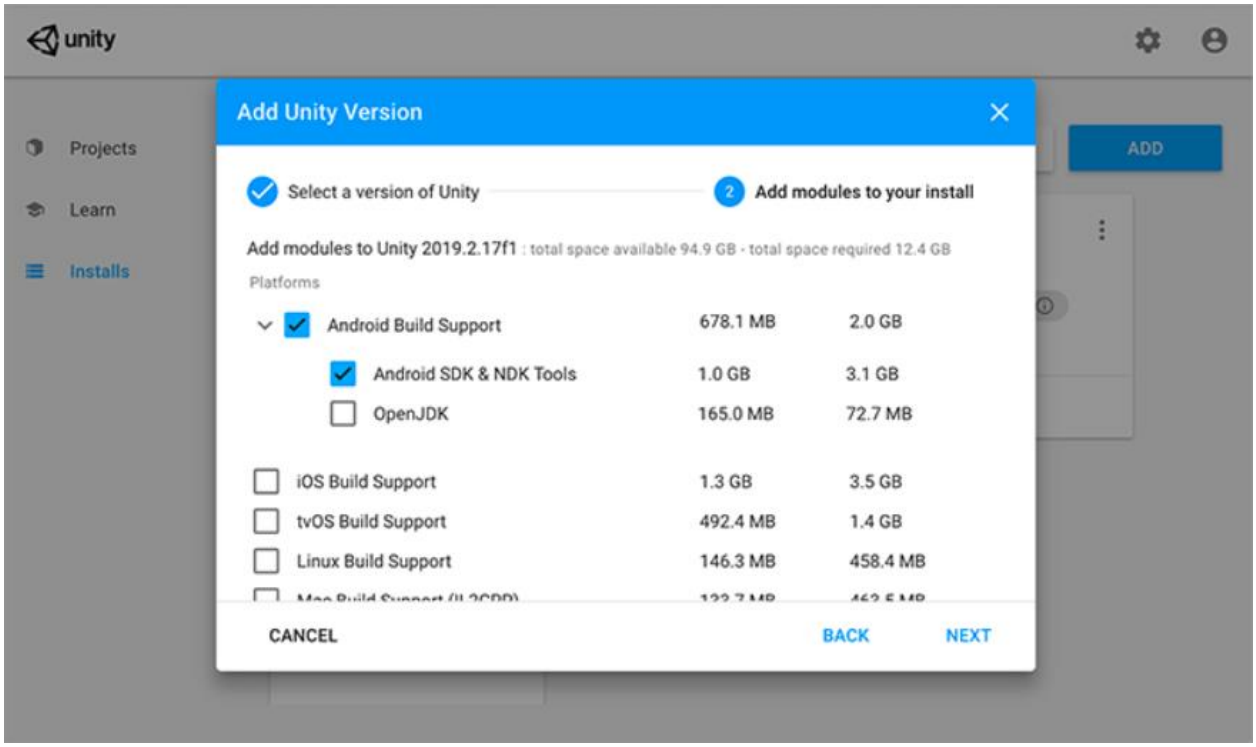

Рис. 1.10. Інтерфейс ігрового рушія та можливість вибору створення застосунку під операційну систему Android

4) використання Appcelerator Titanium, що є ще одним фреймворком, який надає різні можливості розробникам для використання різних веб-технологій заради створення нативних мобільних додатків під Android та інші платформи;

5) використання JavaFX – платформи для створення настільних та мобільних додатків на основі мови програмування Java. Також надає набір інструментів для створення графічних інтерфейсів та роботи з мультимедіа;

6) Android NDK (Native Development Kit) – це інструментарій, що дозволяє розробникам використовувати нативний код, який був написаний мовами C та C++, для створення високопродуктивних компонентів в Android-застосунках.

7) використання платформи Firebase (рис. 1.11) для розробки застосунків, яку пропонує компанія Google. Вона містить набір інструментів та сервісів, таких як база даних у реальному часі, аутентифікація, хмарні сховища, сповіщення тощо, які дозволяють спрощувати розробку мобільних застосунків під операційну систему Android.

Комбінування усіх вищеперерахованих засобів, методів та технологій дозволяють розробникам створювати якісні та сучасні мобільні застосунки під операційну систему Android. Вибір конкретних інструментів та підходів, як і завжди, залежить від вподобань, а також вимог розробника та проєкту.

Готові мобільні застосунки, включно з іграми, можна розміщувати на різних сервісах у публічний доступ. Найпопулярнішим таким сервісом є Google Play (рис. 1.12), де можна знайти різноманітні ігри та застосунки під операційну систему Android [10].

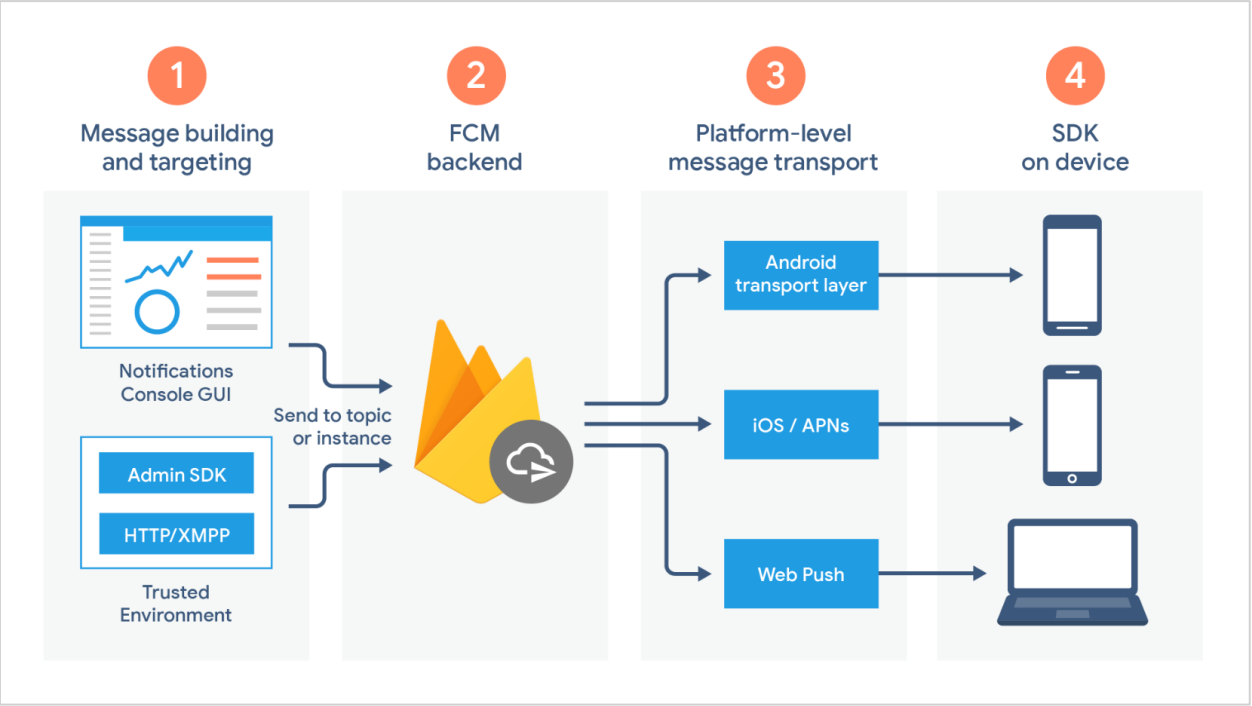

Рис. 1.11. Схема платформи Firebase

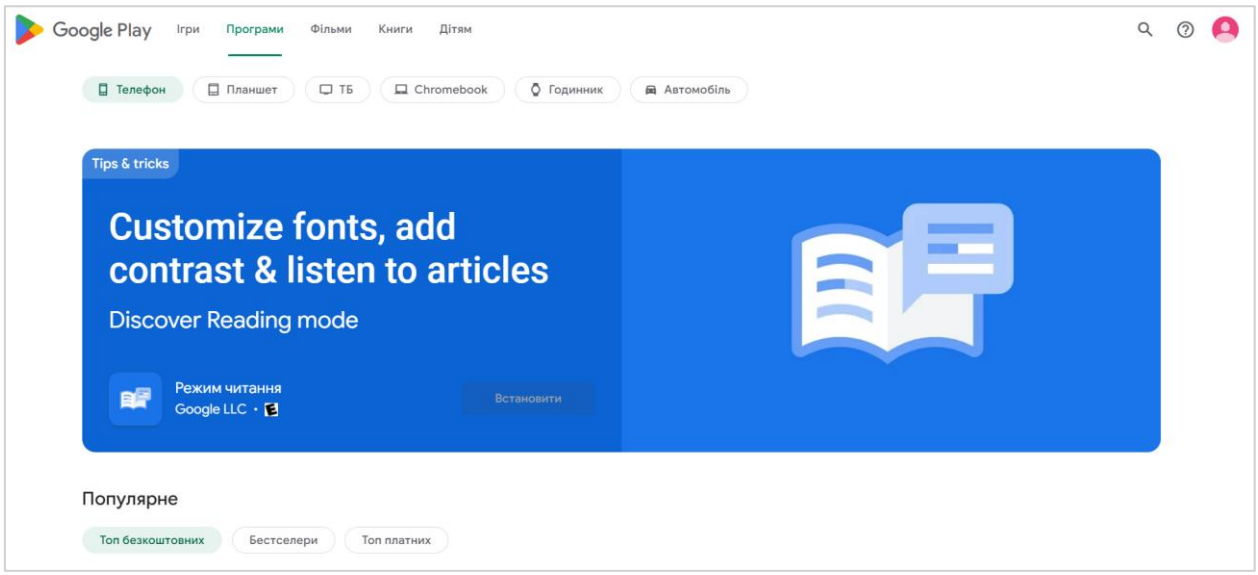

Рис. 1.12. Інтерфейс сервісу для розміщення власних ігор та застосунків під операційну систему Android Google Play

1.6 Висновки до першого розділу. Постановка задач дослідження

Отже, аналіз предметної області показав, що мобільні застосунки посідають важливе місце в нашому сучасному житті та чинять значний вплив на нашу повсякденність. З кожним роком методів, технологій та засобів для їх розроблення стає дедалі більше. Мобільні застосунки можуть використовуватись для розваг та відпочинку, інформаційної діяльності та освіти, а також соціальної взаємодії.

Метою даної роботи є розроблення мобільного застосунку для управління та контролю за фінансами. Він має передбачати можливості для додавання, редагування та видалення транзакцій з вказівкою суми витрат або доходів. Також необхідно передбачити можливість для фіксування суми усіх витрат та доходів й додати вказівку дати для транзакцій, що були здійснені в певний момент часу.

Для досягнення поставленої мети в кваліфікаційній роботі ставились для вирішення наступні завдання дослідження:

1) досліджено основні принципи для розробки мобільних додатків на платформі Android;

2) проаналізовано вимоги до програмного застосунку для обліку фінансів та визначені основні функціональні можливості;

3) спроектовано архітектуру програмного застосунку, а також розроблено користувацький інтерфейс;

4) реалізовано усі необхідні функціональності програмного застосунку, включно з керуванням транзакціями, розрахунком загального балансу та перегляд звітів;

5) проведено тестування та налагодження програмного застосунку для забезпечення його стабільної роботи;

6) проведено оцінку ефективності та зручності використання додатка через проведення користувацького тестування.

РОЗДІЛ 2 ПРАКТИЧНИЙ РОЗДІЛ

2.1 Налаштування середовища для розробки мобільних застосунків

Розробка мобільних застосунків для платформи Android вимагає ретельного налаштування середовища розробки, що включає встановлення необхідних програмних інструментів і конфігурацію проєкту. Основними компонентами, які необхідно налаштувати, є Android SDK та JDK.

2.1.1 Android SDK

Android SDK є набором інструментів, що забезпечують розробникам можливість створення, компіляції, налагодження та тестування застосунків для платформи Android. Android SDK включає такі компоненти, як бібліотеки, відлагоджувальні інструменти, емулятори пристроїв, а також приклади коду та документацію. Використання Android SDK дозволяє розробнику інтегрувати мобільні функції, такі як доступ до сенсорів, камери, геолокації, а також забезпечує сумісність із різними версіями Android.

Для встановлення Android SDK необхідно завантажити Unity Hub з офіційного веб-сайту. Після встановлення необхідно запустити його та вибрати пункт "SDK Manager" у меню інструментів. У SDK Manager вибираються необхідні компоненти, зокрема Android SDK Platform, SDK Tools, SDK Build-Tools та інші залежності, які відповідають мінімальній та цільовій версіям Android, визначеним у проєкті. Також є можливість завантажити всі потрібні інструменти єдиним пакетом.

2.1.2 JDK

Java Development Kit (JDK) є критично важливим інструментом для розробки Android-застосунків, оскільки Android використовує мову програмування Java. JDK містить компілятор, стандартні бібліотеки Java, документацію та інструменти для розробки програмного забезпечення. Для компіляції та виконання Java-програм необхідно мати JDK відповідної версії, сумісної з версією Android SDK.

Для встановлення JDK необхідно завантажити його з офіційного веб-сайту Oracle або альтернативного постачальника, такого як OpenJDK. Після завантаження та встановлення JDK, необхідно додати шлях до JDK у системні змінні середовища, щоб забезпечити доступ до компілятора Java (javac) та інших інструментів JDK з командного рядка або середовища розробки.

У контексті проектів на Unity JDK потрібен лише для компіляції проекту для мобільних пристроїв. Для версії Unity 2021.3.17f1, що використовувався для створення проекту, максимальна та оптимальна версія JDK – 8 [6].

2.1.3 Налаштування конфігурації проєкту для мобільних пристроїв

Після встановлення Android SDK та JDK, необхідно налаштувати Unity для роботи з мобільними платформами. Опис кроків приведений нижче.

1. Вибір платформи розробки

Unity відкривається меню "File" -> "Build Settings". У вікні Build Settings вибирається платформа Android та натискається кнопка "Switch Platform". Це змінює цільову платформу проєкту на Android, що дозволяє використовувати специфічні налаштування та інструменти для Android. Вікно "Build Setting" після встановлення потрібної платформи показане на рисунку 2.1.

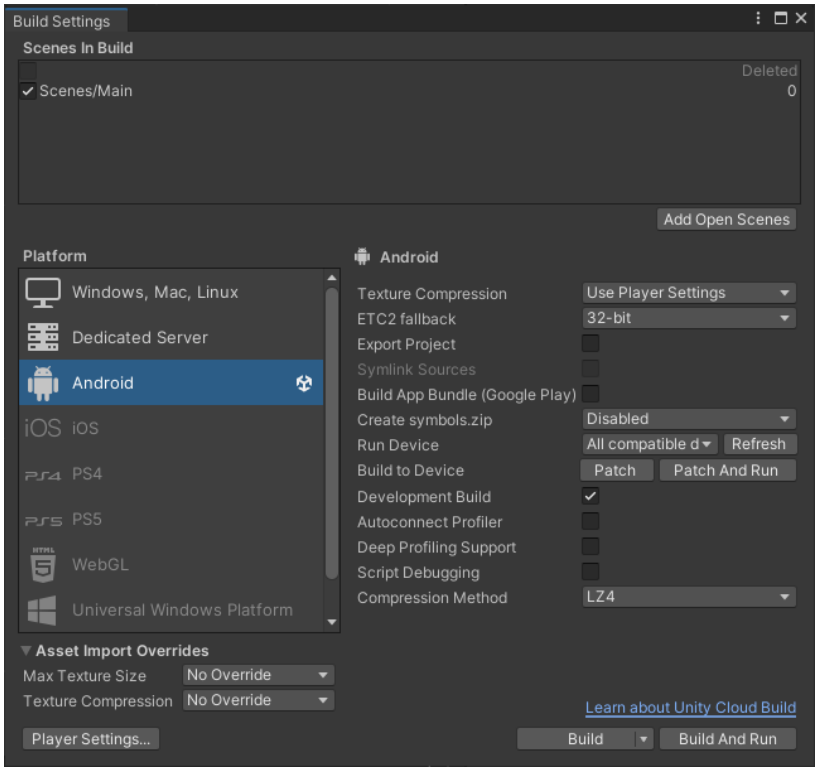

Рис. 2.1. Налаштування Build Settings при виборі цільової платформи Android

2. Налаштування Android проекту

У меню "Edit" -> "Preferences" відкривається вікно налаштувань Unity. У розділі "External Tools" необхідно вказати шляхи до встановлених Android SDK та JDK. Для цього у відповідних полях "SDK" та "JDK" вказуються шляхи до кореневих директорій Android SDK та JDK.

3. Конфігурація Player Settings

У розділі Player Settings налаштовуються параметри проекту для Android. Зокрема, необхідно налаштувати:

a) Company Name та Product Name – поля, визначають назву компанії та продукту що відображаються у метаданих застосунку.

b) Package Name – унікальний ідентифікатор застосунку у форматі com.companyname.productname. Цей ідентифікатор використовується для ідентифікації застосунку у Google Play Market та на пристроях користувачів.

c) Minimum API Level та Target API Level – параметри визначають мінімальну та цільову версії Android, на яких буде працювати застосунок.

Вибір відповідних значень залежить від вимог застосунку та підтримки різних версій Android. У випадку проекта Minimum API Level встановлений до Android 5.1 ‘Lollipop’ (API level 22), а Target API Level – API level 35, що означає, що застосунок підтримує найбільш доступний широкий діапазон версій Android для підтримки.

4. Налаштування дозволів та функціональності

В розділі "Other Settings" у Player Settings налаштовуються дозволи, необхідні для роботи застосунку, такі як доступ до Інтернету, камери, мікрофона тощо. Також у цьому розділі налаштовуються інші параметри, такі як орієнтація екрана, іконка застосунку та налаштування збірки.

5. Встановлення додаткових інструментів

Для тестування застосунку на реальних пристроях необхідно встановити драйвери для підключення Android-пристроїв до комп'ютера. Крім того, можуть знадобитися інструменти для моніторингу продуктивності та налагодження, такі як Android Device Monitor або Logcat.

Налаштування середовища для розробки мобільних застосунків є критично важливим етапом, що забезпечує коректну роботу всіх інструментів та компонентів, необхідних для створення, налагодження та тестування застосунків для платформи Android. Встановлення та конфігурація Android SDK та JDK, а також налаштування Unity для роботи з мобільними пристроями, забезпечують основу для подальшої розробки та оптимізації мобільного застосунку [7].

2.2 Створення сцени та додаткових панелей інтерфейсу

Архітектура застосунку передбачає реалізацію всіх функціональних можливостей на одній основній сцені, де різні вікна та діалоги реалізовані як додаткові панелі інтерфейсу. Такий підхід дозволяє спростити керування сценами, зменшити час завантаження та підвищити ефективність використання ресурсів. У закритому стані додаткові панелі знаходяться за

межами видимості Main Camera, забезпечуючи їх відсутність у полі зору користувача та зменшуючи навантаження на рендеринг.

Основна сцена містить базові елементи інтерфейсу, такі як головне меню, панель навігації та статичні UI елементи. Кожна додаткова панель відповідає за конкретну функціональність, наприклад, за додання нових даних або вибір фільтру за категорією. Для організації цих елементів використовуються Canvas компоненти, які забезпечують коректне відображення UI елементів на різних роздільних здатностях та співвідношеннях сторін екрану. Вигляд сцени у інтерфейсі Unity показано на рисунку 2.2.

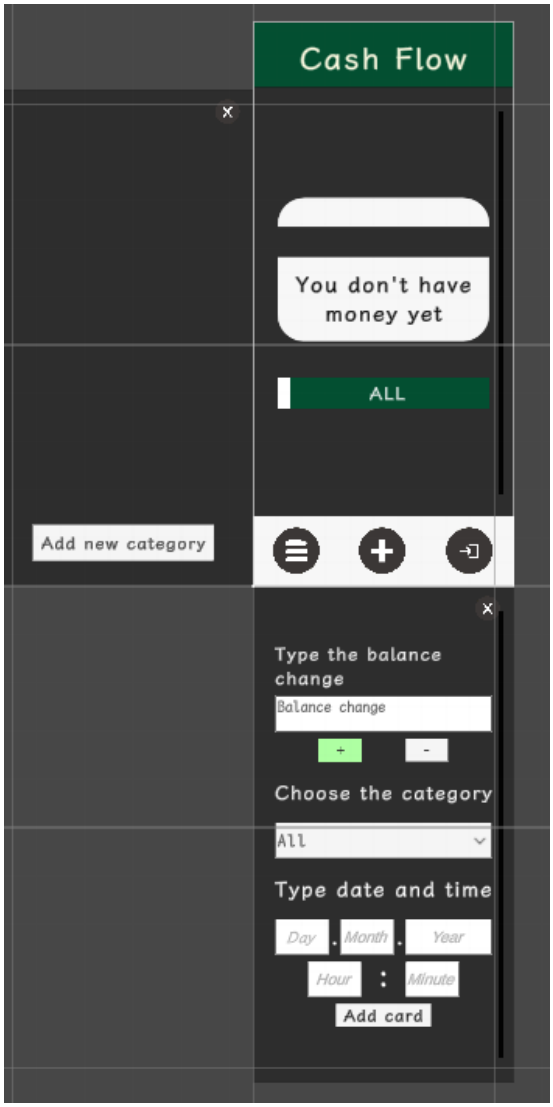

Рис. 2.2. Зображення головної панелі з двома додатковими

Для кожної панелі інтерфейсу налаштовується окремий RectTransform компонент, що визначає її розташування та розміри. У закритому стані панелі мають координати за межами видимості Main Camera, що досягається шляхом зміни параметрів RectTransform, наприклад, зсуву по осі X або Y. У відкритому стані панелі переміщуються до видимої області за допомогою анімацій, забезпечуючи користувачеві доступ до їх функціональності.

Адаптивність інтерфейсу є критичним аспектом розробки мобільних застосунків, оскільки забезпечує коректне відображення елементів на різних пристроях з різними роздільними здатностями та співвідношеннями сторін. Для досягнення адаптивності використовуються різні інструменти та методики, надані Unity [8].

Для забезпечення адаптивності зазвичай використовується режим "Scale With Screen Size", що дозволяє налаштувати референсну роздільну здатність та співвідношення сторін, до яких будуть адаптовані всі елементи інтерфейсу.

Також для адаптивності для пристроїв з різним співвідношенням сторін екрану використовується налаштування анкорів (Anchors) та півотів (Pivots) у RectTransform компонентів. Анкори дозволяють прив'язувати UI елементи до певних частин екрану (кутів, центрів, країв), забезпечуючи їх правильне розташування при зміні розміру екрану. Півоти визначають точку обертання та масштабування елементу. Важливо враховувати, що розміщення відбувається відносно батьківського елементу. Правильне налаштування анкорів та півотів дозволяє забезпечити адаптивне розташування та розміри UI елементів на різних пристроях. Наприклад, розташування кнопки закриття додаткової панелі у правому верхньому куті панелі виглядає як показано на рисунку 2.3.

Рис. 2.3. Розміщення елементу у правому верхньому куті батьківського об’єкту за допомогою анкорів

Щоб інтерфейс виглядав коректно на пристроях з різним співвідношенням сторін, використовується компонент Aspect Ratio Fitter. Він дозволяє підтримувати задане співвідношення сторін для UI елементів, автоматично змінюючи їх розміри відповідно до розміру екрану. Це особливо корисно для елементів, які повинні зберігати свої пропорції, таких як зображення або кнопки з іконками. Для надійності для відмальованих елементів, що не мають розтягуватися, таких як іконки, використовується налаштування Preverse Aspect компоненту Image. Означає що співвідношення сторін зображення змінюватися не буде, а масштабування відбуватиметься за найменшою зі сторін.

Для забезпечення коректної роботи адаптивного інтерфейсу необхідно проводити ретельне тестування на різних пристроях. Unity надає можливість використовувати симулятори різних роздільних здатностей та співвідношень сторін прямо в редакторі, що дозволяє виявити та виправити проблеми з відображенням UI ще до завантаження на реальний пристрій. Частина переліку доступних моделей для Simulator показана на рисунку 2.4.

Рис. 2.4. Моделі смартфонів, доступні для симуляції застосунку у середовищі розробки Unity

2.3 Створення анімацій для переходів між панелями інтерфейсу

У Unity для створення та керування анімаціями використовується вкладка Animation та Animator, що надають потужний функціонал для реалізації різноманітних анімаційних ефектів.

Вкладка Animation в Unity дозволяє створювати анімації для будь-яких об'єктів сцени. Основним інструментом є таймлайн, на якому розташовані ключові кадри (keyframes). Кожен ключовий кадр визначає стан об'єкта в певний момент часу, включаючи його позицію, масштаб, обертання та інші параметри.

Для створення анімації необхідно обрати об'єкт, до якого буде застосовуватись анімація, і відкрити вкладку Animation. Після натискання кнопки "Create" створюється новий Animation Clip, який автоматично

додається до обраного об'єкта. Далі можна додавати ключові кадри, змінюючи параметри об'єкта на таймлайні.

На таймлайні можна змінювати різноманітні параметри об'єктів, такі як позиція, масштаб, обертання, прозорість (alpha) та інші властивості, доступні у компоненті RectTransform та інших компонентах об'єкта. Для додавання ключового кадру необхідно перемістити таймлайн до потрібного моменту часу, змінити параметри об'єкта та натиснути кнопку "Add Keyframe". Unity автоматично інтерполює значення між ключовими кадрами, створюючи плавний перехід між ними. Інтерфейс вкладки Animation з прикладом простої анімації зміни положення (зміна анкорів об’єкту) показана на рисунку 2.5.

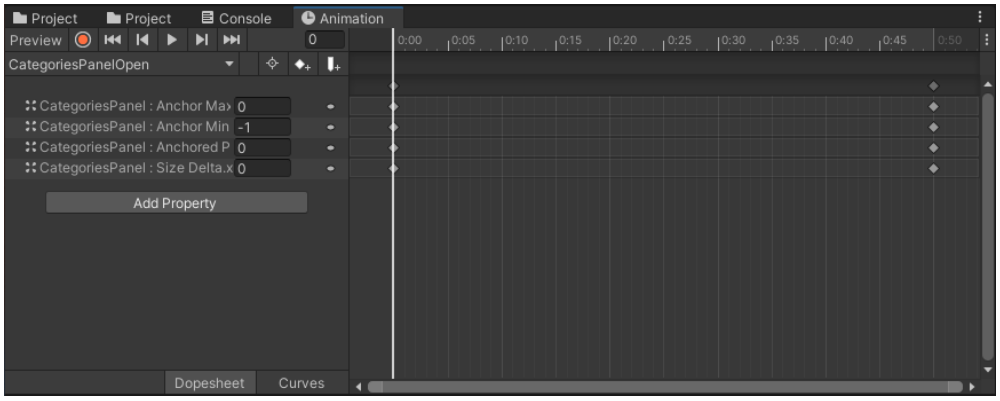

Рис. 2.5. Зміна положення об’єкту за допомогою анімації

Animator є інструментом для керування складними анімаційними станами об'єктів. Він використовує систему графів станів (state machine), що дозволяє легко керувати переходами між різними анімаціями. Кожен стан у Animator представляє окрему анімацію, яку можна створити або імпортувати. Переходи між станами визначаються за допомогою умов (conditions), що базуються на змінних аніматора. Наприклад, перехід може виконуватись при зміні певного параметра (float, int, bool або trigger). Змінні аніматора реалізують базовий принцип керування переходами між анімаціями. Вони дозволяють створювати умови для переходів та змінювати анімаційні стани динамічно під час виконання програми. Для додавання змінної необхідно

відкрити вкладку Animator, перейти до розділу Parameters та додати нову змінну відповідного типу. Потім ця змінна може використовуватись для визначення умов переходів між станами. У якості прикладу приведено налаштування переходу до панелі категорій на рисунку 2.6. Зліва на рисунку видно змінну, за допомогою якої відбувається керування переходами, в центрі – саму схему переходів, а зправа – налаштування переходу від анімації відкритого стану до анімації закритого. У налаштуванні переходу можна побачити, що він відбувається згідно до змінної аніматора.

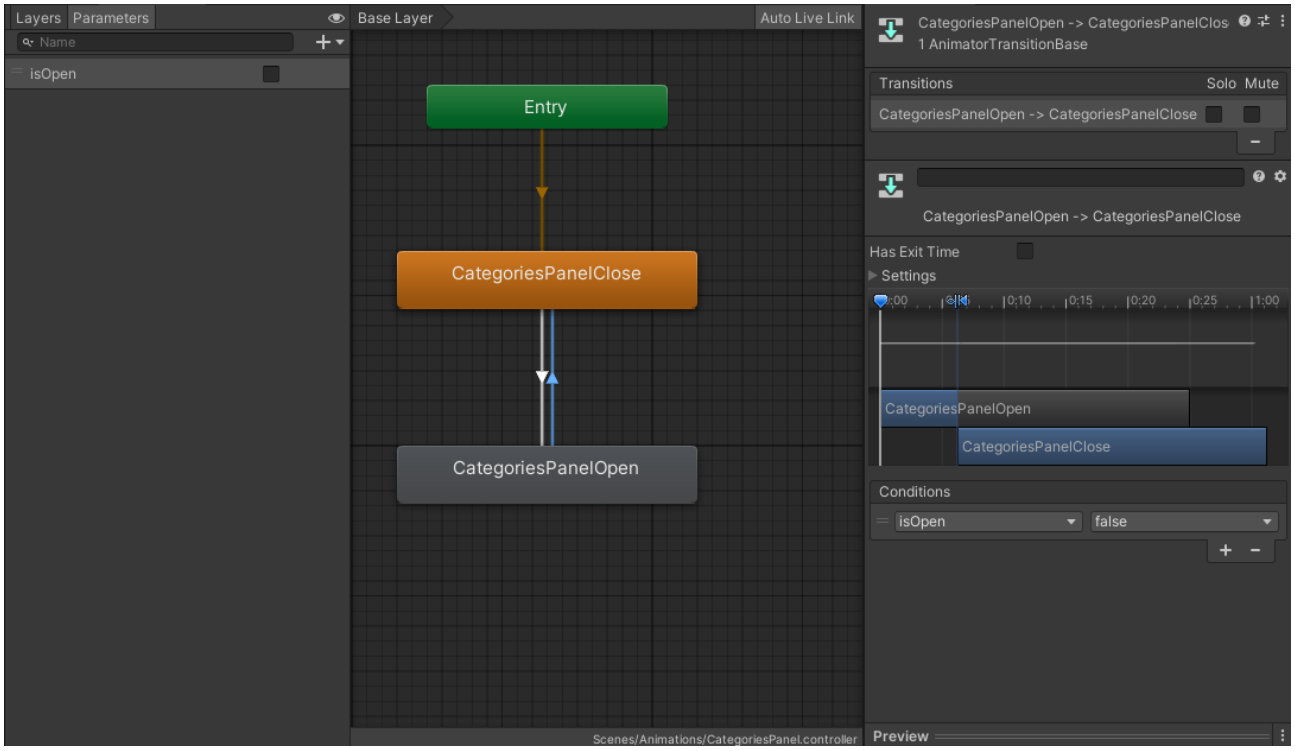

Рис. 2.6. Налаштування вкладки Animator для панелі категорій

В основному для управління змінними аніматора було розроблено клас ShowPanels, в якому я розташував методи з модифікатором доступу public, що дозволяє використовувати їх за допомогою викликів з об’єкту, до якого належить компонент. Ось частина коду, на якій видно принцип запису значень змінних аніматора:

```
public class ShowPanels : MonoBehaviour
```

```
{
    public Animator addPanel;
    public Animator categoryPanel;
    public Animator addCategoryPanel;

    void Start()
    {
        addPanel.SetBool("isOpen", false);
        categoryPanel.SetBool("isOpen", false);
        addCategoryPanel.SetBool("isOpen", false);
    }


    public void AddPanelShow() {
        addPanel.SetBool("isOpen", true);
        ClearForm();
    }

    public void CategoryPanelShow() {
        categoryPanel.SetBool("isOpen", true);
    }

    public void AddCategoryPanelShow() {
        addCategoryPanel.SetBool("isOpen", true);
    }

    public void AddPanelClose() {
        addPanel.SetBool("isOpen", false);
    }

    public void CategoryPanelClose() {
        categoryPanel.SetBool("isOpen", false);
    }

    public void AddCategoryPanelClose() {
        addCategoryPanel.SetBool("isOpen", false);
    }

    public void QuitApp() {
        Application.Quit();
    }
}
```

Функція QuitApp() використовується для виходу з застосунку і розроблена для прив’язки до кнопки виходу.

2.4 Алгоритми взамодії з даними

PlayerPrefs є одним із найпростіших та найпоширеніших способів зберігання та отримання даних у Unity. Він дозволяє зберігати дані у форматі ключ-значення, які зберігаються локально на пристрої користувача. PlayerPrefs підтримує зберігання трьох типів даних: цілих чисел (int), чисел з плаваючою комою (float) та рядків (string).

Основні методи PlayerPrefs включають:

1) PlayerPrefs.SetInt(string key, int value): зберігає ціле число під заданим ключем;
2) PlayerPrefs.GetInt(string key, int defaultValue = 0): отримує ціле число, збережене під заданим ключем. Якщо значення не знайдено, повертає значення за замовчуванням;
3) PlayerPrefs.SetFloat(string key, float value): зберігає число з плаваючою комою під заданим ключем;
4) PlayerPrefs.GetFloat(string key, float defaultValue = 0.0f): отримує число з плаваючою комою, збережене під заданим ключем;
5) PlayerPrefs.SetString(string key, string value): зберігає рядок під заданим ключем;
6) PlayerPrefs.GetString(string key, string defaultValue = ""): отримує рядок, збережений під заданим ключем;
7) PlayerPrefs.DeleteKey(string key): видаляє значення, збережене під заданим ключем;
8) PlayerPrefs.HasKey(string key): перевіряє наявність ключа.

У таблиці 2.1 наведений вміст PlayerPrefs проекту, а також позначено призначення кожного запису. Ініціалізація всіх записів при запуску застосунку виконано в класі PlayerPrefsConfig.

Таблиця 2.1

Вміст сховища PlayerPrefs

Ключ	**Тип**	**Призначення**	**Значення за замовчуванням**
LastCategoryID	int	ID останньої створеної категорії.	2
CurrentCategory	int	ID поточно обраної категорії для фільтрації.	0
Category_0_Name	string	Назва категорії з ID = 0.	"ALL"
Category_0_HEX	string	HEX-код кольору категорії з ID = 0.	"#FFFFFF"
Category_1_Name	string	Назва категорії з ID = 1.	"Food"
Category_1_HEX	string	HEX-код кольору категорії з ID = 1.	"#FFD000"
Category_2_Name	string	Назва категорії з ID = 2.	"Leisure"
Category_2_HEX	string	HEX-код кольору категорії з ID = 2.	"#009EFF"

LastCardID	int	ID останньої створеної картки.	0
CurrentBalance	float	Теперішній загальний баланс.	0.0
CurrentChanching	int	ID картки, яку зараз змінюють. Значення "-1" вказує на стан без змін.	-1
Card_{ID}_Change	float	Зміна балансу у картці з ID = {ID}.	Не встановлено
Card_{ID}_CurrentBalance	float	Загальний баланс після внесеної зміни у картці з ID = {ID}.	Не встановлено
Card_{ID}_Category	int	Категорія картки з ID = {ID} (записується ID категорії).	Не встановлено

Функціонал проекту зображений на блок-схемі на рисунку 2.7.

Рис. 2.7 – Блок-схема функціоналу застосунку “Cash Flow”

Введення даних відбувається за допомогою форми, яка створена за допомогою поєднання об’єктів з компонентами TMPro (текст), TMPro_InputField (поле вводу), Dropdown Menu (спадний список), Toggle Group (група чекбоксів) та об’єкту з компонентом Button (кнопка) для підтвердження обробки форми.

Клас AddCardFormManager служить для обробки даних цієї форми, яка використовується для створення або оновлення карток фінансових транзакцій. Він містить кілька публічних змінних, що представляють елементи форми, такі як текстові поля введення, групи перемикачів, випадаючий список та кнопку підтвердження.

У методі Start для кнопки підтвердження реєструється обробник подій, що викликає метод OnSubmit під час натискання. У методі OnSubmit виконується валідація форми за допомогою методу ValidateForm. Якщо валідація успішна, з текстових полів зчитуються дані, включаючи дату і час, зміну балансу, вибрану категорію та стан перемикача, який вказує на тип транзакції (позитивна або негативна) [9].

```
    int day = int.Parse(dayInputField.text);
    int month = int.Parse(monthInputField.text);
    int year = int.Parse(yearInputField.text);
    int hour = int.Parse(hourInputField.text);
    int minute = int.Parse(minuteInputField.text);
    DateTime dateTime = new DateTime(year, month, day, hour,
minute, 0);
    float balanceChange =
float.Parse(balanceChangeInputField.text);
    bool isPositive = GetToggleValue();
    string category =
categoryDropdown.options[categoryDropdown.value].text;
```

Далі перевіряється, чи існує поточна картка для редагування, використовуючи ID, збережений у PlayerPrefs під ключем CurrentChanging. Якщо ID дорівнює -1, створюється нова картка за допомогою методу CreateCard, інакше існуюча картка оновлюється за допомогою методу UpdateCard.

У методі ValidateForm послідовно перевіряються коректність введених даних для дати, часу та зміни балансу, і якщо будь-яка з перевірок не пройшла, метод повертає false.

Метод ValidateDate перевіряє, чи є значення дня, місяця та року допустимими числами, і чи можна створити об'єкт DateTime з цими значеннями без винятків. Аналогічно, метод ValidateTime перевіряє коректність значень години та хвилини. Метод ValidateBalanceChange перевіряє, чи є значення зміни балансу допустимим числом з плаваючою комою.

Метод GetToggleValue визначає стан перемикача, що вказує на тип транзакції (позитивна чи негативна), повертаючи true для позитивної і false для негативної.

```
foreach (Toggle toggle in toggleGroup.ActiveToggles())
{
    return toggle.name == "Plus";
}
return true;
```

Методи CreateCard та UpdateCard обробляють логіку створення та оновлення карток відповідно. Вони зберігають нові значення змін балансу, категорій та дат у PlayerPrefs. Після цього панель форми закривається, а форма очищається для нових введень.

```
PlayerPrefs.SetFloat("Card_" + ID + "_Change", change);
PlayerPrefs.SetFloat("Card_" + ID + "_CurrentBalance",
currentBalance + change);
PlayerPrefs.SetFloat("CurrentBalance", currentBalance +
change);
PlayerPrefs.SetInt("Card_" + ID + "_Category", categoryID);
PlayerPrefs.SetString("Card_" + ID + "_Date",
dateTimeString);
addPanel.SetBool("isOpen", false);
ClearForm();
```

Метод GetCategoryID знаходить ID категорії за її назвою, шукаючи відповідні значення у PlayerPrefs.

```
for (int categoryID = 0; categoryID <=
PlayerPrefs.GetInt("LastCategoryID"); categoryID++)
{
    if (PlayerPrefs.GetString("Category_" + categoryID +
"_Name") == categoryName)
    {
        return categoryID;
    }
}
return -1;
```

Метод ClearForm очищає всі поля форми та скидає значення поточної картки для редагування в PlayerPrefs.

```
dayInputField.text = "";
monthInputField.text = "";
yearInputField.text = "";
hourInputField.text = "";
minuteInputField.text = "";
balanceChangeInputField.text = "";
PlayerPrefs.SetInt("CurrentChanging", -1);
```

Таким чином, клас AddCardFormManager забезпечує повний цикл обробки введення даних користувача, їх валідацію та збереження, дозволяючи створювати та редагувати картки фінансових транзакцій у додатку на Unity.

Клас CardsGenerate відповідає за генерування та відображення карток фінансових транзакцій у Unity. Цей клас використовує префаб картки для створення нових об'єктів карток та розміщує їх у відповідному батьківському об'єкті. Префаб картки показаний на рисунку 2.8. На ньому введені випадкові дані для демонстрації вигляду картки, при генерації карток як безпосередньо програмного об’єкту, дані змінюються.

У методі Start викликається функція GenerateCards, яка відповідає за створення всіх карток на початку роботи сцени. Після генерації всіх карток, положення батьківського об'єкта змінюється, щоб приховати його за межами видимого екрану.

```
void Start()
{
    GenerateCards();
    parent.localPosition = new
Vector3(parent.localPosition.x, -10000, parent.localPosition.z);
}
```

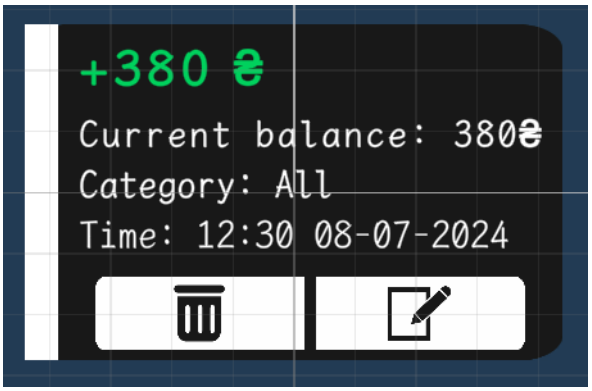

Рис. 2.8 – Префаб картки

У методі Update перевіряється, чи необхідно оновити картки. Якщо це так, знову викликається метод GenerateCards.

```
void Update()
{
    if(env.NeedRefreshCards) {
        GenerateCards();
    }
}
```

Функція GenerateCards відповідає за створення всіх карток транзакцій при запуску сцени або при необхідності оновлення карток. Спочатку вона отримує ідентифікатор останньої картки та поточну категорію з PlayerPrefs. Далі всі старі картки, за винятком спеціальних об'єктів "CardBalance" і "Category", видаляються з батьківського об'єкта. Після цього функція проходить через всі можливі ідентифікатори карток до останнього і перевіряє, чи існують дані для кожної картки в PlayerPrefs. Якщо дані існують і відповідають вибраній категорії, викликається функція CreateCard, яка створює нову картку з відповідними даними. Після створення всіх карток прапорець NeedRefreshCards скидається, а батьківський об'єкт змінює своє положення, щоб приховати його за межами видимого екрану.

```
    private void GenerateCards()
    {
        int lastCardID = PlayerPrefs.GetInt("LastCardID");
        int currentCategory =
PlayerPrefs.GetInt("CurrentCategory");
```

```
        foreach (Transform child in parent)
        {
            if (child.name != "CardBalance" && child.name != 
"Category")
            {
                Destroy(child.gameObject);
            }
        }

        for (int ID = 0; ID <= lastCardID; ID++)
        {
            if (PlayerPrefs.HasKey("Card_" + ID + "_Change"))
            {
                int cardCategoryID = PlayerPrefs.GetInt("Card_" 
+ ID + "_Category");
                if (currentCategory == 0 || cardCategoryID == 
currentCategory)
                {
                    CreateCard(ID);
                }
            }
        }

        env.NeedRefreshCards = false;
        parent.localPosition = new 
Vector3(parent.localPosition.x, -10000, parent.localPosition.z);
    }
```

Функція CreateCard в класі CardsGenerate відповідає за створення нової картки транзакції та її додавання до батьківського об'єкта на сцені. Спочатку створюється новий об'єкт картки з префабу і встановлюється його ім'я, включаючи ідентифікатор картки. Далі функція витягує збережені дані картки з PlayerPrefs, такі як зміна балансу, поточний баланс, категорія, дата та колір. Ці дані використовуються для заповнення відповідних елементів інтерфейсу користувача на картці. Поля тексту, такі як зміна балансу, поточний баланс, назва категорії та дата, отримують свої значення, а також відповідним чином змінюється колір картки. Додатково, функція UpdateTextColor використовується для встановлення кольору тексту зміни балансу залежно від його значення: позитивне, негативне або нульове.

```
    private void CreateCard(int ID)
```

```
    {
        GameObject newCard = Instantiate(cardPrefab, parent);

        newCard.name = "Card_" + ID;

        float change = PlayerPrefs.GetFloat("Card_" + ID +
"_Change");
        float currentBalance = PlayerPrefs.GetFloat("Card_" +
ID + "_CurrentBalance");
        int categoryID = PlayerPrefs.GetInt("Card_" + ID +
"_Category");
        string date = PlayerPrefs.GetString("Card_" + ID +
"_Date");
        string hexColor = PlayerPrefs.GetString("Category_" +
categoryID + "_HEX");

        TextMeshProUGUI balanceChangeText =
newCard.transform.Find("BalanceChange").GetComponent<TextMeshPro
UGUI>();
        TextMeshProUGUI balanceCurrentText =
newCard.transform.Find("BalanceCurrent").GetComponent<TextMeshPr
oUGUI>();
        TextMeshProUGUI categoryNameText =
newCard.transform.Find("CategoryName").GetComponent<TextMeshProU
GUI>();
        TextMeshProUGUI dateText =
newCard.transform.Find("Date").GetComponent<TextMeshProUGUI>();
        Image cardImage =
newCard.transform.Find("Color").GetComponent<Image>();

        balanceChangeText.text = (change > 0 ? "+" : "") +
change.ToString("F2") + " ₴";
        balanceCurrentText.text = "Current balance: " +
currentBalance.ToString("F2") + " ₴";
        categoryNameText.text = "Category: " +
GetCategoryName(categoryID);
        dateText.text = "Date: " + GetDate(date);
        cardImage.color = HexToColor(hexColor);

        UpdateTextColor(balanceChangeText, change);
        UpdateTextColor(balanceCurrentText, currentBalance);
    }
```

Метод GetCardDateTime конвертує строкове представлення дати та часу у формат DateTime.

```
    private DateTime GetCardDateTime(string dateTimeString)
    {
        DateTime dateTime;
```

```
        if (DateTime.TryParse(dateTimeString, out dateTime))
        {
            return dateTime;
        }
        else
        {
            throw new Exception("Помилковий формат дати та
часу");
        }
    }
```

Метод UpdateTextColor змінює колір тексту залежно від значення балансу.

```
    private void UpdateTextColor(TextMeshProUGUI balanceObj,
float balance)
    {
        if (balance > 0)
        {
            balanceObj.color = positiveBalanceColor;
        }
        else if (balance < 0)
        {
            balanceObj.color = negativeBalanceColor;
        }
        else
        {
            balanceObj.color = zeroBalanceColor;
        }
    }
```

Клас CardActions забезпечує функціональність для редагування та видалення карток фінансових транзакцій у Unity. Він містить методи для отримання даних картки, видалення картки, редагування картки та допоміжні методи для взаємодії з елементами UI.

У методі Start виконується ініціалізація посилань на компоненти UI, такі як панель додавання картки, поля вводу значень та дати, група перемикачів та випадаючий список категорій.

```
    void Start()
    {
        addCardPanel = GameObject.Find("Main
Camera/UI/AddCardPanel").GetComponent<Animator>();
```

```
        cashValueInput = GameObject.Find("Main
Camera/UI/AddCardPanel/Content/CashValueInput").GetComponent<TMP
_InputField>();
        dayInput = GameObject.Find("Main
Camera/UI/AddCardPanel/Content/DateInput/DayInput").GetComponent
<TMP_InputField>();
        monthInput = GameObject.Find("Main
Camera/UI/AddCardPanel/Content/DateInput/MonthInput").GetCompone
nt<TMP_InputField>();
        yearInput = GameObject.Find("Main
Camera/UI/AddCardPanel/Content/DateInput/YearInput").GetComponen
t<TMP_InputField>();
        hourInput = GameObject.Find("Main
Camera/UI/AddCardPanel/Content/TimeInput/HourInput").GetComponen
t<TMP_InputField>();
        minuteInput = GameObject.Find("Main
Camera/UI/AddCardPanel/Content/TimeInput/MinuteInput").GetCompon
ent<TMP_InputField>();
        toggleGroup = GameObject.Find("Main
Camera/UI/AddCardPanel/Content/BalanceChangeSign").GetComponent<
ToggleGroup>();
        categoryDropdown = GameObject.Find("Main
Camera/UI/AddCardPanel/Content/CategoryDropdown").GetComponent<T
MP_Dropdown>();
    }
```

Метод DeleteCard видаляє дані картки з PlayerPrefs, оновлює поточний баланс, та встановлює прапорець для необхідності оновлення карток.

```
    public void DeleteCard()
    {
        int cardID = GetCardIDFromName();
        if (cardID == -1)
        {
            Debug.LogWarning($"Не вдалось отримати cardID з
імені: {gameObject.name}");
            return;
        }

        if (PlayerPrefs.HasKey("Card_" + cardID + "_Change"))
        {
            PlayerPrefs.SetFloat("CurrentBalance",
PlayerPrefs.GetFloat("CurrentBalance") -
PlayerPrefs.GetFloat("Card_" + cardID + "_Change"));
            PlayerPrefs.DeleteKey("Card_" + cardID +
"_Change");
            PlayerPrefs.DeleteKey("Card_" + cardID +
"_CurrentBalance");
```

```
            PlayerPrefs.DeleteKey("Card_" + cardID +
"_Category");
            PlayerPrefs.DeleteKey("Card_" + cardID + "_Date");
            env.NeedRefreshCards = true;
        }
        else
        {
            Debug.LogWarning($"Карточка з ID {cardID} не
знайдена.");
        }
    }
```

Функція EditCard в класі CardActions відповідає за редагування існуючої картки транзакції. Спочатку функція отримує ідентифікатор картки з її імені. Якщо ідентифікатор недійсний, функція видає попередження та припиняє виконання. Якщо дані картки існують в PlayerPrefs, ідентифікатор картки зберігається в PlayerPrefs як поточний редагований. Далі з PlayerPrefs витягуються дані картки, такі як зміна балансу, поточний баланс, категорія та дата. Ці дані використовуються для заповнення відповідних полів вводу в інтерфейсі користувача, таких як поле вводу значення зміни, поля вводу дати та часу, а також значення для групи перемикачів та випадаючого списку категорій. Наприклад, значення зміни балансу встановлюється в поле вводу, а відповідні значення дати та часу розбиваються на складові частини і заповнюються у відповідні поля. Функція SetToggleGroupValue використовується для встановлення значення перемикача залежно від знака зміни балансу, а функція GetCategoryIndex знаходить індекс категорії у випадаючому списку. Врешті-решт, панель редагування картки відкривається шляхом встановлення відповідного значення анімації.

```
    public void EditCard()
    {
        int cardID = GetCardIDFromName();
        if (cardID == -1)
        {
            Debug.LogWarning($"Не вдалось отримати cardID з
імені: {gameObject.name}");
            return;
        }
        if (PlayerPrefs.HasKey("Card_" + cardID + "_Change"))
```

```
        {
            PlayerPrefs.SetInt("CurrentChanging", cardID);
            float change = PlayerPrefs.GetFloat("Card_" +
cardID + "_Change");
            float currentBalance = PlayerPrefs.GetFloat("Card_"
+ cardID + "_CurrentBalance");
            int categoryID = PlayerPrefs.GetInt("Card_" +
cardID + "_Category");
            string dateTime = PlayerPrefs.GetString("Card_" +
cardID + "_Date");
            cashValueInput.text = change.ToString();
            DateTime cardDateTime = GetCardDateTime(dateTime);
            dayInput.text = cardDateTime.ToString("dd");
            monthInput.text = cardDateTime.ToString("MM");
            yearInput.text = cardDateTime.ToString("yyyy");
            hourInput.text = cardDateTime.ToString("HH");
            minuteInput.text = cardDateTime.ToString("mm");
            SetToggleGroupValue(change > 0 ? "Plus" : "Minus");
            categoryDropdown.value =
GetCategoryIndex(categoryID);
            addCardPanel.SetBool("isOpen", true);
        }
        else
        {
            Debug.LogWarning($"Карточка с ID {cardID} не
знайдена.");
        }
    }
```

Метод GetCardIDFromName витягує ID картки з її імені, яке починається з "Card_".

```
    private int GetCardIDFromName()
    {
        string name = gameObject.name;
        if (name.StartsWith("Card_"))
        {
            string idString = name.Substring(5);
            int cardID;
            if (int.TryParse(idString, out cardID))
            {
                return cardID;
            }
        }
        return -1;
    }
```

Метод GetCardDateTime конвертує строкове представлення дати та часу у формат DateTime.

```
    private DateTime GetCardDateTime(string dateTimeString)
    {
        DateTime dateTime;
        if (DateTime.TryParse(dateTimeString, out dateTime))
        {
            return dateTime;
        }
        else
        {
            throw new Exception("Помилковий формат дати та
часу");
        }
    }
```

Метод SetToggleGroupValue встановлює відповідне значення для ToggleGroup, залежно від знака зміни балансу.

```
    private void SetToggleGroupValue(string toggleName)
    {
        foreach (Toggle toggle in
toggleGroup.GetComponentsInChildren<Toggle>())
        {
            if (toggle.name == toggleName)
            {
                toggle.isOn = true;
                break;
            }
        }
    }
```

Метод GetCategoryIndex знаходить індекс категорії за її ID у випадаючому списку.

```
    private int GetCategoryIndex(int categoryID)
    {
        for (int i = 0; i < categoryDropdown.options.Count;
i++)
        {
            if (categoryDropdown.options[i].text ==
PlayerPrefs.GetString("Category_" + categoryID + "_Name"))
            {
                return i;
            }
```

```
        }
        return 0;
    }
```

У файлі ChangeCurrentCategory.cs визначено клас, який забезпечує функціонал для зміни поточної категорії в інтерфейсі користувача. На початку відбувається ініціалізація, під час якої клас намагається знайти об'єкт CategoriesPanel у сцені за допомогою методу GameObject.Find. Якщо об'єкт знайдено, він отримує компонент Animator, що буде використовуватись для керування анімацією панелі категорій.

Метод SetCurrentCategory відповідає за встановлення поточної категорії. Він отримує назву об'єкта, на якому викликається, перевіряє чи вона відповідає шаблону "Category_{ID}", і витягує ідентифікатор категорії. Якщо ідентифікатор витягнуто успішно, метод зберігає його в PlayerPrefs, що дозволяє зберегти значення між сесіями. Також встановлюється прапорець NeedRefreshCards у глобальному середовищі env, що сигналізує про необхідність оновлення карток. Після цього анімація панелі категорій закривається.

У файлі CategoriesGenerate.cs описаний клас для генерації категорій на основі збережених даних. Клас містить публічні поля для префабу категорії та батьківського об'єкта, до якого будуть додаватися створені категорії. Під час запуску сцени метод Start викликає приватний метод GenerateCategories, який відповідає за створення категорій.

Метод GenerateCategories спочатку видаляє всі наявні категорії, щоб запобігти дублюванню, а потім циклічно створює нові категорії на основі збережених даних у PlayerPrefs. Якщо для певного ідентифікатора знайдено збережені дані, викликається метод CreateCategory, який створює новий об'єкт категорії, встановлює його назву та заповнює відповідні текстові та графічні поля на основі збережених даних.

Метод HexToColor виконує перетворення кольору з шістнадцяткової форми у формат, що зрозумілий Unity, використовуючи

ColorUtility.TryParseHtmlString. У випадку невдалого парсингу повертається білий колір.

Обидва класи взаємодіють з користувацькими налаштуваннями через PlayerPrefs, що дозволяє зберігати і відновлювати стан інтерфейсу між запусками програми. Клас ChangeCurrentCategory відповідає за динамічну зміну поточної категорії, тоді як CategoriesGenerate забезпечує початкову генерацію категорій на основі попередньо збережених даних.

2.5 Інструкція користувача

Додаток, розроблений на основі Unity, забезпечує зручне управління фінансовими транзакціями через створення, редагування та видалення карток. Користувачеві надається можливість інтерактивно взаємодіяти з інтерфейсом для ефективного ведення обліку фінансів. Для повного розуміння роботи додатку нижче наведено детальну інструкцію щодо взаємодії з усіма його функціональними можливостями.

Після запуску додатку користувач потрапляє на головний екран, де відображаються всі створені фінансові картки. Картки представляють собою графічні об'єкти, що містять інформацію про транзакції: зміну балансу, поточний баланс, категорію, дату та час транзакції. Спочатку додаток завантажує всі необхідні дані з PlayerPrefs і генерує картки, відображаючи їх у відповідному батьківському об'єкті. Завантаження даних та генерація графічних об’єктів відбувається автоматично. Вигляд головного меню з карткою показаний на рисунку 2.9.

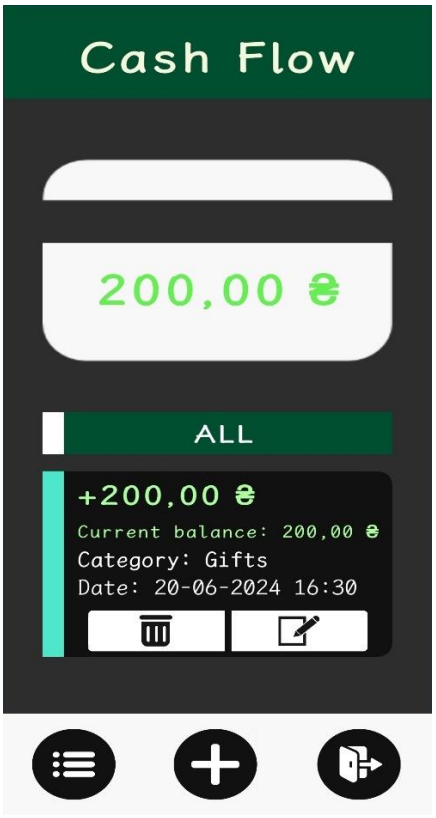

Рис. 2.9. Головна панель застосунку “Cash Flow”

Для створення нової картки транзакції необхідно натиснути кнопку, яка відкриває панель додавання картки. На цій панелі розміщено кілька полів вводу, що дозволяють користувачу ввести суму транзакції, дату та час, вибрати категорію транзакції з випадаючого списку та встановити знак зміни балансу через групу перемикачів. Користувач вводить суму транзакції в поле CashValueInput. Дата та час транзакції вводяться у відповідні поля DayInput, MonthInput, YearInput, HourInput та MinuteInput. Знак зміни балансу встановлюється за допомогою вибору відповідного перемикача у групі ToggleGroup, де користувач може вибрати між позитивною або негативною зміною. Категорія транзакції вибирається з випадаючого списку CategoryDropdown. Для кожного з полей вводу додані підказки та заголовки для того, щоб інтуїтивно було зрозуміло що потрібно вводити. Як це виглядає показано на рисунку 2.10. Всі поля я обов’язкові та мають валідацію, тому у разі помилки можна побачити відповідне повідомлення (див. рис. 2.11).

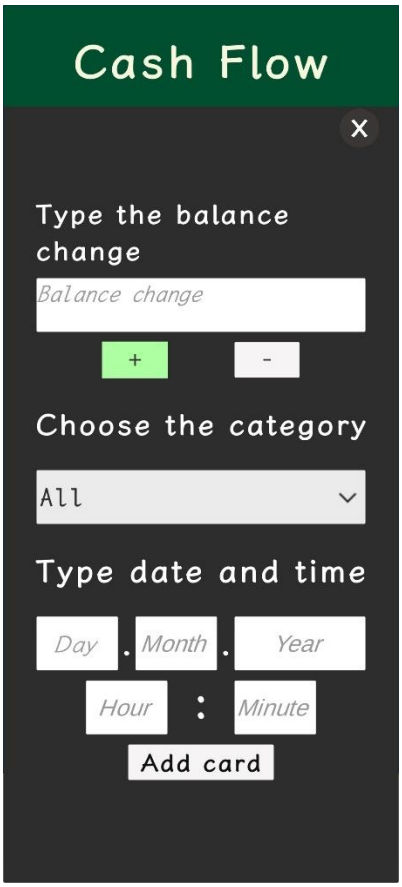

Рис. 2.10. Панель для додання даних нової картки

Рис. 2.11. Повідомлення помилки про невірне введення даних

Після введення всіх даних користувач натискає кнопку підтвердження на панелі додавання картки. Додаток автоматично зберігає всі введені дані у PlayerPrefs та створює нову картку за допомогою функції, яка додає новий об'єкт картки до батьківського об'єкта на сцені. Дані нової картки відображаються у відповідних елементах інтерфейсу, включаючи зміну та поточний баланс, категорію, дату та час транзакції. Також у кожної картки є 2 кнопки: редагувати та видалити.

Для редагування існуючої картки користувач натискає на кнопку на потрібній картці, що відкриває панель редагування. Додаток витягує дані картки з PlayerPrefs і заповнює ними відповідні поля вводу на панелі додання картки. Користувач може змінити суму транзакції, дату та час, вибрати іншу категорію або змінити знак балансу. Після внесення змін необхідно натиснути кнопку підтвердження, що призведе до збереження нових даних у PlayerPrefs і оновлення відображення картки. Приклад вигляду панелі при редагуванні показаний на рисунку 2.12.

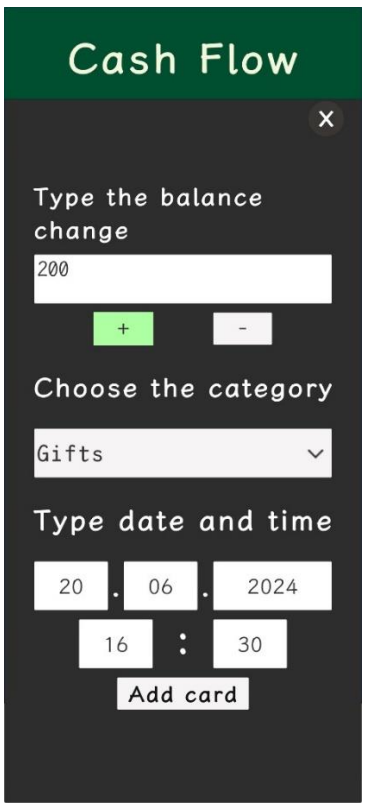

Рис. 2.12. Редагування даних картки

Для видалення картки користувач натискає відповідну кнопку на картці. Додаток видаляє всі пов'язані з карткою дані з PlayerPrefs, включаючи зміну та поточний баланс, категорію та дату транзакції. Потім функція DeleteCard оновлює поточний баланс шляхом віднімання зміни, пов'язаної з видаленою карткою, і встановлює прапорець для необхідності оновлення карток. Картка видаляється з головної панелі.

Кожна картка має відповідну категорію, яка визначається користувачем при створенні або редагуванні картки. Категорії зберігаються у PlayerPrefs і можуть бути вибрані з випадаючого списку на панелі додавання або редагування картки (див рисунки 2.10 та 2.12). Колір категорії також зберігається і відображається на картці, надаючи користувачу візуальну індикацію категорії транзакції. Повний перелік категорій відокремлений на спеціальній панелі (показано на рисунку 2.13).

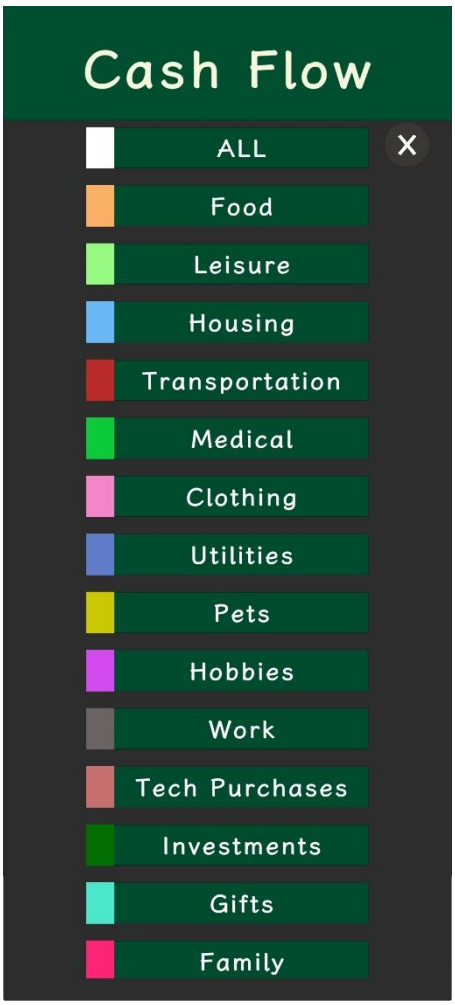

Рис. 2.13. Панель категорій

Користувач може відфільтрувати відображення карток за категоріями, обравши певну категорію. Додаток відображає лише ті картки, які відповідають вибраній категорії, що дозволяє зручно переглядати лише потрібні транзакції. При цьому категорія “ALL” відображує картки всіх інших категорій. Функція GenerateCards забезпечує коректне оновлення відображення карток при зміні вибраної категорії. Наприклад, якщо обрати категорію, в якій немає поточних даних (див. рис. 2.14), можна побачити загальний баланс рахунку, назву обраної категорії та відсутність карток.

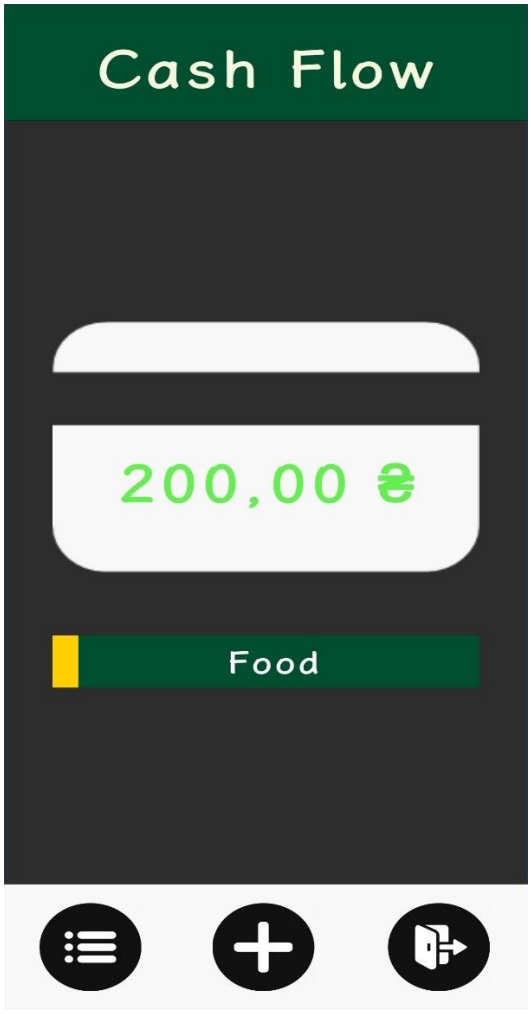

Рис. 2.14. Робота фільтрації відображених даних за категорією

Таким чином, додаток надає зручний інтерфейс для управління фінансовими транзакціями, дозволяючи користувачу легко створювати, редагувати та видаляти картки, а також фільтрувати їх за категоріями. Усі дії виконуються за допомогою інтуїтивно зрозумілих елементів інтерфейсу, що забезпечує ефективне та приємне користування додатком.

ВИСНОВОК

Кваліфікаційна робота присвячена розробці мобільного застосунку для обліку та контролю за фінансами на платформі Android. Це завдання є надзвичайно актуальним у контексті сучасного розвитку мобільних технологій та зростання необхідності ефективного управління особистими фінансами. Метою роботи було створення зручного та функціонального інструменту, який дозволить користувачам автоматизувати процеси обліку фінансів, забезпечуючи швидкий доступ до фінансової інформації та допомагаючи досягати фінансових цілей.

У першому розділі роботи було проведено детальний аналіз предметної області, зокрема дослідження сучасних методів та підходів до розробки мобільних застосунків для обліку фінансів. Особливу увагу було приділено аналізу існуючих рішень, що дозволило визначити ключові тенденції та технології, які використовуються для створення таких додатків. Було досліджено методи аналізу особистих фінансів, такі як фінансова звітність, SWOT-аналіз та матриця Ейзенхауера, що допомогло визначити основні функціональні можливості застосунку.

У другому розділі роботи було обрано оптимальні програмні засоби для реалізації проекту та описано процес розробки застосунку для обліку фінансів Для розробки застосунку було використано мови програмування Java та C#, а також офіційне інтегроване середовище розробки Unity. Це забезпечило потужні можливості для створення мобільного застосунку з високою продуктивністю та зручністю у використанні. Було розглянуто різні інструменти та фреймворки для розробки мобільних додатків, включаючи Android SDK, JDK, Google API та інші, що дозволило забезпечити широку функціональність застосунку. Було детально описано процес розробки застосунку для обліку фінансів. Реалізовано основні компоненти, такі як архітектура додатку, користувацький інтерфейс, алгоритми обробки даних та інтеграція з різними сервісами. Було розроблено та протестовано ключові

елементи застосунку, включаючи додавання, редагування та видалення транзакцій, а також генерування звітів про доходи та витрати. Проведене тестування проекту підтвердило його функціональність та відповідність вимогам.

Розроблений проект демонструє високу ефективність та відповідає сучасним вимогам індустрії програмного забезпечення. Використання сучасних технологій дозволило досягти високої гнучкості та адаптивності коду, що полегшує його подальшу підтримку та розвиток. Результати роботи можуть бути корисними для інших розробників та дослідників у сфері програмної інженерії, які прагнуть створювати інноваційні продукти, що відповідають сучасним тенденціям та очікуванням користувачів.

ЛІТЕРАТУРА

1. Приклад застосунку для управління та контролю за фінансами Personal Capital. [Електронний ресурс]. – Режим доступу: https://www.personalcapital.com/

2. Приклад застосунку для управління та контролю за фінансами Quicken. [Електронний ресурс]. – Режим доступу: https://www.quicken.com/

3. Приклад застосунку для управління та контролю за фінансами PocketGuard. [Електронний ресурс]. – Режим доступу: https://pocketguard.com/

4. Приклад застосунку для управління та контролю за фінансами Wally. [Електронний ресурс]. – Режим доступу: https://www.wally.me/

5. Chechetova, N. F. & Chechetova-Terashvili, T. M. (2019). Financial literacy as a key to the success of personal finance management. World Science, 2(10(50), 14-20. http://doi.org/10.31435/rsglobal_ws/31102019/6725.

6. Unity 3D [Електронний ресурс] – Режим доступу: https://docs.unity.com

7. Майк Гейг. Розробка ігор на Unity за 24 години, 2020. 464 с.

8. Алекс Окіта. Вивчаємо програмування на C# з Unity 3D, друге видання, 2019. 690 с.

9. Офіційний документація C# [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/overview

10. Cross platform app frameworks in 2021 [Електронний ресурс]. – Режим доступу: https://www.netsolutions.com/insights/cross-platform-appframeworks-in-2021

ДОДАТОК А. Лістинг програми

Лістинг AddCardFormManager.cs

```
using UnityEngine;
using TMPro;
using UnityEngine.UI;
using System;
using System.Collections.Generic;

public class AddCardFormManager : MonoBehaviour
{
    public TMP_InputField dayInputField;
    public TMP_InputField monthInputField;
    public TMP_InputField yearInputField;
    public TMP_InputField hourInputField;
    public TMP_InputField minuteInputField;
    public TMP_InputField balanceChangeInputField;
    public ToggleGroup toggleGroup;
    public TMP_Dropdown categoryDropdown;
    public Button submitButton;

    public GameObject errorPanel;
    public Animator addPanel;

    void Start()
    {
        submitButton.onClick.AddListener(OnSubmit);
    }

void OnSubmit()
    {
        if (ValidateForm())
        {
            // Отримання даних
            int day = int.Parse(dayInputField.text);
```

```
            int month = int.Parse(monthInputField.text);
            int year = int.Parse(yearInputField.text);
            int hour = int.Parse(hourInputField.text);
            int minute = int.Parse(minuteInputField.text);
            DateTime dateTime = new DateTime(year, month, day, hour,
minute, 0);

            float balanceChange =
float.Parse(balanceChangeInputField.text);
            bool isPositive = GetToggleValue();
            string category =
categoryDropdown.options[categoryDropdown.value].text;

            // Обробка даних форми
            int currentChangingID =
PlayerPrefs.GetInt("CurrentChanging");
            if (currentChangingID == -1)
            {
                // Додавання нової картки
                CreateCard(isPositive, balanceChange, category,
dateTime);
            }
            else
            {
                // Зміна існуючої картки
                UpdateCard(currentChangingID, isPositive,
balanceChange, category, dateTime);
            }
        }
        else
        {
            errorPanel.SetActive(true);
        }
    }

    bool ValidateForm()
    {
```

```
        bool isValid = true;

        if (!ValidateDate())
        {
            Debug.Log("Помилка дати.");
            isValid = false;
        }

        if (!ValidateTime())
        {
            Debug.Log("Помилка часу.");
            isValid = false;
        }

        if (!ValidateBalanceChange())
        {
            Debug.Log("Помилка зміни балансу.");
            isValid = false;
        }

        return isValid;
    }

    bool ValidateDate()
    {
        int day, month, year;

        if (!int.TryParse(dayInputField.text, out day) || day < 1 ||
day > 31)
        {
            return false;
        }
        if (!int.TryParse(monthInputField.text, out month) || month <
1 || month > 12)
        {
            return false;
```

```
        }
        if (!int.TryParse(yearInputField.text, out year) || year < 1)
        {
            return false;
        }

        try
        {
            DateTime date = new DateTime(year, month, day);
        }
        catch
        {
            return false;
        }

        return true;
    }

    bool ValidateTime()
    {
        int hour, minute;

        if (!int.TryParse(hourInputField.text, out hour) || hour < 0
|| hour > 23)
        {
            return false;
        }
        if (!int.TryParse(minuteInputField.text, out minute) || minute
< 0 || minute > 59)
        {
            return false;
        }

        return true;
    }
```

```
    bool ValidateBalanceChange()
    {
        float balanceChange;

        if (!float.TryParse(balanceChangeInputField.text, out
balanceChange))
        {
            return false;
        }

        return true;
    }

    bool GetToggleValue()
    {
        foreach (Toggle toggle in toggleGroup.ActiveToggles())
        {
            return toggle.name == "Plus";
        }

        return true;
    }

    void CreateCard(bool isPositive, float balanceChange, string
category, DateTime dateTime)
    {
        // Індекс нової картки
        int ID = PlayerPrefs.GetInt("LastCardID") + 1;
        PlayerPrefs.SetInt("LastCardID", ID);

        // Зміна баланса
        float currentBalance = PlayerPrefs.GetFloat("CurrentBalance");
        float change = (int)((isPositive ? balanceChange : -
balanceChange) * 100) / 100f;
        PlayerPrefs.SetFloat("Card_" + ID + "_Change", change);
        PlayerPrefs.SetFloat("Card_" + ID + "_CurrentBalance",
currentBalance + change);
```

```
        PlayerPrefs.SetFloat("CurrentBalance", currentBalance +
change);


        // Категорія
        int categoryID = GetCategoryID(category);
        PlayerPrefs.SetInt("Card_" + ID + "_Category", categoryID);


        // Дата і час
        string dateTimeString = dateTime.ToString("yyyy-MM-dd
HH:mm:ss");
        PlayerPrefs.SetString("Card_" + ID + "_Date", dateTimeString);


        // Закрити панель форми
        addPanel.SetBool("isOpen", false);
        ClearForm();
        env.NeedRefreshCards = true;
    }


    void UpdateCard(int cardID, bool isPositive, float balanceChange,
string category, DateTime dateTime)
    {
        PlayerPrefs.SetFloat("CurrentBalance",
PlayerPrefs.GetFloat("CurrentBalance") - PlayerPrefs.GetFloat("Card_"
+ cardID + "_Change"));


        // Зміна балансу
        float currentBalance = PlayerPrefs.GetFloat("CurrentBalance");
        float change = (int)((isPositive ? balanceChange : -
balanceChange) * 100) / 100f;
        PlayerPrefs.SetFloat("Card_" + cardID + "_Change", change);
        PlayerPrefs.SetFloat("Card_" + cardID + "_CurrentBalance",
currentBalance + change);
        PlayerPrefs.SetFloat("CurrentBalance", currentBalance +
change);


        // Категорія
        int categoryID = GetCategoryID(category);
        PlayerPrefs.SetInt("Card_" + cardID + "_Category",
categoryID);
```

```
        // Дата і час
        string dateTimeString = dateTime.ToString("yyyy-MM-dd
HH:mm:ss");
        PlayerPrefs.SetString("Card_" + cardID + "_Date",
dateTimeString);


        // Закрити панель форми
        addPanel.SetBool("isOpen", false);
        ClearForm();
        env.NeedRefreshCards = true;
    }


    int GetCategoryID(string categoryName)
    {
        for (int categoryID = 0; categoryID <=
PlayerPrefs.GetInt("LastCategoryID"); categoryID++)
        {
            if (PlayerPrefs.GetString("Category_" + categoryID +
"_Name") == categoryName)
            {
                return categoryID;
            }
        }
        return -1; // Якщо категорія не знайдена
    }


    void ClearForm()
    {
        dayInputField.text = "";
        monthInputField.text = "";
        yearInputField.text = "";
        hourInputField.text = "";
        minuteInputField.text = "";
        balanceChangeInputField.text = "";


        // Скинути значення CurrentChanging
```

```
        PlayerPrefs.SetInt("CurrentChanging", -1);
    }
}
```

Лістинг ButtonAudio.cs

```
 using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ButtonAudio : MonoBehaviour
{
    private AudioSource audioSource;

    void Awake()
    {
        audioSource = GetComponent<AudioSource>();
    }

    public void PlaySound(AudioClip audioClip)
    {
        audioSource.clip = audioClip;
        audioSource.Play();
    }
}
```

Лістинг CardActions.cs

```
using System;
using UnityEngine;
using UnityEngine.UI;
using TMPro;

public class CardActions : MonoBehaviour
{

    private Animator addCardPanel;
    private TMP_InputField cashValueInput;
    private TMP_InputField dayInput;
    private TMP_InputField monthInput;
    private TMP_InputField yearInput;
    private TMP_InputField hourInput;
    private TMP_InputField minuteInput;
    private ToggleGroup toggleGroup;
    private TMP_Dropdown categoryDropdown;

    void Start()
    {
        addCardPanel = GameObject.Find("Main
Camera/UI/AddCardPanel").GetComponent<Animator>();
        cashValueInput = GameObject.Find("Main
Camera/UI/AddCardPanel/Content/CashValueInput").GetComponent<TMP_Input
Field>();
        dayInput = GameObject.Find("Main
Camera/UI/AddCardPanel/Content/DateInput/DayInput").GetComponent<TMP_I
nputField>();
```

```
        monthInput = GameObject.Find("Main
Camera/UI/AddCardPanel/Content/DateInput/MonthInput").GetComponent<TMP
_InputField>();
        yearInput = GameObject.Find("Main
Camera/UI/AddCardPanel/Content/DateInput/YearInput").GetComponent<TMP_
InputField>();
        hourInput = GameObject.Find("Main
Camera/UI/AddCardPanel/Content/TimeInput/HourInput").GetComponent<TMP_
InputField>();
        minuteInput = GameObject.Find("Main
Camera/UI/AddCardPanel/Content/TimeInput/MinuteInput").GetComponent<TM
P_InputField>();
        toggleGroup = GameObject.Find("Main
Camera/UI/AddCardPanel/Content/BalanceChangeSign").GetComponent<Toggle
Group>();
        categoryDropdown = GameObject.Find("Main
Camera/UI/AddCardPanel/Content/CategoryDropdown").GetComponent<TMP_Dro
pdown>();
    }

    public void DeleteCard()
    {
        int cardID = GetCardIDFromName();
        if (cardID == -1)
        {
            Debug.LogWarning($"Не удалось получить cardID из имени:
{gameObject.name}");
            return;
        }

        // Видалення усіх даних про картку з PlayerPrefs
        if (PlayerPrefs.HasKey("Card_" + cardID + "_Change"))
        {
            PlayerPrefs.SetFloat("CurrentBalance",
PlayerPrefs.GetFloat("CurrentBalance") - PlayerPrefs.GetFloat("Card_"
+ cardID + "_Change"));
            PlayerPrefs.DeleteKey("Card_" + cardID + "_Change");
            PlayerPrefs.DeleteKey("Card_" + cardID +
"_CurrentBalance");
            PlayerPrefs.DeleteKey("Card_" + cardID + "_Category");
            PlayerPrefs.DeleteKey("Card_" + cardID + "_Date");
            env.NeedRefreshCards = true;
        }
        else
        {
            Debug.LogWarning($"Карточка с ID {cardID} не найдена.");
        }
    }

    public void EditCard()
    {
        int cardID = GetCardIDFromName();
        if (cardID == -1)
        {
            Debug.LogWarning($"Не удалось получить cardID из имени:
{gameObject.name}");
            return;
```

```
        }

        if (PlayerPrefs.HasKey("Card_" + cardID + "_Change"))
        {
            PlayerPrefs.SetInt("CurrentChanging", cardID);

            // Отримання даних картки та вивід в лог
            float change = PlayerPrefs.GetFloat("Card_" + cardID +
"_Change");
            float currentBalance = PlayerPrefs.GetFloat("Card_" +
cardID + "_CurrentBalance");
            int categoryID = PlayerPrefs.GetInt("Card_" + cardID +
"_Category");
            string dateTime = PlayerPrefs.GetString("Card_" + cardID +
"_Date");

            cashValueInput.text = change.ToString();

            DateTime cardDateTime = GetCardDateTime(dateTime);

            dayInput.text = cardDateTime.ToString("dd");
            monthInput.text = cardDateTime.ToString("MM");
            yearInput.text = cardDateTime.ToString("yyyy");
            hourInput.text = cardDateTime.ToString("HH");
            minuteInput.text = cardDateTime.ToString("mm");

            // Встановлення значення для ToggleGroup
            SetToggleGroupValue(change > 0 ? "Plus" : "Minus");

            // Встановлення значення для Dropdown
            categoryDropdown.value = GetCategoryIndex(categoryID);

            addCardPanel.SetBool("isOpen", true);
        }
        else
        {
            Debug.LogWarning($"Карточка с ID {cardID} не найдена.");
        }
    }

    private int GetCardIDFromName()
    {
        string name = gameObject.name;
        if (name.StartsWith("Card_"))
        {
            string idString = name.Substring(5); // Прибираємо частину
строчки після "Card_"
            int cardID;
            if (int.TryParse(idString, out cardID))
            {
                return cardID;
            }
        }
        return -1; // Повертаємо -1 якщо не вийшло отримати ID
    }

    private DateTime GetCardDateTime(string dateTimeString)
```

```
    {
        DateTime dateTime;
        if (DateTime.TryParse(dateTimeString, out dateTime))
        {
            return dateTime;
        }
        else
        {
            throw new Exception("Помилковий формат дати та часу");
        }
    }

    private void SetToggleGroupValue(string toggleName)
    {
        foreach (Toggle toggle in
toggleGroup.GetComponentsInChildren<Toggle>())
        {
            if (toggle.name == toggleName)
            {
                toggle.isOn = true;
                break;
            }
        }
    }

    private int GetCategoryIndex(int categoryID)
    {
        for (int i = 0; i < categoryDropdown.options.Count; i++)
        {
            if (categoryDropdown.options[i].text ==
PlayerPrefs.GetString("Category_" + categoryID + "_Name"))
            {
                return i;
            }
        }
        return 0; // Default to first category if not found
    }
}
```

Лістинг CardGenerate.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using TMPro;

public class CardsGenerate : MonoBehaviour
{
    public GameObject cardPrefab; // Префаб карточки
    public Transform parent; // Род об'єкт для карток

    public Color positiveBalanceColor = Color.green;
    public Color zeroBalanceColor = Color.gray;
    public Color negativeBalanceColor = Color.red;
```

```
    void Start()
    {
        GenerateCards();
        parent.localPosition = new Vector3(parent.localPosition.x, -
10000, parent.localPosition.z);
    }

    void Update()
    {
        if(env.NeedRefreshCards) {
            GenerateCards();
        }
    }

    private void GenerateCards()
    {
        int lastCardID = PlayerPrefs.GetInt("LastCardID");
        int currentCategory = PlayerPrefs.GetInt("CurrentCategory");

        // Очистка старих карток перед генерацією нових
        foreach (Transform child in parent)
        {
            if (child.name != "CardBalance" && child.name !=
"Category")
            {
                Destroy(child.gameObject);
            }
        }

        for (int ID = lastCardID; ID >= 0; ID--)
        {
            if (PlayerPrefs.HasKey("Card_" + ID + "_Change"))
            {
                int cardCategoryID = PlayerPrefs.GetInt("Card_" + ID +
"_Category");

                // Фільтрація по категорії
                if (currentCategory == 0 || cardCategoryID ==
currentCategory)
                {
                    CreateCard(ID);
                }
            }
        }

        env.NeedRefreshCards = false;
        parent.localPosition = new Vector3(parent.localPosition.x, -
10000, parent.localPosition.z);
    }

    private void CreateCard(int ID)
    {
        // Створення нової картки
        GameObject newCard = Instantiate(cardPrefab, parent);

        // Заповнення даних картки
        newCard.name = "Card_" + ID;
```

```
        float change = PlayerPrefs.GetFloat("Card_" + ID + "_Change");
        float currentBalance = PlayerPrefs.GetFloat("Card_" + ID +
"_CurrentBalance");
        int categoryID = PlayerPrefs.GetInt("Card_" + ID +
"_Category");
        string date = PlayerPrefs.GetString("Card_" + ID + "_Date");
        string hexColor = PlayerPrefs.GetString("Category_" +
categoryID + "_HEX");

        // Пошук текстових полів в картці та заповнення їх даними
        TextMeshProUGUI balanceChangeText =
newCard.transform.Find("BalanceChange").GetComponent<TextMeshProUGUI>(
);
        TextMeshProUGUI balanceCurrentText =
newCard.transform.Find("BalanceCurrent").GetComponent<TextMeshProUGUI>
();
        TextMeshProUGUI categoryNameText =
newCard.transform.Find("CategoryName").GetComponent<TextMeshProUGUI>()
;
        TextMeshProUGUI dateText =
newCard.transform.Find("Date").GetComponent<TextMeshProUGUI>();
        Image cardImage =
newCard.transform.Find("Color").GetComponent<Image>();


        balanceChangeText.text = (change > 0 ? "+" : "") +
change.ToString("F2") + " ₴";
        balanceCurrentText.text = "Current balance: " +
currentBalance.ToString("F2") + " ₴";
        categoryNameText.text = "Category: " +
GetCategoryName(categoryID);
        dateText.text = "Date: " + GetDate(date);
        cardImage.color = HexToColor(hexColor);

        UpdateTextColor(balanceChangeText, change);
        UpdateTextColor(balanceCurrentText, currentBalance);
    }

    private string GetCategoryName(int categoryID)
    {
        return PlayerPrefs.GetString("Category_" + categoryID +
"_Name");
    }

    private Color HexToColor(string hex)
    {
        Color color;
        if (ColorUtility.TryParseHtmlString(hex, out color))
        {
            return color;
        }
        return Color.white;
    }

    private string GetDate(string dateTime)
    {
```

```
        DateTime cardDateTime = GetCardDateTime(dateTime);

        string date = cardDateTime.ToString("dd-MM-yyyy");
        string time = cardDateTime.ToString("HH:mm");

        return date + " " + time;
    }

    private DateTime GetCardDateTime(string dateTimeString)
    {
        DateTime dateTime;
        if (DateTime.TryParse(dateTimeString, out dateTime))
        {
            return dateTime;
        }
        else
        {
            throw new Exception("Помилковий формат дати та часу");
        }
    }

    private void UpdateTextColor(TextMeshProUGUI balanceObj, float
balance)
    {
        if (balance > 0)
        {
            balanceObj.color = positiveBalanceColor;
        }
        else if (balance < 0)
        {
            balanceObj.color = negativeBalanceColor;
        }
        else
        {
            balanceObj.color = zeroBalanceColor;
        }
    }
}
```

Лістинг CategoriesGenerate.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;
using UnityEngine.UI;

public class CategoriesGenerate : MonoBehaviour
{
    public GameObject categoryPrefab; // Префаб для категорії
    public Transform parent; // Родительский об'єкт для категорій

    void Start()
    {
        GenerateCategories();
    }
```

```
    private void GenerateCategories()
    {
        int lastCategoryID = PlayerPrefs.GetInt("LastCategoryID");

        // Очистка старих категорій перед генерацією нових
        foreach (Transform child in parent)
        {
            Destroy(child.gameObject);
        }

        for (int ID = 0; ID <= lastCategoryID; ID++)
        {
            // Перевірка наявності даних про категорію
            if (PlayerPrefs.HasKey("Category_" + ID + "_Name"))
            {
                CreateCategory(ID);
            }
        }
    }

    private void CreateCategory(int ID)
    {
        // Створення нової категорії
        GameObject newCategory = Instantiate(categoryPrefab, parent);

        // Заповнення даних категорії
        newCategory.name = "Category_" + ID;

        string categoryName = PlayerPrefs.GetString("Category_" + ID +
"_Name");
        string hexColor = PlayerPrefs.GetString("Category_" + ID +
"_HEX");

        // Пошук текстового поля і зображення в категорії, заповнення
їх даними
        TextMeshProUGUI nameText =
newCategory.transform.Find("Name").GetComponent<TextMeshProUGUI>();
        Image colorImage =
newCategory.transform.Find("Color").GetComponent<Image>();

        nameText.text = categoryName;
        colorImage.color = HexToColor(hexColor);
    }

    private Color HexToColor(string hex)
    {
        Color color;
        if (ColorUtility.TryParseHtmlString(hex, out color))
        {
            return color;
        }
        return Color.white; // Повертаємо білий колір, якщо парсінг не
вдався
    }
}
```

Лістинг ChangeCurentCategory.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using TMPro;

public class ChangeCurrentCategory : MonoBehaviour
{
    private Animator categoriesPanelAnim;
    void Start ()
{
    // Знаходимо об'єкт сцени за допомогою Find
    GameObject categoriesPanel = GameObject.Find("Main
Camera/UI/CategoriesPanel");

    if (categoriesPanel != null)
    {
        categoriesPanelAnim =
categoriesPanel.GetComponent<Animator>();
    }
    else
    {
        Debug.LogError("Unable to find CategoriesPanel GameObject in
the scene.");
    }
}

    public void SetCurrentCategory()
    {
        string name = gameObject.name;
        int categoryID;

        // Перевірка на формат "Category_{ID}"
        if (name.StartsWith("Category_"))
        {
            string idString = name.Substring("Category_".Length); //
Вилучення частини строки після "Category_"
            if (int.TryParse(idString, out categoryID))
            {
                PlayerPrefs.SetInt("CurrentCategory", categoryID);
                env.NeedRefreshCards = true;
                categoriesPanelAnim.SetBool("isOpen", false);
            }
            else
            {
                Debug.LogWarning($"Failed to parse category ID from
name: {gameObject.name}");
            }
        }
        else
        {
            Debug.LogWarning($"Invalid category name format:
{gameObject.name}");
        }
```

```
    }
}
```

Лістинг env.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class env : MonoBehaviour
{
    public static bool NeedRefreshCards = true;
    public static bool NeedRefreshCurrentCategory = false;
}
```

Лістинг FadeOutPanelRemove.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class FadeOutPanelRemove : MonoBehaviour
{
    public void SetInactive() {
        gameObject.SetActive(false);
    }
}
```

Лістинг OrientationManager.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class OrientationManager : MonoBehaviour
{
    void Start()
    {
        Screen.orientation = ScreenOrientation.AutoRotation;
        Screen.autorotateToPortrait = true;
        Screen.autorotateToPortraitUpsideDown = true;
        Screen.autorotateToLandscapeLeft = false;
        Screen.autorotateToLandscapeRight = false;
    }
}
```

Лістинг PlayerPrefsConfig.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerPrefsConfig : MonoBehaviour
{
```

```
    void Start()
    {
        // CATEGORY:
        // LastCategoryID - int - ID останньої створенної категорії,
відлік починається з 0, за замовченням - 2 (всього 3 категорії)
        // CurrentCategory - int - ID поточно обраної категорії для
фільтрації, за замовченням - 0

        // Category_{ID}_Name - string - назва категорії з ID = {ID}
        // Category_{ID}_HEX - int - HEX-код кольору категорії з ID =
{ID}

        if (!PlayerPrefs.HasKey("LastCategoryID")) {
            PlayerPrefs.SetInt("LastCategoryID", 15);
        }

        // Стандартні категорії
        if (!PlayerPrefs.HasKey ("Category_0_Name")) {
            PlayerPrefs.SetString("Category_0_Name", "ALL");
        }
        if (!PlayerPrefs.HasKey ("Category_0_HEX")) {
            PlayerPrefs.SetString("Category_0_HEX", "#FFFFFF");
        }

        if (!PlayerPrefs.HasKey ("Category_1_Name")) {
            PlayerPrefs.SetString("Category_1_Name", "Food");
        }
        if (!PlayerPrefs.HasKey ("Category_1_HEX")) {
            PlayerPrefs.SetString("Category_1_HEX", "#FFD000");
        }

        if (!PlayerPrefs.HasKey ("Category_2_Name")) {
            PlayerPrefs.SetString("Category_2_Name", "Leisure");
        }
        if (!PlayerPrefs.HasKey ("Category_2_HEX")) {
            PlayerPrefs.SetString("Category_2_HEX", "#009EFF");
        }

        if (!PlayerPrefs.HasKey ("Category_3_Name")) {
            PlayerPrefs.SetString("Category_3_Name", "Housing");
        }
        if (!PlayerPrefs.HasKey ("Category_3_HEX")) {
            PlayerPrefs.SetString("Category_3_HEX", "#69B8F5");
        }

        if (!PlayerPrefs.HasKey ("Category_4_Name")) {
            PlayerPrefs.SetString("Category_4_Name",
"Transportation");
        }
        if (!PlayerPrefs.HasKey ("Category_4_HEX")) {
            PlayerPrefs.SetString("Category_4_HEX", "#B92C2C");
        }

        if (!PlayerPrefs.HasKey ("Category_5_Name")) {
            PlayerPrefs.SetString("Category_5_Name", "Medical");
        }
        if (!PlayerPrefs.HasKey ("Category_5_HEX")) {
```

```
            PlayerPrefs.SetString("Category_5_HEX", "#0CCA3A");
        }

        if (!PlayerPrefs.HasKey ("Category_6_Name")) {
            PlayerPrefs.SetString("Category_6_Name", "Clothing");
        }
        if (!PlayerPrefs.HasKey ("Category_6_HEX")) {
            PlayerPrefs.SetString("Category_6_HEX", "#F585C9");
        }

        if (!PlayerPrefs.HasKey ("Category_7_Name")) {
            PlayerPrefs.SetString("Category_7_Name", "Utilities");
        }
        if (!PlayerPrefs.HasKey ("Category_7_HEX")) {
            PlayerPrefs.SetString("Category_7_HEX", "#617CC8");
        }

        if (!PlayerPrefs.HasKey ("Category_8_Name")) {
            PlayerPrefs.SetString("Category_8_Name", "Pets");
        }
        if (!PlayerPrefs.HasKey ("Category_8_HEX")) {
            PlayerPrefs.SetString("Category_8_HEX", "#CAC706");
        }

        if (!PlayerPrefs.HasKey ("Category_9_Name")) {
            PlayerPrefs.SetString("Category_9_Name", "Hobbies");
        }
        if (!PlayerPrefs.HasKey ("Category_9_HEX")) {
            PlayerPrefs.SetString("Category_9_HEX", "#D34BEE");
        }

        if (!PlayerPrefs.HasKey ("Category_10_Name")) {
            PlayerPrefs.SetString("Category_10_Name", "Work");
        }
        if (!PlayerPrefs.HasKey ("Category_10_HEX")) {
            PlayerPrefs.SetString("Category_10_HEX", "#6A6363");
        }

        if (!PlayerPrefs.HasKey ("Category_11_Name")) {
            PlayerPrefs.SetString("Category_11_Name", "Tech
Purchases");
        }
        if (!PlayerPrefs.HasKey ("Category_11_HEX")) {
            PlayerPrefs.SetString("Category_11_HEX", "#C87070");
        }

        if (!PlayerPrefs.HasKey ("Category_12_Name")) {
            PlayerPrefs.SetString("Category_12_Name", "Investments");
        }
        if (!PlayerPrefs.HasKey ("Category_12_HEX")) {
            PlayerPrefs.SetString("Category_12_HEX", "#046F04");
        }

        if (!PlayerPrefs.HasKey ("Category_13_Name")) {
            PlayerPrefs.SetString("Category_13_Name", "Gifts");
        }
        if (!PlayerPrefs.HasKey ("Category_13_HEX")) {
```

```
            PlayerPrefs.SetString("Category_13_HEX", "#4DE7CB");
        }

        if (!PlayerPrefs.HasKey ("Category_14_Name")) {
            PlayerPrefs.SetString("Category_14_Name", "Family");
        }
        if (!PlayerPrefs.HasKey ("Category_14_HEX")) {
            PlayerPrefs.SetString("Category_14_HEX", "#FD2575");
        }

        if (!PlayerPrefs.HasKey ("Category_15_Name")) {
            PlayerPrefs.SetString("Category_15_Name", "Other");
        }
        if (!PlayerPrefs.HasKey ("Category_15_HEX")) {
            PlayerPrefs.SetString("Category_15_HEX", "#131313");
        }

        if (!PlayerPrefs.HasKey ("CurrentCategory")) {
            PlayerPrefs.SetInt("CurrentCategory", 0);
        }


        // CARD:
        // LastCardID - int - ID останньої створенної картки, відлік
починається з 0
        // CurrentBalance - float - теперішній загальний баланс
        // CurrentChanching - int - ID картки, яку зараз змінюють, "-
1" при стані без змін

        // Card_{ID}_Change - float - зміна балансу у картці з ID =
{ID}
        // Card_{ID}_CurrentBalance - float - загальний баланс після
внесеної в нього зміни у картці з ID = {ID}
        // Card_{ID}_Category - int - категорія картки з ID = {ID}
(записується ID категорії)
        // Card_{ID}_Date -  дата картки з ID = {ID}

        if (!PlayerPrefs.HasKey("LastCardID")) {
            PlayerPrefs.SetInt("LastCard", 0);
        }

        if (!PlayerPrefs.HasKey("CurrentBalance")) {
            PlayerPrefs.SetFloat("CurrentBalance", 0.0f);
        }

        if (!PlayerPrefs.HasKey("CurrentChanching")) {
            PlayerPrefs.SetInt("CurrentChanching", -1);
        }
    }
}
```

Лістинг ScrollBarVisibility.cs

```
using UnityEngine;
using UnityEngine.UI;

public class ScrollbarVisibility : MonoBehaviour
```

```
{
    public ScrollRect scrollRect;
    public Scrollbar scrollbar;
    public float hideDelay = 0.5f;

    private CanvasGroup canvasGroup;
    private Animator scrollbarAnimator;
    private float timer;
    private bool isScrolling;

    void Start()
    {
        canvasGroup = scrollbar.GetComponent<CanvasGroup>();
        if (canvasGroup == null)
        {
            canvasGroup =
scrollbar.gameObject.AddComponent<CanvasGroup>();
        }

        scrollbar.value = 1;
        scrollbarAnimator =
scrollbar.gameObject.GetComponent<Animator>();
        if (scrollbarAnimator == null)
        {
            Debug.LogError("Animator component is missing on the
scrollbar.");
            return;
        }

        scrollbarAnimator.SetBool("isHide", true);

        scrollRect.onValueChanged.AddListener(OnScroll);

        canvasGroup.alpha = 0;
    }

    void Update()
    {
        if (!isScrolling && canvasGroup.alpha > 0)
        {
            timer -= Time.deltaTime;
            if (timer <= 0)
            {
                scrollbarAnimator.SetBool("isHide", true);
            }
        }
    }

    void OnScroll(Vector2 position)
    {
        scrollbarAnimator.SetBool("isHide", false);
        timer = hideDelay;
        isScrolling = true;
        Invoke("StopScrolling", hideDelay);
    }

    void StopScrolling()
```

```
    {
        isScrolling = false;
    }
}
```

Лістинг ShowCurrentCategory.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using TMPro;

public class ShowCurrentCategory : MonoBehaviour
{
    public TextMeshProUGUI text;
    public Image color;

    void Update()
    {
        ChangeCategory();
    }

    private void ChangeCategory()
    {
        int currentCategoryID = PlayerPrefs.GetInt("CurrentCategory");
        text.text = PlayerPrefs.GetString("Category_" +
currentCategoryID.ToString() + "_Name");
        color.color = HexToColor(PlayerPrefs.GetString("Category_" +
currentCategoryID.ToString() + "_HEX"));
    }

    private Color HexToColor(string hex)
    {
        Color color;
        if (ColorUtility.TryParseHtmlString(hex, out color))
        {
            return color;
        }
        return Color.white;
    }
}
```

Лістинг ShowGlobalBalance.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;

public class ShowGlobalBalance : MonoBehaviour
{
    public TextMeshProUGUI globalBalance;
    public Color positiveBalanceColor = Color.green;
    public Color zeroBalanceColor = Color.gray;
    public Color negativeBalanceColor = Color.red;
```

```
    void Update()
    {
        ShowBalance();
    }

    private void ShowBalance()
    {
        float balance = PlayerPrefs.GetFloat("CurrentBalance");
        globalBalance.text = ((balance <= 99999.99f) ?
balance.ToString("F2") : "Over than 100k") + " ₴";
        UpdateTextColor(balance);
    }

    private void UpdateTextColor(float balance)
    {
        if (balance > 0)
        {
            globalBalance.color = positiveBalanceColor;
        }
        else if (balance < 0)
        {
            globalBalance.color = negativeBalanceColor;
        }
        else
        {
            globalBalance.color = zeroBalanceColor;
        }
    }
}
```

Лістинг ShowPanel.cs

```
using UnityEngine;
using TMPro;
using UnityEngine.UI;
using System;
using System.Collections.Generic;
public class ShowPanels : MonoBehaviour
{
    public Animator addPanel;
    public Animator categoryPanel;
    public Animator addCategoryPanel;


    public TMP_InputField dayInputField;
    public TMP_InputField monthInputField;
    public TMP_InputField yearInputField;
    public TMP_InputField hourInputField;
    public TMP_InputField minuteInputField;
    public TMP_InputField balanceChangeInputField;
    public TMP_Dropdown categoryDropdown;


    void Start()
    {
        addPanel.SetBool("isOpen", false);
```

```
        categoryPanel.SetBool("isOpen", false);
        addCategoryPanel.SetBool("isOpen", false);
    }


    public void AddPanelShow() {
        addPanel.SetBool("isOpen", true);
        ClearForm();
    }

    public void CategoryPanelShow() {
        categoryPanel.SetBool("isOpen", true);
    }

    public void AddCategoryPanelShow() {
        addCategoryPanel.SetBool("isOpen", true);
    }

    public void AddPanelClose() {
        addPanel.SetBool("isOpen", false);
    }

    public void CategoryPanelClose() {
        categoryPanel.SetBool("isOpen", false);
    }

    public void AddCategoryPanelClose() {
        addCategoryPanel.SetBool("isOpen", false);
    }

    public void QuitApp() {
        Application.Quit();
    }


    private void ClearForm()
    {
        dayInputField.text = "";
        monthInputField.text = "";
        yearInputField.text = "";
        hourInputField.text = "";
        minuteInputField.text = "";
        balanceChangeInputField.text = "";
        categoryDropdown.value = 0;

        // Скинути значення CurrentChanging
        PlayerPrefs.SetInt("CurrentChanging", -1);
    }
}
```