МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри_____ Є. О. Давиденко

підпис

«___» _____2024 р.

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

Клієнтський мобільний застосунок для більярдного клубу

Спеціальність «Інженерія програмного забезпечення»

121 – КРБ.1 – 408.22120801

Здобувач _____В. В. Димитрієв

підпис

«__» _____ 2024 р.

Керівник канд. техн. наук, доцент _____ Г. В. Горбань

підпис

«__» _____ 2024 р.

Консультант канд. техн. наук, доцент _____ А. О. Алексєєва

підпис

«__» _____ 2024 р.

Миколаїв – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Чорноморський національний університет імені Петра Могили

Факультет комп'ютерних наук

Кафедра інженерії програмного забезпечення

ЗАТВЕРДЖУЮ

Зав. кафедри _____ Є. О. Давиденко

«_____» _____ 2024 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи бакалавра

Видано здобувач групи 408 факультету комп'ютерних наук

_____Димитрієв Валерій Вадимович_____

(прізвище, ім'я, по батькові здобувач)

1. Тема кваліфікаційної роботи

Клієнтський мобільний застосунок для більярдного клубу

_____

_____

Затверджена наказом по ЧНУ від «22» _____грудня_____ 2023 р. № ___269_____

2. Строк представлення кваліфікаційної роботи «____» _____ 20__ р.

3. Очікуваний результат роботи та початкові дані, якщо такі потрібні

Очікуваним результатом є мобільний застосунок для більярдного клубу_____

_____

4. Перелік питань, що підлягають розробці:

- дослідження предметної області та аналіз існуючих аналогів;
- формування специфікації вимог до програмного забезпечення;
- визначення архітектури для проєктування програмного забезпечення;
- моделювання та проєктування програмного забезпечення;
- розробка програмного забезпечення;
- здійснення тестування роботи програмного забезпечення;

5. Перелік графічних матеріалів

__Презентація_____

_

6. Завдання до спеціальної частини

__Аналіз охорони праці на робочих місцях фахівців з інформаційних технологій__

7. Консультанти:

Консультант	Кафедра (організація)	Частина роботи
Алєксєєва А. О.	Кафедра екології	Спеціальна частина з охорони праці

Керівник роботи _____канд. техн. наук, доцент Горбань Г. В._____

(посада, прізвище, ім’я, по батькові)

_____

(підпис)

Завдання прийнято до виконання

_____Димитрієв Валерій Вадимович_____

(прізвище, ім’я, по батькові)

_____

(підпис)

Дата видачі завдання «____» _____ 2024р.

КАЛЕНДАРНИЙ ПЛАН
виконання бакалаврської кваліфікаційної роботи

Тема: «Клієнтський мобільний застосунок для більярдного клубу»

№	Найменування роботи	Початок	Закінчення	Примітки
1	Розробка та затвердження завдання на виконання КРБ	20.02.2024	23.02.2024	Виконано
2	Огляд літератури за темою роботи	24.02.2024	26.02.2024	Виконано
3	Складання календарного плану КРБ	26.02.2024	27.02.2024	Виконано
4	Аналіз предметної області	28.02.2024	01.03.2024	Виконано
5	Розробка проєктних рішень	04.03.2024	10.03.2024	Виконано
6	Моделювання та конструювання ПЗ	22.03.2024	26.03.2024	Виконано
7	Кодування ПЗ, тестування та апробація розробленого ПЗ, аналіз результатів тестування	28.03.2024	22.05.2024	Виконано
8	Розробка спеціальної частини з охорони праці	05.04.2024	22.04.2024	Виконано
9	Оформлення КРБ та презентації	24.04.2024	10.05.2024	Виконано
10	Відгук керівника КРБ	21.05.2024	22.05.2024	Виконано
11	Попередній захист	03.06.2024	05.06.2024	Виконано
12	Рецензування	13.06.2024	14.06.2024	Виконано
13	Захист кваліфікаційної роботи	25.06.2024	26.06.2024	Виконано

Розробив здобувач Димитрієв В.В. _____
(прізвище, ім'я, по батькові) *(підпис)*
«24» січня 2024 р.

Керівник роботи канд. техн. наук, доцент Горбань Г. В.
(посада, прізвище, ім'я, по батькові) *(підпис)*
«24»січня 2024 р.

АНОТАЦІЯ

до кваліфікаційної роботи бакалавра

«Клієнтський мобільний застосунок для більярдного клубу»

Здобувач 408 гр,: Димитрієв Валерій Вадимович

Керівник: канд. техн. наук, доцент Горбань Г. В..

Розробка мобільного застосунку для більярдного клубу "Класік" є актуальною в контексті зростаючого інтересу до цифрових технологій у сфері розважальних закладів. Застосунок спрямований на поліпшення взаємодії клієнтів з клубом через зручне відстеження годин гри, доступ до актуальної клубної інформації, підтримку статистики особистих ігор та участь у турнірах.

Об'єктом роботи є процес розробки мобільного застосунку для більярдного клубу "Класік", включаючи аналіз потреб клієнтів, вивчення можливостей технологій та створення ефективного інструменту для управління клубом та покращення взаємодії зі споживачами.

Предметом роботи є технологічні та функціональні аспекти розробки мобільного застосунку, зокрема використання бази даних засобами Firebase, розроблення необхідних таблиць та опис сутностей з атрибутами.

Метою роботи є розробка застосунку, спрямованого на оптимізацію процесів відстеження часу гри, отримання актуальної інформації про клуб, ведення статистики особистих ігор та участь у турнірах.

У першому розділі представлено аналіз предметної області, аналіз готових рішень та специфікацію вимог до застосунку. У другому розділі представлено проєктування застосунку, а саме розробку діаграм та моделей. У третьому розділі представлено вибір технологій розробки, модель бази даних та мокап системи. У четвертому розділі представлено реалізацію застосунку.

КРБ викладена на 52 сторінки, вона містить 4 розділи, 26 ілюстрацій, 7 таблиць, ___ джерел в переліку посилань

Ключові слова: мобільний застосунок, більярдний клуб, Firebase, Swift, x-code.

ABSTRACT

to the qualification work of the bachelor

"Client mobile application for billiards club"

Student 408 gr,: Dimitriev Valery VadimovychSupervisor: candidate technical Sciences, Associate Professor G. V. Horban.

The development of a mobile application for the "Classic" billiard club is relevant in the context of the growing interest in digital technologies in the field of entertainment establishments. The application is aimed at improving customer interaction with the club through convenient tracking of game hours, access to current club information, support of personal game statistics and participation in tournaments.

The object of the work is the process of developing a mobile application for the "Klasik" billiard club, including the analysis of customer needs, the study of technology possibilities and the creation of an effective tool for managing the club and improving interaction with consumers.

The subject of the work is the technological and functional aspects of developing a mobile application, in particular the use of a database by means of Firebase, the development of the necessary tables and the description of entities with attributes.

The purpose of the work is to develop an application aimed at optimizing the processes of tracking game time, obtaining up-to-date information about the club, keeping statistics of personal games and participating in tournaments.

The first section presents the analysis of the subject area, the analysis of ready-made solutions and the specification of application requirements. The second section presents the design of the application, namely the development of diagrams and models. The third section presents a selection of development technologies, a database model, and a mockup of the system. The fourth chapter presents the implementation of the application.

QWB is laid out on 52 pages, it contains 4 sections, 26 illustrations, 7 tables, ___ sources in the list of references

Keywords: mobile application, pool club, Firebase, Swift, x-code.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

UML (Unified Modeling Language) – уніфікована мова моделювання;

БД – База даних;

VC – ViewController;

ПЗ – Програмне забезпечення;

ПС – Програмна система;

SMM – Маркетинг у соціальних мережах;

КПК – Кишеньковий персональний комп'ютер;

БК – Більярдний клуб;

MVM – Model-View-Model;

ДсанПіН – Державні санітарні норми і правила;

ПК – Персональний комп’ютер;

ДНАОП – Державні нормативні акти про охорону праці;

Swift – Багатопарадигмова компільована мова програмування, розроблена компанією Apple для того, щоб співіснувати з Objective C і бути стійкішою до помилкового код ;

Firebase – Хмарна база даних, яка дозволяє користувачам зберігати і отримувати збережену інформацію, а також мати зручні засоби і методи взаємодії з нею;

IDE XCode – Середовище розробки мобільного застосунку;

Rational Rose – Середовище розробки діаграм для проєктування інтерфесу.

ВСТУП

Зважаючи на потреби клієнтів клубу та зростаючу популярність мобільних технологій, метою даної кваліфікаційної роботи є розробка імовірного рішення у вигляді мобільного застосунку для більярдного клубу «Класік».

Цей проєкт має важливе значення не лише для розважальної галузі, але й для сектора комерції. Він надає можливість ефективного контролю за клубом, адже мобільна платформа є найбільш гнучкою для реалізації різноманітних функцій і завжди легко доступна для користувачів, навіть у порівнянні з комп'ютерною платформою.

Об’єктом роботи є процес розробки мобільного застосунку для більярдного клубу «Класік«, включаючи аналіз потреб клієнтів, вивчення можливостей технологій та створення ефективного інструменту для управління клубом та покращення взаємодії зі споживачами.

Предметом роботи є технологічні та функціональні аспекти розробки мобільного застосунку, зокрема використання бази даних засобами Firebase, розроблення необхідних таблиць та опис сутностей з атрибутами.

Метою роботи є розробка застосунку, спрямованого на оптимізацію процесів відстеження часу гри, отримання актуальної інформації про клуб, ведення статистики особистих ігор та участь у турнірах.

Для досягнення мети потрібно вирішити наступні **завдання**:

1) розробка інтерфейсу застосунку, що забезпечить зручне та ефективне відстеження часу гри;

2) імплементація функціоналу для отримання актуальної інформації про клуб та його події;

3) створення можливості ведення статистики особистих ігор користувачів;

4) розробка модуля для реєстрації та участі у турнірах;

5) оптимізація застосунку для зменшення витрат часу та людських ресурсів, потрібних для організації подібних заходів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної області

На сьогодні мобільні застосунки є найзручнішим способом використання ПС, оскільки у кожну секунду можна отримати інформацію будь-якого роду. В конкретному випадку – свою статистику у грі в більярд. Все більше і більше програмних рішень переходить на мобільні платформи задля зручності використання, швидкого відклику системи а також швидкості розробки і отримання готового продукту.

Оскільки останніми роками більярдний спорт стає дедалі популярнішим, клієнти клубу потребують вже не просто гри, а статистики, на яку вони зможуть сміливо орієнтуватись, виправляти свої помилки аналізувати дії. Також, в клубах бракує інформації про сам клуб, їх акціях а також подіях, в силу того, що робота професійних SMM робітників коштує необґрутовано дорого і не завжди ефективно, у той час, коли вся інформація може знаходитись незалежно від соціальних мереж або інших сервісів.

На сьогодні фізична гра потребує й доповнення з боку інформаційних технологій для полегшення контролю обліку, а також внести більшу спортивну мотивацію для гравців. Для полегшення контролю обліку використовують ПК, але програмні рішення для мобільних платформ на сьогодні не використовуються.

Зручність мобільних платформ порівняно з ПК є їх універсальність та портативність, а також можливість швидкої розробки багатофункціонального застосунку для вирішення завдань будь-якої складності.

На основі вище описаного було прийняте рішення написати клієнтський застосунок «Більярдний клуб «Класік»

З мобільним застосунком «Більярдний клуб «Класік» будуть працювати:

- клієнти;
- маркер;

– більярдні коучі.

Маркер – особа що прислужує при грі на більярді і веде рахунок очок.

Нікнейм – особисте, переважно, вигадане, ім’я, яким називають себе користувачі інтернету в різноманітних чатах, форумах, месенджерах або інших застосунках.

Клієнт – людина, яка оформила замовлення.

Ко́учинг – метод здійснення консалтингу й тренінгу, в якому спеціальна людина, «коуч», допомагає іншим людям досягнути певної цілі в професії чи в особистому житті.

Замовлення – доручення виготовлення, виконання або підготування продукту.

Грошовий чек – документ, оплату якого здійснюють шляхом виплати грошей фірмі.

Банківська картка – пластикова пластина, яка використовується для ідентифікації її користувача, для способу фіксації інформації і як аналог платіжних засобів. У кожної картки є свій номер, строк придатності та підпис картки (CVV-код).

Розробка застосунків для мобільних пристроїв – це процес, при якому застосунки розробляються для невеликих портативних пристроїв, таких як КПК, смартфони або телефони. Ці програми можуть бути встановлені на пристрій в процесі виробництва, завантажені користувачем з допомогою різних платформ для поширення або бути вебзастосунками, які обробляються на стороні клієнта (JavaScript) або сервера.

1.2 Аналіз готових рішень

На даний момент на ринку програмного забезпечення існує багато команд розробників, що надають комплекс послуг з розробки мобільних застосунків під певну задачу. Тому перед початком розробки власного програмного забезпечення проведено аналіз вже існуючих, схожих систем, основною метою якого було виявлення найбільш вдалих рішень з погляду користувача, а також визначення

недоліків в роботі та відмінності в існуючих комплексах для їх наступного врахування під час розробки. Але аналогів даному застосунку на даний момент немає, тож були розглянуті сервіси, які найбільш співпадають як структурно так і ідейно з бажаним, тож було розглянуто наступні сервіси :

– Microinvest Більярд Pro (є найближчим в плані внутрішньої структури застосунку) (див. рис. 1.1) [3]:

Рисунок 1.1 – Microinvest Більярд Pro

Microinvest Більярд Pro – продукт для автоматизації розважальної індустрії. За допомогою Microinvest Склад Pro і спеціалізованого контролера, розробленого компанією Microinvest, оператори можуть включати і вимикати освітлення більярдних столів.

Переваги:

- централізоване управління об’єктом;
- різноманітні звіти, що дозволяють повний контроль над усіма робочими процесами;
- контроль над знижками;
- підвищення швидкості та якості обслуговування клієнтів;
- можливість управляти 8 окремими столами одночасно;
- система не вимагає спеціальних доопрацювань існуючих систем;
- можливість продажу за артикулами;

– повна інтеграція з Microinvest Склад Pro;

– мінімальна можливість зловживань і виникнення помилок з боку персоналу і багато ін.

Недоліки:

– порівняно низька мобільність;

– обмеженість адміністрацією та управлінням закладу;

– повна відсутність внутрішньої системи авторизації.

PROBILLIARD – система автоматизації proBilliard передбачає облік, управління і контроль діяльності більярдних клубів (найбільш наближена по функціоналу), представлена на рис. 1.2-1.4 [3]:

Рисунок 1.2 – Аутентифікація в ПЗ PROBILLIARD

Рисунок 1.3 – Додання нового клієнта у БД

Рисунок 1.4 – Статус столів

Переваги:

- авторизація користувача;
- присутня база даних клієнтів;
- використання особистого рахунку клієнта;
- оплата послуг;
- тарифікація послуг.

Недоліки:

- порівняно низька мобільність;
- відсутність функціоналу для клієнта;
- клієнт не має переваг при наявності даного ПЗ;
- клієнт не має можливості забронювати стіл без дзвінка в клуб;
- клієнт має жодної інформації щодо кількості вільних столів;
- маркер не має можливості створити турнір або видалити користувача з БД.

1.3 Специфікація вимог

Призначення та межі проєкту

Призначення системи:

Розробка мобільного застосунку для зручної бронювання столів, перегляду цін та рейтингової позиції користувачів у клубі.

Погодження, що ухвалені в програмній документації:

Відображення інформації про замовлення, вільні столи, спортивні події, виграші, час, ціни та клієнтів.

Межі проєкту ПЗ:

Реалізація функціональності, описаної у зазначеному обсязі, для мобільних пристроїв на платформах Android та iOS.

Загальний опис

Сфера застосування:

Мобільний застосунок для клієнтів нічного клубу, що дозволяє здійснювати різні операції, пов'язані з відвідуванням закладу.

Характеристики користувачів:

Клієнти клубу, які мають мобільні пристрої з операційними системами Android або iOS.

Загальна структура і склад системи:

Мобільний застосунок, що складається з функціональних модулів для реєстрації, перегляду рейтингу, вибору столів, оплати, перегляду подій та знижкових карт.

Загальні обмеження:

Обмеження, пов'язані з функціональністю та обмеженнями платформ Android і iOS.

Функції системи:

- реєстрація та вхід;
- перегляд рейтингу та статистики виграшів;
- меню;
- вибір столу для бронювання;
- введення банківських реквізитів для оплати;
- перегляд списку спортивних подій;
- використання знижкових карт постійних клієнтів.

Вимоги до інформаційного забезпечення

Джерела і зміст вхідної інформації (даних):

- інформація про замовлення;

- інформація про вільні столи;
- інформація про спортивні події;
- інформація про виграші;
- інформація про час;
- інформація про ціни;
- інформація про клієнтів.

Нормативно-довідкова інформація:

- дані замовлень;
- статус столу;
- список замовлень;
- список подій;
- дані статистики на діаграмі;
- дані статистики у вікні середнього часу;
- ціна за годину;
- дані клієнта.

Висновки до розділу 1

У даному розділі виконано ретельний аналіз предметної області з урахуванням актуальних вимог та потреб користувачів. На основі цього аналізу було проведено огляд існуючих рішень на ринку, щоб виявити їхні переваги та обмеження. Крім того, було визначено два ключові аналоги, які стали об'єктом подальшого порівняльного аналізу.

У цьому розділі ретельно розглянуто функціональність, інтерфейсні можливості та ефективність кожного з аналогів з урахуванням вимог проєкту. Проаналізовано їхні сильні та слабкі сторони з метою ідентифікації найбільш оптимальних рішень для впровадження в розроблюваний вебзастосунок онлайн-клініки.

Отримані результати аналізу стануть важливою основою для формулювання конкретних завдань та цілей проєкту, а також визначення його унікальної пропозиції.

2 ПРОЄКТУВАННЯ ВЕБЗАСТОСУНКУ ОНЛАЙН-КЛІНІКИ

2.1 Розробка діаграми використання системи

Діаграма варіантів використання, або Use Case Diagram, є одним з ключових інструментів в розробці програмного забезпечення, який використовується для опису функціональних вимог до системи. Вона відображає взаємодію між користувачами (акторами) і системою через різні сценарії використання (варіанти використання). Ця діаграма допомагає розробникам і зацікавленим сторонам зрозуміти, як система повинна функціонувати і які задачі вона повинна виконувати.

Контекстну діаграму можна переглянути на рис. 2.1.

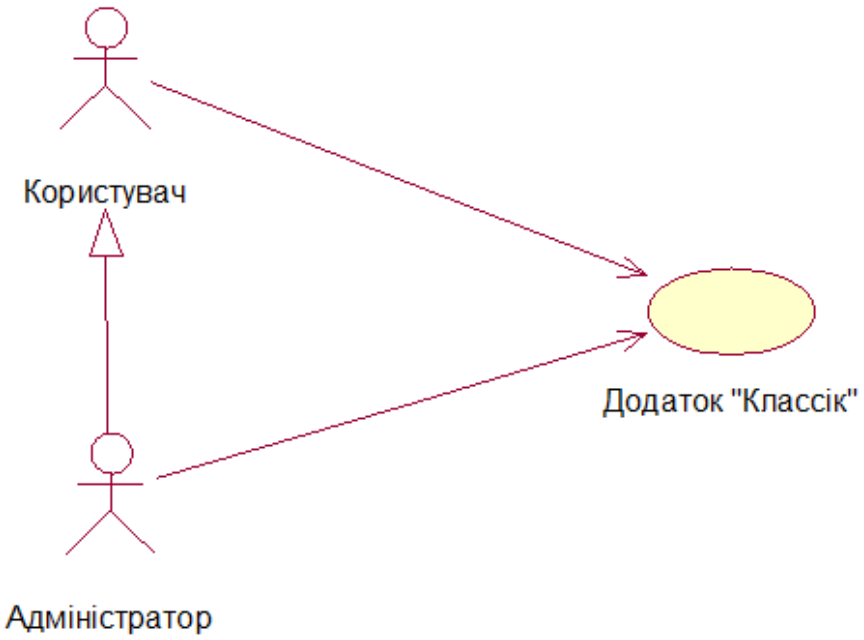

Рисунок 2.1 – Загальна діаграма варіантів використання

Діаграма варіантів використання клієнтського застосунку, представлено на рис. 2.2.

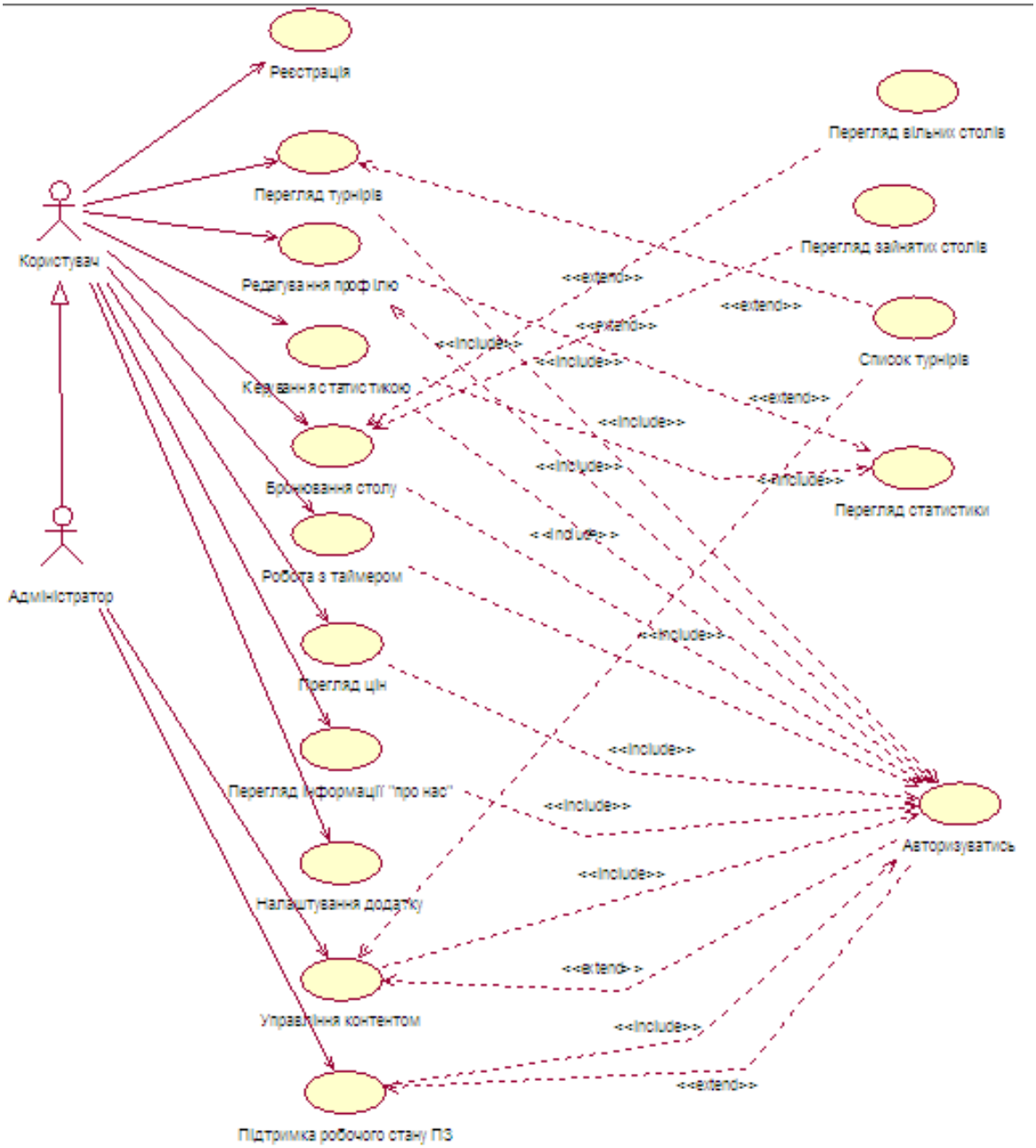

Рисунок 2.2 – Діаграма варіантів використання

Сценарій 1 – Реєстрація користувача

Короткий опис: Користувач реєструється у додатку для подальшого користування системою зі збереженням налаштувань та статусу акаунту.

Основний сценарій:

– користувач натискає кнопку реєстрації;
– система відображає форму для введення реєстраційних даних;
– користувач заповнює форму та надсилає дані;
– система зберігає введені дані.

Альтернативний сценарій:

– якщо поля форми заповнені некоректно, система відображає повідомлення про помилку;

– користувач виправляє помилки та повторно надсилає дані;

– система перевіряє валідність та створює новий обліковий запис.

Сценарій 2 – Перегляд турнірів

Короткий опис: Користувач переглядає доступні турніри та записується на обраний.

Основний сценарій:

– користувач натискає кнопку «Гра»;

– система відправляє запит на сервер для отримання списку доступних турнірів;

– система відображає список турнірів;

– користувач обирає турнір і натискає кнопку «Записаться»;

– система відправляє запит на сервер для отримання деталей обраного турніру (дата, час, внесок, список учасників) та відображає їх на екрані;

– користувач підтверджує участь натисканням кнопки запису.

Альтернативний сценарій:

– якщо обраний турнір вже заповнений, система пропонує вибрати інший турнір;

– користувач обирає інший турнір і повторює дії з пункту 4 основного сценарію.

Сценарій 3 – Редагування профілю

Короткий опис: Користувач редагує персональні дані у своєму профілі.

Основний сценарій:

– користувач натискає кнопку «Редагувати профіль»;

– система відображає форму для редагування профілю з існуючими даними;

– користувач змінює потрібні дані та зберігає їх;

– система зберігає зміни.

Альтернативний сценарій:

– користувач натискає кнопку «Редагувати профіль», але не змінює жодних даних;

– система повертає користувача до вікна профілю без змін.

Сценарій 4 – Керування статистикою

Короткий опис: Користувач керує своєю статистикою виграшів та програшів.

Основний сценарій:

– користувач натискає кнопку «Статистика»;

– система завантажує дані статистики користувача та відображає їх на екрані;

– користувач змінює статистику (додає виграні або програні партії) та зберігає зміни;

– система зберігає оновлену статистику.

Альтернативний сценарій:

– користувач ініціює гру з іншим користувачем;

– система надає форму для підтвердження гри;

– користувачі грають, а система оновлює статистику гравців.

Сценарій 5 – Бронювання столу

Короткий о Користувач бронює та переглядає доступні столи.

Основний сценарій:

– користувач натискає кнопку «Столи»;

– система запитує сервер про інформацію про вільні та зайняті столи, відображає їх на екрані;

– користувач обирає стіл та вводить дані для бронювання;

– система перенаправляє користувача на сторінку оплати;

– користувач проводить оплату;

– система заносить інформацію про бронювання.

Альтернативний сценарій:

- користувач натискає кнопку «Столи»;
- система показує інформацію про столи;
- користувач вводить дані для бронювання, але скасовує операцію;
- система показує повідомлення про необхідність оплати за бронювання.

Ці сценарії детально описують послідовність дій користувачів в межах кожного функціонального блоку системи.

Діаграма діяльності є графічним інструментом, що використовується в UML (Unified Modeling Language) для моделювання поведінки системи або бізнес-процесу. Вона дозволяє візуалізувати послідовність дій або робіт, які виконуються в рамках процесу, а також взаємодії між різними елементами системи. Діаграма діяльності допомагає зрозуміти, як здійснюється виконання певних задач, які кроки слідують один за одним, і як вони взаємодіють між собою.

Побудовано діаграму діяльності для варіанта використання «Робота з таймером». Представлена на рис. 2.3.

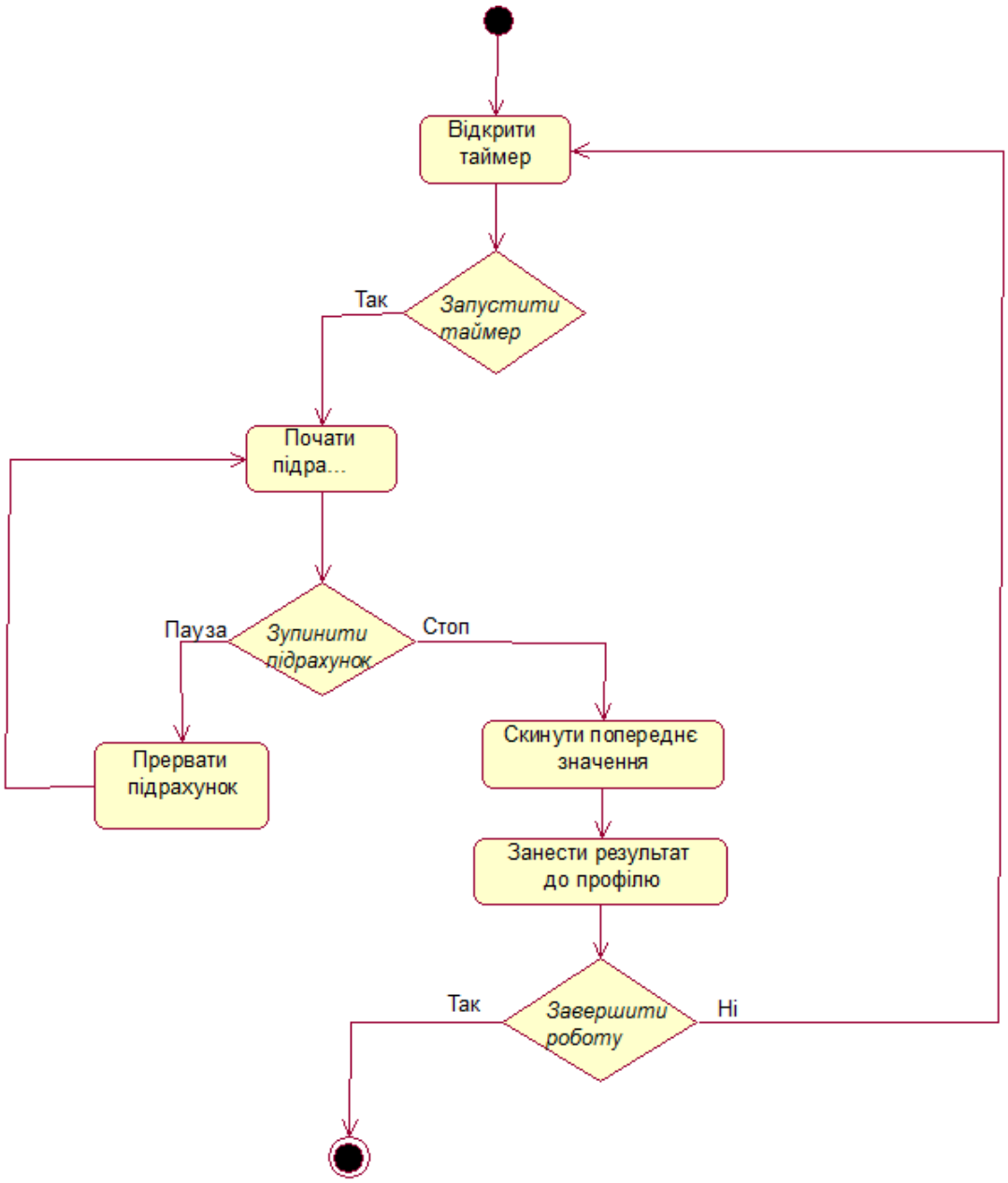

Рисунок 2.3 –Діаграма діяльності для варіанта використання «Робота з таймером»

Побудовано діаграму діяльності для варіанта використання «Робота з таймером». Представлена на рис. 2.4.

Рисунок 2.4 –Діаграма діяльності для варіанта використання «Перегляд профілю»

Дані діаграми дозволять розробити гнучку логіку для застосунку.

2.2 Розробка діаграми класів

Діаграма класів є одним з основних інструментів в UML (Unified Modeling Language) і використовується для візуалізації структури і взаємозв'язків між класами в системі програмного забезпечення. Вона дозволяє моделювати об'єктно-орієнтовані системи, які складаються з класів, їх атрибутів і методів, а також зв'язків між класами.

Діаграму класів, представлено на рис. 2.5.

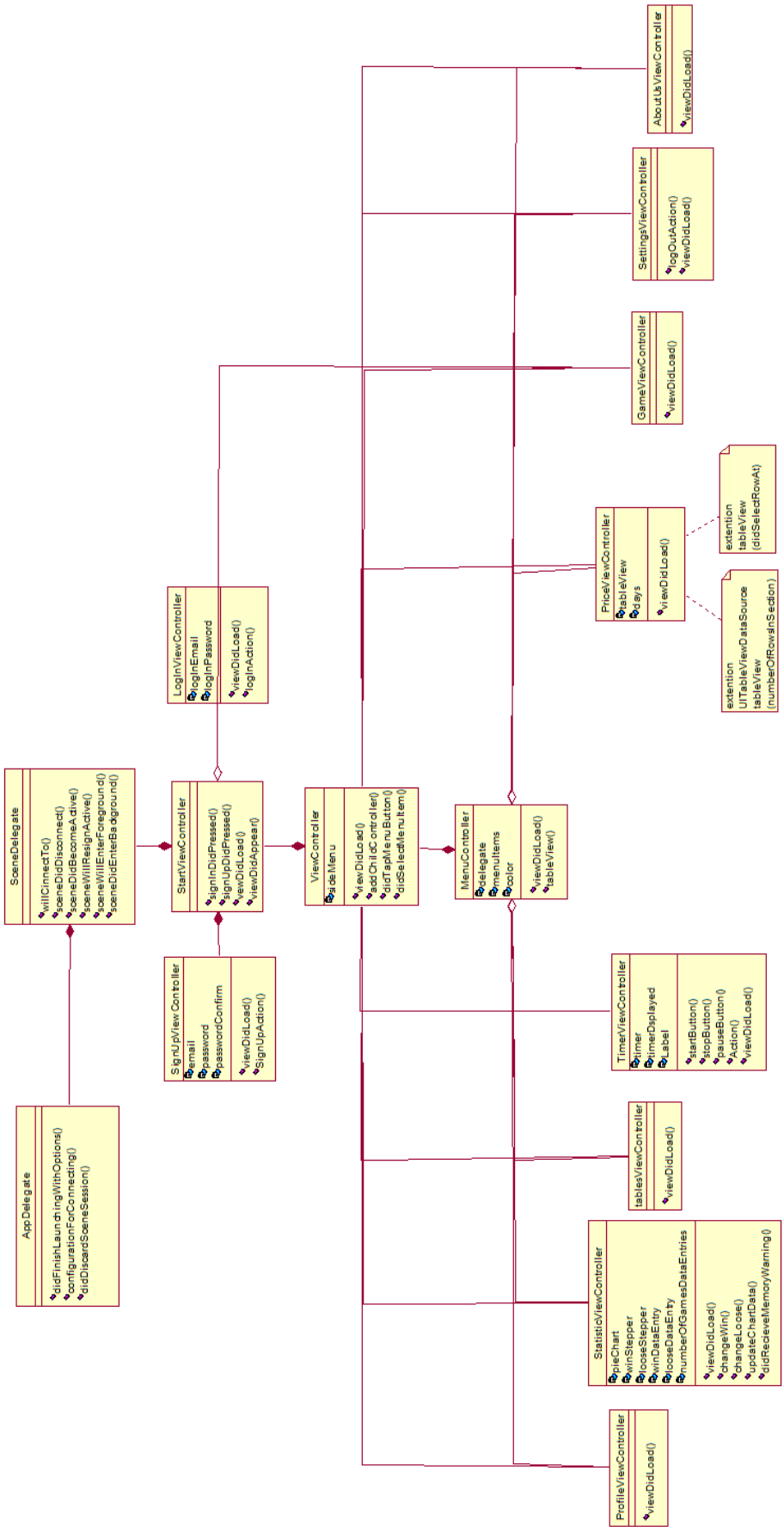

Рисунок 2.5 – Діаграма класів

Клас AppDelegate (Викликається на етапі запуску програми) має функції та параметри запуску застосунку. Існування програми без нього неможливе.

Клас SceneDelegate має набір функцій і параметрів для запуску форм, має зв’язок «композиція» з класом: AppDelegate, що містить параметри запуску застосунку.

Клас ViewController містить в собі глобальні змінні, викликає функції переходу до залежних класів, функції та файли підключення фреймворку, має зв’язок «композиція» з класом: SceneDelegate що має параметри для існування ViewController.

Клас MenuController містить в собі функціональний та анімаційний опис головного меню, містить функцію відображення viewDidLoad. Існування неможливе без звязку «композиція» з класом ViewController.

Клас ProfileViewController містить методи обробки інформації про користувача, має зв’язок асоціації від MenuController, оскільки в методі класу відбувається виклик класу MenuController а також залежність з класом ViewController.

Клас StatisticViewController містить методи обробки інформації про статистику гравця має зв’язок асоціації з MenuController, оскільки в методі класу відбувається виклик класу MenuController а також залежність з класом ViewController.

Клас TablesViewController містить методи обробки інформації про вільні і зайняті столи має зв’язок асоціації з MenuController, оскільки в методі класу відбувається виклик класу MenuController а також залежність з класом ViewController.

Клас TimerViewController містить методи обробки інформації про час має зв’язок асоціації з MenuController, оскільки в методі класу відбувається виклик класу MenuController а також залежність з класом ViewController.

Клас PriceViewController містить опис елементів інтерфейсу, інформації про ціни, має зв’язок асоціації від MenuController, оскільки в методі класу відбувається виклик класу MenuController а також залежність з класом ViewController.

Клас GameViewController містить методи, що допомагають створити сесію гри з іншим користувачем має зв’язок асоціації від MenuController, оскільки в методі класу відбувається виклик класу MenuController а також залежність з класом ViewController.

Клас SettingsViewController містить методи, що допомагають налаштувати програму під користувача має зв’язок асоціації від MenuController, оскільки в методі класу відбувається виклик класу MenuController а також залежність з класом ViewController.

Клас AboutUsViewController містить опис елементів інтерфейсу, інформації про клуб, має зв’язок асоціації від MenuController, оскільки в методі класу відбувається виклик класу MenuController а також залежність з класом ViewController.

2.3 Розробка діаграми послідовностей дій системи

Діаграма послідовності є одним із видів діаграм в UML (Unified Modeling Language), призначеним для моделювання послідовності взаємодії між об'єктами в рамках системи програмного забезпечення. Вона ілюструє, як об'єкти взаємодіють один з одним у конкретному процесі або сценарії, показуючи послідовність повідомлень, які вони обмінюються.

Діаграма послідовності системи представлена на рис. 2.6.

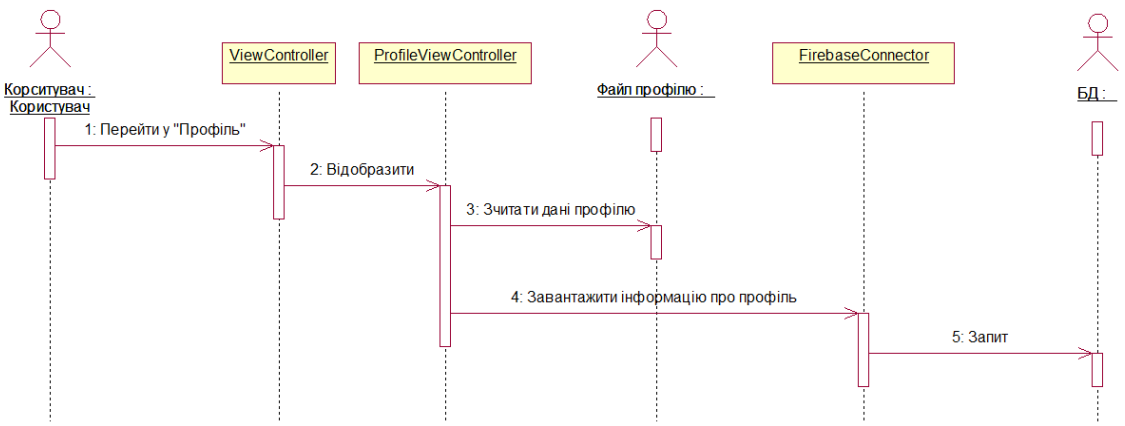

Рисунок 2.6 – Діаграма послідовності системи

Для відображення переходів між вікнами мобільного застосунку «Більярдний клуб Класік» побудовано діаграму переходів станів, яка представлена на рис. 2.7.

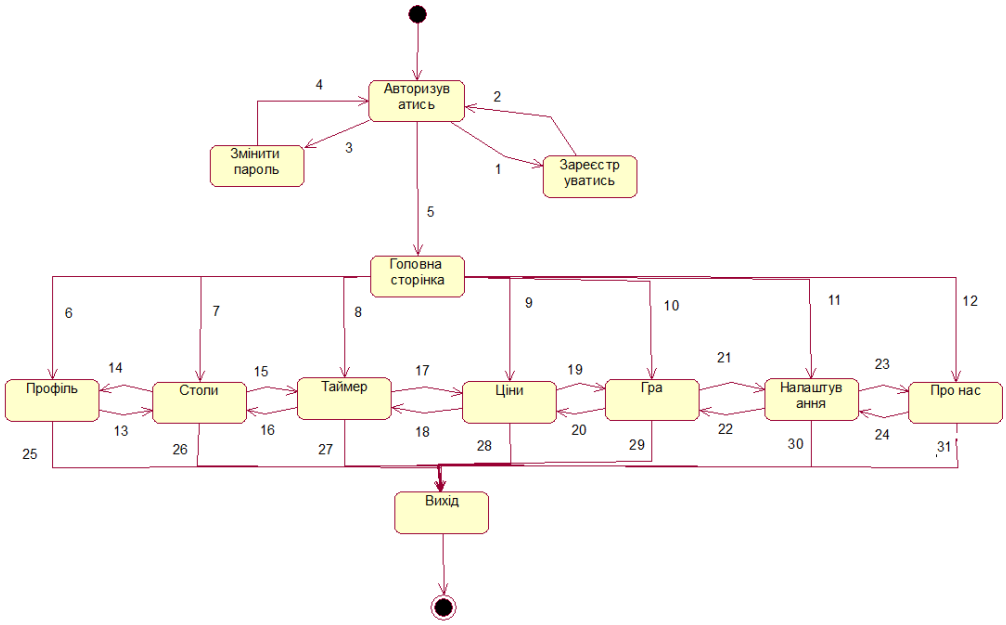

Рисунок 2.7 – Діаграма станів мобільного застосунку «Більярдний клуб Класік»

Таблиця 2.1 – Специфікація діаграми переходів станів

Номер	Умова/Дія
1	2
1	Ініціювання відкриття форми реєстрації / Відкриття форми реєстрації.
2	Ініціювання повернення до вікна авторизації / Відкриття форми авторизації.
3	Ініціювання відкриття форми зміни паролю / Відкриття форми зміни паролю.
4	Ініціювання повернення до вікна авторизації / Відкриття форми авторизації.
5	Ініціювання переходу на головну сторінку / Відкриття форми головної сторінки
6	Ініціювання переходу на вікно «Профіль» / Відкритя вікна «Профіль»
7	Ініціювання переходу на вікно «Столи» / Відкритя вікна «Столи»
8	Ініціювання переходу на вікно «Таймер» / Відкритя вікна «Таймер»
9	Ініціювання переходу на вікно «Ціни» / Відкритя вікна «Ціни»
10	Ініціювання переходу на вікно «Гра» / Відкритя вікна «Гра»
11	Ініціювання переходу на вікно «Налаштування» / Відкритя вікна «Налаштування»
12	Ініціювання переходу на вікно «Про нас» / Відкритя вікна «Про нас»
13	Ініціювання переходу на вікно «Столи» / Відкритя вікна «Столи»

Продовження таблиці 2.1

1	2
14	Ініціювання переходу на вікно «Профіль» / Відкритя вікна «Профіль»
15	Ініціювання переходу на вікно «Таймер» / Відкритя вікна «Таймер»
16	Ініціювання переходу на вікно «Столи» / Відкритя вікна «Столи»
17	Ініціювання переходу на вікно «Ціни» / Відкритя вікна «Ціни»
18	Ініціювання переходу на вікно «Таймер» / Відкритя вікна «Таймер»
19	Ініціювання переходу на вікно «Гра» / Відкритя вікна «Гра»
20	Ініціювання переходу на вікно «Ціни» / Відкритя вікна «Ціни»
21	Ініціювання переходу на вікно «Налаштування» / Відкритя вікна «Налаштування»
22	Ініціювання переходу на вікно «Гра» / Відкритя вікна «Гра»
23	Ініціювання переходу на вікно «Про нас» / Відкритя вікна «Про нас»
24	Ініціювання переходу на вікно «Налаштування» / Відкритя вікна «Налаштування»
25,26, 27,28, 29,30,31	Ініціювання виходу з додатку / Вихід з додатку

Діаграма переходів станів є типом діаграми в UML (Unified Modeling Language), яка моделює різні стани системи або об'єкта та переходи між цими станами відповідно до зовнішніх подій або внутрішніх умов.

Висновки до розділу 2

У другому розділі кваліфікаційної роботи бакалавра спроєктовано клієнтский мобільний застосунок для більярдного клубу.

- розроблено наступні моделі та діаграми:
- діаграму варіантів використання системи;
- діаграму класів системи;
- діаграму переходів станів системи;
- діаграму послідовностей.

3 ВИБІР ТЕХНОЛОГІЙ РОЗРОБКИ ТА МОДЕЛЬ БАЗИ ДАНИХ

3.1 Вибір технологій розробки

Мобільні застосунки відіграють ключову роль у сучасному цифровому світі, забезпечуючи користувачів зручними та інтуїтивно зрозумілими засобами для виконання різноманітних завдань, від спілкування до управління фінансами і розваг. Ринок мобільних застосунків продовжує зростати, залучаючи мільйони користувачів та генеруючи значні доходи. Однією з провідних платформ для розробки мобільних застосунків є операційна система iOS від Apple. Запущена у 2007 році, iOS швидко завоювала популярність завдяки своїй надійності, безпеці та продуктивності.

iOS працює на різноманітних пристроях Apple, включаючи iPhone, iPad, iPod touch та Apple Watch. Кожна нова версія iOS приносить покращення у функціональність, безпеку та користувацький досвід, що робить її привабливою для розробників та користувачів. Платформа iOS підтримує багатозадачність, має інтеграцію з іншими сервісами Apple, такими як iCloud, Apple Music та Apple Pay, що дозволяє створювати комплексні та зручні застосунки [1].

Основою для розробки застосунків на iOS є мова програмування Swift, яка була представлена Apple у 2014 році. Swift є сучасною мовою програмування, що поєднує простоту використання з високою продуктивністю та безпекою. Вона дозволяє розробникам писати більш чистий та надійний код, зменшуючи кількість помилок і покращуючи загальну продуктивність застосунків. Swift тісно інтегрована з інструментами та фреймворками, розробленими Apple, що робить процес розробки ще більш ефективним.

Одним із ключових інструментів для розробки застосунків на iOS є Xcode – інтегроване середовище розробки (IDE), розроблене Apple. Xcode містить усе необхідне для створення, тестування та розгортання застосунків, включаючи редактор коду, інструменти для налагодження, симулятори пристроїв та інструменти для аналізу продуктивності. Xcode також підтримує інтеграцію з

іншими сервісами Apple, такими як TestFlight для тестування застосунків та App Store для розповсюдження.

Розробка застосунків для iOS вимагає від розробників знання принципів дизайну інтерфейсів користувача, які відповідають вимогам Apple. Це включає використання гайдлайнів Human Interface Guidelines (HIG), які визначають правила і рекомендації щодо створення інтерфейсів, що забезпечують інтуїтивність, естетику та зручність використання. Відповідність цим стандартам допомагає створювати застосунки, які легко сприймаються користувачами та забезпечують високий рівень взаємодії.

Безпека є однією з основних переваг iOS. Apple приділяє особливу увагу захисту даних користувачів, впроваджуючи різноманітні механізми безпеки, такі як шифрування даних, захист від зловмисного програмного забезпечення та контроль за доступом до системних ресурсів. Це забезпечує користувачам високу ступінь захисту їхніх особистих даних та конфіденційності.

Завдяки своїй надійності, високій продуктивності та багатому набору інструментів для розробників, iOS залишається однією з провідних платформ для розробки мобільних застосунків. Вона забезпечує розробникам можливості для створення інноваційних та функціональних застосунків, які задовольняють потреби користувачів і відповідають сучасним стандартам якості та безпеки.

Для успішної розробки застосунків на платформі iOS необхідно використовувати спеціальне програмне забезпечення, яке забезпечує всі необхідні інструменти для створення, тестування та розгортання застосунків. Одним із основних інструментів є Xcode – інтегроване середовище розробки (IDE), розроблене Apple. Xcode включає редактор коду, компілятор, симулятори пристроїв та інструменти для налагодження і тестування, що робить його потужним і універсальним інструментом для розробників.

Xcode підтримує як мову програмування Swift, так і Objective-C, що дозволяє розробникам використовувати всі можливості платформи iOS. Редактор коду в Xcode надає зручні інструменти для автозаповнення, підсвічування синтаксису та аналізу коду в реальному часі, що допомагає розробникам швидше

писати та налагоджувати код. Крім того, Xcode включає інструменти для візуального дизайну інтерфейсів користувача, такі як Interface Builder, який дозволяє створювати інтерфейси за допомогою перетягування елементів.

Однією з ключових функцій Xcode є інтеграція з різноманітними сервісами Apple, такими як TestFlight для бета-тестування застосунків та App Store Connect для управління застосунками в App Store. Це дозволяє розробникам легко розповсюджувати свої застосунки та отримувати зворотний зв'язок від користувачів. Xcode також включає інструменти для аналізу продуктивності застосунків, такі як Instruments, що дозволяє виявляти та виправляти проблеми з продуктивністю та ефективністю.

Ще одним важливим інструментом для розробки на iOS є Swift Playgrounds – інтерактивне середовище для вивчення мови програмування Swift. Swift Playgrounds дозволяє розробникам експериментувати з кодом, бачити результати в режимі реального часу та вивчати основи програмування на Swift. Це особливо корисно для початківців, які тільки починають свою кар'єру у розробці застосунків на iOS [2, 6].

Розробка застосунків на iOS також вимагає знання принципів дизайну інтерфейсів користувача, які відповідають вимогам Apple. Для цього використовуються гайдлайни Human Interface Guidelines (HIG), які визначають правила та рекомендації щодо створення інтерфейсів, що забезпечують інтуїтивність, естетику та зручність використання. Відповідність цим стандартам допомагає створювати застосунки, які легко сприймаються користувачами та забезпечують високий рівень взаємодії.

Крім того, розробники можуть використовувати різноманітні бібліотеки та фреймворки, такі як CocoaPods та Carthage, для управління залежностями та інтеграції сторонніх бібліотек у свої проєкти. Це значно спрощує процес розробки та дозволяє швидко додавати нові функціональні можливості до застосунків.

Ще одним важливим аспектом розробки на iOS є використання служб і сервісів, таких як Firebase, для розширення функціональних можливостей застосунків. Firebase надає різноманітні інструменти для аутентифікації

користувачів, зберігання даних, аналітики та надсилання повідомлень, що дозволяє створювати комплексні та функціональні застосунки [2].

Використання спеціального програмного забезпечення для розробки на iOS забезпечує високу якість, продуктивність та безпеку застосунків, що є ключовими факторами для успішного залучення користувачів та підтримки їхньої задоволеності. Завдяки широкому набору інструментів і сервісів, розробники можуть створювати інноваційні застосунки, які відповідають найвищим стандартам якості та безпеки.

Firebase Analytics є потужним інструментом для збирання та аналізу даних про поведінку користувачів у мобільних і вебзастосунках. Він дозволяє розробникам отримувати детальну інформацію про те, як користувачі взаємодіють із застосунком, що допомагає приймати обґрунтовані рішення щодо покращення функціональності та користувацького досвіду. Firebase Analytics забезпечує автоматичний збір даних про події та поведінкові параметри, що дозволяє розробникам зосередитися на аналізі отриманих даних і прийнятті стратегічних рішень.

Одна з основних переваг Firebase Analytics полягає у його інтеграції з іншими сервісами Firebase, такими як Firebase Crashlytics, Firebase Remote Config та Firebase Cloud Messaging. Це дозволяє розробникам отримувати комплексний огляд роботи застосунка і швидко реагувати на виявлені проблеми. Наприклад, за допомогою Firebase Crashlytics можна відстежувати аварійні завершення роботи застосунка, а Firebase Remote Config дозволяє змінювати налаштування застосунка в режимі реального часу без необхідності випускати нову версію.

Firebase Analytics забезпечує збирання даних у режимі реального часу, що дозволяє розробникам оперативно реагувати на зміни у поведінці користувачів. Це особливо корисно для виявлення проблемних зон у застосунку та оптимізації користувацького досвіду. Крім того, Firebase Analytics дозволяє сегментувати користувачів за різними критеріями, такими як географічне розташування, тип пристрою або версія операційної системи, що допомагає краще розуміти потреби різних груп користувачів і адаптувати застосунок відповідно до їхніх очікувань.

Ще однією важливою функцією Firebase Analytics є можливість створення спеціальних подій і параметрів, які дозволяють розробникам відстежувати конкретні дії користувачів у застосунку. Це може включати, наприклад, відстеження покупок, перегляд сторінок або виконання певних дій у застосунку. Завдяки цьому розробники можуть отримувати більш детальну інформацію про поведінку користувачів і ефективність різних функцій застосунка.

Firebase Analytics також підтримує інтеграцію з Google Ads, що дозволяє розробникам оцінювати ефективність рекламних кампаній і оптимізувати маркетингові стратегії. Це допомагає залучати нових користувачів і збільшувати залучення існуючих, що сприяє загальному успіху застосунка. Аналіз даних про взаємодію користувачів з рекламними оголошеннями дозволяє визначити, які кампанії приносять найбільший прибуток і залучають найбільш лояльних користувачів.

Крім того, Firebase Analytics надає розробникам інструменти для створення звітів і візуалізації даних. Це дозволяє легко аналізувати зібрані дані і виявляти тенденції у поведінці користувачів. Інтерфейс Firebase Analytics є інтуїтивно зрозумілим і зручним для користувачів, що дозволяє швидко знаходити необхідну інформацію і приймати обґрунтовані рішення.

Використання Firebase Analytics у розробці застосунків забезпечує розробникам глибоке розуміння поведінки користувачів і ефективності різних функцій застосунка. Це дозволяє оптимізувати користувацький досвід, підвищувати залученість користувачів і досягати більш високих результатів. Завдяки інтеграції з іншими сервісами Firebase і можливості створення спеціальних подій і параметрів, Firebase Analytics є потужним інструментом для забезпечення успіху мобільних і вебзастосунків.

Firebase Cloud Messaging (FCM) є потужним інструментом для надсилання повідомлень та сповіщень на мобільні пристрої та веббраузери. Цей сервіс дозволяє розробникам легко інтегрувати функціональність повідомлень у свої застосунки, забезпечуючи оперативний зв'язок з користувачами. FCM підтримує різні типи повідомлень, включаючи сповіщення, дані повідомлення та

повідомлення з високим пріоритетом, що дозволяє адаптувати систему повідомлень відповідно до потреб застосунка.

Однією з головних переваг FCM є його здатність надсилати повідомлення на різні платформи, включаючи iOS, Android та веббраузери. Це забезпечує єдиний підхід до управління повідомленнями і дозволяє охоплювати широку аудиторію користувачів. FCM використовує надійну та масштабовану інфраструктуру Google, що забезпечує високу надійність і швидкість доставки повідомлень.

FCM надає розробникам можливість сегментувати користувачів і надсилати повідомлення тільки певним групам користувачів. Це дозволяє персоналізувати сповіщення і підвищувати їхню ефективність. Наприклад, можна надсилати спеціальні пропозиції користувачам, які часто використовують певну функцію застосунка, або нагадування тим, хто давно не відкривав застосунок. Така сегментація допомагає підвищити залученість користувачів і утримувати їх у застосунку.

Ще однією важливою функцією FCM є можливість надсилання повідомлень з високим пріоритетом, які доставляються негайно і відображаються на екрані навіть у режимі блокування пристрою. Це особливо корисно для застосунків, які потребують оперативного інформування користувачів про важливі події або оновлення. Наприклад, застосунки для новин можуть використовувати високопріоритетні повідомлення для інформування користувачів про екстрені новини.

FCM також підтримує функціональність повідомлень з даними, які дозволяють надсилати приховані повідомлення з даними, які обробляються застосунком у фоновому режимі. Це корисно для синхронізації даних або оновлення вмісту застосунка без необхідності інформувати користувача. Така функціональність забезпечує безшовний користувацький досвід і підвищує загальну ефективність застосунка.

Інтеграція FCM у застосунок є досить простою завдяки докладній документації та зручним інструментам розробки. Firebase надає SDK для різних платформ, що спрощує процес впровадження і забезпечує швидкий старт роботи

з FCM. Крім того, Firebase Console надає зручний інтерфейс для управління повідомленнями, налаштування кампаній і аналізу ефективності повідомлень.

Використання Firebase Cloud Messaging у застосунках забезпечує розробникам ефективний інструмент для оперативного зв'язку з користувачами, підвищення залученості та утримання користувачів у застосунку. Завдяки широким можливостям сегментації, підтримці різних типів повідомлень і надійній інфраструктурі, FCM є потужним інструментом для забезпечення успішного функціонування мобільних і вебзастосунків. Це дозволяє розробникам створювати інноваційні рішення, які відповідають високим стандартам якості та забезпечують відмінний користувацький досвід.

Firebase Auth є потужним інструментом для аутентифікації користувачів у мобільних і вебзастосунках. Він надає розробникам можливість легко інтегрувати різні методи аутентифікації, такі як електронна пошта і пароль, телефонні номери, а також сторонні постачальники, такі як Google, Facebook, Twitter та інші. Це забезпечує гнучкість і зручність для користувачів, дозволяючи їм вибирати найбільш зручний спосіб входу в застосунок.

Однією з головних переваг Firebase Auth є його простота інтеграції. Firebase надає SDK для різних платформ, що спрощує процес впровадження аутентифікації у застосунок. Крім того, Firebase Console надає зручний інтерфейс для управління користувачами, налаштування параметрів аутентифікації та моніторингу активності користувачів. Це дозволяє розробникам швидко налаштовувати аутентифікацію і зосередитися на основних функціональних можливостях застосунка.

Firebase Auth підтримує різні методи аутентифікації, що забезпечує гнучкість і зручність для користувачів. Наприклад, аутентифікація за допомогою електронної пошти і пароля є одним з найбільш поширених методів, який забезпечує високий рівень безпеки і зручність використання. Крім того, Firebase Auth підтримує аутентифікацію за допомогою телефонних номерів, що дозволяє користувачам входити в застосунок за допомогою одноразових кодів підтвердження, що надсилаються на їхні телефони.

Сторонні постачальники аутентифікації, такі як Google, Facebook і Twitter, також підтримуються Firebase Auth, що дозволяє користувачам входити в застосунок за допомогою своїх облікових записів у соціальних мережах. Це забезпечує зручність і швидкість входу для користувачів, а також спрощує процес реєстрації нових користувачів. Крім того, використання соціальних мереж для аутентифікації підвищує рівень безпеки, оскільки сторонні постачальники забезпечують застосункові механізми захисту даних користувачів.

Firebase Auth забезпечує високу безпеку даних користувачів завдяки впровадженню сучасних методів шифрування і захисту від зловмисних атак. Це включає захист від фішингу, атаки грубої сили та інших типів атак, що забезпечує безпеку облікових записів користувачів і захист їхніх особистих даних. Крім того, Firebase Auth підтримує багатофакторну аутентифікацію (MFA), що забезпечує додатковий рівень захисту для користувачів.

Ще однією важливою функцією Firebase Auth є можливість управління користувачами через Firebase Console. Це включає створення, редагування та видалення облікових записів користувачів, а також моніторинг активності користувачів і аналіз даних про використання застосунка. Така функціональність дозволяє розробникам ефективно управляти користувацькими обліковими записами і забезпечувати високий рівень обслуговування користувачів.

Інтеграція Firebase Auth у застосунки забезпечує розробникам потужний інструмент для аутентифікації користувачів, який поєднує високу безпеку, зручність використання та гнучкість. Завдяки підтримці різних методів аутентифікації, інтеграції зі сторонніми постачальниками і можливості управління користувачами через Firebase Console, Firebase Auth є незамінним інструментом для забезпечення надійної і безпечної аутентифікації у мобільних і вебзастосунках. Це дозволяє розробникам створювати інноваційні рішення, які відповідають високим стандартам безпеки і забезпечують відмінний користувацький досвід.

Firebase Realtime Database є потужним інструментом для зберігання і синхронізації даних у режимі реального часу для мобільних і вебзастосунків. Ця

база даних дозволяє розробникам створювати інтерактивні та динамічні застосунки, які можуть оперативно обмінюватися даними між користувачами і пристроями. Основною перевагою Realtime Database є її здатність забезпечувати миттєву синхронізацію даних, що дозволяє застосункам реагувати на зміни в режимі реального часу.

Realtime Database використовує модель даних на основі JSON, що робить її простою у використанні та легкою для інтеграції. Дані зберігаються у вигляді ключ-значення, що дозволяє швидко і ефективно зберігати і отримувати доступ до інформації. Ця модель є дуже гнучкою і дозволяє зберігати різноманітні типи даних, від простих текстових значень до складних об'єктів.

Однією з основних функцій Realtime Database є можливість синхронізації даних у режимі реального часу між усіма підключеними клієнтами. Це особливо корисно для застосунків, які потребують оперативного обміну інформацією між користувачами, таких як чати, спільні документи або онлайн-ігри. Завдяки миттєвій синхронізації, користувачі можуть бачити зміни в застосунку одразу після їх внесення, що забезпечує безшовний і інтерактивний користувацький досвід.

Firebase Realtime Database також підтримує офлайн-режим, що дозволяє застосункам продовжувати працювати навіть при відсутності інтернет-з'єднання. Дані зберігаються локально на пристрої користувача і автоматично синхронізуються з базою даних, коли з'єднання відновлюється. Це забезпечує безперебійну роботу застосунка і збереження даних користувачів у будь-яких умовах.

Безпека даних у Firebase Realtime Database забезпечується за допомогою правил безпеки, які дозволяють контролювати доступ до даних на основі автентифікації користувачів і інших умов. Розробники можуть налаштовувати правила доступу, щоб забезпечити захист конфіденційної інформації і запобігти несанкціонованому доступу до даних. Крім того, дані передаються по захищеному каналу, що забезпечує додатковий рівень безпеки.

Інтеграція Firebase Realtime Database у застосунки є досить простою завдяки докладній документації та зручним інструментам розробки. Firebase надає SDK для різних платформ, що спрощує процес впровадження і забезпечує швидкий старт роботи з базою даних. Крім того, Firebase Console надає зручний інтерфейс для управління базою даних, моніторингу активності та налаштування правил безпеки.

Використання Firebase Realtime Database у розробці застосунків забезпечує розробникам потужний інструмент для зберігання і синхронізації даних у режимі реального часу. Завдяки підтримці офлайн-режиму, гнучкості моделі даних та можливості налаштування правил безпеки, Realtime Database є незамінним інструментом для створення інтерактивних і динамічних застосунків, які забезпечують відмінний користувацький досвід і відповідають високим стандартам безпеки. Це дозволяє розробникам створювати інноваційні рішення, які відповідають потребам користувачів і забезпечують високу продуктивність і надійність.

Firebase Storage є потужним інструментом для зберігання та управління файлами, такими як зображення, відео, аудіо та інші мультимедійні дані, у мобільних і вебзастосунках. Цей сервіс надає розробникам надійну та масштабовану інфраструктуру для зберігання великих обсягів даних без необхідності налаштовувати власну серверну інфраструктуру. Використання Firebase Storage дозволяє значно спростити процес управління файлами і забезпечити високу надійність і доступність даних.

Однією з основних переваг Firebase Storage є його інтеграція з іншими сервісами Firebase, такими як Firebase Authentication і Firebase Realtime Database. Це дозволяє легко керувати доступом до файлів і забезпечувати безпеку даних користувачів. Наприклад, розробники можуть налаштувати правила безпеки, які дозволяють завантажувати або завантажувати файли тільки авторизованим користувачам або певним групам користувачів. Це забезпечує захист конфіденційної інформації і запобігає несанкціонованому доступу до даних.

Firebase Storage використовує потужну інфраструктуру Google Cloud Storage, що забезпечує високу надійність, швидкість і масштабованість. Це дозволяє зберігати великі обсяги даних і обробляти запити на завантаження і завантаження файлів з високою продуктивністю. Крім того, дані зберігаються у зашифрованому вигляді, що забезпечує додатковий рівень безпеки.

Інтеграція Firebase Storage у застосунки є досить простою завдяки докладній документації та зручним інструментам розробки. Firebase надає SDK для різних платформ, що спрощує процес впровадження і забезпечує швидкий старт роботи з сховищем. Крім того, Firebase Console надає зручний інтерфейс для управління файлами, моніторингу активності і налаштування правил безпеки.

Ще однією важливою функцією Firebase Storage є можливість генерації URL-адрес для доступу до файлів, що дозволяє легко ділитися файлами з іншими користувачами або вбудовувати мультимедійні дані у застосунок. Це забезпечує зручність і гнучкість у роботі з файлами, дозволяючи розробникам створювати багатофункціональні застосунки з інтерактивним контентом.

Firebase Storage також підтримує обробку файлів у фоновому режимі, що дозволяє застосункам продовжувати працювати навіть при завантаженні великих файлів. Це забезпечує безшовний користувацький досвід і запобігає збоїв у роботі застосунка. Крім того, Firebase Storage надає можливість автоматичного збереження версій файлів, що дозволяє відновлювати попередні версії файлів у разі необхідності.

Використання Firebase Storage у розробці застосунків забезпечує розробникам потужний інструмент для зберігання і управління файлами, який поєднує високу надійність, безпеку і масштабованість. Завдяки інтеграції з іншими сервісами Firebase, підтримці різних платформ і можливості генерації URL-адрес для доступу до файлів, Firebase Storage є незамінним інструментом для створення багатофункціональних застосунків з інтерактивним контентом. Це дозволяє розробникам створювати інноваційні рішення, які відповідають високим стандартам якості і забезпечують відмінний користувацький досвід.

3.2 Моделювання бази даних

Оскільки Firebase відноситься до СУБД парадигми NoSQL а саме до документо-орієнтованих СУБД, то ж логічна модель не проєктується.

Документо-орієнтована система керування базами даних– система керування базами даних, спеціально призначена для зберігання ієрархічних структур даних (документів) і зазвичай реалізована за допомогою підходу NoSQL. В основі документо-орієнтованих СКБД лежать документні сховища, котрі мають структуру дерева (іноді лісу). Структура дерева починається з кореневого вузла і може містити кілька внутрішніх і листових вузлів. Листові вузли містять дані, які при додаванні документа заносяться в індекси, що дозволяє навіть при досить складній структурі знаходити місце (шлях) шуканих даних.

Фізична модель представлена у вигляді сутностей з атрибутами (див. рис. 3.1):

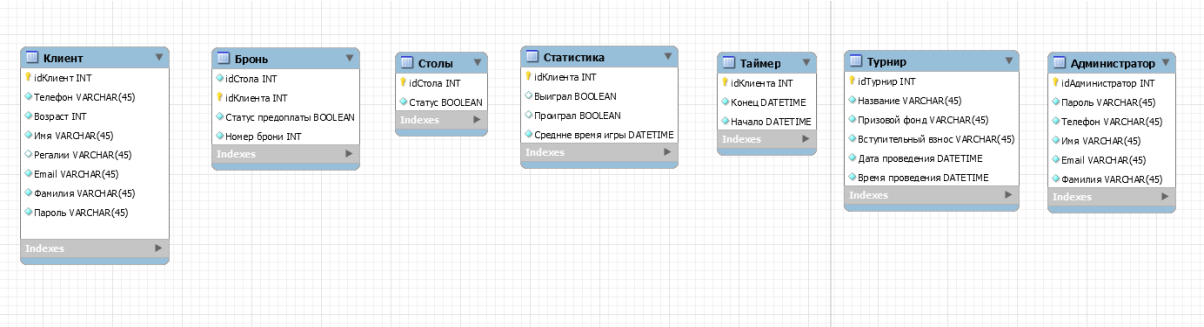

Рисунок 3.1 – Фізична модель БД мобільного застосунку «Більярдний клуб Класік»

Таблиця 3.1 – Специфікація до сутності «Клієнт»

Ідентифікатор поля	Ознака ключа	Тип даних	Обмеження
1	2	3	4
idКлієнт	PK	INT	NOT NULL
Телефон		VARCHAR(45)	NOT NULL
Вік		INT	NOT NULL
Ім’я		CHAR(45)	NOT NULL

Продовження таблиці 3.1

1	2	3	4
Регаліі		VARCHAR(45)	
Email		CHAR(45)	NOT NULL
Прізвище		CHAR(45)	NOT NULL
Пароль		CHAR(45)	NOT NULL

Таблиця 3.2 Специфікація до сутності «Бронь»

Ідентифікатор поля	Ознака ключа	Тип даних	Обмеження
1	2	3	4
idСтола	PK	INT	NOT NULL
idКлієнта	FK	INT	NOT NULL
Статус передплати		BOOLEAN	NOT NULL
Номер бронювання		INT	NOT NULL

Таблиця 3.3 Специфікація до сутності «Столи»

Ідентифікатор поля	Ознака ключа	Тип даних	Обмеження
1	2	3	4
idСтола	PK	INT	NOT NULL
Статус		BOOLEAN	NOT NULL

Таблиця 3.4 Специфікація до сутості «Статистика»

Ідентифікатор поля	Ознака ключа	Тип даних	Обмеження
1	2	3	4
idКлієнта	PK	INT	NOT NULL
Виграв		BOOLEAN	NOT NULL

Програв		BOOLEAN	NOT NULL
Середній час гри		DATETIME	NOT NULL

Таблиця 3.5 Специфікація до сутості «Таймер»

Ідентифікатор поля	Ознака ключа	Тип даних	Обмеження
1	2	3	4
idКлієнта	PK	INT	NOT NULL
Кінець		DATETIME	NOT NULL
Початок		DATETIME	

Таблиця 3.6 Специфікація до сутності «Турнір»

Ідентифікатор поля	Ознака ключа	Тип даних	Обмеження
1	2	3	4
idТурніру	PK	INT	NOT NULL
Назва		VARCHAR(45)	NOT NULL
Призовой фонд		VARCHAR(45)	NOT NULL
Вступний внесок		VARCHAR(45)	NOT NULL
Дата проведення		DATETIME	NOT NULL
Час проведення		DATETIME	NOT NULL

Таблиця 3.7 Специфікація до сутності «Адміністратор»

Ідентифікатор поля	Ознака ключа	Тип даних	Обмеження
1	2	3	4
idАдміністратор	PK	INT	NOT NULL
Пароль		CHAR(45)	NOT NULL
Телефон		CHAR(45)	NOT NULL
Ім’я		VARCHAR(45)	NOT NULL
Email		VARCHAR(45)	NOT NULL
Прізвище		VARCHAR(45)	NOT NULL

3.3 Розробка мокапу системи

Інтерфейс користувача (ІК) – засіб зручної взаємодії користувача з інформаційною системою.

Сукупність засобів для обробки та відображення інформації, якнайбільш пристосованих для зручності користувача; у графічних системах інтерфейс користувача втілюється багатовіконним режимом, змінами кольору, розміру, видимості (прозорість, напівпрозорість, невидимість) вікон, їхнім розташуванням, сортуванням елементів вікон, гнучкими налаштуваннями як самих вікон, так і окремих їх елементів (файли, теки, ярлики, шрифти тощо), доступністю багатокористувацьких налаштувань.

Прототипи вікон застосунку зображені на рис. 3.2-3.6.

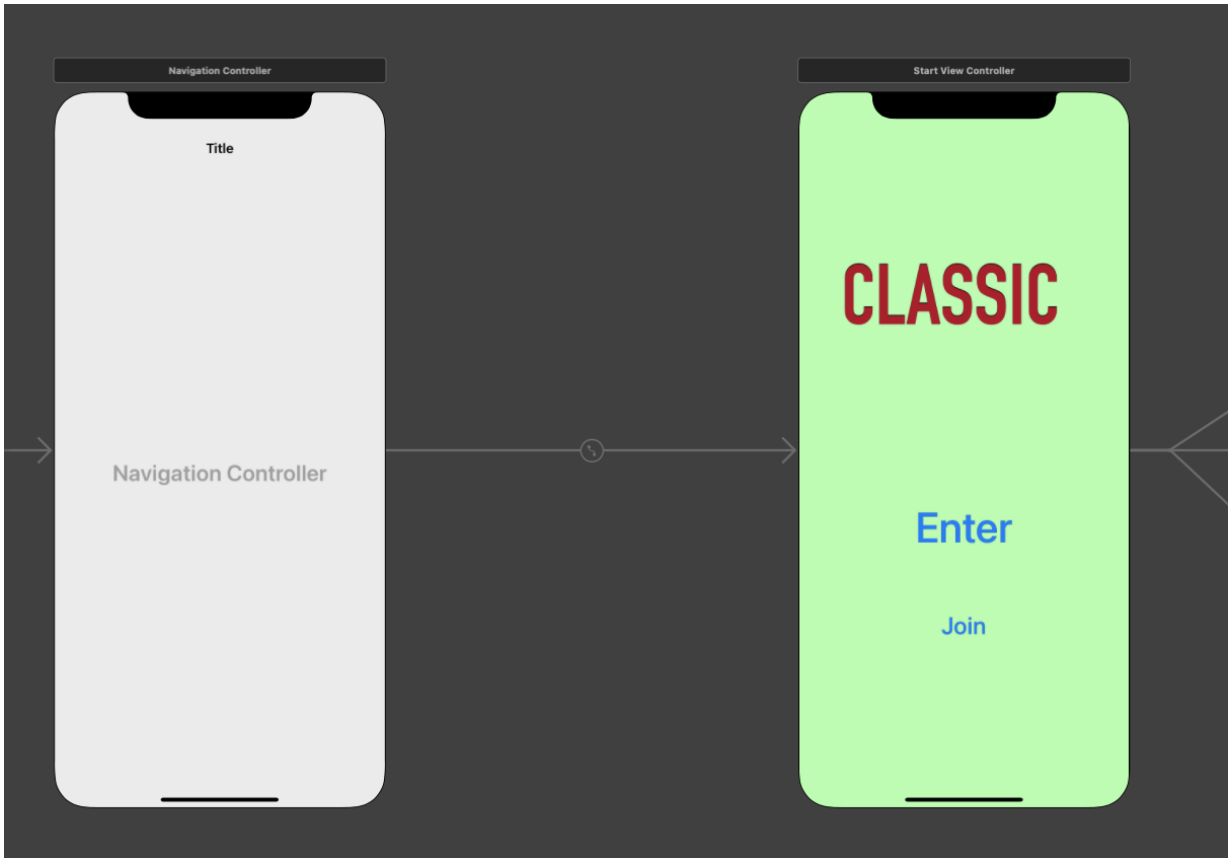

Рисунок 3.2 – Прототип стартового вікна

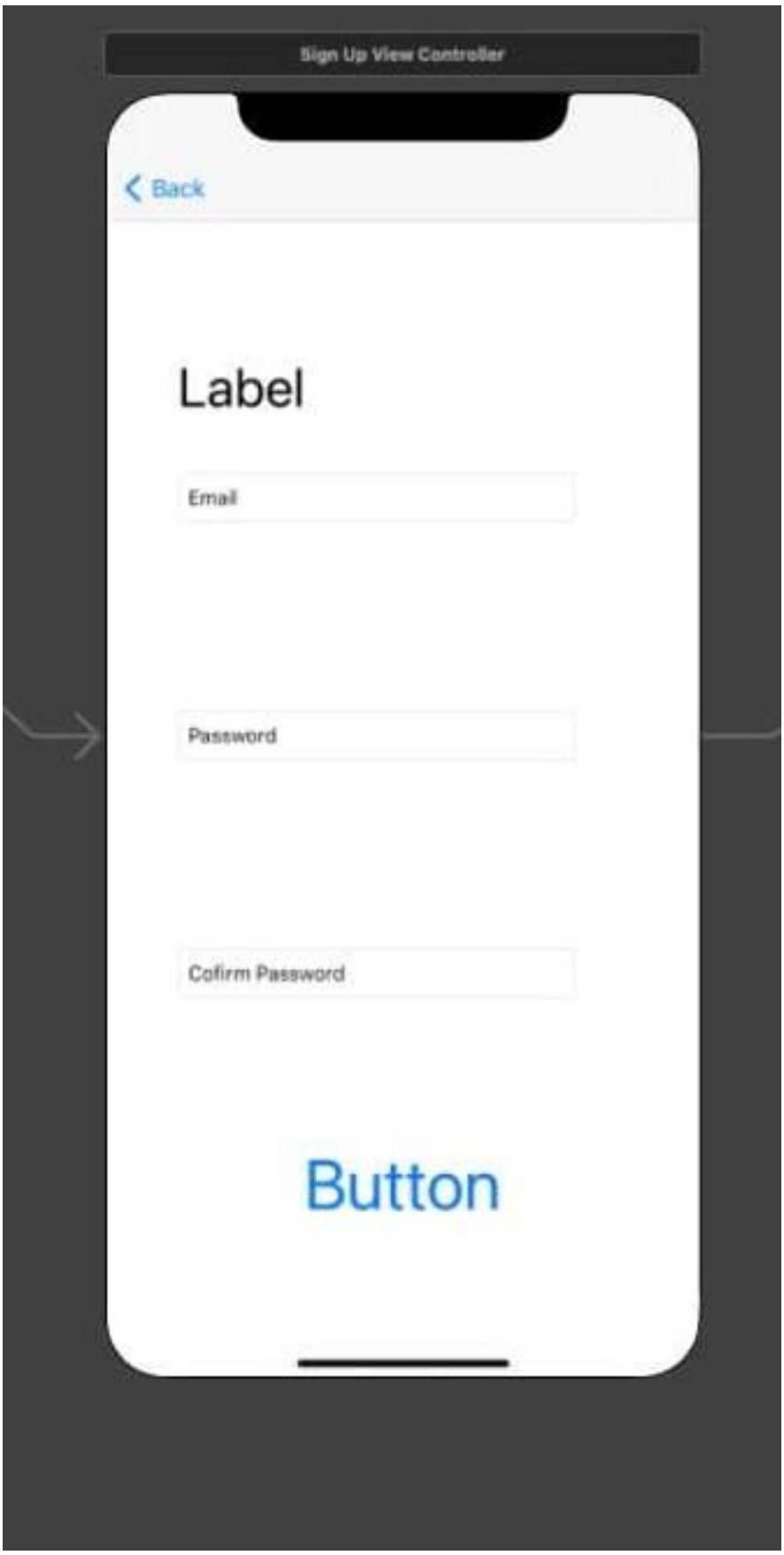

Рисунок 3.3 – Прототип вікна реєстрації

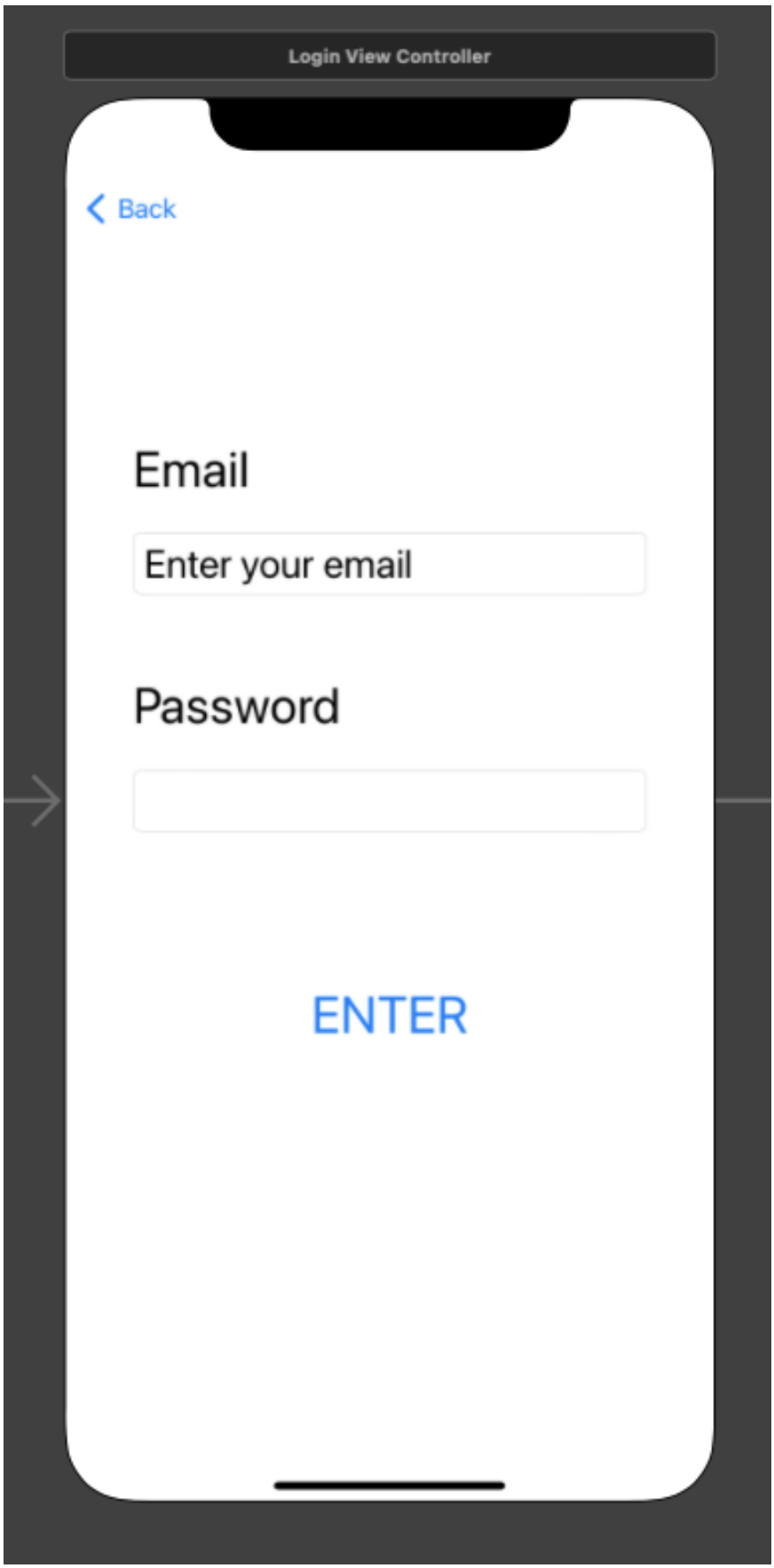

Рисунок 3.4 – Прототип вікна авторизації

Рисунок 3.5 – Прототип вікна «Про нас»

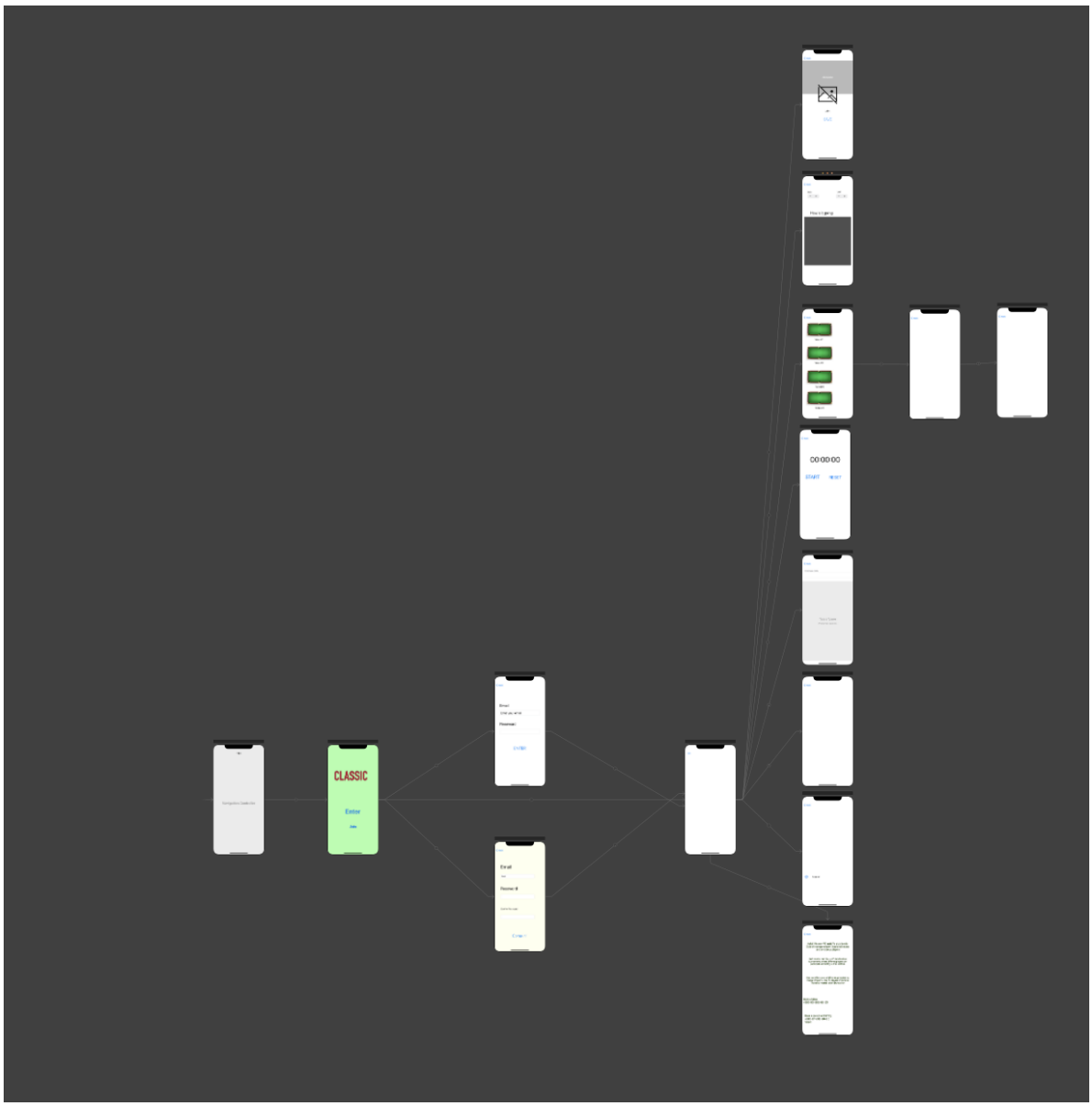

Рисунок 3.6 – Загальна схема-зв’язок усіх вікон

Дані мокапи та схеми взаємодії дозволять реалізувати функціонал застосунку у повному обсязі

Висновки до розділу 3

У третьому розділі кваліфікаційної роботи представлено вибір технологій розробки.

Досліджено та обрано технології та пакети що надає провайдер Firebase. А саме: Firebase Auth, Firebase Security Manager, Firebase Console, Firebase Analytics.

Спроєктовано та відображено модель та структуру бази даних.

Розроблено мокап застосунку.

4 РЕЗУЛЬТАТИ РОЗРОБКИ

В кваліфікаційній роботі велася розробка мобільного застосунку під ОС iOS «Більярдний клуб Класік». Він включає наступні модулі:

4.1 Модуль авторизації

При завантажені ПО система відображає користувачу вікно, яке дає користувачу вибір: зареєструватися у застосунку, увійти, якщо у нього вже є аккаунт, або продовжити, якщо був виконаний вхід у систему раніше (рис. 4.1-4.5).

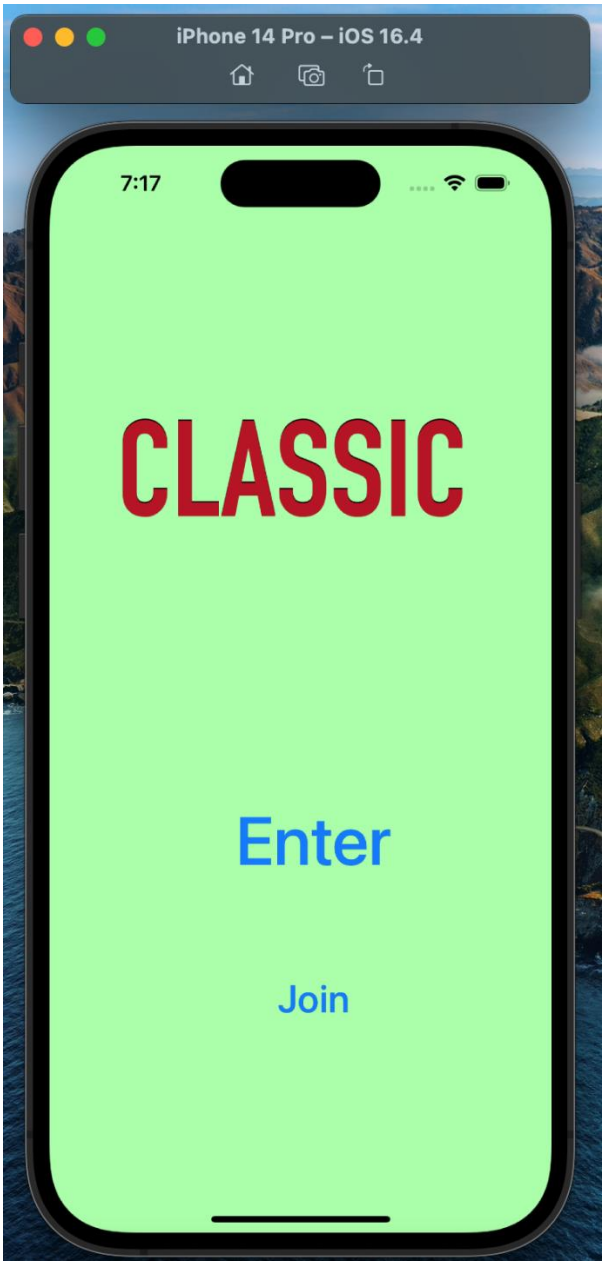

Рисунок 4.1 – Вікно вибору для авторизації

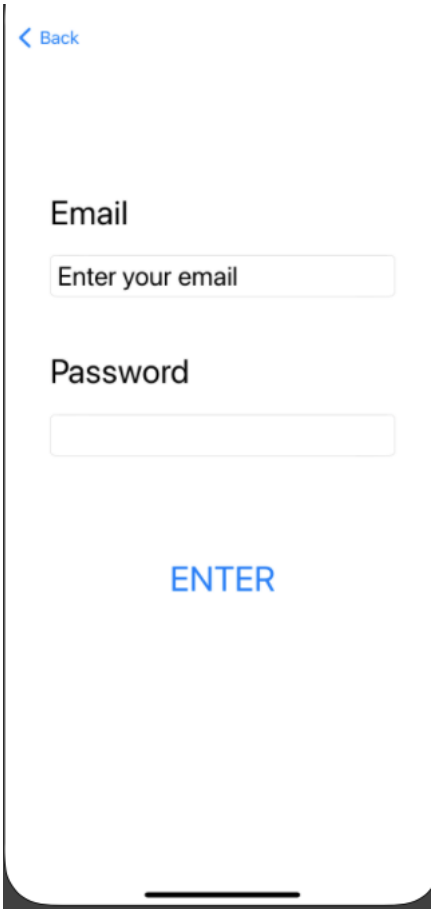

Рисунок 4.2 – Вікно входу у систему

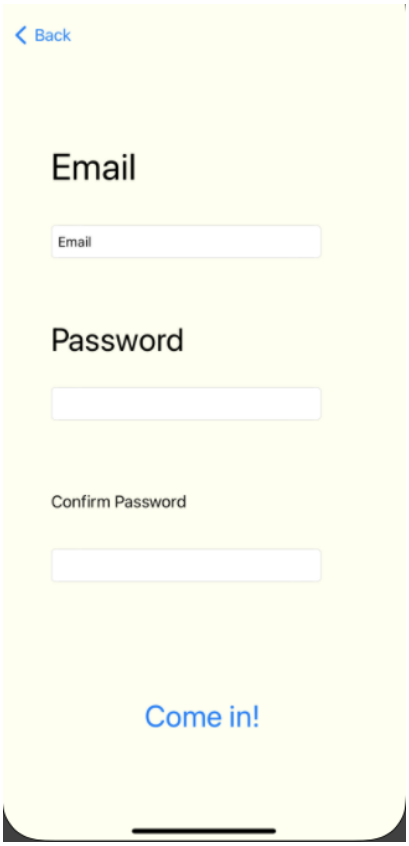

Рисунок 4.3 – Вікно реєстрації

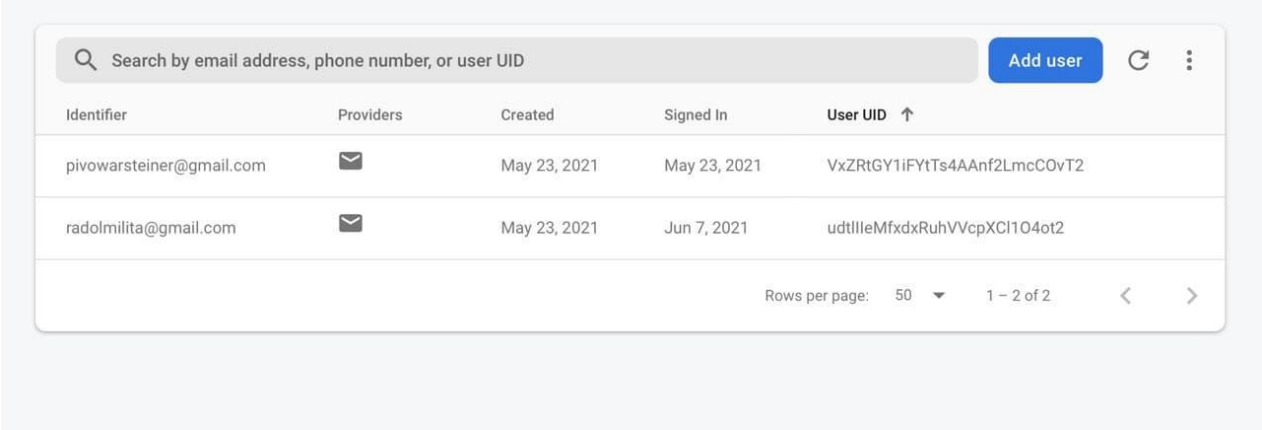

Рисунок 4.4 – Існуючі користувачі до реєстрації

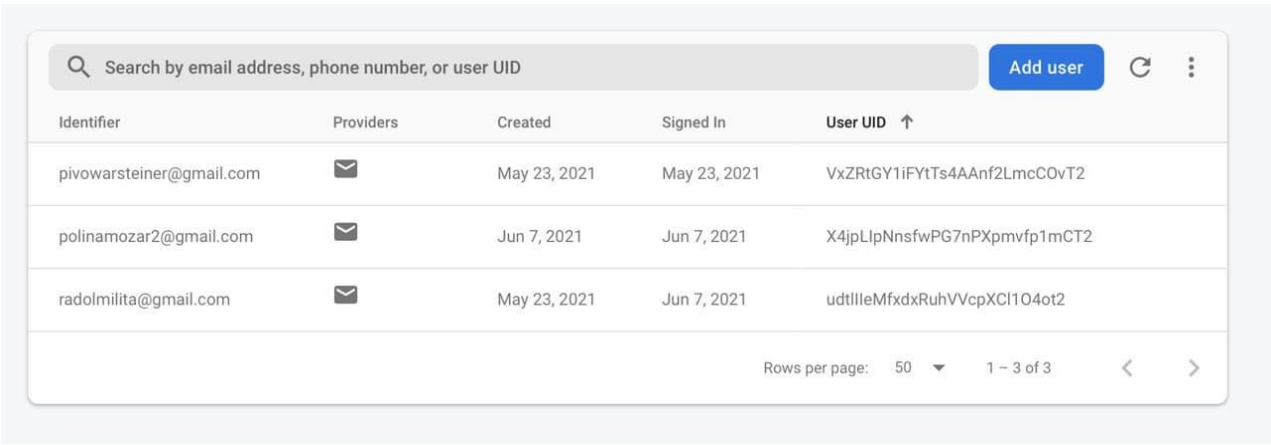

Рисунок 4.5 – Новий користувач у системі

4.2 Модуль навігації

При успішній авторизації, система відображає користувачу головне меню (рис. 4.6). Для переходу по підпрограмам системи користувач повинен натиснути кнопу на панелі навігації або провети пальцем по екрану зліва направо. Буде дійсне для будь-якого модулю при переході.

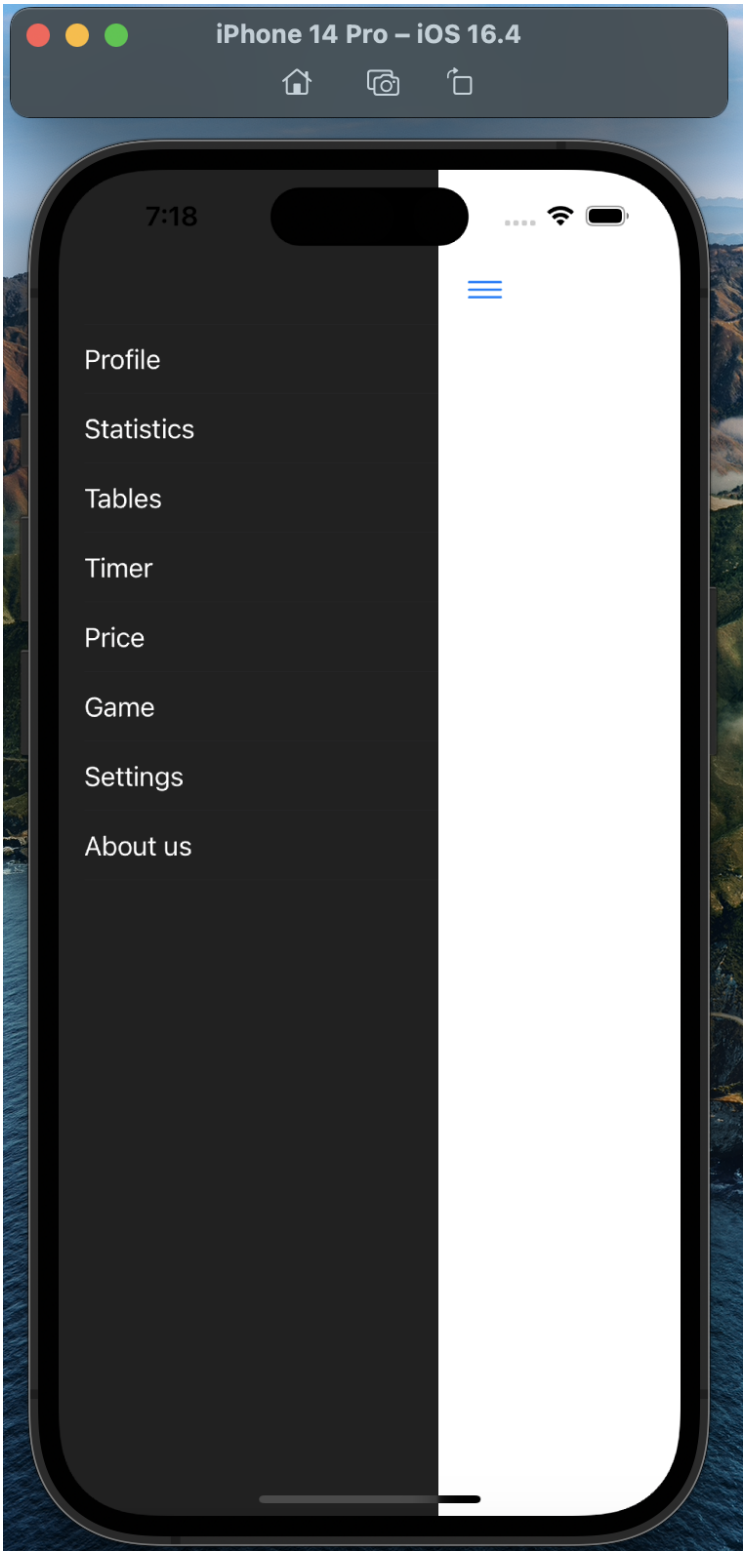

Рисунок 4.6 – Панель навігації

Модуль «Profile»

Піля головної сторінки користувач може обрати будь-яку точку програми. Але першим що йому потрібно буде зробити – створити профіль. Користувач заповнює дану йому форму й додає фотографію у свій профіль. Після чого, користувач може бачити свій заповнений профіль, а якщо при заповнені були

допущені помики, або не додані певні дані, користувач може їх змінити, шляхом натискання кнопки редагування профілю.

4.3 Модуль «підрахунку статистики»

При переході з головного екрану на модуль статистики буде показана кругова діаграма співвідношення виграшних партій до програшних з можливістю редагування (рис. 4.7).

Рисунок 4.7 – Модуль підрахунку статистики

4.4 Модуль «Бронювання столів»

Також, після створення профілю, користувачу відкривається можливість бронювати стіл. Але, бронювання буде неможливим без прив’язки платіжної карти, що б залишати заставу. Тому, для того щоб забронювати стіл, необхідно буде заповнити форму прив’язки платіжної карти. Після прив’язки з’явиться можливість обрати стіл, який клієнт бажає забронювати. Після вибору бажаного столу, з’явится форма з вибором часу, на який користувач бажає забронювати стіл. Після заповнення форми, з’явиться push-повідомлення про бронювання столу, яке буде неможливо прибрати з панелі повідомлень. За 5 хвилин до запланованного часу бронювання, нагадування прийде знову зі звуковим сигналом (рис.4.8).

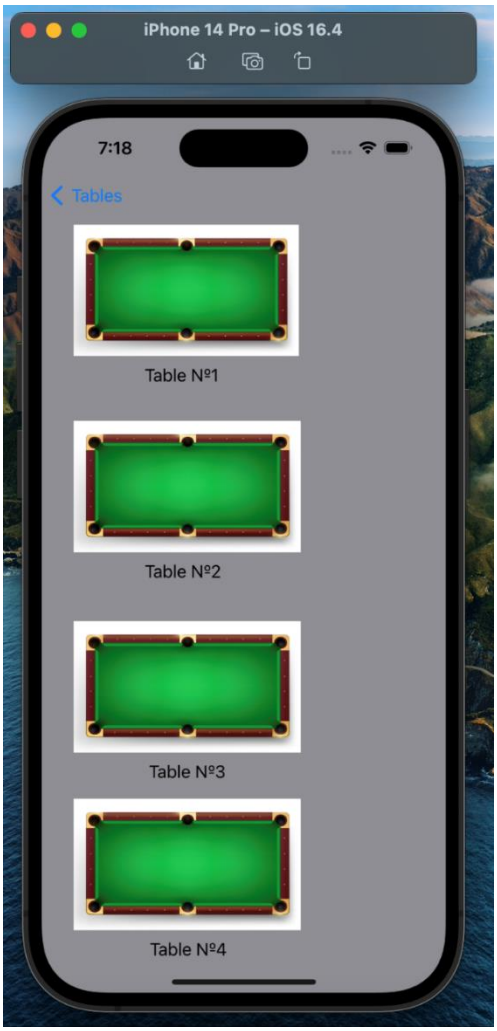

Рисунок 4.8 – Модуль бронювання столів

4.5 Модуль «Timer»

Функція таймер також стає доступною після створення профілю, та відображається у ньому. Таймер керується користувачем, за допомогою кнопок «Стоп», «Пауза» та «Старт». При початку підрахунку часу, приходить push-повідомлення, що інформує про початок рахунку. При своєчасній зупинці, інформація про підрахований час заноситься у БД. Якщо користувач забуває зупинити таймер, з плином однієї години, він отримує push-повідомлення, що інформує його про те, що він забув зупинити таймер і якщо за цей час була закінчена гра, то діалогове вікно запропонує користувачу видалити запис, або, якщо гра продовжується – продовжити підрахунок

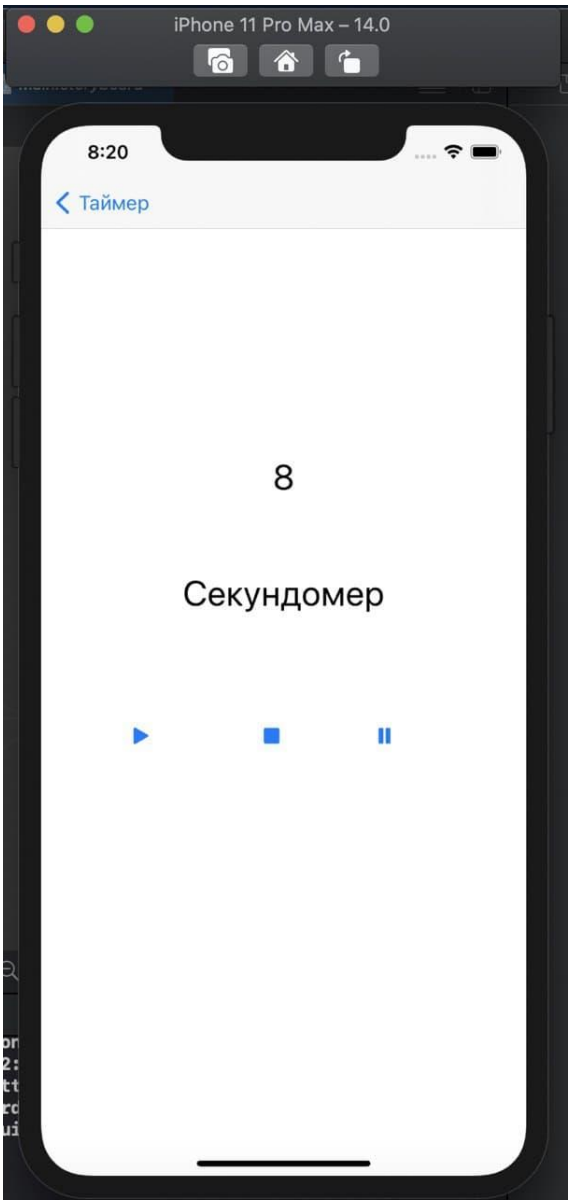

Рисунок 4.9– Модуль таймер

4.6 Модуль «Price»

Модуль ціни доступний і без створення профілю, адже не має прив’язки до клієнта. Модуль ціни відображає актуальні ціни на годину гри. Змінюється лише при зміні цін (див. рис.4.10).

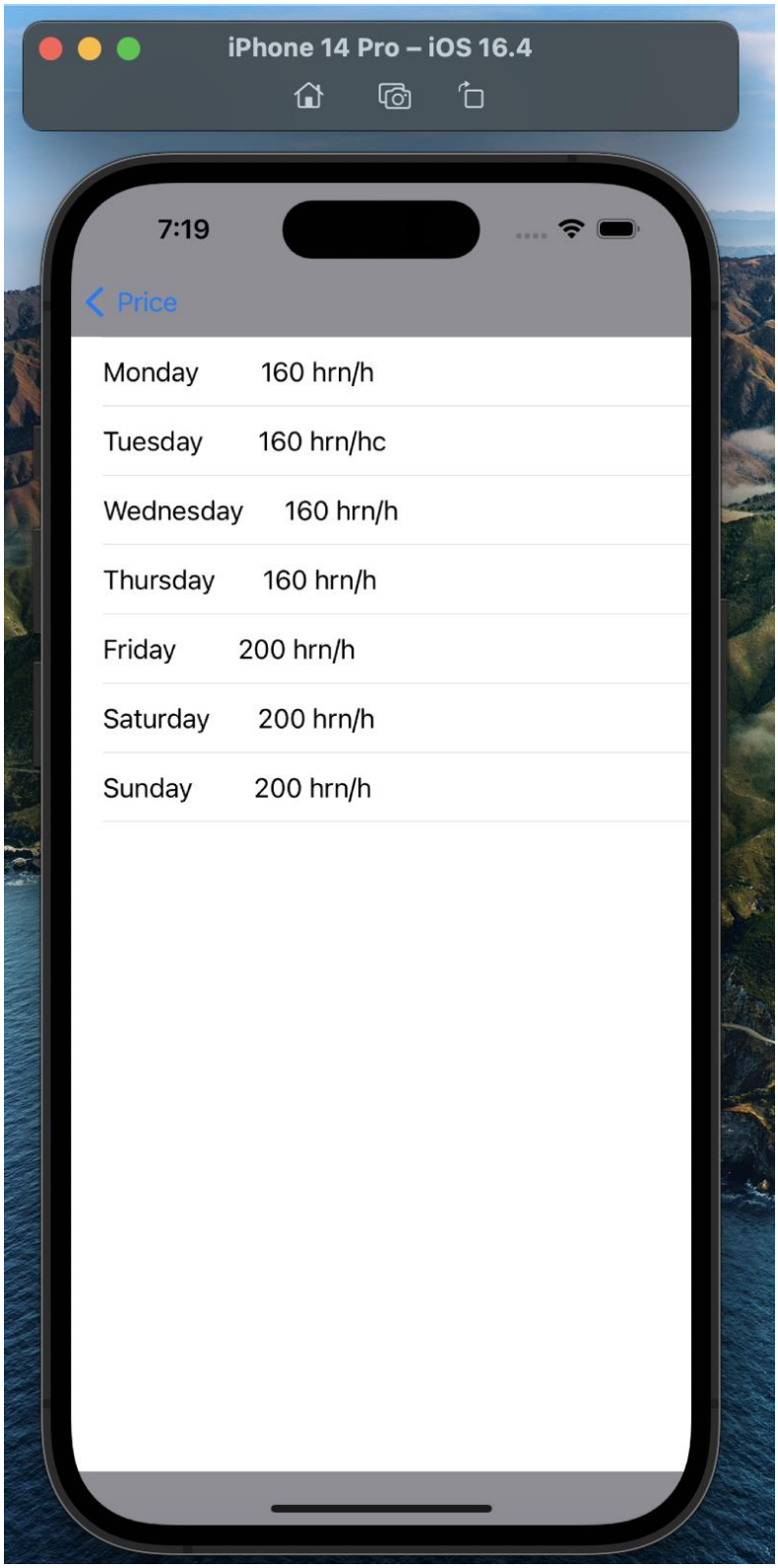

Рисунок 4.10 – Модуль «Price»

4.7 Модуль «Game»

Модуль «Game» доступний після створення профілю. Модуль дає змогу записатися на турніри, що відбуваються в клубі, як комерційні, так і спонсовані ФСБУ(Федерацією спортивного більярду України). Для запису на турнір необхідно заповнити форму і сплатити вступний внесок. Якщо користувач ще жодного разу не бронював столи, то буде видана форма прив’язки банківскої картки. Після сплати вступного внеску, користувача буде занесено до списку учасників. За день до турніру буде послано push-повідомлення про те, що користувача зписано на турнір. Якщо у користувача немає змоги прийти на турнір, він може відмінити запис, та повернути вступий внесок. Якщо користувач не відміняє запис, то за дві години до почтааку турніру, йому прийде ще одне push-повідомлення про проведення турніру (на даному етапі відмінити запис неможливо).

Модуль «Settings»

Модуль налаштування призначений для виходу з облікового запису, зміни розміру шрифту у додатку, а також змінити кольорву гамму при порушеннях сприйняття кольорів. При виборі пункту «порушення зору», буде видано вікно з вибором проблеми у клієнта, якщо такі вади є, то кольори інтерфейсу зміняться для комфортного користування додатком (див. рис 4.11).

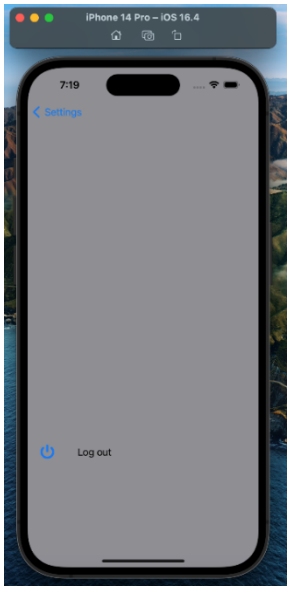

Рисунок 4.11 – Модуль налаштування

Модуль «About us»

Модуль з інформацією про клуб. Не потребує змін, тому несе інформацію, яку не можна змінювати. Представлено на рисунку 4.12

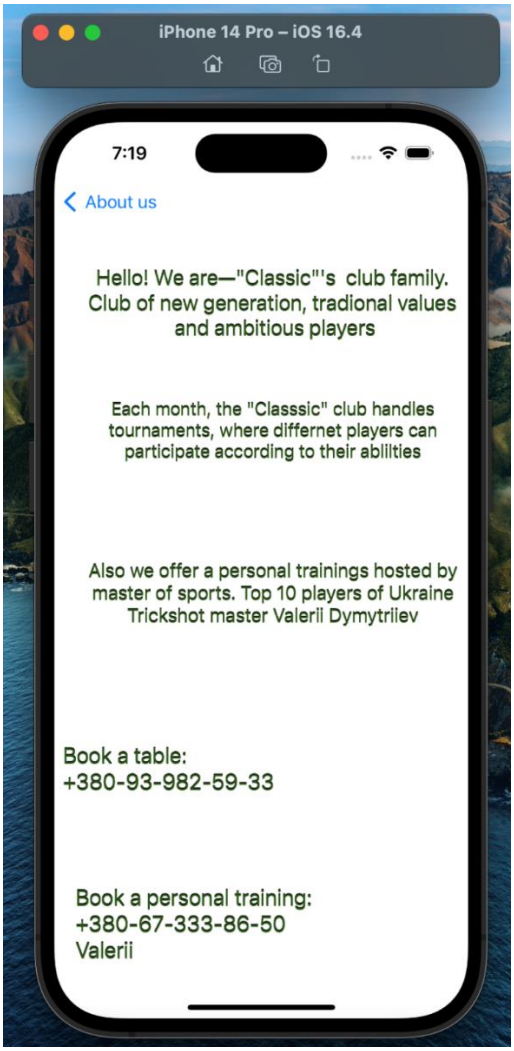

Рисунок 4.12 – Модуль «Про нас»

Всі вищеописані компоненти забезпечують стабільний і задовільний користувацький досвід. Вся інформація зібрана та розподілена по інтуітивно зрозумілому, мінімалістичному інтерфейсу.

Висновки до розділу 4

У четвертому розділі кваліфікаційної роботи представлено реалізацію мобільного застосунку, а саме розроблені вікна, поля, контроллери.
Представлено оптимізацію та додано підтримку додатку для пристроїв, що несуть на борту версії ОС iOS вище за 14.0.

ВИСНОВКИ

Під час написання кваліфікаційної роботи було досягнуто кількох ключових результатів, відповідно до поставлених завдань. По-перше, було розроблено інтерфейс додатку, що забезпечує зручне та ефективне відстеження часу гри. Інтерфейс було ретельно спроектовано з урахуванням зручності для користувачів, що дозволило зробити його інтуїтивно зрозумілим та легким у використанні. По-друге, імплементовано функціонал, який дозволяє отримувати актуальну інформацію про клуб та його події. Це включає можливість перегляду розкладу заходів, новин та спеціальних пропозицій клубу, що забезпечує користувачів актуальною та корисною інформацією.

Також створено можливість ведення статистики особистих ігор користувачів. Ця функція дозволяє гравцям відстежувати свої досягнення, аналізувати гру та покращувати свої навички, що робить додаток ще більш корисним та привабливим для постійних клієнтів. Окрім цього, було розроблено модуль для реєстрації та участі у турнірах. Цей модуль спрощує процес організації турнірів, дозволяючи користувачам швидко та легко реєструватися на змагання, а адміністраторам клубу – ефективно управляти подіями.

Завдяки оптимізації додатку, значно зменшено витрати часу та людських ресурсів, необхідних для організації подібних заходів. Оптимізація включала покращення продуктивності додатку, зниження навантаження на сервери та забезпечення швидкої та стабільної роботи системи. У результаті, додаток став не лише корисним інструментом для користувачів, але й ефективним засобом для управління клубом, що підвищило загальну ефективність роботи закладу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Види мобільних додатків: https://training.qatestlab.com/blog/technical-articles/types-of-mobile-applications/ (дата звернення 11.05.2021)

2. Документація Firebase URL: https://firebase.google.com/docs (дата звернення 20.05.2021)

3. Сайт розробників Microivest Billiard Pro URL: https://unipro.com.ua/ru/programma-dlya-bilyarda-microinvest-bilyard-pro/ (дата звернення 17.06.2021)

4. C.Kobryn, G.Booch, I.Jacobson, J.Rumbaugh UML Distilled: A Brief Guide to the Standard Object Modeling Language, Print2print, 2016. 192 с

5. J.Osis, U.Donins Topological UML Modeling: An Improved Approach for Domain Modeling and Software Development– Elsevier, 2017. – 234 с

6. Документація Swift URL: https://swiftbook.ru/ (дата звернення 11.05.2021)

7. Документація CocoaPods URL: https://guides.cocoapods.org/ (дата звернення 11.05.2021)

8. Документація XCode URL: https://developer.apple.com/documentation/(дата звернення 11.05.2021)

9. Ткачук К. Н., Халімовський М. О., Зацарний В. В. та ін. Основи охорони праці: Підручник. 2-ге вид., допов. і перероб. – К.: Основа, 2006. – 444 с.

10. Основи охорони праці: Підручник / За ред. проф. В.В.Березуцького – Х.: Факт, 2005. – 480 с.

11. Жидецький В.Ц., Джигирей В.С., Мельников О.В. Основи охорони праці. Навчальний посібник. – Вид. 4-те, доповнене. Львів: Афіша, 2000. – 350 с.

12. Econometrics and Data Science. 1st Ed. T.C.Nokeri; – Apress. 2022. – 228 с.

ДОДАТОК А – ЛІСТИНГ КОДУ ПРОГРАМИ

Код програми представлено у лістингах А.1 – А.14.

Лістинг А.1 – Код AppDelegate

```
import UIKit
import Firebase
@main
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?
    func application(_ application: UIApplication,
didFinishLaunchingWithOptions launchOptions:
[UIApplication.LaunchOptionsKey: Any]?) -> Bool {
        FirebaseApp.configure()
        return true
    }

    // MARK: UISceneSession Lifecycle

    func application(_ application: UIApplication,
configurationForConnecting connectingSceneSession: UISceneSession,
options: UIScene.ConnectionOptions) -> UISceneConfiguration {
        // Called when a new scene session is being created.
        // Use this method to select a configuration to create the
new scene with.
        return UISceneConfiguration(name: "Default Configuration",
sessionRole: connectingSceneSession.role)
    }

    func application(_ application: UIApplication,
didDiscardSceneSessions sceneSessions: Set<UISceneSession>) {
        // Called when the user discards a scene session.
        // If any sessions were discarded while the application was
not running, this will be called shortly after
application:didFinishLaunchingWithOptions.
        // Use this method to release any resources that were
specific to the discarded scenes, as they will not return.
    }
}
```

Лістинг А.2 – Код SceneDelegate

```
import UIKit
class SceneDelegate: UIResponder, UIWindowSceneDelegate {
    var window: UIWindow?
    func scene(_ scene: UIScene, willConnectTo session:
UISceneSession, options connectionOptions: UIScene.ConnectionOptions) {
        // Use this method to optionally configure and attach the
UIWindow window to the provided UIWindowScene scene.
        // If using a storyboard, the window property will
automatically be initialized and attached to the scene.
        // This delegate does not imply the connecting scene or
session are new (see application:configurationForConnectingSceneSession
instead).
        guard let _ = (scene as? UIWindowScene) else { return }
    }
```

```
        func sceneDidDisconnect(_ scene: UIScene) {
            // Called as the scene is being released by the system.
            // This occurs shortly after the scene enters the
background, or when its session is discarded.
            // Release any resources associated with this scene that can
be re-created the next time the scene connects.
            // The scene may re-connect later, as its session was not
necessarily discarded (see application:didDiscardSceneSessions instead).
        }
        func sceneDidBecomeActive(_ scene: UIScene) {
            // Called when the scene has moved from an inactive state to
an active state.
            // Use this method to restart any tasks that were paused (or
not yet started) when the scene was inactive.
        }
        func sceneWillResignActive(_ scene: UIScene) {
            // Called when the scene will move from an active state to
an inactive state.
            // This may occur due to temporary interruptions (ex. an
incoming phone call).
        }
        func sceneWillEnterForeground(_ scene: UIScene) {
            // Called as the scene transitions from the background to
the foreground.
            // Use this method to undo the changes made on entering the
background.
        }
        func sceneDidEnterBackground(_ scene: UIScene) {
            // Called as the scene transitions from the foreground to
the background.
            // Use this method to save data, release shared resources,
and store enough scene-specific state information
            // to restore the scene back to its current state.
        }
    }
```

Лістинг А.3 – Код StartViewController

```
    import UIKit
    import FirebaseAuth
    class StartViewController: UIViewController {
        // MARK: - Defining buttons action
        @IBAction func signInDidPressed(_ sender: Any) {
            self.performSegue(withIdentifier: "LogInSeuge", sender:
self)
        }
        @IBAction func SignUpDidPressed(_ sender: Any) {
            self.performSegue(withIdentifier: "SignUpSeuge", sender:
self)
        }
        override func viewDidLoad() {
            super.viewDidLoad()
            // Do any additional setup after loading the view.
        }
        // MARK: - already loggedIn
        override func viewDidAppear(_ animated: Bool) {
            super.viewDidAppear(animated)
            if Auth.auth().currentUser != nil{
```

```
                 self.performSegue(withIdentifier: "alreadyLoggedIn",
sender: nil)
             }
         }
         /*
         // MARK: - Navigation

         // In a storyboard-based application, you will often want to do
a little preparation before navigation
         override func prepare(for segue: UIStoryboardSegue, sender:
Any?) {
             // Get the new view controller using segue.destination.
             // Pass the selected object to the new view controller.
         }
         */
     }
```

Лістинг А.4 – Код LoginViewController

```
     import UIKit
     import FirebaseAuth
     class LoginViewController: UIViewController {
         // MARK: - Defining of Elements
         @IBOutlet weak var logInEmail: UITextField!
         @IBOutlet weak var logInPassword: UITextField!
         override func viewDidLoad() {
             super.viewDidLoad()
             // Do any additional setup after loading the view.
         }
         // MARK: - Logging in
         @IBAction func logInAction(_ sender: Any) {
             Auth.auth().signIn(withEmail: logInEmail.text!, password:
logInPassword.text!) { (user, error) in
           if error == nil{
             self.performSegue(withIdentifier: "loginToHome", sender:
self)
                           }
            else{
             let alertController = UIAlertController(title: "Error",
message: error?.localizedDescription, preferredStyle: .alert)
             let defaultAction = UIAlertAction(title: "OK", style:
.cancel, handler: nil)
             alertController.addAction(defaultAction)
             self.present(alertController, animated: true, completion:
nil)
                 }
       }
         }
         /*
         // MARK: - Navigation
         // In a storyboard-based application, you will often want to do
a little preparation before navigation
         override func prepare(for segue: UIStoryboardSegue, sender:
Any?) {
             // Get the new view controller using segue.destination.
             // Pass the selected object to the new view controller.
         }
         */
     }
```

Лістинг А.5 – Код SignUpViewController

```
import UIKit
import FirebaseAuth
class SignUpViewController: UIViewController {
    // MARK: - Defining elements
    @IBOutlet weak var emai: UITextField!
    @IBOutlet weak var password: UITextField!

    @IBOutlet weak var passwordConfirm: UITextField!
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view.
    }
    // MARK: - Sign Up operation
    @IBAction func SignUpAction(_ sender: Any) {
        if password.text != passwordConfirm.text {
        let alertController = UIAlertController(title: "Password
Incorrect", message: "Please re-type password", preferredStyle: .alert)
        let defaultAction = UIAlertAction(title: "OK", style:
.cancel, handler: nil)

        alertController.addAction(defaultAction)
        self.present(alertController, animated: true, completion:
nil)
                }
        else{
        Auth.auth().createUser(withEmail: emai.text!, password:
password.text!){ (user, error) in
         if error == nil {
           self.performSegue(withIdentifier: "signupToHome", sender:
self)
                            }
         else{
           let alertController = UIAlertController(title: "Error",
message: error?.localizedDescription, preferredStyle: .alert)
           let defaultAction = UIAlertAction(title: "OK", style:
.cancel, handler: nil)
            alertController.addAction(defaultAction)
            self.present(alertController, animated: true,
completion: nil)
                }
                    }
                }
    }
}
```

Лістинг А.6 – Код ViewController

```
import SideMenu
import UIKit
import FirebaseAuth

public var sideMenu: SideMenuNavigationController?
class ViewController: UIViewController, MenuControllerDelegate{
    // MARK: - Side Menu Initialization

    public let profileController = ProfileViewController()
```

```
        public let statisticController = StatisticViewController()
        public let tablesController = TablesViewController()
        public let timerController = TimerViewController()
        public let priceController = PriceViewController()
        public let gameController = GameViewController()
        public let settingsController = SettingsViewController()
        public let aboutUsController = AboutUsViewController()

        override func viewDidLoad() {
            super.viewDidLoad()
            let menu = MenuController(with: SideMenuItem.allCases)
            menu.delegate = self
            sideMenu = SideMenuNavigationController(rootViewController:
menu)

            sideMenu?.leftSide = true
            SideMenuManager.default.leftMenuNavigationController =
sideMenu
            SideMenuManager.default.addPanGestureToPresent(toView: view)
            addChildController()
        }
        // MARK: - Navigation Side Menu
        private func addChildController(){
            addChild(profileController)
            addChild(statisticController)
            addChild(tablesController)
            addChild(timerController)
            addChild(priceController)
            addChild(gameController)
            addChild(settingsController)
            addChild(aboutUsController)

            view.addSubview(profileController.view)
            view.addSubview(statisticController.view)
            view.addSubview(tablesController.view)
            view.addSubview(timerController.view)
            view.addSubview(priceController.view)
            view.addSubview(gameController.view)
            view.addSubview(settingsController.view)
            view.addSubview(aboutUsController.view)


            profileController.view.frame = view.bounds
            statisticController.view.frame = view.bounds
            tablesController.view.frame = view.bounds
            timerController.view.frame = view.bounds
            priceController.view.frame = view.bounds
            gameController.view.frame = view.bounds
            settingsController.view.frame = view.bounds
            aboutUsController.view.frame = view.bounds
            profileController.didMove(toParent: self)
            statisticController.didMove(toParent: self)
            tablesController.didMove(toParent: self)
            timerController.didMove(toParent: self)
            priceController.didMove(toParent: self)
            gameController.didMove(toParent: self)
            settingsController.didMove(toParent: self)
            aboutUsController.didMove(toParent: self)
```

```
            profileController.view.isHidden = true
            statisticController.view.isHidden = true
            tablesController.view.isHidden = true
            timerController.view.isHidden = true
            priceController.view.isHidden = true
            gameController.view.isHidden = true
            settingsController.view.isHidden = true
            aboutUsController.view.isHidden = true
        }
        @IBAction func didTapMenuButton(){
            present(sideMenu!,animated: true)
        }
        func didSelectMenuItem(named: SideMenuItem) {
            sideMenu?.dismiss(animated: true, completion: { [weak self]
in
                self?.title = named.rawValue
                switch named {
                case .profile:
                    self?.profileController.view.isHidden = false
                    self?.statisticController.view.isHidden = true
                    self?.tablesController.view.isHidden = true
                    self?.timerController.view.isHidden = true
                    self?.priceController.view.isHidden = true
                    self?.gameController.view.isHidden = true
                    self?.settingsController.view.isHidden = true
                    self?.aboutUsController.view.isHidden = true
                    self?.performSegue(withIdentifier: "ProfileLink",
sender: self)
                case .statistic:
     self?.profileController.view.isHidden = true
                    self?.statisticController.view.isHidden = false
                    self?.tablesController.view.isHidden = true
                    self?.timerController.view.isHidden = true
                    self?.priceController.view.isHidden = true
                    self?.gameController.view.isHidden = true
                    self?.settingsController.view.isHidden = true
                    self?.aboutUsController.view.isHidden = true
                    self?.performSegue(withIdentifier: "StatisticLink",
sender: self)
                case .tables:
                    self?.profileController.view.isHidden = true
                    self?.statisticController.view.isHidden = true
                    self?.tablesController.view.isHidden = false
                    self?.timerController.view.isHidden = true
                    self?.priceController.view.isHidden = true
                    self?.gameController.view.isHidden = true
                    self?.settingsController.view.isHidden = true
                    self?.aboutUsController.view.isHidden = true
                    self?.performSegue(withIdentifier: "TablesLink",
sender: self)
                case .timer:
                    self?.profileController.view.isHidden = true
                    self?.statisticController.view.isHidden = true
                    self?.tablesController.view.isHidden = true
                    self?.timerController.view.isHidden = false
                    self?.priceController.view.isHidden = true
                    self?.gameController.view.isHidden = true
                    self?.settingsController.view.isHidden = true
```

```
                    self?.aboutUsController.view.isHidden = true
                    self?.performSegue(withIdentifier: "TimerLink",
sender: self)
                case .price:
                    self?.profileController.view.isHidden = true
                    self?.statisticController.view.isHidden = true
                    self?.tablesController.view.isHidden = true
                    self?.timerController.view.isHidden = true
                    self?.priceController.view.isHidden = false
                    self?.gameController.view.isHidden = true
                    self?.settingsController.view.isHidden = true
                    self?.aboutUsController.view.isHidden = true
                    self?.performSegue(withIdentifier: "PriceLink",
sender: self)
                case .game:
                    self?.profileController.view.isHidden = true
                    self?.statisticController.view.isHidden = true
                    self?.tablesController.view.isHidden = true
                    self?.timerController.view.isHidden = true
                    self?.priceController.view.isHidden = true
                    self?.gameController.view.isHidden = false
                    self?.settingsController.view.isHidden = true
                    self?.aboutUsController.view.isHidden = true
                    self?.performSegue(withIdentifier: "GameLink",
sender: self)
                case .settings:
                    self?.profileController.view.isHidden = true
                    self?.statisticController.view.isHidden = true
                    self?.tablesController.view.isHidden = true
                    self?.timerController.view.isHidden = true
                    self?.priceController.view.isHidden = true
                    self?.gameController.view.isHidden = true
                    self?.settingsController.view.isHidden = false
                    self?.aboutUsController.view.isHidden = true
                    self?.performSegue(withIdentifier: "SettingsLink",
sender: self)
                case .aboutUs:
                    self?.profileController.view.isHidden = true
                    self?.statisticController.view.isHidden = true
                    self?.tablesController.view.isHidden = true
                    self?.timerController.view.isHidden = true
                    self?.priceController.view.isHidden = true
                    self?.gameController.view.isHidden = true
                    self?.settingsController.view.isHidden = true
                    self?.aboutUsController.view.isHidden = false
                    self?.performSegue(withIdentifier: "AboutUsLink",
sender: self)
                }
            })
        }
    }
    // MARK: - Menu Controller
    protocol MenuControllerDelegate{
        func didSelectMenuItem(named: SideMenuItem)
    }
    enum SideMenuItem: String, CaseIterable{
        case profile = "Профиль"
        case statistic = "Статистика"
```

```
        case tables = "Столы"
        case timer = "Таймер"
        case price = "Цены"
        case game = "Игра"
        case settings = "Настройки"
        case aboutUs = "О нас"
    }
    class MenuController: UITableViewController{
        public var delegate:MenuControllerDelegate?
        public let menuItems:[SideMenuItem]
        public let color = UIColor(red: 33/255.0, green: 33/255.0, blue:
33/255.0, alpha: 1)
        init(with menuItems: [SideMenuItem]){
            self.menuItems = menuItems
            super.init(nibName: nil, bundle: nil)
            tableView.register(UITableViewCell.self,
forCellReuseIdentifier: "cell")
        }

        required init?(coder: NSCoder) {
            fatalError("init(coder:) has not been implemented")
        }
        override func viewDidLoad() {
            super.viewDidLoad()
            tableView.backgroundColor = color
            view.backgroundColor = color
        }
        override func tableView(_ tableView: UITableView,
numberOfRowsInSection section: Int) -> Int {
            return menuItems.count
        }
        override func tableView(_ tableView: UITableView, cellForRowAt
indexPath: IndexPath) -> UITableViewCell {
            let cell = tableView.dequeueReusableCell(withIdentifier:
"cell", for: indexPath)
            cell.textLabel?.text = menuItems[indexPath.row].rawValue
            cell.textLabel?.textColor = .white
            cell.backgroundColor = color
            cell.contentView.backgroundColor = color
            return cell
        }
        override func tableView(_ tableView: UITableView, didSelectRowAt
indexPath: IndexPath) {
            tableView.deselectRow(at: indexPath, animated: true)
            let selectedItem = menuItems[indexPath.row]
            delegate?.didSelectMenuItem(named: selectedItem)
        }
    }
```

Лістинг А.7 – Код ProfileViewController

```
    import UIKit

    class ProfileViewController: UIViewController {

        override func viewDidLoad() {
            super.viewDidLoad()
            view.backgroundColor = .systemGray
        }
```

```
}
```

Лістинг А.8 – Код StatisticViewController

```
import UIKit
import Charts

class StatisticViewController: UIViewController {
    @IBOutlet weak var pieChart: PieChartView?

    @IBOutlet weak var iosStepper: UIStepper?
    @IBOutlet weak var macStepper: UIStepper?

    var iosDataEntry = PieChartDataEntry(value: 0)
    var macDataEntry = PieChartDataEntry(value: 0)
    var numberOfDownloadsDataEntries = [PieChartDataEntry]()
    override func viewDidLoad() {
        super.viewDidLoad()
        pieChart?.chartDescription?.text = ""
        iosDataEntry.value = iosStepper?.value ?? 0
        iosDataEntry.label = "Выиграл"
        macDataEntry.value = macStepper?.value ?? 0
        macDataEntry.label = "Проиграл"
        numberOfDownloadsDataEntries = [iosDataEntry, macDataEntry]
        updateChartData()
    }
    @IBAction func changeiOS(_ sender: UIStepper) {
        iosDataEntry.value = sender.value
        updateChartData()
    }
    @IBAction func changeMac(_ sender: UIStepper) {
        macDataEntry.value = sender.value
        updateChartData()
    }
    func updateChartData() {
        let chartDataSet = PieChartDataSet(entries:
numberOfDownloadsDataEntries, label: nil)
        let chartData = PieChartData(dataSet: chartDataSet)
        let colors = [UIColor(named:"macColor"),
UIColor(named:"iosColor")]
        chartDataSet.colors = colors as! [NSUIColor]
        pieChart?.data = chartData
    }
    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }
}
```

Лістинг А.9 – Код TablesViewController

```
import UIKit
class TablesViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
        view.backgroundColor = .systemGray
    }
}
```

Лістинг А.10 – Код TimerViewController

```
import UIKit
import Foundation
class TimerViewController: UIViewController {
    var timer = Timer()
    var timerDisplayed = 0

    @IBOutlet weak var Label: UILabel!
    @IBAction func startButton(_ sender: Any) {
        timer = Timer.scheduledTimer(timeInterval: 1, target: self,
selector: #selector(Action), userInfo: nil, repeats: true)
    }
    @IBAction func stopButton(_ sender: Any) {
        timer.invalidate()
        timerDisplayed = 0
        Label.text = "0"
    }
    @IBAction func pauseButton(_ sender: Any) {
        timer.invalidate()
    }

    @objc func Action(){
        timerDisplayed += 1
        Label.text = String(timerDisplayed)
    }
    override func viewDidLoad() {
        super.viewDidLoad()

        view.backgroundColor = .white
    }
}
```

Лістинг А.11 – Код PriceViewController

```
import UIKit

class PriceViewController: UIViewController{
    @IBOutlet var tableView: UITableView!
    let days = [
    "Понедельник    72 грн/час",
    "Вторник        72 грн/час",
    "Среда          72 грн/час",
    "Четверг        78 грн/час",
    "Пятница        78 грн/час",
    "Суббота        78 грн/час",
    "Воскресенье    78 грн/час"]
    override func viewDidLoad() {
        super.viewDidLoad()
        tableView?.delegate = self
        tableView?.dataSource = self
        view.backgroundColor = .systemGray
    }
}
extension PriceViewController: UITableViewDelegate{
    func tableView(_ tableView: UITableView, didSelectRowAt
indexPath: IndexPath) {
        print ("Такие у нас приятные цены")
```

```
        }
    }
    extension PriceViewController: UITableViewDataSource{
        func tableView(_ tableView: UITableView, numberOfRowsInSection
section: Int) -> Int {
            return days.count
        }
        func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {
            let priceCell =
tableView.dequeueReusableCell(withIdentifier: "priceCell", for: indexPath)
            priceCell.textLabel?.text = days [indexPath.row]
            return priceCell
        }
    }
```

Лістинг А.12 – Код GameViewController

```
    import UIKit

    class GameViewController: UIViewController {

        override func viewDidLoad() {
            super.viewDidLoad()

            view.backgroundColor = .white
        }

    }
```

Лістинг А.13 – Код SettingsViewController

```
    import UIKit
    import FirebaseAuth
    class SettingsViewController: UIViewController {
        @IBAction func logOutAction(_ sender: Any) {
            do {
                        try Auth.auth().signOut()
                    }
                catch let signOutError as NSError {
                        print ("Error signing out: %@", signOutError)
                    }

                    let storyboard = UIStoryboard(name: "Main", bundle:
nil)
                    let initial =
storyboard.instantiateInitialViewController()
                    UIApplication.shared.keyWindow?.rootViewController =
initial
        }

        override func viewDidLoad() {
            super.viewDidLoad()
            view.backgroundColor = .systemGray
        }
    }
```

Лістинг А.14 – Код AboutUsViewController

```
import UIKit

class AboutUsViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        view.backgroundColor = .white
        }
    }
```