Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка мобільного застосунку для обліку на шиншиловій фермі

Виконав: студент IV курсу, групи СНс-42
спеціальності 122 Комп'ютерні науки
(шифр і назва спеціальності)

_____ Сороківський О.В.
(підпис) (прізвище та ініціали)

Керівник _____ Назаревич О.Б.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Шимчук Г.В.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Рецензент _____ _____
(підпис) (прізвище та ініціали)

Тернопіль
2021

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«__» _____ 2021 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

Студенту Сороківському Олександру Вікторовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка мобільного застосунку для обліку на шиншиловій фермі

Керівник роботи Назаревич Олег Богданович, кандидат технічних наук, доцент кафедри КН
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «__» _____ 2021 року № 4/7-824

2. Термін подання студентом завершеної роботи

3. Вихідні дані до роботи Літературні та Інтернет-джерела

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ.

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

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи хорони праці	Гурик О.Я., доцент кафедри МТ		

7. Дата видачі завдання 7 червня 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи		*Виконано*
2.	Підбір джерел про бізнес ведення шиншил та		*Виконано*
	шиншилові ферми		
3.	Переклад та опрацювання джерел про шиншилові ферми		*Виконано*
4.	Виконання дослідження щодо вимог до застосунку для		*Виконано*
	шиншилової ферми		
5	Розроблення застосунку для шиншилової ферми		
5.	Оформлення розділу «Аналіз предметної області»		*Виконано*
6.	Оформлення розділу «Практичний розділ»		*Виконано*
7.	Виконання завдання до підрозділу «Безпека життєдіяльності»		*Виконано*
8.	Виконання завдання до підрозділу «Основи хорони		*Виконано*
	праці»		
9.	Оформлення кваліфікаційної роботи		*Виконано*
10.	Нормоконтроль		*Виконано*
11.	Перевірка на плагіат		*Виконано*
12.	Попередній захист кваліфікаційної роботи		*Виконано*
13.	Захист кваліфікаційної роботи		

Студент _____ Сороківський О.В
(підпис) (прізвище та ініціали)

Керівник роботи _____ Назаревич О.Б
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка мобільного застосунку для обліку на шиншиловій фермі // Кваліфікаційна робота освітнього рівня «Бакалавр» // Сороківський Олександр Вікторович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СНс-42 // Тернопіль, 2021 // С 51. , рис. – 21, табл. – 2, кресл. – 0, додат. – 2, бібліогр. – 30.

Ключові слова: МОБІЛЬНИЙ ЗАСТОСУНОК, PYTHON, FLUTTER, DART, POSTGRESQL, API, ШАБЛОНИ ООП.

Метою кваліфікаційної роботи є розробка мобільного застосунку для обліку на шиншиловій фермі. Під час розробки були використано такі технології як: Python для розробки серверної частини застосунку, Dart для розробки мобільної версії застосунку та PostgreSQL для роботи з базою даних.

Для реалізації серверу був використаний фреймворк для мови програмування Python під назвою Flask задача якого обробка запитів, які поступають на сервер, а також взаємодія з базою даних.

Для реалізації мобільного застосунку було використано фреймворк, для мови програмування Dart під назвою Flutter. За допомогою цього фреймворку було побудовано всі елементи інтерфейсу системи, а також реалізовано їх динамічне оновлення. Взаємодія між застосунком та сервером відбувається за допомогою API запитів застосунка до серверу.

За допомогою PostgreSQL та бібліотеки для мови програмування Python під назвою Psycopg2 було реалізовано взаємодію з базою даних на стороні серверу.

ANNOTATION

Mobile application development for chinchilla farm accounting // Qualification work of Bachelor educational degree // Sorokivskyi Oleksandr // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Computer Science Department, SNs-42 group // Ternopil, 2021 // Pages – 51, figures – 21, tables – 2, sketches – 0, addendums – 2, references – 30.

Keywords: mobile application, Python, Flutter, Dart, PostgreSQL, API, OOP patternts.

The purpose of the qualification work is to develop a mobile application for accounting on a chinchilla farm. During the development, these technologies were used: Python for server part development, Dart for development of the mobile application, and PostgreSQL to work with databases.

Python framework called Flask was used to implement the server, whose task is to process requests that are coming to the server and interacting with the database.

Dart framework called Flutter was used to implement the mobile application. With the help of this framework, all user interface elements were created and achieved their dynamic refreshment. Interaction between the application and the server was done via API requests from the application to the server.

Interaction between server and database was done via PostgreSQL and Python programming language library called Psycopg2.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Мобільний застосунок (також відомий як додаток) – програмне забезпечення, призначене для роботи на смартфонах, планшетах та інших мобільних пристроях.

ООП – об’єктно орієнтоване програмування.

БД – База даних.

API (англ. Application Programming Interface) – прикладний програмний інтерфейс.

JSON (англ. JavaScript Object Notation) – запис об’єктів JavaScript, текстовий формат обміну даними.

ПК – персональний комп’ютер.

SDK (англ. Software Development Kit) – набір із засобів розробки, утиліт і документації.

МП – мова програмування.

ЗМІСТ

ВСТУП

Актуальність теми. Внаслідок розвитку технологій розробки мобільних пристроїв і їх широкого поширення людство почало їх використовувати для автоматизації як і локальних процесів, таких як рутинні задачі до глобальних процесів, таких як боротьба з всесвітньою епідемією. Через те, що мобільні пристрої присутні у більшості людей автоматизація локальних процесів стає все популярнішою. Тому, для автоматизації процесу обліку на шиншиловій фермі була запропоновано розробку застосунку як інструмент цієї автоматизації.

Мета і задачі дослідження. Метою даної кваліфікаційної роботи освітнього рівня «Бакалавр» є розробка мобільного застосунку для обліку на шиншиловій фермі. Для досягнення поставленої задачі необхідно виконати наступні цілі:

- дослідити основні функції конкурентних рішень для того, щоб їх реалізувати у застосунку;
- дослідити доступні програмні засоби та інструменти для реалізації мобільного застосунку;
- проаналізувати можливі варіанти використання та архітектурні особливості системи;
- розробити мобільний застосунок для обліку на шиншиловій фермі;
- провести опис створених програмних рішень клієнтської та серверної частин;
- провалідувати і протестувати мобільний застосунок для обліку на шиншиловій фермі.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Визначення доцільності розробки

Розмноження тварин – це контрольоване розмноження свійських тварин для добування їжі, матеріалів тощо. Людство модифікувало свійських тварин з давніх часів для того, щоб вони краще підпадали під людські потреби. Вибіркове розмноження включає використання знань з багатьох наукових галузей. Одні з найважливіших це генетика, статистика, репродуктивна фізіологія, комп’ютерні науки і молекулярна генетика.

З часом, розмноження тварин почало набувати економічного характеру. Однією з перших тварин, яких почали вирощувати і продавати не за харчову цінність, а за матеріальну – шиншили. Існує два види шиншил: довгохвоста і куцохвоста. Торгівля хутром шиншил розпочалось в шістнадцятому столітті.

Їх хутро набуло популярності через надзвичайну м'якість. Цю м’якість шиншилам забезпечує її волосяний покрив, з кожної цибулини якого, в середньому, росте шістдесят волосків. Зазвичай колір хутра є однорідним, що дозволяє робити з нього як і дрібний одяг, так і більшого розміру. Лише одне довге пально, яке зроблене з хутра шиншил, потребує не менше ніж 150 шкір тварин, через те, що шиншили є невеликими. Через це, люди почали полювати на шиншил і тим самим спричинили вимирання цілого їх видно, а інші два – під загрозою вимирання. На даний момент полювання на диких шиншил є нелегальним, але дикі шиншили досі знаходять на межі вимирання через браконьєрство.

В середньому, бізнес, який базується на розведенні шиншил є прибутковим при належному догляді. Дослідження показує, що вирощування шиншил прибуткове методом статистичної оцінки плодючості одного самця. На таблиці 1.1 зображено результати спостережень.

Таблиця 1.1 – Показники народжуваності шиншил

	Середнє	Середньоквадратичне відхилення	Медіана
Загальна кількість виводків одного самця	25.2	21.1	19
Загальна кількість виводків від одного самця, яких було відселено	23	19.4	18
Загальна кількість нащадків народжених від одного самця	47.2	40.8	34.5
Загальна кількість нащадків від одного самця, яких було відселено	39.4	34.2	29.5
Кількість народжених на виводок	1.87	0.81	2
Кількість відселених на виводок	1.56	0.83	2
Тривалість репродуктивності особи (днів)	1927	984	1716
Процент вирощених нащадків з народжених	83.5	11.4	85.1

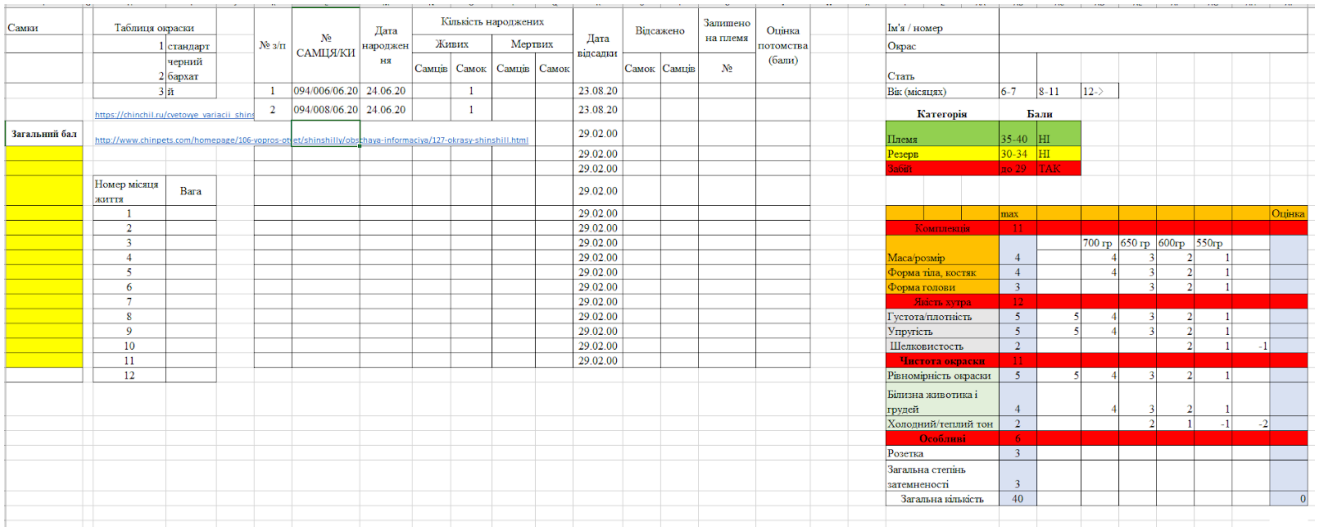
Дані для таблиці збирались протягом п’яти років. Протягом цього часу віл одного самця було отримано більше двадцяти п’яти виводків або ж сорока семи осіб, в середньому. Більшість самців були готові до репродукції починаючи з семи до шістнадцяти місяців. Отже, виходячи з цих даних бізнес розведення шиншил можна розцінювати прибутковим.

Розведення шиншил потребує строгих температурних умов. Загалом, шиншили здатні витримати холод, але є дуже чутливими до тепла. Температура навколишнього середовища шиншил повинна коливатись від вісімнадцяти градусів цельсію до двадцяти семи. При високій вогкості вищі температури можуть спричинити тепловий удар для тварини. Тому, найкращий варіант це слідкувати і за температурою і за вогкістю приміщення. Клітки шиншил є з дном сіткою та суцільним. Також, розведення шиншил потребує строгої дієти, адже якість хутра залежить від їжі і її нормах.

Через прибутковість бізнесу, в світі з'являється все більше ферм шиншил. В середньому, щоб почати такий бізнес потрібно закупити п’ять сімей тварин. Одна сім’я складається з чотирьох самок та одного самця. В результаті власник отримає двадцять п’ять осіб. Так, як їх розведення потребує строгої дисципліни виникає потреба в документуванні всіх подій, які пов'язані з фермою.

1.2 Аналіз конкурентних рішень

Найпопулярніше рішення серед фермерів, кількість шиншил в яких не перевищує двісті це Google Spreadsheets. Google Spreadsheets – це веб ресурс призначений для роботи з таблицями. Приклад інтерфейсу ресурсу зображений на рисунку 1.1



Самки	Таблиця окраски		№ з/п	№ САМЦЯ/КІ	Дата народження	Кількість народжених				Дата відсадки	Відсажено		Залишено на племя	Оцінка потомства (бали)
						Живих		Мертвих						
						Самців	Самок	Самців	Самок		Самок	Самців	№	
	1	стандарт												
	2	черний бархат												
	3	й	1	094/006/06.20	24.06.20		1			23.08.20				
	https://chinchil.ru/cvetovye_variacii_shinshill/		2	094/008/06.20	24.06.20		1			23.08.20				
Загальний бал	http://www.chinpets.com/homepage/106-vopros-otvet/shinshilly/obshhaya-informaciya/127-okrasy-shinshill.html									29.02.00				
										29.02.00				
										29.02.00				
	Номер місяця життя	Вага								29.02.00				
	1									29.02.00				
	2									29.02.00				
	3									29.02.00				
	4									29.02.00				
	5									29.02.00				
	6									29.02.00				
	7									29.02.00				
	8									29.02.00				
	9									29.02.00				
	10									29.02.00				
	11									29.02.00				
	12													

Ім'я / номер			
Окрас			
Стать			
Вік (місяцях)	6-7	8-11	12->

Категорія	Бали	
Племя	35-40	НІ
Резерв	30-34	НІ
Забій	до 29	ТАК

	max						Оцінка
Комплекція	11						
			700 гр	650 гр	600гр	550гр	
Маса/розмір	4		4	3	2	1	
Форма тіла, костяк	4		4	3	2	1	
Форма голови	3			3	2	1	
Якість хутра	12						
Густота/плотність	5	5	4	3	2	1	
Упругість	5	5	4	3	2	1	
Шелковистость	2				2	1	-1
Чистота окраски	11						
Рівномірність окраски	5	5	4	3	2	1	
Білизна животика і грудей	4		4	3	2	1	
Холодний/теплий тон	2			2	1	-1	-2
Особливі	6						
Розетка	3						
Загальна степінь затемненості	3						
Загальна кількість	40						0

Рисунок 1.1 – Приклад використовування Google Spreadsheets як засіб обліку шиншил

Переваги використання такого інструменту:

1. Портативність. Так, як таблиці знаходять в хмарному сховищі доступ можна отримати з любого пристрою, який має доступ в Інтернет.

2. Ресурсоефективність. Ресурси пристрою, на якому переглядається чи редагується документ майже не задіюються.

3. Ціна. Враховуючи, що компанія Google забезпечує до п’яти гігабайт хмарного сховища безкоштовно, для ферм до двісті осіб цей інструмент буде безкоштовним.

Недоліки використання:

1. Складність в налаштуванні. Налаштування повноцінного процесу обліку в таблицях це довгий і важкий процес, який потребує фундаментальних знань як і в області розведення шиншил так і в галузі комп’ютерних наук. Не всі фермери початківці зможуть налаштувати таку систему без сторонньої допомоги спеціалістів.

2. Низька масштабованість. Зі збільшенням популяції шиншил збільшується і час на внесення даних про них.

Ще одним рішенням є програма “Szynszyle”. Ця програма була створена польською асоціацією шиншиловодів близько двадцяти років назад, але досі

користується популярністю в Польщі. Програма підтримується операційними системами починаючи від Windows 95 та закінчуючи поточною версією Windows – Windows 10. До функцій програми належать:

- реєстрація самців та самок;
- реєстрація сімей;
- архів.

Приклад інтерфейсу програми зображений на рисунку 1.2.

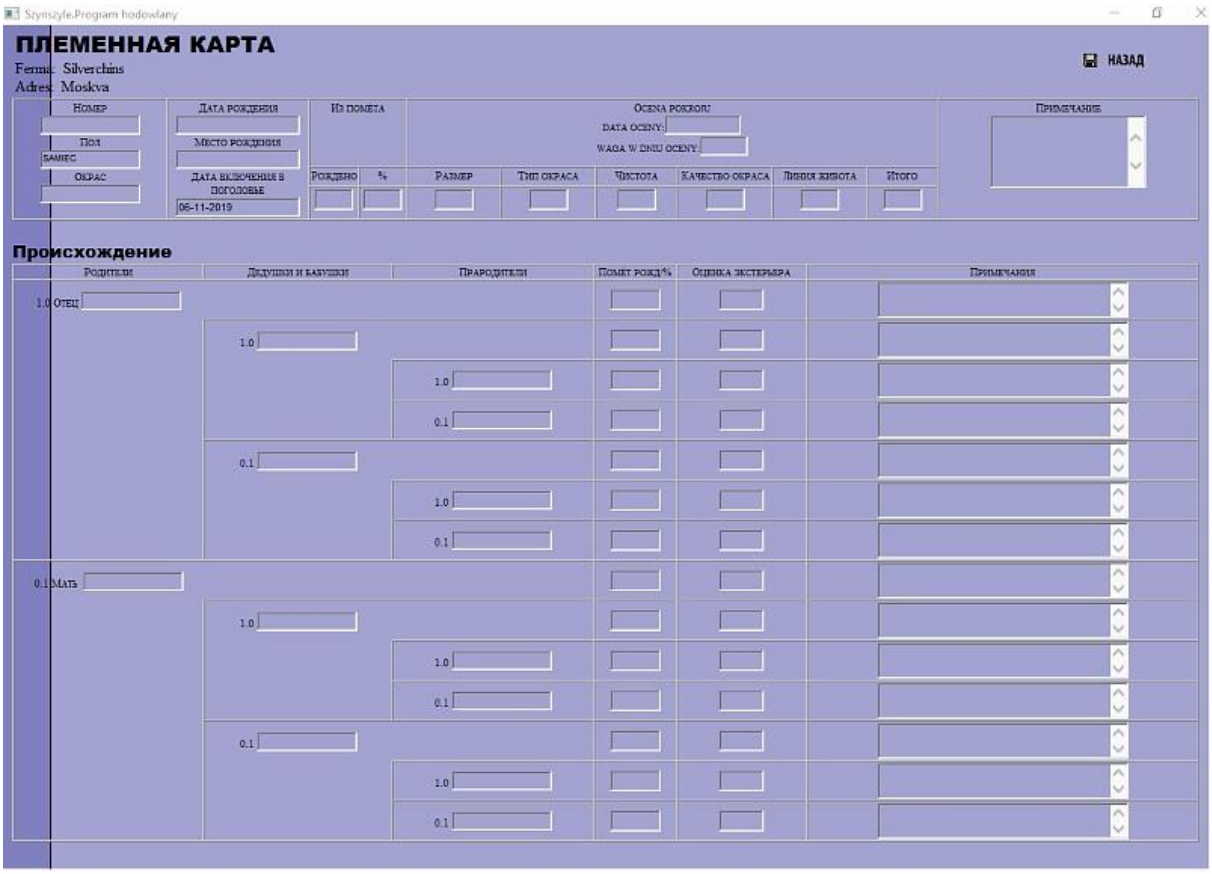


Рисунок 1.2 – Інтерфейс програми

Переваги програми:

1. Ціна. Програма безкоштовна і підтримується лише силами фермерів.

2. Простота у використанні. В програмі зручно представлені лише найнеобхідніші поля, що не ускладнює інтерфейс.

3. Кросплатформеність. Програма підтримується як і старими версіями системи Windows так і найсучаснішими.

Недоліки:

1. Локалізація. Програма підтримує лише дві мови – російську і польську.

2. Кросплатформеність. Програма працює лише на Windows, що ускладнює роботу з програмою. У випадку, коли фермеру потрібно внести в програму не покидаючи ферми, він не зможе цього зробити.

3. Безпека. У разі поломки жорсткого диску ПК буде неможливо відновити дані, внесені в програму.

Однією з програм для розведення не лише шиншил, а й решта тварин є “ZooEasy”. Ця програма містить більше функцій чим попередні інструменти. Вона включає такі функції:

- додавання, видалення, копіювання та редагування інформації про особу;
- додавання фотографії особи;
- вкладка для внесення замірів тварин;
- функції для складання звітів тощо.

Ліцензування цієї програми працює по підписці. Види підписок, а також їх функції представлені в таблиці 1.2.

Таблиця 1.2 – Тарифні плани “ZooEasy”

Назва підписки	Назавжди безкоштовна	Єдиний користувач	Мульти-користувач	Безлімітна
Ціна в євро	0	5	17	270

Продовження таблиці 1.2

Назва підписки	Назавжди безкоштовна	Єдиний користувач	Мульти-користувач	Безлімітна
Кількість користувачів	1	1 (або невелика організація)	2 або більше, Організація середнього розміру	Велика організація
Ліміт осіб тварин	25	Безлімітно	Безлімітно	Безлімітно
Кількість користувачів типу “Менеджмент”	1	1	2+ (додаткові користувачі платні)	Безлімітно
Кількість користувачів типу “Селекціонер”	0	0	0+ (додаткові користувачі платні)	Безлімітно
Кількість користувачів типу “Читач”	0	0	0+ (додаткові користувачі платні)	Безлімітно
Імпорт даних	-	+	+	+
Підтримка	+	+	+	+

Сама програма набагато функціональніший чим її попередні аналоги. Вигляд інтерфейсу програми зображений на рисунку 1.3.

Рисунок 1.3 – Вигляд програми “ZooEasy”

Переваги програми:

1. Широкий функціонал. Програма містить безліч корисних функцій.

2. Види інформації для внесення. Інструмент налічує понад 20 різних характеристик про кожну особі.

3. Легка в освоєнні. Незважаючи на багато можливостей програма не містить складних чи нагромаджених елементів інтерфейсу.

4. Безплатна версія. Містить безкоштовний тип підписки для ознайомлення з програмою.

Недоліки програми:

1. Безпека. При поломці ПК, а саме пристрою зберігання інформації, всі дані втрачаються і потрібно буде вносити інформацію знову.

2. Зав’язаність на єдиній платформі. Програма лише доступна для операційної системи Windows і не призначена для мобільних пристроїв чи інших ОС.

3. Ціна. Програма не є безкоштовною, хоть і не потребує додаткових ресурсів зі сторони розробника. Безкоштовна версія розроблена лише для ознайомлення з програмою.

Інструмент Agritec – це також засіб для обліку, статистики та звітності по розведенню тварин. Основною відмінністю Agritec є те, що він кросплатформений. Приклад інтерфейсу програми зображений на рисунку 1.4

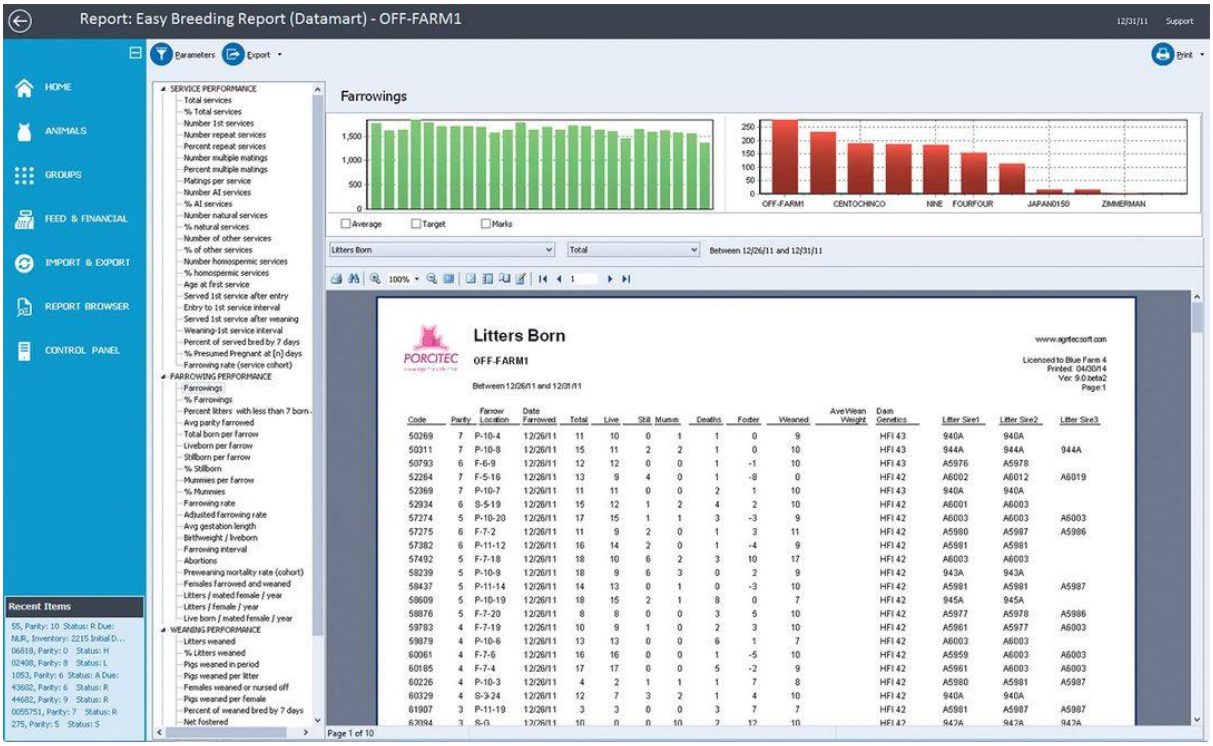


Рисунок 1.4 Інтерфейс Agritec

Переваги Agritec:

1. Кросплатформеність. Програма підтримує найпопулярніші платформи, а також, містить веб-версію.

2. Багатофункціональність. Програма надає широкий спектр статистичних інструментів та засобів для внесення інформації

3. Масштабованість. Agritec підтримує збереження інформації починаючи від п’ятсот і до двадцяти тисяч осіб.

4. Безкоштовна версія. Застосунок містить безкоштовну версію з базовим функціоналом.

Недоліки:

1. Не підходить для фермерів початківців. Ця програма не призначена для фермерів початківців, в яких є до 500 осіб. Більшість функцій застосунку не буде використовуватись в такому випадку.

2. Ціна. Застосунок хоть і містить безкоштовну версію, але її функціонал обмежений. До того ж, ця опція не працює для ферм, в яких більше п’ятсоти осіб.

1.3 Призначення мобільного додатку

Розробка мобільного застосунку для шиншилової ферми призначена для допомоги фермерам в обліку тварин, а також, отриманні базової статистики.

Застосунок реалізує рівень інтерфейсу між клієнтом (фермером) та серверною частиною в трирівневій клієнт-серверній архітектурі.

Таким чином застосунок дозволить спростити та пришвидшити роботу з даними ферми, а також забезпечити цілісність даних.

1.4 Вимоги до розробки мобільного застосунку

Проаналізувавши предметну область можна сформувати загальні вимоги до розробки застосунку:

1. Вимоги до інтерфейсу:

- інтерфейс має бути простим і зрозумілим користувачам любої вікової категорії;
- інтерфейс повинен складатись зі знайомих компонентів;
- розробка мобільного застосунку виконується під Android, мінімальна версія SDK – 23.

2. Вимоги до доступу та захисту:

- доступ до аутентифікаційних даних повинен мати лише сервер;
- вхід користувача захищений паролем;
- дані для входу передаються лише раз під час аутентифікації користувача. Наступні дії користувача можливі лише за наявності аутентифікаційного токену;
- пароль при введенні не повинен відображатись.

3. Вимоги до функціональних можливостей:

- можливість створення нового користувача додатком;
- можливість додавання інформації про шиншилу;
- можливість редагування та видалення інформації про шиншилу;
- можливість працювати з додатком без підключення до інтернету;
- можливість отримання даних за допомогою сканування QR коду.

1.5 Пошук актантів та варіантів використання

На рисунку 1.5 зображена діаграма варіантів використання з акторами та прецедентами, що виникають у розробленій системі.

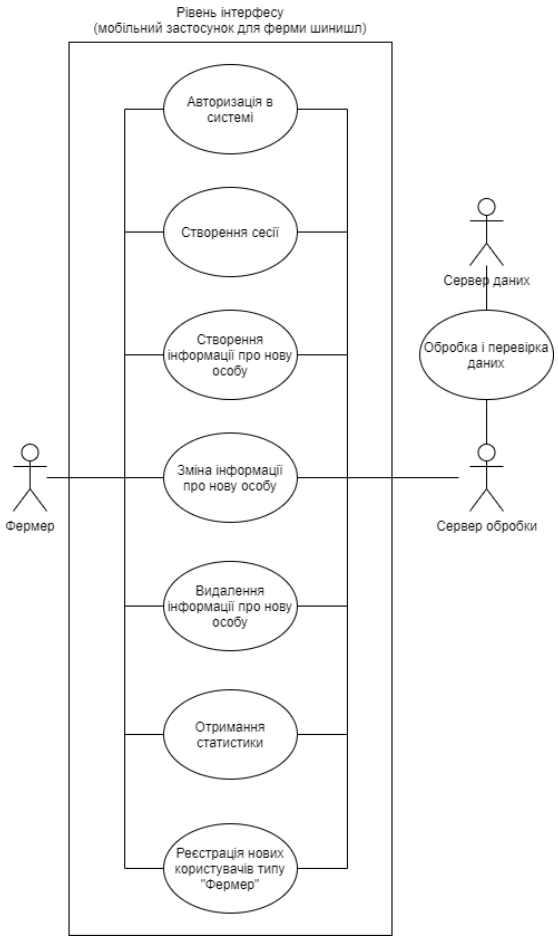


Рисунок 1.5 – Діаграма використання системи мобільного застосунку

В системі виділяються три основних актори – Фермер (актор, що використовує застосунок), Сервер обробки (актор, який обробляє запити та дані, які поступають на сервер) та Сервер даних (актор, що отримує та надає дані для серверу обробки)

Прецедентами в даній схемі – це вимоги до застосунку, тобто, основні дії, які може виконати актор під час користування додатку.

Через те, що архітектура системи є клієнт-серверною, то для зображення ролей рівнів обробки та даних з якими взаємодіє Фермер через додаток було використано акторів.

Обробка запитів та даних, які надходять від Фермера відбуваються на Сервері оброки. Після отримання даних Сервер обробки обмінюється інформацією з Сервером даних.

РОЗДІЛ 2. ПРАКТИЧНИЙ РОЗДІЛ

2.1 Технології, які використовувались для розробки

Для того, щоб додаток був кросплатформенним використано фреймворк “Flutter”. Ця розробка від компанії Google славиться своєю кросплатформеністю. За допомогою цього фреймворку та мови програмування Dart можна розробляти програми для найпопулярніших операційних систем для ПК, застосунки на IOS та Android та веб-сторінки.

Загалом, фреймворк “Flutter” – це більше як frontend частина розробки, що заставляє задуматись над сервером обробки запитів та даних. Для реалізації цього серверу була використана мова програмування Python та фреймворк Flask. Ця мова програмування відома своєю простотою у вивченні та великою кількість бібліотек, що значно пришвидшує розробку програм.

Фреймворк Flask призначений для створення HTTP серверів, які працюють по REST протоколу.

Отож, в реалізації даного застосунку всі дані надсилаються та отримуються по протоколу HTTP між сервером, який написаний з використанням фреймворку Flask та клієнтською частиною, яка написана на мові програмування Dart.

Як основна база даних, в якій зберігаються всі дані використовується PostgresQL. Її було обрано через її нереляційний тип, а також через швидкість роботи.

2.2 Проектування архітектури мобільного застосунку для ферми шиншил

Для того, щоб реалізувати всі потрібні функції застосунку, він повинен містити декілька екранів для того, щоб не перевантажувати інтерфейс елементами. Для цього було розроблено наступні екрани:

- екран авторизації, за допомогою якого користувач проходить аутентифікацію;
- екран, на якому розміщені основні елементи роботи з інформацією про шиншил. На цьому екрані користувач може додати інформацію про нову особу, може обрати особу для перегляду і редагуванню інформації про неї;
- екран для сканування QR коду для швидшого отримання інформації про особу;
- екран, на якому розміщена основна інформація про особу, а також інструменти для її редагування;
- екран для генерації QR коду особи;
- екран для додавання інформації про нову особу;
- екран статистики, на якому зображено базову статистику ферми.

Для реалізації цих екранів була використана навігаційна архітектура. Фреймворк Flutter має вбудовані інструменти для створення і підтримки такої архітектури. Основними інструментами є Navigator та Route. Ці два класи створенні для простої, але фукнціональної реалізації переходів між екранами проекту. Дана реалізація застосунку не передбачає використання Route, через те, що застосунок не містить складної логіки переходів між екранами. Граф можливих зв’язків між екранами зображений на рисунку 2.1

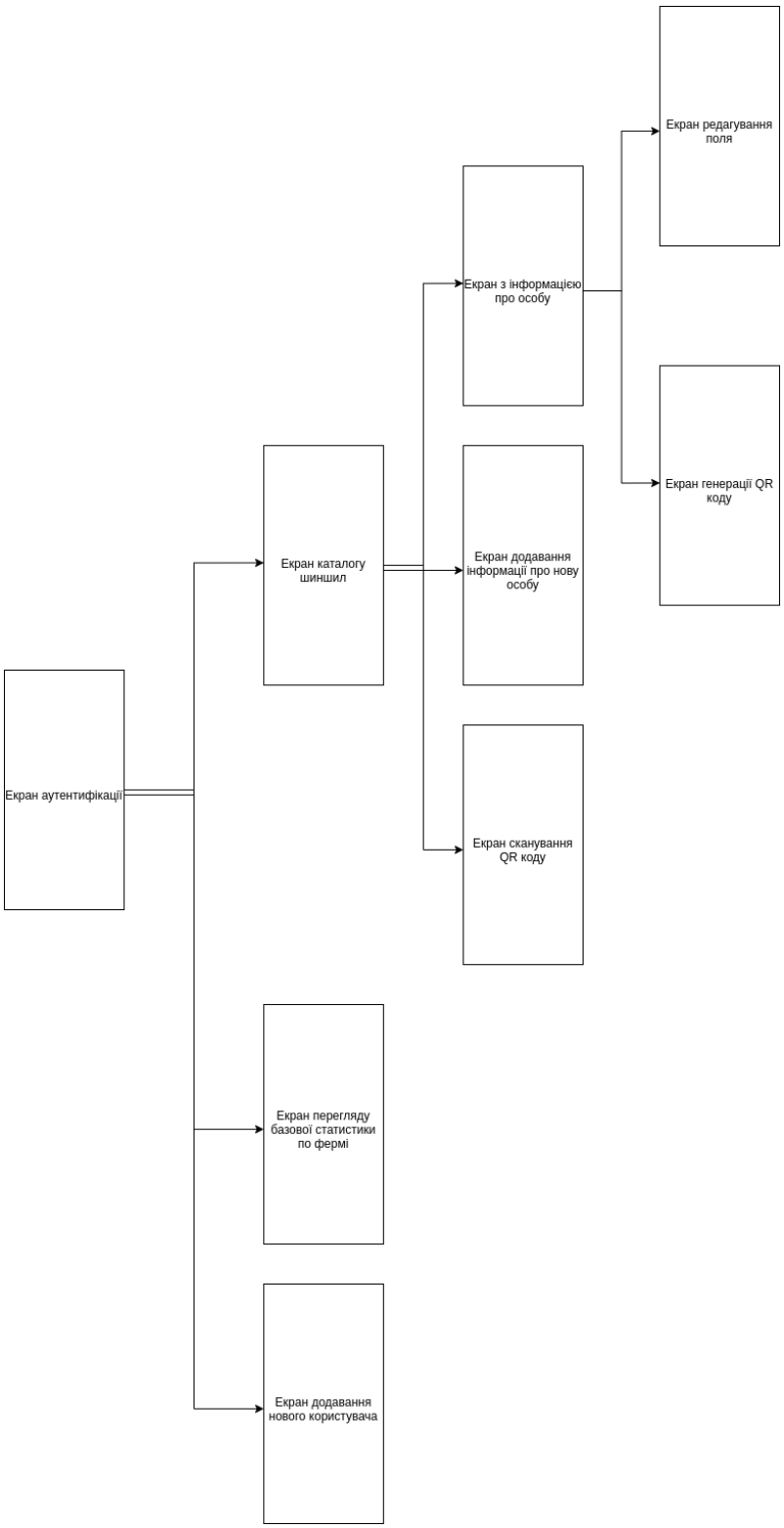


Рисунок 2.1 – Граф можливих зв’язків між екранами

На графі схематично зображені екрани програми і шляхи можливих переходів.

При запуску застосунку відкривається екран аутентифікації. Це зроблено для того, щоб не авторизовані користувачі не могли ніяким чином використати застосунок.

Після авторизації користувач попадає на екран каталогу шиншил. З цього екрану можна переміститись на екрани, які зв’язані з редагуванням чи додаванням інформації про шиншил.

Внизу екрану з інформацією про шиншил розміщене меню з переходами на екран статистики та на екран додавання нового користувача.

На екран додавання інформації про нову особу можна перейти з екрану інформації про шиншил. Таким ж чином здіснюється перехід на екран сканування QR коду та на екран отримання інформації про особу.

Екран з інформацією про особу містить в собі переходи на екран редагування конкретного поля та на екран генерації QR коду для обнарої особи.

2.3 Опис програмних рішень

Зв’язок застосунку з сервером було розроблено розроблено як систему API запитів зі сторони застосунку до сервера, адже застосунок щільно взаємодіє з сервером.

2.3.1 Опис програмних рішень на стороні сервера

Основний функціонал класу розділений на декілька модулів. Це зроблено для того, щоб умовно розділити логіку по різних файлах. Сервер містить чотири основні модулі:

- модуль роботи з базою даних;
- модуль обробки запитів;
- модуль допоміжних функцій;

- модуль конфігурації.

Модуль для роботи з базою даних реалізований в класі DbWorker. Під час ініціалізації модулю відбувається перевірка на наявність створеної бази і схеми таблиці. Під час налаштування сервера адміністратор повинен створити базу, а також користувача для неї з правами суперкористувача для подальшої роботи програми. Після перевірки підготовлених даних програмою вона перевіряє чи існує потрібна їй таблиця. Якщо таблиці немає, то програма її автоматично створює. Доступи до бази реалізовано в наступних методах цього класу:

- add_profile – записує інформацію про нову особу;
- remove_profile – видаля інформацію про особу з бази;
- update_profile – отримує оновлену інформацію про особу та заносить її в базу;
- get_data(records=False) – повертає інформацію про всіх осіб в базі. Параметр records вказує на форматування відповіді;
- get_fields – повертає інформацію про всі поля таблиці. Використовується для автоматичного додавання полів в застосунку;
- get_data_by_id – повертає інформацію про певну особу;
- get_data_by_family_id – повертає інформацію про сім’ю по її номеру.

Модуль обробки запитів – це модуль, який відповідає за обробку API запитів. Кожен з його методів загортається в один з готових декораторів фреймворку Flask. В цьому модулі було використано лише декоратор Route, який відповідає за реєстрацію відповідної функції за відповідним URL. Реалізація Route включає в собі підтримку таких методів як GET, POST, PUT, DELETE тощо. Також за допомогою спеціальної шаблонної мови можна зазначати параметри з якими викликається функція обробки та їх типи.

Результати серверних функції діляться на два типи:

- повернення відповіді у форматі json. Це відбувається за допомогою функції jsonify;
- повернення відповіді у вигляді HTTP коду. Для цього використовується клас Response, який для ініціалізації потребує задання параметра status та необов’язкове задання параметру response.

Функції модулю обробки даних можна розділити на такі:

- функції для роботи з інформацією про осіб;
- функції для авторизації користувачів;
- функції для створення статистики.

Функції для роботи з інформацією містять в собі валідацію та обробку інформації. Для перевірки оновлень даних використовується система ідентифікаторів оновлення. Кожен підключений користувач надсилає запити на отримання ідентифікатора поточного оновлення. До функцій роботи з інформацією належать:

- get_update_id – приймає GET запит та повертає поточний ідентифікатор оновлення. Якщо при звірці ідентифікаторів вони є різними, то відбувається запит за адресою “/get”, обробником якої є фукнція send_info, що повертає інформацію про всіх осіб;
- add_info – використовується для додання інформації про нову особу. Перед викликом функції занесення інформації про нову особу в базу даних створюється параметр in_family_id який відповідає за ідентифікатор шиншили в сім’ї;
- update_info – відповідає за оновлення даних про шиншилу в системі;
- send_info – функція, яка обробляє запити на отримання всієї інформації по шиншилах;
- send_fields – функція, яка повертає інформацію про всі поля таблиці з інформацією про шиншил;

- upload_profile_image – функція, яка обробляє запити в яких надсилається зображення шиншили. Під час обробки функція конвертує закодоване зображення в base64 формат та обраховує md5 хеш для кожного зображення для подальшої звірки на оновлення;
- get_profile_image_hash – функція, яка повертає хеш зображення певної шиншили;
- download_profile_image – надсилає зображення профілю шиншили з серверу, закодовуючи його у base64 формат.

Функції для авторизації реалізують базову логіку авторизації користувачів з верифікацією токену. До них входять:

- create_user – додає нового користувача до системи;
- login_user – реалізує процес авторизації користувача. При правильні комбінації логіну і паролю повертає токен автентифікації;
- unlogin_user – видаляє токен автентифікації користувача

Функції для побудови статистики використовують бібліотеку Pandas. Для побудови статистики створені такі функції:

- get_born_stats – повертає статистику по народжуваності за останній місяць;
- get_most_profitable – повертає інформацію про сімей, в яких найбільше новонароджених за останній місяць;
- get_jigged_count – повертає інформацію про кількість відстаджених за останній місяць.

Модуль допоміжних функцій містить собі функції, які не належать до конкретної теми, а є лише допоміжними. До таких належать:

- calculate_in_family_id – призначена для обрахунку нового ідентифікатора в певній сім’ї;
- create_id – реалізує створення фермерського ідентифікатора шиншили. Такий ідентифікатор відрізняється від звичайного, оскільки він базується на сім’ї, даті народження, статі тощо.

Модуль конфігурації містить в собі основні конфігурації бази даних.

2.3.2 Опис програмних рішень на стороні мобільного застосунку

Розробка мобільного застосунку проводилась з використанням мови програмування Dart та її фреймворком Flutter. Для кожного застосунку існує декілька типів файлів для його роботи – файли логіки програми, файли ресурсів та файли конфігурацій. Усі файли, що були розроблені, знаходяться в додатках.

Файли логіки в даному застосунку представляють основний його код. Для чистоти програмного коду логіку було розділено на декілька файлів для зручності підтримки програми.

NetworkWorker – один з базових класів програми, який забезпечує надсилання запитів на сервер. Для того, щоб запити не зупиняли весь додаток методи класу є корутинами. Даних клас містить реалізацію черги для запитів, які ще не були відправлені. Наприклад, якщо користувач вніс якісь зміни в застосунку, але не був підключений до інтернету, то NetworkWorker збереже параметри запиту і надішле їх на сервер коли появиться доступ до інтернету. За перевірко доступу до інтернету відповідає метод checkServer. Цей метод пробує надіслати тестовий запит до серверу. Якщо відповіді не отримано, або ж забит не вдалось надіслати, то програма переходит в режим оффлайн. Під час навігації по застосунку часто відправляються запити, які перевіряють наявність оновлень. Під час відправки цих запитів здійснюється повторна перевірка серверу. Після успішного під’єднання до серверу відбувається надсилання збережених змін, а також оновлення даних, якщо потрібно.

Клас DataManager – є основним класом, який реалізує доступ до даних. Крім бази даних на сервері, застосунок використовує власну базу, яка розміщена в пам’яті пристрою. Під час оновлення даних вона заповнюється завантажуючи дані з серверної БД. _saveData – це основний метод класу,

який відповідає за зберігання даних. Для зберігання даних клас робить запит на сервер, а також записує в локальну БД.

Клас ImagesManager – це клас для роботи з зображеннями. Основна функція класу це запис та читання зображень профілів шиншил з пам’яті пристрою. Також, під час запису нового зображення розраховується його md5 хеш для звірки версії зображення на сервері та в застосунку. У випадку не співпадіння застосунок завантажить нову версію замінивши нею стару.

Генерацію полів для введення даних розроблено з використанням шаблону Фабрика. Класи, які пов’язані з цією реалізацією знаходяться в директорії EditFields. Застосунок містить чотири види полів: поле вибору, поле дати, поле для тексту та поле для цифр. Кожне з цих полів наслідує абстрактний клас поля AbstractFieldView, який наслідує StatefulWidget фреймворку.

Flutter містить в собі два типи віджетів – Stateful та Stateless. Stateful тип – це віджет, вигляд і дані якого в процесі будуть мінятись. Наприклад, до таких віджетів можна віднести ті, в якому, після натискання кнопки, відображаються дані, які завантажуються з сервера в реальному часі. Stateless тип – це віджет, вигляд і дані якого не будуть змінюватись в процесі користування. Зазвичай, які дані подаються при ініціалізації такого віджета, такі і будуть протягом всього його життєвого циклу.

Для успішного наслідування StatefulWidget до нього також має бути приєднаний клас, який відповідає за стан даного віджета. Обов’язковим методом для реалізації стану є метод build, який викликається щоразу при оновленні інформації. AbstractFieldView містить основні методи для валідації та зберігання інформації з полів вводу.

Застосунок містить в собі файл конфігурації в якому прописано основні параметри для кожного поля. До таких параметрів належать: тип поля, обов’язковість поля, заголовок, текст помилки, можливі вибори поля.

Типи полів визначені в переліченому типу даних, який реалізований в класі FieldType.

Фабрику в даному застосунку реалізує клас EditFieldsFactory, мета якого створити поле відповідно до заданого його типу. Реалізації кожного типу поля описані в класах ChoiceFieldView, DateFieldView, NumberFieldView, TextFieldView. UML діаграма шаблону в контексті застосунку зображена на рисунку 2.2.

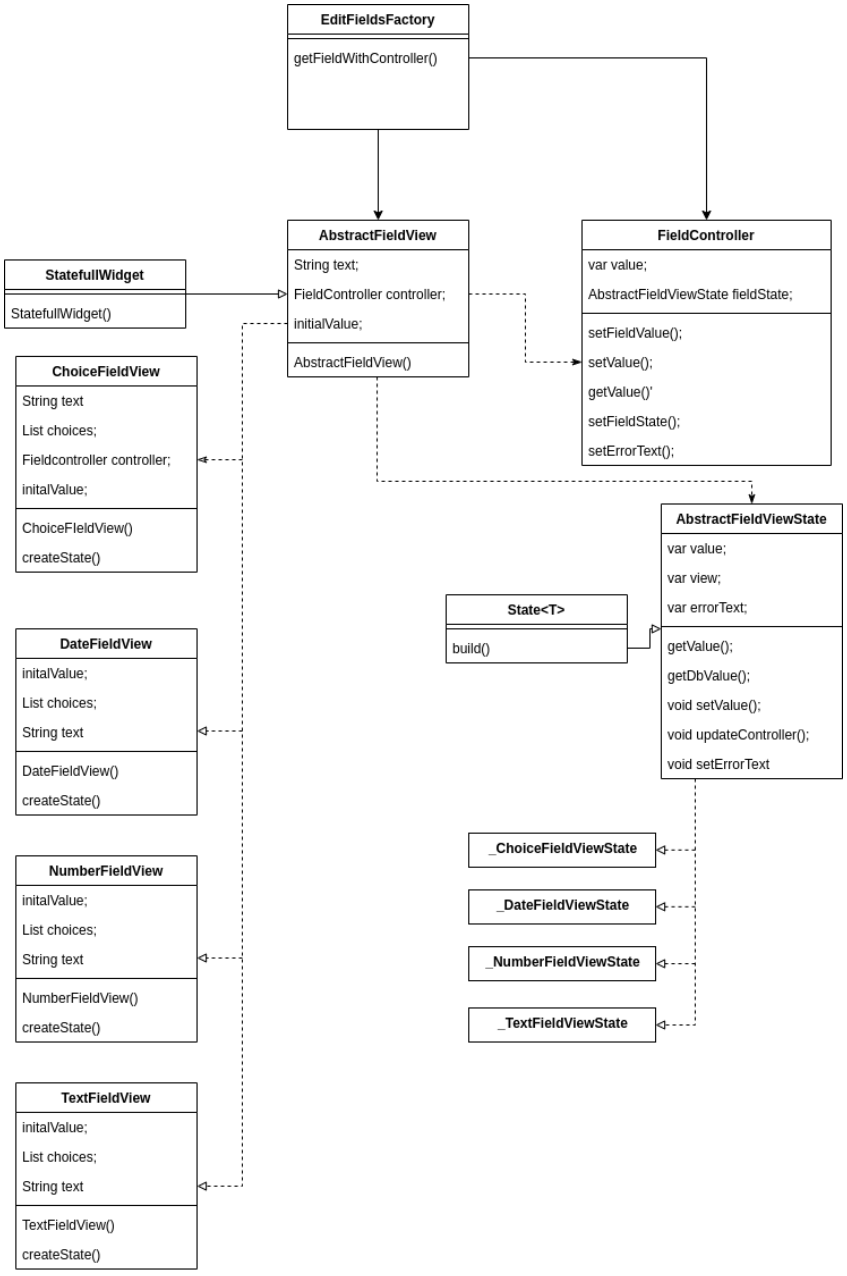


Рисунок 2.2 – UML діаграма класів для створення полів вводу

Для доступу до даних цих класів використовується шаблон Спостерігач. Коли поле зафіксувало оновлення даних, воно сповіщає про це спостерігача, пізніше з якого ці дані отримуються застосунком. Зв’язок класу реалізації спостерігача FieldController з іншими класами системи зображений на UML діаграмі на рисунку 2.2.

LoginView – це клас, призначенням якого є автентифікація користувача. Основною функцією є login() яка зчитує дані з полів логіну і паролю та надсилає їх на сервер. При успішній авторизації сервер повертає токен авторизації користувача.

Statistics – клас, який відповідає за відображення базової статистики по фермі. До цієї статистики входить: статистика по народжуваності за останній місяць, статистика про сімей, в яких найбільше новонароджених за останній місяць, статистика про кількість відсаджених за останній місяць. За обрахунок статистики відповідає клас StatisticsManager. Цей клас завантажує статистичні дані з сервера, якщо додаток встановив з’єднання з сервером, якщо ж ні – викликає власні методи обрахунку потрібної статистики. Така структура розроблена для того, щоб на пристрої користувача не задіювались додаткові ресурси, а більшості обробки даних проводилось на стороні серверу.

2.4 Опис інтерфейсу мобільного застосунку для ферми шиншил

При запуску мобільного застосунку відкривається екран авторизації, який зображений на рисунку 2.3.

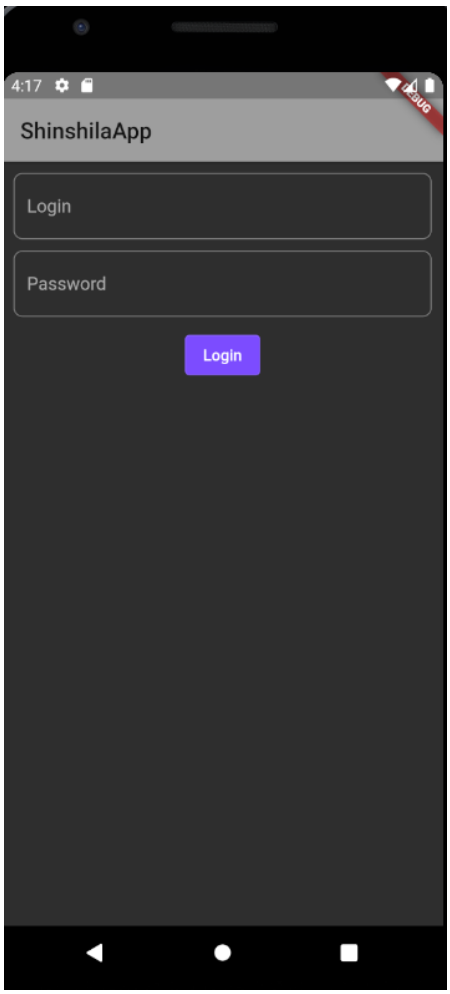


Рисунок 2.3 – Екран авторизації

Якщо користувач ввів правильні вхідні дані то користувач переходить на домашню сторінкою, якою є каталог всіх осіб ферми. Домашню сторінку зображено на рисунку 2.4.

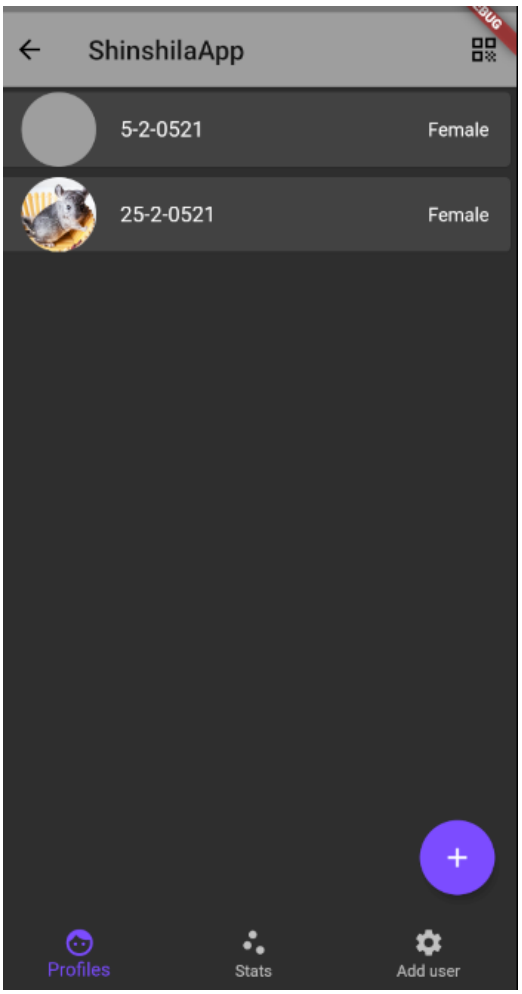


Рисунок 2.4 – Вигляд домашньої сторінки

На домашній сторінці списком розміщені елементи осіб з короткою інформацією, на яких відображено три речі: фотографія особи, її фермерський ідентифікатор та стать. Внизу екрану розміщені три кнопки для навігації між основними екранами програми. Кнопка Profiles відповідає за екран на якому розміщені основні елементи роботи з інформацією про шиншил.Кнопка Stats переключає на екран, на якому розміщена базова статистика. Кнопка Add user переміщає на екран для додавання нового користувача.

Клацнувши на один з доданих профілів шиншилл можна перейти на сторінку інформації про особу, яка зображена на рисунку 2.5.

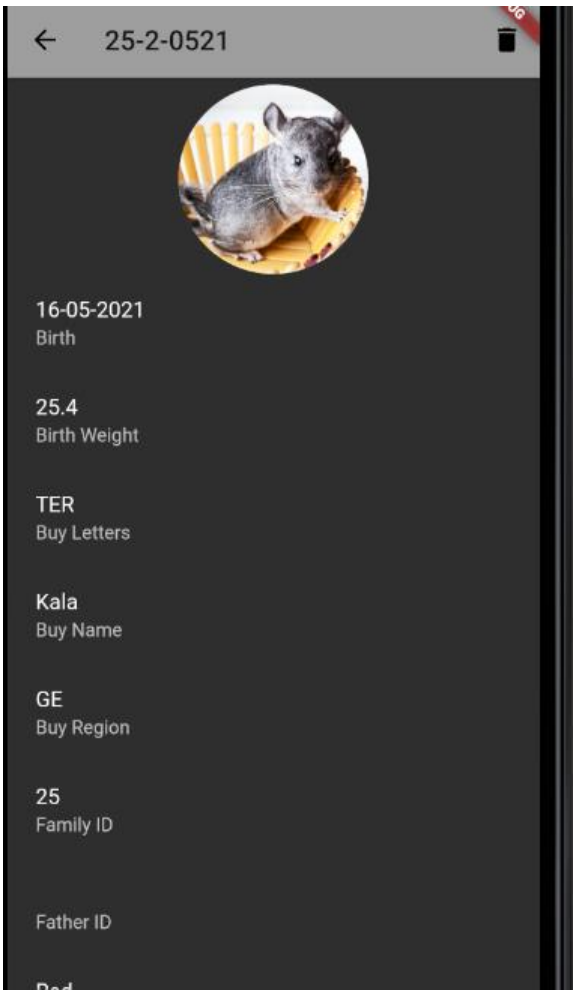
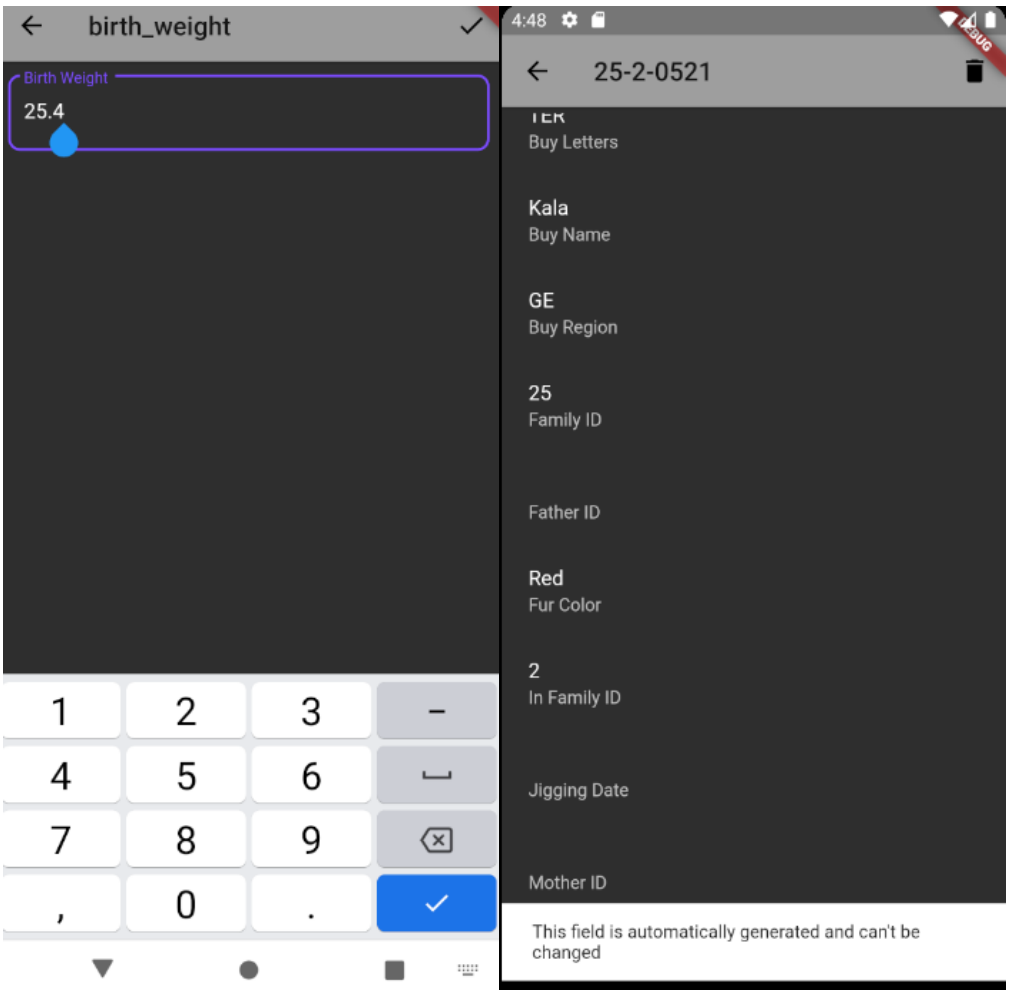


Рисунок 2.5 – Екран з відображення основної інформації про шиншилу

Натиснувши на поле застосунок переключає поточний екран на екран редагування інформації. Не всю інформацію можна редагувати. Якщо діє заборона на редагування певного поля, то на екрані з'явиться повідомлення про те, що поле не може бути редагованим. Приклади екрану редагування інформації поля без заборони та з забороною зображений на рисунку 2.6а та 2.5б.



а) б)

Рисунок 2.6 – Приклад екрану редагування інформації (а – вигляд не забороненого поля; б – вигляд забороненого поля)

Також на екрані перегляду інформації можна видалити інформацію про особу нажавши на іконку корзини в правому верхньому куті.

На домашній сторінці над списком вкладок можна перейти у вікно додавання інформації про нову шиншиллу методом натискання на кнопку “+”. Приклад екрану створення нової шиншили зображений на рисунку 2.7

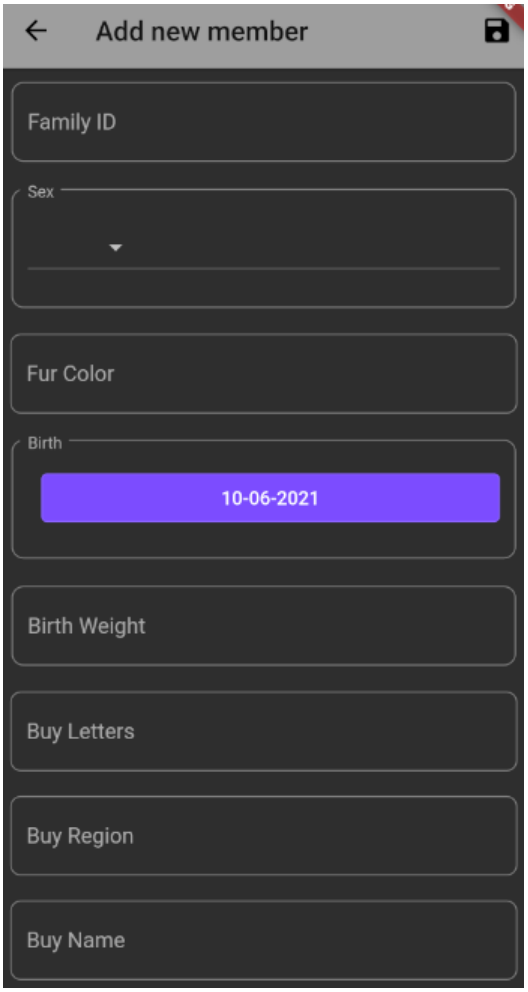
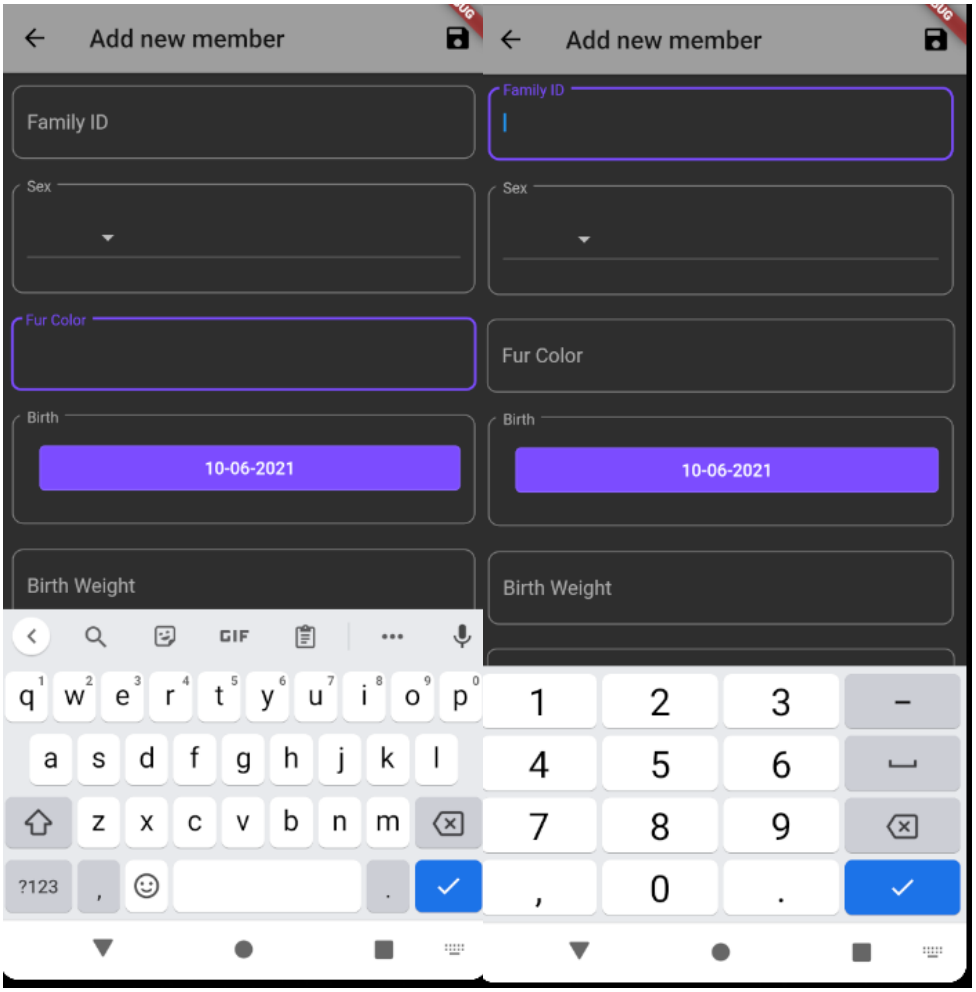


Рисунок 2.7 – Вигляд екрану для додавання інформації про шиншилу

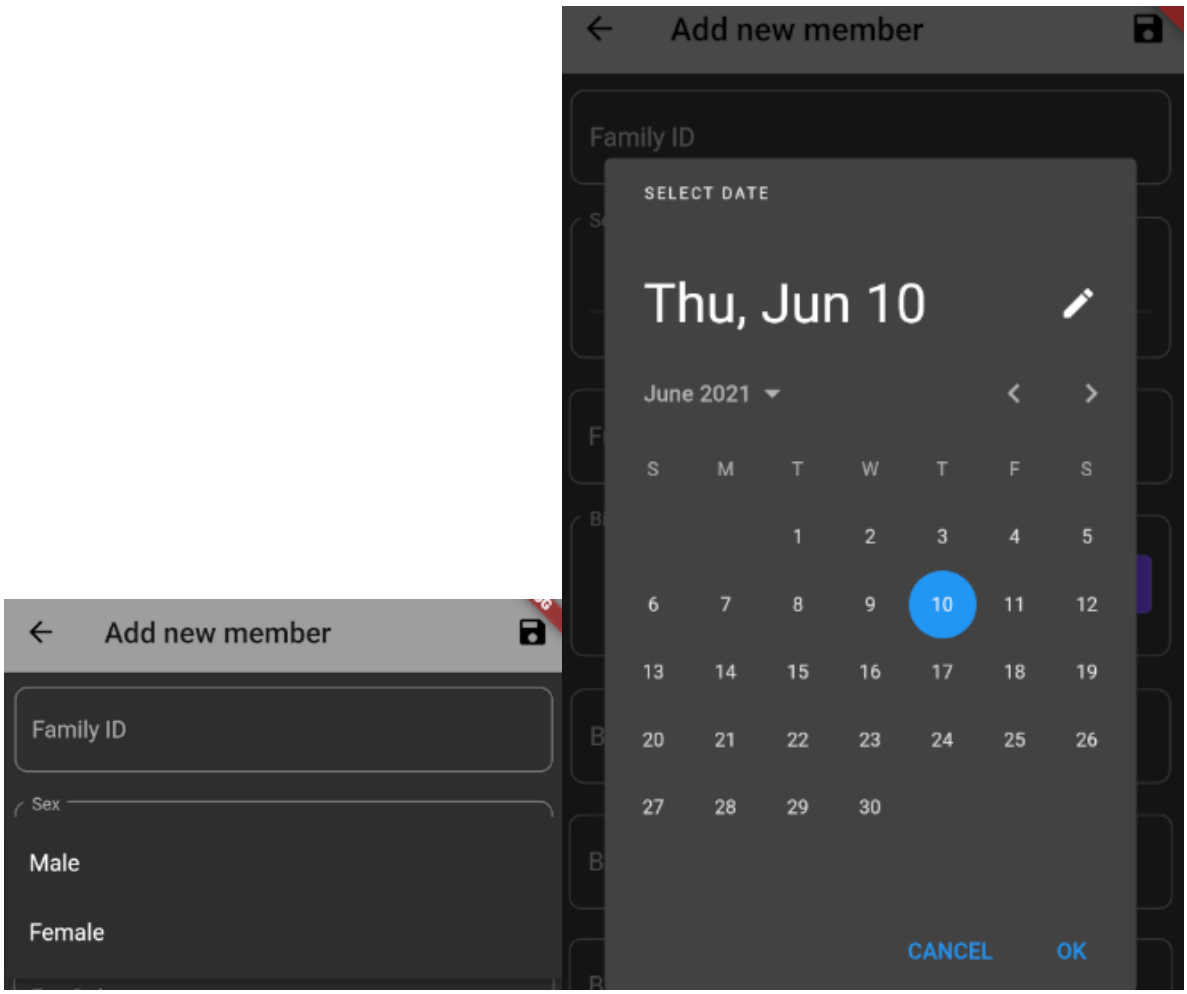
Екран містить поля, які згенеровані за допомогою шаблону програмування Фабрика, про який було описано вище. На рисунку 2.8а зображено вигляд взаємодії з текстовим полем, а на рисунку 2.8б – вигляд взаємодії з чисельним полем.



а) б)

Рисунок 2.8 – Вигляд взаємодії з полями (а – текстовим, б – чисельним)

На рисунку 2.9а зображена взаємодія з полем вибору, а на рисунку 2.9б – з полем вибору дати.



а) б)

Рисунок 2.9 – Вигляд взаємодії з полями (а – полем вибору, б – полем вибору дати)

Для того, щоб зберегти зміни, потрібно нажати на іконку дискети в правому верхньому кутку екрану. Для скасування додавання потрібно нажати на іконку стрілки, яка поверне на головний екран.

Екран статистики містить два базові графіки: секторну діаграму, та стовпцеву діаграму. Секторна діаграма показує кількість народжених самок та самців і їх розподіл за поточний місяць, а також кількість померлих. Стовпцева діаграма показує топ п’ять сімей по кількості народжуваності в поточному місяці. Висота стовпця – це кількість осіб, які народились в поточному місця в поточної сім’ї. Разом з графіками сторінка статистики містить інформацію про кількість осіб, яких відсаджують чи відсадили

протягом цього місяця. Вигляд екрану зі статистикою зображений на рисунку 2.10.

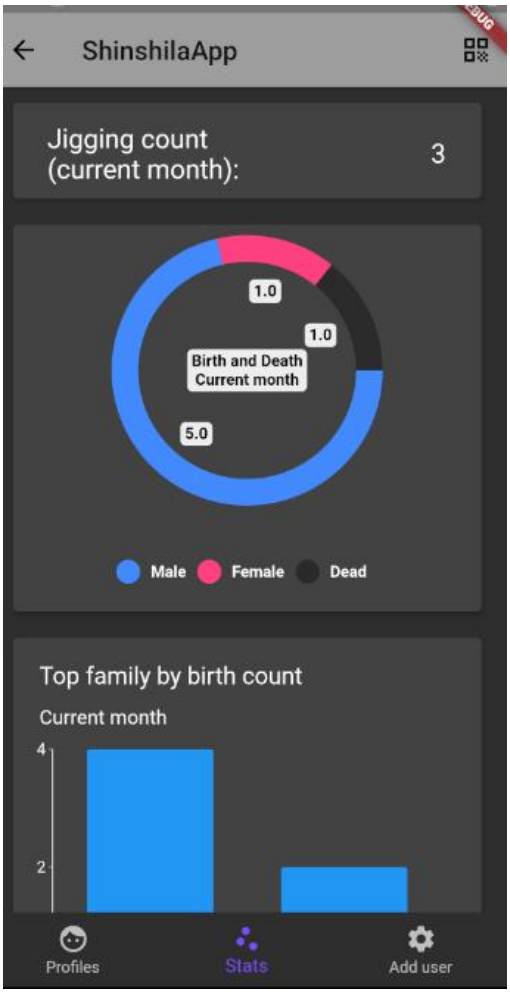


Рисунок 2.10 – Вигляд екрану зі статистикою

На екрані додавання користувача розміщено пару полів для додавання логіну і паролю нового користувача. Така спрощена схема дозволяє швидко додавати нових користувачів, що є зручним для дрібних фермерів.

2.5 Тестування та валідація мобільного застосунку для ферми шиншил

Для того, щоб провалідувати програмний код мобільного застосунку для ферми шиншил було використано IDE Android Studio, яке є фаворитом з

подібних йому програм, що використовуються для розробки мобільних застосунків. За допомогою утиліти “Інспекція коду” була проведена валідація, яка зображена на рисунку 2.11.

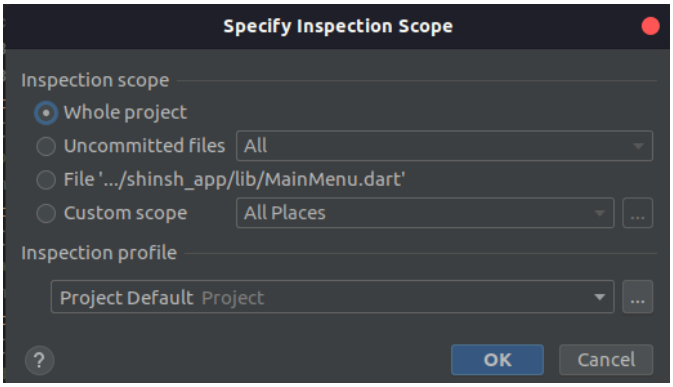


Рисунок 2.11 – Запуск інструменту “Інспекція коду”

Після перевірки утиліти було знайдено помилки для двох категорій “Android” та “General”. Помилки першої категорії пов'язані з адаптацією файлів системи Андроїд до фреймворку Flutter. На рисунку 2.12 зображені помилки першої категорії.

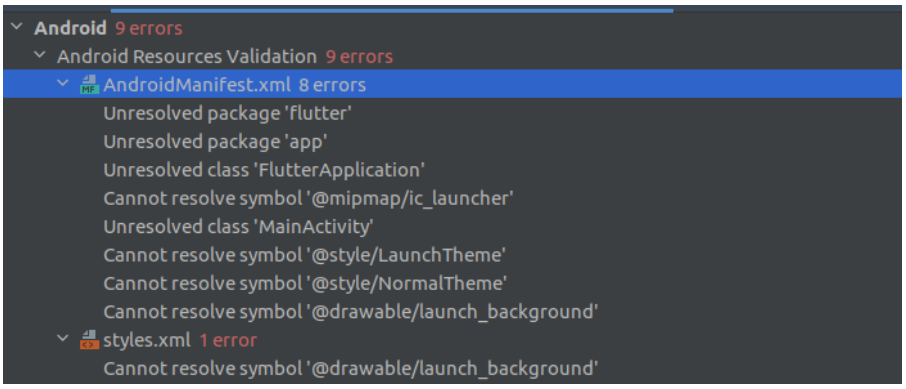


Рисунок 2.12 – Помилки першої категорії

Помилки другої категорій виникають також через те, що розробка проводиться за допомогою Flutter. Ці помилки вирішаться під час збирання пакету програми. На рисунку 2.13 зображені помилки другої категорії.

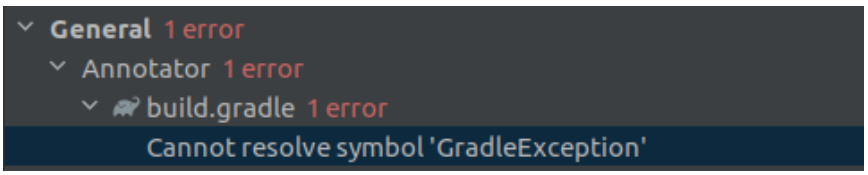


Рисунок 2.13 – Помилки другої категорії

До категорій попереджень відноситься “Proofreading”. Ці попередження, переважно, пов’язані з неправильним написанням слів, або ж зі словами, яких не існує в англійській мові. Ця категорія дуже корисна, оскільки дозволяє якісно перейменувати елементи коду, які названі погано. На рисунку 2.14 зображено помилки Proofreading.

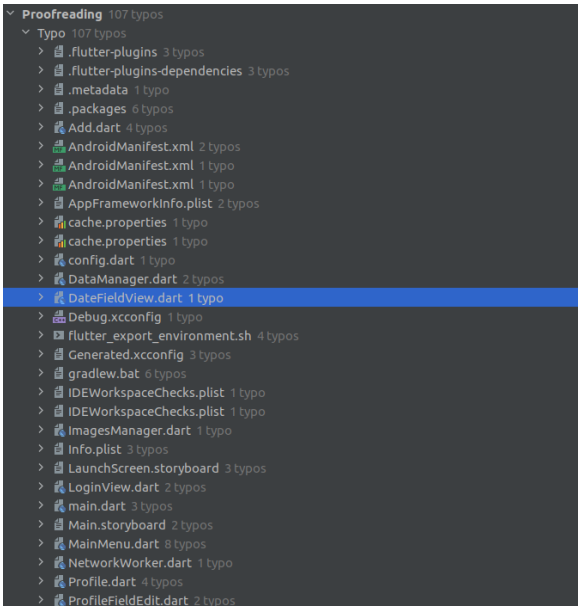


Рисунок 2.13 – Помилки категорії Proofreading

Після перевірки коду інструментом “Інспекція” було проведено тестування базових властивостей полів. Для цього було перевірено основний екран, в якому використовуються всі типи полів для вводу – екран додавання інформації про нову особу.

Для тестування полів було проведено ряд тестів. Перший тест полягає в перевірці на збереження даних при пустоті обов’язкових полів. Результат тестування зображений на рисунку 2.14.

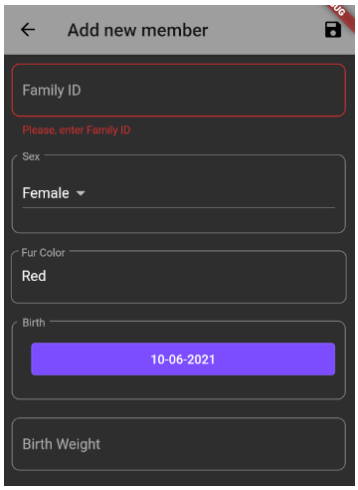


Рисунок 2.14 – Результат тестування збереження при пустоті обов’язкових полів

З результатів тесту видно, що застосунок не зберіг введені дані, а надав червону рамку для пустого поля, яке повинне мати значення. Таку ж перевірку було зроблено для поля типу Choice. Результати перевірки зображені на рисунку 2.15.

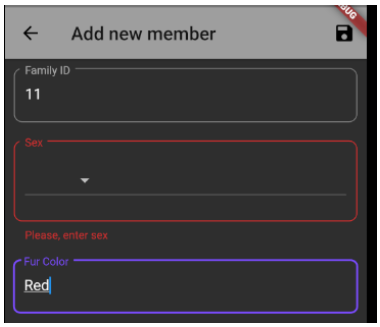


Рисунок 2.15 – Результат тестування поля типу Choice

Тестування решти функцій відбувалось відбувалось в ручному режимі методом чорного ящика. Це означає, що було перевірено коректне виконання функцій застосунку і серверу.

РОЗДІЛ 3. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

3.1 Шляхи підвищення життєдіяльності людини

Робота програміста, зазвичай, не передбачає важкого фізичного навантаження, що має негативний вплив на здоров'я. Однак, дослідження показують, що фізичні тренування значно покращують стан та функції основних органів і систем людини.

Загалом розрізняють два види фізичної роботи – статичну і динамічну.

Під час статичної роботи розхід енергії збільшується, що потребує підвищеного обміну речовин. Така робота дуже швидко викликає втому, адже довготривале скорочення і напруження м'язів та відсутність умов для кровообігу є основними факторами для цього.

Якщо під час роботи виникають нагрузки які потребують рух кінцівок чи інших частин ціла, то таку роботу називають динамічною. За рахунок того, що навантаження розподіляється не постійно, а поетапно, в людини не виникає застою кровопостачання і кисневого обміну, з чого слідує, що людина менше втомлюється.

Стан м’язів, а також їх здатність працювати залежить від навантаження. Якщо програмісту правильно підібрати середні величини ритму і навантаження, то це підвищить його продуктивність та відтермінує час, коли програміст відчує втому.

Підбираючи навантаження варто взяти до уваги професійну діяльність, особистість людини, її фізичний стан. Робота програміста, зазвичай, не передбачає тренованості людини, що призводить до погіршення стану серцево-судинної, дихальної та центральної нервової систем.

Для покращення стану цих систем, а також підвищення життєдіяльності людини потрібно змінити свій звичайний раціон на раціональне харчування. Також, попри зміну раціону існують інші засоби для

покращення цих показників, такі як: фармакологічні препарати, лікарські рослини, фізіотерапія, масаж, загартовування та інші.

Однак, харчування є найважливішим фактором, що впливає на організм людини, оскільки воно забезпечує людину енергетичними ресурсами з яких утворюються гормони та ферменти, які в свою чергу є регуляторами обміну речовин у тканинах. Відповідно до цього можна сформувати три принципи раціонального харчування:

1. Енергетична цінність раціонального харчування повинна відповідати енергетичним затратам організму.

2. Відповідність хімічного складу їжі фізіологічним потребам організму в харчових продуктах.

3. Різноманітність споживання продуктів.

Якщо слідувати цим трьом правилам, то хорчування буде більш здоровіше і насиченіше. Щоб визначити потреби енергії дорослої людини вчені розрізняють п’ять груп фізичної зайнятості:

- I група – особи, зайняті переважно розумовою працею (керівники);
- II група – особи, зайняті легкою фізичною працею (санітарки, викладачі, тренера);
- III група – особи, зайняті середньою за ступенем важкості фізичною працею (верстатники, слюсарі, хіміки);
- IV група – особи, зайняті важкою фізичною працею (будівельники, механізатори);
- V група – особи, зайняті особливо важкою фізичною працею (сталевари, лісоруби, землекопи).

Програмісти підпадають під першу групу фізичної зайнятості, відповідно треба враховувати це при складанні раціону.

До психологічних засобів підвищення життєдіяльності і відновлення працездатності належать: психотерапія, психопрофілактика та психогігієна.

Психопрофілактика це набір вправ які навчають пацієнтів м’язової релаксації вмінню контролювати мимовільну розумові діяльність та інше. Серед методів, які дають можливість захисту психіки людини від шкідливих дій і настроїти її на подолання труднощів, стресових станів, на першому місці стоїть психічна саморегуляція. Психічна саморегуляція – це дія людини на саму себе за допомогою слів та відповідних їм уявних образів. Отже, слова, мова, уявні образи умовно рефлекторним шляхом впливають на функціональний стан різних органів і систем позитивно чи негативно. Психогігієна включає мистецтво взаємовідносин між людьми, духовну гармонію людини і природи, комфортні умови побуту, різні види відпочинку.

Під час розробки та використання застосунку потрібно не забувати про шляхи підвищення життєдіяльності, оскільки в більшості випадків особи, які цим займаються підпадають під І група фізичної зайнятості.

3.2 Інструкція для обслуговуючого персоналу на випадок виникнення аварії, пожежі

Вирощування шиншил потребує строгого дотримання умов. Ключовими умовами, які можуть спричинити пожежі є температурниий режим, який потрібно дотримувати завжди на одному рівні, та сухість приміщення. Також, часто власники ферм зберігають інвентар, до якого входить сушена трава, в одному приміщенні з шиншилами. Невдале коротке замикання чи поломка в нагрівальних елементах можуть спричинити пожежу, яка буде загрожувати життю обслуговуючого персоналу ферми.

У випадку пожежі обслуговуючий персонал, спершу, повинен негайно повідомити про пожежу оперативно-рятувальну службу цивільного захисту по телефону за номером 101. Під час повідомлення потрібно володіти інформацією про адресу, кількість поверхів будівлі, місце виникнення пожежі і наявність людей. Також, служба може запросити ваше прізвище для подальшої кооперації.

Далі слідує етап підготовки до евакуації. На цьому етапі потрібно негайно і спокійно повідомити всіх про термінову евакуацію використовуючи систему оповіщення, якщо така є. Після цього потрібно перерахувати усіх евакуйованих і у випадку невиявлення когось із працівників негайно з’ясувати де його востаннє бачили і передати цю інформацію представнику оперативно-рятувальної служби, яка прибула до місця виклику.

Наступний етап це гасіння загоряння або осередку пожежі до прибуття оперативно-рятувальної служби. Для цього потрібно відключити будівлю від електрики, і негайно почати гасіння застосовуючи первинні засоби для пожежогасіння. При наявності систем протипожежного захисту перевірити їх спрацювання або привести їх у дію.

Якщо немає прямої загрози, то потрібно евакуювати матеріальні цінності, згідно із заздалегідь розробленим планом першочерговості евакуації ,а також печатку, штампи та, по можливості, тварин, які є на фермі.

При зустрічі з оперативно-рятувальною службою цивільного захисту потрібно проінформувати керівника служби, що прибула про таке: чи є в будинку люди, яким загрожує пожежа, а якщо є то їх кількість, місце, де виникла пожежа, яке приміщення горить і куди може розповсюдитись вогонь та дим і вказати місце розташування пожежних гідрантів.

Враховуючи те, що користування застосунком відбуватиметься на території ферми потрібно знати як діяти у випадку виникнення пожежі.

3.3 Вимоги до профілактичних медичних оглядів для працівників ПК

Під час довготривалої роботи з ПК у програмістів можуть виникати проблеми пов'язані з серцево-судинною системою, головні болі, болі в суглобах тощо. Тому, працівники, які працюють з пк підлягають обов’язковими медичним оглядам. Обов’язковість проведення медичних

оглядів для працівників, які працюють за електронно-обчислювальними машинами, попросту комп'ютерами, передбачена розд. 6 Державних санітарних правил і норм роботи з візуальними дисплейними терміналами електронно-обчислювальних машин ДСанПіН 3.3.2.007-98, затверджених постановою Головного державного санітарного лікаря України від 10.12.98 р. № 7 [5].

Працюючі з візуальними дисплейними терміналами (ВДТ) електронно-обчислювальних машин (ЕОМ) колективного використання та персональних ЕОМ (ПЕОМ) підлягають обов'язковим медичним оглядам: попереднім — при влаштуванні на роботу і періодичним — протягом трудової діяльності.

Такі медичні огляди проводяться відповідно до вимог Порядку проведення медичних оглядів працівників певних категорій, затвердженого наказом МОЗ від 21.05.2007 р. № 246 [3].

Медичні огляди мають проводитись з періодичністю раз на два роки та у терапевта, невропатолога та офтальмолога. Також, за наявності медичних показань можуть залучати лікарів інших спеціальностей.

Огляд проводиться з метою:

- вчасного виявлення гострих та хронічних захворювань, загальних та виробничих факторів і трудового процесу;
- забезпечення постійної періодичності спостереження за станом здоров'я в умовах дії шкідливих та небезпечних виробничих факторів;
- розробки індивідуальних та групових програм оздоровчих та реабілітаційних заходів працівника, що віднесені за результатами медичного огляду до групи ризику;
- проведення відповідних оздоровчих заходів.

Огляд проводиться також, щоб перевірити людину на придатність. Для останнього повинна бути підтверджена гострота зору, показники реакції, акомодації, стану бінокулярного апарату ока тощо. При цьому також

враховується стан організму в цілому. Але, питання про придатність до роботи для кожного працівника вирішується персонально. Під час цього враховуються його особливості, умови праці та результати додаткових медичних оглядів.

До протипоказань для роботи з ПК належать усі хронічні форми психічних захворювань, ендокринні захворювання, тяжкий ступінь бронхіальної системи, гіпертонічна хвороба III стадії та інші захворювання.

На період проходження медогляду роботодавець зберігає за працівником місце роботи і середній заробіток. А після медогляду інформує працівника про можливість чи неможливість продовжувати роботу.

Оскільки кваліфікаційна робота бакалавра виконувалась з використанням ПК, були враховані відповідні вимоги щодо профілактичних оглядів.

ВИСНОВКИ

У процесі розробки кваліфікаційної роботи було спроектовано та розроблено мобільний застосунок для обліку на шиншиловій фермі під платформу Android, з використанням мов програмування Dart та Python.

В першому розділі кваліфікаційної роботи освітнього рівня «Бакалавр»:

- Проведено аналіз предметної області, досліджено конкурентні рішення.
- Встановлено основні вимоги до застосунку обліку на шиншиловій фермі.
- Виділено основні сутності, що присутні в мобільному застосунку.

В другому розділі кваліфікаційної роботи:

- Описано процес проектування, розробки і тестування мобільного застосунку для обліку на шиншиловій фермі.
- Проаналізовано варіанти використання та архітектурні особливості системи;
- Описано програмні рішення як частини застосунку, так і частини серверу.
- Протестовано та провалідовано мобільний застосунок.

В розділі «Безпека життєдіяльності, основи охорони праці» розглянуті шляхи підвищення життєдіяльності людини, подано інструкцію для обслуговуючого персоналу на випадок виникнення аварії, пожежі та було розглянуто вимоги до профілактичних медичних оглядів для працівників ПК.

ПЕРЕЛІК ДЖЕРЕЛ

1. Андроїд [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Android_(operating_system) – Дата доступу: 3.06.2021.

2. Досліджування народжуваності шиншил [Електронний ресурс] – Режим доступу до ресурсу: https://www.researchgate.net/publication/346921355_The_analysis_of_chinchilla_Chinchilla_lanigera_M_male_reproduction_The_case_of_a_leading_Polish_Breeding_Farm – Дата доступу: 6.06.2021.

3. МОЗ від 21.05.2007 р. № 246 [Електронний ресурс] – Режим доступу до ресурсу: http://search.ligazakon.ua/l_doc2.nsf/link1/RE14113.html – Дата доступу: 9.06.2021.

4. Мова програмування Dart [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Dart_(programming_language) – Дата доступу: 3.06.2021.

5. Наказ N 246 [Електронний ресурс] – Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/v0007282-98#Text – Дата доступу: 9.06.2021.

6. Об'єктно-орієнтоване програмування [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Object-oriented_programming – Дата доступу: 2.06.2021.

7. Про компанію Google [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Google – Дата доступу: 3.06.2021.

8. Про Flutter [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Flutter_(software) – Дата доступу: 4.06.2021.

9. Хеш-функція [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Hash_function – Дата доступу: 5.06.2021.

10. Шаблони проектування [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Design_Patterns – Дата доступу: 1.06.2021.

11. Шаблон Спостерігач [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Observer_pattern – Дата доступу: 2.06.2021.

12. . Шаблон Фабричний метод [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Factory_method_pattern – Дата доступу: 2.06.2021.

13. Шиншила [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Chinchilla – Дата доступу: 6.06.2021.

14. Що таке чистий код [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/company/jugru/blog/490718/ – Дата доступу: 10.06.2021.

15. Agritec [Електронний ресурс] – Режим доступу до ресурсу: https://www.agritecsoft.com/en/ – Дата доступу: 7.06.2021.

16. Android Studio [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/studio – Дата доступу: 9.06.2021.

17. API [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/API – Дата доступу: 5.06.2021.

18. Dart [Електронний ресурс] – Режим доступу до ресурсу: https://dart.dev/ – Дата доступу: 4.06.2021.

19. DrawIO[Електронний ресурс] – Режим доступу до ресурсу: https://app.diagrams.net/ – Дата доступу: 8.06.2021.

20. Flask [Електронний ресурс] – Режим доступу до ресурсу: https://flask.palletsprojects.com/en/2.0.x/– Дата доступу: 1.06.2021.

21. Flutter [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/flutter/flutter – Дата доступу: 4.06.2021.

22. Flutter Charts [Електронний ресурс] – Режим доступу до ресурсу: https://pub.dev/packages/charts_flutter – Дата доступу: 10.06.2021.

23. Github [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/ – Дата доступу: 10.06.2021.

24. Google Spreadsheets [Електронний ресурс] – Режим доступу до ресурсу: https://docs.google.com/spreadsheets – Дата доступу: 7.06.2021.

25. HTTP [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol – Дата доступу: 6.06.2021.

26. JetBrains [Електронний ресурс] – Режим доступу до ресурсу: https://www.jetbrains.com/ru-ru/ – Дата доступу: 8.06.2021.

27. PostgreSQL [Електронний ресурс] – Режим доступу до ресурсу: https://www.postgresql.org/ – Дата доступу: 5.06.2021

28. Python [Електронний ресурс] – Режим доступу до ресурсу: https://www.python.org/ – Дата доступу: 1.06.2021.

29. Use case діаграма [Електронний ресурс] – Режим доступу до ресурсу: https://www.lucidchart.com/pages/uml-use-case-diagram – Дата доступу: 8.06.2021.

30. ZooEasy [Електронний ресурс] – Режим доступу до ресурсу: https://www.zooeasy.com/ – Дата доступу: 7.06.2021.

ДОДАТКИ

Додаток А

Код клієнтського модуля на серверній частині

Лістинг файлу server.py

```
from flask import Flask, request, Response, jsonify
from db_worker import DbWorker
from login_db_worker import LoginDBWorker
import json
import uuid
import base64
import hashlib
import utils
import config
from datetime import datetime
import pandas as pd


app = Flask(__name__)
worker = DbWorker()
login_worker = LoginDBWorker()

last_update_id = '25'

def extract_auth_data(data):
    auth_data = dict()
    request_data = dict()

    for field in data:
        if field in config.auth_fields:
            auth_data[field] = data[field]
        else:
            request_data[field] = data[field]
```

```
        return auth_data, request_data


    @app.route('/add', methods=['POST'])
    def add_info():
        global last_update_id

        last_update_id = uuid.uuid4()
        encrypted_data = json.loads(request.data)
        request_data                                                =
login_worker.decrypt_request_data(encrypted_data)
        auth_data, data = extract_auth_data(request_data)

        if login_worker.check_api_token(auth_data):

            family_data                                             =
worker.get_data_by_family_id(int(data['family_id']))
            data['in_family_id']                                    =
utils.calculate_in_family_id(family_data, data['sex'])

            data['profile_id'] = utils.create_id(data)
            worker.add_profile(**data)

            return Response(status=200)
        else:
            return ("Invalid API token", 404)

    @app.route('/update', methods=['POST'])
    def update_info():
        global last_update_id

        encrypted_data = json.loads(request.data)
        request_data                                                =
login_worker.decrypt_request_data(encrypted_data)
```

```
            auth_data, data = extract_auth_data(request_data)


            if login_worker.check_api_token(auth_data):
                last_update_id = str(uuid.uuid4())
                id_ = data.pop("_id")
                data['profile_id'] = utils.create_id(data)
                worker.update_profile(id_, **data)
                return Response(status=200, response=last_update_id)
            else:
                return ("Invalid API token", 404)

        @app.route('/delete_profile', methods=['POST'])
        def delete_profile():
            global last_update_id

            encrypted_data = json.loads(request.data)
            request_data                                                =
login_worker.decrypt_request_data(encrypted_data)
            auth_data, data = extract_auth_data(request_data)
            if login_worker.check_api_token(auth_data):
                last_update_id = str(uuid.uuid4())
                id_ = data.pop("_id")
                worker.remove_profile(id_)
                return Response(status=200, response=last_update_id)
            else:
                return ("Invalid API token", 404)


        @app.route('/get', methods=['POST'])
        def send_info():
            encrypted_data = json.loads(request.data)
            request_data                                                =
login_worker.decrypt_request_data(encrypted_data)
            auth_data, data = extract_auth_data(request_data)
```

```
            if login_worker.check_api_token(auth_data):
                return jsonify(worker.get_data(records=True))
            else:
                return ("Invalid API token", 404)

        @app.route('/get_fields', methods=['POST'])
        def send_fields():
            encrypted_data = json.loads(request.data)
            request_data                                                    =
login_worker.decrypt_request_data(encrypted_data)
            auth_data, data = extract_auth_data(request_data)


            if login_worker.check_api_token(data):
                return jsonify(worker.get_fields(records=True))
            else:
                return ("Invalid API token", 404)

        @app.route('/get_update_id', methods=['POST'])
        def get_update_id():
            encrypted_data = json.loads(request.data)
            request_data                                                    =
login_worker.decrypt_request_data(encrypted_data)
            auth_data, data = extract_auth_data(request_data)

            if login_worker.check_api_token(auth_data):
                return jsonify({'updateId': last_update_id})
            else:
                return ("Invalid API token", 404)


        @app.route('/status', methods=['GET'])
        def status():
            return Response(status=200)
```

```
@app.route('/upload_profile_image', methods=['POST'])
def upload_profile_image():
    global last_update_id

    encrypted_data = json.loads(request.data)
    request_data = login_worker.decrypt_request_data(encrypted_data)
    auth_data, data = extract_auth_data(request_data)


    if login_worker.check_api_token(data):
        id_ = data.pop("_id")
        file_name = f'images/{id_}.png'
        file = open(file_name, 'wb')
        img_data = base64.b64decode(data['image'])
        img_hash = hashlib.md5(data['image'].encode('utf-8')).hexdigest()
        file.write(img_data)
        file.close()

        last_update_id = str(uuid.uuid4())

        worker.update_profile(id_, image_url=file_name, _image_hash=img_hash)
        return jsonify({'updateId': last_update_id})
    else:
        return ("Invalid API token", 404)

@app.route('/get_profile_image_hash', methods=['POST'])
def get_profile_image_hash():
    encrypted_data = json.loads(request.data)
    request_data = login_worker.decrypt_request_data(encrypted_data)
```

```
            auth_data, data = extract_auth_data(request_data)



            if login_worker.check_api_token(auth_data):
                profile_data = worker.get_data_by_id(data['_id'])
                if not profile_data.empty:
                    image_hash = profile_data['_image_hash'].item()
                    return jsonify({'hash': image_hash})
                else:
                    return ("Profile not founded", 404)
            else:
                return ("Invalid API token", 404)

        @app.route('/download_profile_image', methods=['POST'])
        def download_profile_image():
            encrypted_data = json.loads(request.data)
            request_data                                                    =
login_worker.decrypt_request_data(encrypted_data)
            auth_data, data = extract_auth_data(request_data)



            if login_worker.check_api_token(auth_data):
                profile_data = worker.get_data_by_id(data['_id'])
                if not profile_data.empty:
                    image_url = profile_data['image_url'].item()
                    if image_url:
                        file = open(image_url, 'rb')
                        image_data = base64.b64encode(file.read())
                        response = {'image': image_data}
                    else:
                        response = {'image': None}

                    return jsonify(response)
                else:
```

```
                return ("Profile does not contain profile image",
404)
        else:
            return ("Invalid API token", 404)

    @app.route('/get_born_stats', methods=['POST'])
    def get_born_stats():
        encrypted_data = json.loads(request.data)
        request_data                                              =
login_worker.decrypt_request_data(encrypted_data)
        auth_data, data = extract_auth_data(request_data)


        if login_worker.check_api_token(auth_data):
            data = worker.get_data()
            current_month = datetime.today().month
            data['birth']      =      pd.to_datetime(data['birth'],
format='%d-%m-%Y')
            data['death_date']                                     =
pd.to_datetime(data['death_date'], format='%d-%m-%Y')
            birth_data     =     data[data['birth'].dt.month     ==
current_month]
            death_data    =    data[data['death_date'].dt.month    ==
current_month]

            male_birth    =    len(birth_data[birth_data['sex']    ==
'Male'])
            female_birth   =   len(birth_data[birth_data['sex']   ==
'Female'])

            response_data   =   {'dead':   len(death_data),   'male':
male_birth, 'female': female_birth}
            return jsonify(response_data)
```

```
    @app.route('/get_most_profitable', methods=['POST'])
    def get_most_profitable():
        encrypted_data = json.loads(request.data)
        request_data =
login_worker.decrypt_request_data(encrypted_data)
        auth_data, data = extract_auth_data(request_data)


        if login_worker.check_api_token(auth_data):
            data = worker.get_data()
            current_month = datetime.today().month
            data['birth']  =  pd.to_datetime(data['birth'],
format='%d-%m-%Y')
            birth_data  =  data[(data['birth'].dt.month  ==
current_month) & (data['alive'] == 'Alive')]
            response_data =
birth_data['family_id'].value_counts().to_dict()
            return jsonify(response_data)

    @app.route('/get_jigged_count', methods=['POST'])
    def get_jigged_count():
        encrypted_data = json.loads(request.data)
        request_data =
login_worker.decrypt_request_data(encrypted_data)
        auth_data, data = extract_auth_data(request_data)


        if login_worker.check_api_token(auth_data):
            data = worker.get_data()
            current_month = datetime.today().month
            data['jigging_date'] =
pd.to_datetime(data['jigging_date'], format='%d-%m-%Y')
            jigging_data  =  data[(data['jigging_date'].dt.month  ==
current_month) & (data['alive'] == 'Alive')]
            return jsonify(len(jigging_data))
```

```
@app.route('/get_pub_key', methods=['GET'])
def get_pub_key():
    return jsonify({'e': login_worker.pubkey.e, 'n': f'k:
{login_worker.pubkey.n}'})

@app.route('/login', methods=['POST'])
def login():
    encrypted_data = json.loads(request.data)
    data = login_worker.decrypt_request_data(encrypted_data)

    if login_worker.check_login(data):
        user_api_token =
login_worker.get_data_by_login(data['login'], 'api_token')
        response_data = {'api_token': user_api_token}
        encrypted_response_data =
login_worker.encrypt_reqest_data(response_data, data['n'],
data['e'])
        return jsonify(encrypted_response_data)

    else:
        return ("User not found", 404)


@app.route('/register', methods=['POST'])
def register_user():
    encrypted_data = json.loads(request.data)
    data = login_worker.decrypt_request_data(encrypted_data)

    if login_worker.check_api_token(data):
        login_worker.register_user(data['new_user_login'],
data['new_user_password_hash'])

        return Response(status=200)
```

```
    return ("Bad request", 400)


if __name__ == "__main__":
    app.run(host="localhost", port=6000, debug=True)
```

Лістинг файлу login_db_worker.py

```
import psycopg2
import config
import secrets
import rsa
import pandas as pd
import base64
import json

class LoginDBWorker():
  def __init__(self):

    self.conn = None
    self.cur = None

    self.update_session()
    self._check_table(config.users_table_name)

    self.pubkey, self.privkey = rsa.newkeys(512)


  def update_session(self):
    if self.conn:
         self.conn.close()
    if self.cur:
         self.cur.close()
```

```
            self.conn = psycopg2.connect(dbname=config.db_name,
user=config.db_username,
password=config.db_password,
                                         host=config.db_host)
            self.cur = self.conn.cursor()

        def _check_table(self, table_name):
            self.cur.execute(f"SELECT * FROM information_schema.tables
where table_name='{table_name}'")
            if not self.cur.rowcount:
                self.cur.execute(config.users_table_query)
                self.conn.commit()

            return True

        def encrypt_reqest_data(self, data, n, e):
            user_pubkey = rsa.PublicKey(n=n, e=e)
            encrypted_data = dict()
            for field, value in data.items():
                if field not in ['image']:
                    string_value = json.dumps(value)
                    encrypted_data[field]                          =
self.encrypt(string_value, user_pubkey)
                else:
                    encrypted_data[field] = value

            encrypted_data['n'] = f'k: {self.pubkey.n}'
            encrypted_data['e'] = self.pubkey.e

            return encrypted_data

        def decrypt_request_data(self, data):
            decrypted_data = dict()
```

```
            for field, value in data.items():
                if field not in ['n', 'e', 'image']:
                    decrypted_data[field]                                    =
json.loads(self.decrypt(value))
                else:
                    decrypted_data[field] = int(value)

            return decrypted_data

        def check_api_token(self, data):
            if 'api_token' not in data:
                return False

            login_api_token   =   self.get_data_by_login(data['login'],
'api_token')

            if login_api_token == data['api_token']:
                return True
            else:
                return False

        def check_login(self, data):
            password_hash    =    self.get_data_by_login(data['login'],
'password_hash')
            if not password_hash:
                return False
            if password_hash == data['password_hash']:
                return True
            else:
                return False

        @staticmethod
        def generate_token():
            return secrets.token_hex(16)
```

```
        def encrypt(self, message, pubkey):
          encoded_message = message.encode('utf-8')
          encrypted_message = rsa.encrypt(encoded_message, pubkey)
          base64_encrypted_message =
base64.b64encode(encrypted_message).decode('utf-8')
          return base64_encrypted_message

        def decrypt(self, message):
          encoded_message = base64.b64decode(message)
          return rsa.decrypt(encoded_message,
self.privkey).decode('utf-8')

        def get_data_by_login(self, login, data_col):
          query = f"SELECT * FROM {config.users_table_name};"
          data_df = pd.read_sql_query(query, self.conn)

          login_data = data_df[data_df['login'] == login]
          if not len(login_data):
               return False

          return login_data[data_col].item()

        def register_user(self, login, password_hash):
          api_token = self.generate_token()
          try:
               self.cur.execute(f"INSERT INTO
{config.users_table_name} (login, password_hash, api_token) VALUES
('{login}', '{password_hash}', '{api_token}');")
          except psycopg2.errors.UniqueViolation:
               self.update_session()
               return False
          self.conn.commit()

          return api_token
```

Лістинг файлу db_worker.py

```
import psycopg2
import config
import re
import pandas as pd


class DbWorker(object):
    def __init__(self):

        self.conn = None
        self.cur = None

        self.update_session()
        self._check_table(config.shinshila_table_name)


    def update_session(self):
        if self.conn:
            self.conn.close()
        if self.cur:
            self.cur.close()

        self.conn = psycopg2.connect(dbname=config.db_name,
                                     user=config.db_username,
password=config.db_password,
                                     host=config.db_host)
        self.cur = self.conn.cursor()

    def _check_table(self, table_name):
```

```
            # self.update_session()

            self.cur.execute(f"SELECT * FROM
information_schema.tables where table_name='{table_name}'")
            if not self.cur.rowcount:
                self.cur.execute(config.shinshila_table_query)
                self.conn.commit()

            return True

        def add_profile(self, **kwargs):
            # self.update_session()

            fields =
self._format_to_psql_list(kwargs.keys()).replace("'", '')
            values = self._format_to_psql_list(kwargs.values())
            print(values)
            self.cur.execute(f"INSERT INTO
{config.shinshila_table_name} {fields} VALUES {values};")
            self.conn.commit()

        def _format_to_psql_list(self, list):
            list_line = f'({[field for field in list]})'
            line = re.sub(r"[\[|\]]", '', list_line)
            return line

        def remove_profile(self, id_):
            # self.update_session()

            self.cur.execute(f"DELETE FROM
{config.shinshila_table_name} WHERE _id='{id_}';")
            self.conn.commit()

        def update_profile(self, id_, **kwargs):
            # self.update_session()
```

```
            pairs = self._get_update_pairs(**kwargs)
            query = f"UPDATE {config.shinshila_table_name} SET
{pairs} WHERE _id='{id_}';"
            self.cur.execute(query)
            self.conn.commit()

        def _get_update_pairs(self, **kwargs):
            pairs = list()
            for key, value in kwargs.items():
                if not value:
                    continue
                pairs.append(f"{key} = '{value}'")

            return ', '.join(pairs)

        def get_data(self, records=False):
            # self.update_session()

            query = f"SELECT * FROM
{config.shinshila_table_name};"
            data_df = pd.read_sql_query(query, self.conn)

            if records:
                return data_df.to_dict('records')
            else:
                return data_df

        def get_fields(self, records=True):
            # self.update_session()

            query = f"SELECT column_name, data_type FROM
information_schema.columns WHERE table_name =
'{config.shinshila_table_name}';"
```

```
            fields = pd.read_sql_query(query,
self.conn)['column_name'].tolist()
            return fields

        def get_data_by_id(self, id_):
            # self.update_session()

            query = f"SELECT * FROM {config.shinshila_table_name}
WHERE _id = '{id_}';"
            return pd.read_sql_query(query, self.conn)

        def get_data_by_family_id(self, family_id):
            query = f"SELECT * FROM {config.shinshila_table_name}
WHERE family_id = {family_id}"
            return pd.read_sql_query(query, self.conn)
```

Лістинг файлу utils.py

```
from datetime import datetime

def calculate_in_family_id(family_data, sex):
    in_family_ids = family_data['in_family_id'].tolist()
    for i in range(1 if sex == 'Male' else 2, 999, 2):
        if i not in in_family_ids:
            return i

def create_id(data):
    profile_id = list()

    profile_id.append(str(data['family_id']))
    profile_id.append(str(data['in_family_id']))

    birth_date = datetime.strptime(data['birth'], '%d-%m-%Y')
```

```
        birth_date_id_part                                                  =
f'{birth_date.month:02d}{str(birth_date.year)[2:]}'
        profile_id.append(birth_date_id_part)

        if 'buy_letters' in data and data['buy_letters']:
            profile_id.append(data['buy_letters'])
        if 'buy_region' in data and data['buy_region']:
            profile_id.append(data['buy_region'])
        if 'buy_name' in data and data['buy_name']:
            profile_id.append(data['buy_name'])

        return '-'.join(profile_id)
```

Лістинг файлу config.py

```
db_name = 'shinshila_db'
db_username = 'python'
db_password = 'python'
db_host = 'localhost'

users_table_name = 'users_db'

shinshila_table_name = 'person_table'
ext = 'CREATE EXTENSION IF NOT EXISTS "uuid-ossp";'
shinshila_table_query  = '''CREATE TABLE {} (
    _id uuid DEFAULT uuid_generate_v4(),

    profile_id VARCHAR(100),
    family_id INT NOT NULL,
    sex VARCHAR(50) NOT NULL,
    fur_color VARCHAR(50),
    birth VARCHAR(50) NOT NULL,
    alive VARCHAR(50) NOT NULL,
    in_family_id INT,
```

```
    birth_weight FLOAT,
    buy_letters VARCHAR(50),
    buy_region VARCHAR(50),
    buy_name VARCHAR(50),
    jigging_date VARCHAR(50),
    death_date VARCHAR(50),
    father_id VARCHAR(100),
    mother_id VARCHAR(100),
    image_url VARCHAR(50),
    _image_hash VARCHAR(100),

    PRIMARY KEY(_id)
    );'''.format(shinshila_table_name)

main_insert_params = 'id, family_id, sex, birth'

users_table_query = '''CREATE TABLE {} (
    login VARCHAR(100),
    password_hash VARCHAR(100),
    api_token VARCHAR(100),

    PRIMARY KEY (login)
    )'''.format(users_table_name)

auth_fields = ['login', 'api_token', 'e', 'n']
```

Додаток В

Код мобільного застосунку для обліку ферми шиншил

Лістинг файлу Add.dart

```
import 'package:flutter/material.dart';
import 'package:shinsh_app/DataManager.dart';
import 'package:shinsh_app/EditFields/EditFieldsFactory.dart';
import 'package:shinsh_app/EditFields/FieldType.dart';
import 'package:shinsh_app/config.dart';

import 'EditFields/FieldController.dart';



class AddWidget extends StatefulWidget{
  final DataManager dataManager;
  final EditFieldsFactory editFieldsFactory = EditFieldsFactory();

  AddWidget({this.dataManager});

  @override
  State<StatefulWidget> createState() {
    return _AddWidgetState();
  }
}

class _AddWidgetState extends State<AddWidget>{
  Map<String, FieldController> _fieldsControllers = {};
  Map<String, dynamic> _savedTexts = {};
  Map<String, String> _fieldsErrors = {};
```

```
        List<Widget> listWidgets;

        @override
        void initState() {
          listWidgets = [];
          for (var i = 0; i < allFields.length; i++){
            Widget fieldWidget =_valueRowBuilder(i);
            if (fieldWidget != null)
              listWidgets.add(fieldWidget);
          }
          super.initState();
        }

        @override
        Widget build(BuildContext context) {

          return Scaffold(
            appBar: AppBar(
              actions: [
                IconButton(icon:  Icon(Icons.save),  onPressed:  ()
{_save(context);})
              ],
              title: Text("Add new member")
            ),
            body: CustomScrollView(
              shrinkWrap: false,
              slivers: [
                SliverList(
                  delegate: SliverChildListDelegate(
                    listWidgets,
                    addAutomaticKeepAlives: true,
                  ),
                ),
              ],
            )
```

```
          );
        }

        Widget _valueRowBuilder(int index) {
          String fieldName = fieldsConfig.keys.toList()[index];
          if (fieldName.startsWith("_"))
            return null;

          var fieldData = fieldsConfig[fieldName];

          if (fieldData['generated'] == true)
            return null;

          if (_savedTexts.containsKey(fieldName) == false)
            _savedTexts[fieldName] = null;
          if (_fieldsErrors.containsKey(fieldName) == false)
            _fieldsErrors[fieldName] = null;


          Map                    fieldWithController                    =
widget.editFieldsFactory.getFieldWithController(fieldData, null);

          _fieldsControllers[fieldName]                                  =
fieldWithController['controller'];

          return fieldWithController['fieldView'];
        }

        _save(BuildContext context){
          bool save = true;

          for(MapEntry<String,      dynamic>      savedEntry      in
_savedTexts.entries){
            String fieldName = savedEntry.key;
            var fieldValue = savedEntry.value;
```

```
            fieldValue = _fieldsControllers[fieldName].getValue();
            _savedTexts[fieldName] = fieldValue;

            if (fieldValue == null &&
fieldsConfig[fieldName]['required'] == true){

_fieldsControllers[fieldName].setErrorText(fieldsConfig[fieldName]['
errorText']);
              save = false;
            }
            else
              _fieldsControllers[fieldName].setErrorText(null);
          }
          if (save == true){
            Map nonEmptySavedTexts = {};
            for (MapEntry<String, dynamic> savedEntry in
_savedTexts.entries)
              if (savedEntry.value != null)
                nonEmptySavedTexts[savedEntry.key] =
savedEntry.value;

            widget.dataManager.addProfile(nonEmptySavedTexts);
            Navigator.pop(context, save);
          }
        }
      }
```

Лістинг файлу

```
import 'package:shinsh_app/EditFields/FieldType.dart';

const Map config = const {
  "endpoint": "http://10.0.2.2:6000",
  // "endpoint": "http://127.0.0.1:5000",
```

```
  "profilesDataFileName": "profiles.data",
  "profilesImagesDataFileName": "profilesImages.data",
  "imagesDir": "images"
};

const Map fieldsConfig = const {
  "family_id": {
    "type": FieldType.number,
    "required": true,
    "title": "Family ID",
    "errorText": "Please, enter Family ID"
  },

  "sex": {
    "type": FieldType.choice,
    "title": "Sex",
    "required": true,
    "errorText": "Please, enter sex",
    "choices": ["Male", "Female"],
    "dbChoices": ["M", "F"]
  },

  "fur_color": {
    "type": FieldType.text,
    "title": "Fur Color",
    "required": false,
    "errorText":
    "Please, enter fur color"
  },

  "alive": {
    "type": FieldType.choice,
    "title": "Life Status",
    "required": true,
    "errorText": "Error",
```

```
  "choices": ["Alive", "Dead"],
  "dbChoices": ["D", "A"]
},

"birth": {
  "type": FieldType.date,
  "title": "Birth",
  "required": true,
  "errorText": "Please, enter Birth"
},

"in_family_id": {
  "type": FieldType.number,
  "title": "In Family ID",
  "required": false,
  "errorText": "Please, enter In Family ID",
  "generated": true
},

"birth_weight": {
  "type": FieldType.number,
  "title": "Birth Weight",
  "required": false,
  "errorText": "Error"
},

"buy_letters": {
  "type": FieldType.text,
  "title": "Buy Letters",
  "required": false,
  "errorText": "Error"
},

"buy_region": {
  "type": FieldType.text,
```

```
  "title": "Buy Region",
  "required": false,
  "errorText": "Error"
},

"buy_name": {
  "type": FieldType.text,
  "title": "Buy Name",
  "required": false,
  "errorText": "Error"
},

"jigging_date": {
  "type": FieldType.date,
  "title": "Jigging Date",
  "required": false,
  "errorText": "Error",
  "generated": true
},

"father_id": {
  "type": FieldType.text,
  "title": "Father ID",
  "required": false,
  "errorText": "Error"
},

"mother_id": {
  "type": FieldType.text,
  "title": "Mother ID",
  "required": false,
  "errorText": "Error"
},

"profile_id": {
```

```
    "type": FieldType.text,
    "title": "ProfileId",
    "required": false,
    "errorText": "Error",
    "generated": true
  },

  "death_date": {
    "type": FieldType.date,
    "title": "Shinshila death date",
    "required": false,
    "errorText": "Error",
    "generated": false
  }


};

List allFields = fieldsConfig.keys.toList();
// List allFields = ['family_id'];
```

Лістинг файлу CryptoManager.dart

```
import 'dart:convert';
import 'dart:math';
import 'dart:typed_data';
import 'package:encrypt/encrypt.dart' as enc;
import 'package:pointycastle/export.dart';

class CryptoManager{

  RSAPublicKey publicKey;
  RSAPrivateKey privateKey;
  enc.RSA rsa;
```

```
        CryptoManager(){
          generateRSAkeyPair(bitLength: 1024);
          rsa    =    enc.RSA(publicKey:    publicKey,    privateKey:
privateKey);

        }

        String encryptMessage(String message, BigInt n, BigInt e){
          RSAPublicKey publicKey = RSAPublicKey(n, e);
          enc.RSA serverRsa = enc.RSA(publicKey: publicKey);
          enc.Encrypted                cypherText                =
serverRsa.encrypt(Utf8Encoder().convert(message));

          return cypherText.base64;

        }

        String decryptMessage(String message){
          Uint8List                decodedMessage                =
Base64Decoder().convert(message);
          enc.Encrypted              encryptedMessage              =
enc.Encrypted(decodedMessage);

          String                 decryptedMessage                =
String.fromCharCodes(rsa.decrypt(encryptedMessage));
          return decryptedMessage;
        }

        AsymmetricKeyPair<RSAPublicKey,               RSAPrivateKey>
generateRSAkeyPair({int bitLength = 2048}) {
          final _sGen = Random.secure();
          final secureRandom = SecureRandom('Fortuna');

secureRandom.seed(KeyParameter(Uint8List.fromList(List.generate(32,
(_) => _sGen.nextInt(255)))));
```

```
          final keyGen = RSAKeyGenerator()
            ..init(ParametersWithRandom(
                RSAKeyGeneratorParameters(BigInt.parse('65537'),
bitLength, 64),
                secureRandom));

          final pair = keyGen.generateKeyPair();

          publicKey = pair.publicKey as RSAPublicKey;
          privateKey = pair.privateKey as RSAPrivateKey;
        }

        Map addPublicKeyToMap(Map targetMap){
          targetMap['n'] = publicKey.modulus.toString();
          targetMap['e'] = publicKey.exponent.toString();
          return targetMap;
        }

        Map encryptMap(Map targetMap, n, e){
          Map encryptedMap = {};

          targetMap.forEach((key, value) {
            if (key == 'image')
              encryptedMap[key] = value;
            else
              encryptedMap[key]  =  encryptMessage(jsonEncode(value),
n, e);
          });

          encryptedMap = addPublicKeyToMap(encryptedMap);
          return encryptedMap;
        }

        Map decryptMap(Map targetMap){
```

```
    Map decryptedMap = {};

    targetMap.forEach((key, value) {
      if (['n', 'e', 'image'].contains(key))
        decryptedMap[key] = value;
      else
        decryptedMap[key] = jsonDecode(decryptMessage(value));
    });

    return decryptedMap;
  }


}
```

Лістинг файлу DataManager.dart

```
import 'dart:convert';
import 'dart:io';

import 'package:path_provider/path_provider.dart';
import 'package:shinsh_app/ImagesManager.dart';
import 'package:shinsh_app/NetworkManager.dart';
import 'package:uuid/uuid.dart';

import 'config.dart';

class DataManager{
  NetworkManager networkWorker;
  ImagesManager imagesManager;
  List _profilesData = [];
  var uuid = Uuid();

  DataManager({this.networkWorker, this.imagesManager});
```

```
        Future<List> _getProfilesData() async {
          await _readData();
          bool updatesAvailable = await
networkWorker.checkUpdates();
          if (updatesAvailable)
            await _downloadData();

          return _profilesData;
        }

        Future<bool> dataUpdated() async {
          await _getProfilesData();
          return true;
        }

        int getProfilesCount() {
          if (_profilesData != null)
            return _profilesData.length;
          else
            return 0;
        }


        Map getProfileByIndex(int index) {
          if (_profilesData != null)
            return _profilesData[index];
          else
            return null;
        }

        int getProfileById(String _id){
          int idx = 0;
          for (Map profileData in _profilesData){
            if (profileData['_id'] == _id)
              return idx;
```

```
            idx ++;
          }
        }

        _downloadData() async{
          _profilesData = await networkWorker.getData();
          _saveData();
        }

        _readData() async{
          try {
            final directory = await
getApplicationDocumentsDirectory();
            final file =
File('${directory.path}/${config['profilesDataFileName']}');
            String text = await file.readAsString();
            _profilesData = json.decode(text);
            print('Loaded data from local storage');
          } catch (e) {
            print("Couldn't read data file \n ERROR: $e");
            return null;
          }
        }

        void _saveData() async{
          final directory = await
getApplicationDocumentsDirectory();
          final file =
File('${directory.path}/${config['profilesDataFileName']}');
          String text = json.encode(_profilesData);
          await file.writeAsString(text);
        }

        void addProfile(Map profileData) async{
          profileData['_id'] = uuid.v4();
```

```
    _profilesData.add(profileData);
    _saveData();
    networkWorker.addData(profileData);
  }

  void updateProfile(Map updatedProfileData) async {
    for (int i = 0; i < _profilesData.length; i++){
      Map profileData = _profilesData[i];
      if (profileData['_id'] == updatedProfileData['_id'])
        _profilesData[i] = updatedProfileData;
    }
    _saveData();
    networkWorker.updateData(updatedProfileData);
  }

  void deleteProfile(int profileIndex){
    String profileId = getProfileByIndex(profileIndex)['_id'];
    _profilesData.removeAt(profileIndex);
    networkWorker.deleteProfile(profileId);
  }
}
```

Лістинг файлу GenerateQRView.dart

```
import 'dart:io';
import 'dart:ui';

import 'package:flutter/cupertino.dart';
import 'package:flutter/material.dart';
import 'package:path_provider/path_provider.dart';
import 'package:qr_flutter/qr_flutter.dart';
import 'package:share/share.dart';

class GenerateQRView extends StatefulWidget{
  final String _id;
```

```
        GenerateQRView(this._id);

        @override
        State<StatefulWidget>            createState()            =>
_GenerateQRViewState();
      }

      class _GenerateQRViewState extends State<GenerateQRView>{


        @override
        Widget build(BuildContext context) {

          QrImage qrImage = QrImage(
            data: widget._id,
            backgroundColor: Colors.white,
            size: 250,
          );

          return Scaffold(
              appBar: AppBar(
                  title: Text("Generate QR"),
              ),
            body: Container(
              width: double.infinity,
              child: Column(
                crossAxisAlignment: CrossAxisAlignment.center,
                mainAxisAlignment: MainAxisAlignment.center,
                children: <Widget>[
                  Padding(padding:     EdgeInsets.all(15),     child:
qrImage),
                  Row(
                    mainAxisAlignment:
MainAxisAlignment.spaceAround,
```

```
                    children: [
                      OutlinedButton(
                        child: Text("Share"),
                        onPressed: () {
                          _shareQr();
                        },
                        style: TextButton.styleFrom(
                            primary:
Theme.of(context).textTheme.button.color,
                            backgroundColor:
Theme.of(context).accentColor
                        ),
                      ),
                    ],
                  )
                ],
              ),
            )
          );
        }

        _createQrPicture()async{
          final qrValidationResult = QrValidator.validate(
            data: widget._id,
            version: QrVersions.auto,
            errorCorrectionLevel: QrErrorCorrectLevel.L,
          );

          final qrCode = qrValidationResult.qrCode;

          final painter = QrPainter.withQr(
            qr: qrCode,
            color: const Color(0xFF000000),
            gapless: true,
```

```
            embeddedImageStyle: null,
            embeddedImage: null,
          );

          final picData = await painter.toImageData(2048, format:
ImageByteFormat.png);

          Directory tempDir = await getTemporaryDirectory();
          String tempPath = tempDir.path;
          final                          ts                          =
DateTime.now().millisecondsSinceEpoch.toString();
          String path = '$tempPath/$ts.png';

          final buffer = picData.buffer;
          await File(path).writeAsBytes(
              buffer.asUint8List(picData.offsetInBytes,
picData.lengthInBytes)
          );

          return path;
        }

        _shareQr() async{
          String qrPath = await _createQrPicture();
          await Share.shareFiles(
              [qrPath],
              mimeTypes: ["image/png"],
              subject: 'My QR code',
              text: 'Please scan me'
          );
        }
      }
```

Лістинг файлу NetworkManager.dart

```
import 'dart:convert';
import 'dart:io';

import 'package:charts_flutter/flutter.dart';
import 'package:decimal/decimal.dart';
import 'package:shared_preferences/shared_preferences.dart';
import 'package:shinsh_app/CryptoManager.dart';
import 'package:shinsh_app/config.dart';
import 'package:http/http.dart' as http;
import 'package:connectivity/connectivity.dart';

class NetworkManager {
  String url = config["endpoint"];
  String lastUpdateId = '2';
  int lastOfflineId = 0;
  List<List> offlineRequests = [];
  CryptoManager cryptoManager = CryptoManager();
  Map<String, BigInt> serverPubKey = {};


  NetworkManager(){

Connectivity().onConnectivityChanged.listen(connectionStatusChanged)
;
  }

  Map<String, BigInt> _processServerPublicKey(String n, int
e){
    n = n.replaceAll("k: ", "");

    BigInt processedN = BigInt.tryParse(n);
    BigInt processedE = BigInt.from(e);

    return {'n': processedN, 'e': processedE};
  }
```

```
        getServerPublicKey() async{
          String pubKeyUrl = url + '/get_pub_key';
          if (await checkServer()) {
            http.Response response = await http.get(pubKeyUrl);

            Map responseMap = jsonDecode(response.body);
            serverPubKey = _processServerPublicKey(responseMap['n'],
responseMap['e']);
          }
        }

        login(String loginValue, String password) async{
          final prefs = await SharedPreferences.getInstance();
          if (await checkServer()) {
            await getServerPublicKey();
            Map loginData = {'login': loginValue, 'password_hash':
password};
            Map                encryptedLoginData                =
cryptoManager.encryptMap(loginData,              serverPubKey['n'],
serverPubKey['e']);

            String loginUrl = url + '/login';
            http.Response loginResponse = await http.post(loginUrl,
body: json.encode(encryptedLoginData));
            if (loginResponse.statusCode == 200) {
              Map loginResponseMap = jsonDecode(loginResponse.body);
              Map            decryptedLoginResponseMap            =
cryptoManager.decryptMap(loginResponseMap);
              serverPubKey                                        =
_processServerPublicKey(decryptedLoginResponseMap['n'],
decryptedLoginResponseMap['e']);
              prefs.setString('login', loginValue);
              prefs.setString('apiToken',
decryptedLoginResponseMap['api_token']);
```

```
            return true;
          }
          else
            return false;
        }
      }

      register(String newUserLogin, String newUserPassword) async{
        String registerUrl = url + "/register";
        if (await checkServer()) {
          Map data = {"new_user_login": newUserLogin,
"new_user_password_hash": newUserPassword};
          http.Response response = await makeRequest(registerUrl,
data);
          if (response.statusCode == 200)
            return true;
        }
      }

      Future<Map> signRequest(Map requestData) async{
        final prefs = await SharedPreferences.getInstance();
        requestData['login'] = prefs.getString('login');
        requestData['api_token'] = prefs.getString('apiToken');

        return requestData;
      }

      Future<http.Response> makeRequest(String url, requestData)
async{
        await getServerPublicKey();
        Map signedRequestData = await signRequest(requestData);
        Map encryptedRequestData =
cryptoManager.encryptMap(signedRequestData, serverPubKey['n'],
serverPubKey['e']);
```

```
        http.Response response = await http.post(url, body:
jsonEncode(encryptedRequestData));
        return response;
      }

      connectionStatusChanged(ConnectivityResult result) async {
        if (result != ConnectivityResult.none)
          executeChangesList();
      }

      executeChangesList() {
        for (List request in offlineRequests){
          request[0](request[1]);
        }
        offlineRequests = [];
      }

      Future<bool> checkServer() async{
        String statusUrl = url + '/status';
        bool serverWorking = false;
        try {
          http.Response response = await http.get(statusUrl);
          if (response.statusCode == 200)
            serverWorking = true;
        }
        on SocketException{
          serverWorking = false;
        }
        catch(e){
          print("Another network error");
          print(e);
          serverWorking = false;
        }
        return serverWorking;
      }
```

```
        void addData(Map data) async {
          String addUrl = url + "/add";
          if (await checkServer()) {
            http.Response response = await makeRequest(addUrl,
data);
            if (response.statusCode == 200)
              lastUpdateId = response.body;
          }
          else
            offlineRequests.add([addData, data]);
        }

        void updateData(Map data) async{
          String updateUrl = url + "/update";
          if (await checkServer()){
            http.Response response = await makeRequest(updateUrl,
data);
            lastUpdateId = response.body;
          }
          else
            offlineRequests.add([updateData, data]);
        }

        Future getFields() async {
          String getFieldsUrl = url + "/get_fields";
          if (await checkServer()) {
            http.Response response = await makeRequest(getFieldsUrl,
{});
            List fields = json.decode(response.body);
            return fields;
          }
          else
            return null;
        }
```

```
        Future getData() async {
          String getDataUrl = url + "/get";
          if (await checkServer()) {
            http.Response response = await makeRequest(getDataUrl,
{});
            return json.decode(response.body);
          }
          else
            return null;
        }

        Future<bool> checkUpdates() async {
          String getUpdatesInfoUrl = url + "/get_update_id";
          if (await checkServer()) {
            http.Response response = await
makeRequest(getUpdatesInfoUrl, {});
            Map data = json.decode(response.body);
            if (data['updateId'] != null && data['updateId'] !=
lastUpdateId)
              return true;
            else
              return false;
          }
          else
            return false;
        }

        uploadProfileImage(data) async{
          if (await checkServer()) {
            String imageUploadUrl = url + "/upload_profile_image";
            http.Response response = await
makeRequest(imageUploadUrl, data);
            lastUpdateId = response.body;
          }
```

```
          else
            offlineRequests.add([uploadProfileImage, data]);
        }

        Future<String> downloadProfileImage(String profileId) async
{
          if (await checkServer()) {
            String imageDownloadUrl = url +
"/download_profile_image";
            Map data = {"_id": profileId};
            http.Response response = await
makeRequest(imageDownloadUrl, data);
            return json.decode(response.body)['image'];
          }

          else
            return null;
        }

        Future<String> getProfileImageHash(String profileId) async {
          if (await checkServer()) {
            String getProfileImageHashUrl = url +
"/get_profile_image_hash";
            Map data = {"_id": profileId};
            http.Response response = await
makeRequest(getProfileImageHashUrl, data);
            if (response.statusCode == 200)
              return json.decode(response.body)['hash'];
          }
          else
            return null;
        }

        Future<bool> deleteProfile(String profileId) async {
          if (await checkServer()){
```

```
            String deleteProfileUrl = url + "/delete_profile";
            Map data = {"_id": profileId};
            http.Response response = await
makeRequest(deleteProfileUrl, data);
            if (response.statusCode == 200)
              return true;
          }
          return false;
        }

        Future<Map> getBirthData() async{
          if (await checkServer()){
            String getBirthDataUrl = url + '/get_born_stats';
            http.Response response = await
makeRequest(getBirthDataUrl, {});
            if (response.statusCode == 200)
              return jsonDecode(response.body);
          }
        }

        Future<Map> getTopFamiliesData() async{
          if (await checkServer()){
            String getBirthDataUrl = url + '/get_most_profitable';
            http.Response response = await
makeRequest(getBirthDataUrl, {});
            if (response.statusCode == 200)
              return jsonDecode(response.body);
          }
        }

        Future<int> getJiggingCount() async{
          if (await checkServer()){
            String getBirthDataUrl = url + '/get_jigged_count';
            http.Response response = await
makeRequest(getBirthDataUrl, {});
```

```
            print(response.body);
            if (response.statusCode == 200)
              return int.parse(response.body);
          }
        }
      }
```

Лістинг файлу Statistics.dart

```
      import 'package:flutter/material.dart';
      import
'package:flutter_staggered_grid_view/flutter_staggered_grid_view.dar
t';
      import 'package:pie_chart/pie_chart.dart';
      import 'package:charts_flutter/flutter.dart' as charts;
      import 'package:shinsh_app/NetworkManager.dart';

      class StatisticsWidget extends StatefulWidget {
        NetworkManager networkManager;

        StatisticsWidget(this.networkManager);

        @override
        State<StatefulWidget>            createState()            =>
_StatisticsWidgetState();
      }


      class _StatisticsWidgetState extends State<StatisticsWidget> {
        Map<String, double> birthData;
        List<charts.Series<FamilyData, String>> topFamiliesData;
        int jiggingCount;

        @override
        void initState(){
```

```
          widget.networkManager.getBirthData().then((value)        =>
setBirthData(value));
          widget.networkManager.getTopFamiliesData().then((value)  =>
_preprocessFamiliesData(value));
          widget.networkManager.getJiggingCount().then((value)     =>
setJiggingCountValue(value));
          super.initState();
        }

        void setBirthData(Map data){
          Map<String, double> convertedData = {};

          data.forEach((key, value) {
            convertedData[key] = value.toDouble();
          });
          this.birthData = convertedData;
          setState(() {
          });
        }

        void _preprocessFamiliesData(Map data){
          List<FamilyData> tempFamiliesData = [];

          data.forEach((key, value) {
            tempFamiliesData.add(FamilyData(key, value));
          });

          topFamiliesData = [charts.Series<FamilyData, String>(
              id: "Top birth families",
              domainFn: (FamilyData sales, _) => sales.FamilyId,
              measureFn: (FamilyData sales, _) => sales.BornCount,
              data: tempFamiliesData
          )];
          print(tempFamiliesData);
          setState(() {});
```

```
  }

  void setJiggingCountValue(data){
    jiggingCount = data;
  }


  @override
  Widget build(BuildContext context) {
    return StaggeredGridView.count(
      crossAxisCount: 1,
      crossAxisSpacing: 12,
      mainAxisSpacing: 12,
      padding: EdgeInsets.symmetric(horizontal: 16, vertical:
8),
      children: [
        getJiggedCountStat(),
        getBrithDeathStatistics(),
        getTopFamiliesStats(),
      ],
      staggeredTiles: [
        StaggeredTile.extent(1, 80),
        StaggeredTile.extent(1, 300),
        StaggeredTile.extent(1, 300)
      ],
    );
  }

  Widget getLoadingCard(){
    return Container(
      child: Card(
        child: Padding(
          padding: EdgeInsets.all(10),
          child: Text("loading..."),
        ),
```

```
            ),
          );
        }

        Widget getBrithDeathStatistics(){
          if (birthData == null)
           return getLoadingCard();

          return Container(
              child: Card(
                child: Padding(
                  padding: EdgeInsets.all(10),
                  child: PieChart(
                    dataMap: birthData,
                    centerText: "Birth and Death\nCurrent month",
                    chartType: ChartType.ring,
                    colorList:  [Colors.black38,  Colors.pinkAccent,
Colors.blueAccent],
                    legendOptions:     LegendOptions(legendPosition:
LegendPosition.bottom, showLegendsInRow: true),
                  ),
                ),
                elevation: 5,
              )
          );
        }

        Widget getTopFamiliesStats(){
          if (topFamiliesData == null)
            return getLoadingCard();


          return Container(
              child: Card(
                child: Padding(
```

```
                    padding: EdgeInsets.all(10),
                    child: charts.BarChart(
                      topFamiliesData,
                      animate: true,
                      vertical: true,
                      domainAxis: charts.OrdinalAxisSpec(
                        renderSpec: charts.SmallTickRendererSpec(
                          labelStyle: charts.TextStyleSpec(
                            color: charts.MaterialPalette.white
                          )
                        )
                      ),
                      primaryMeasureAxis: charts.NumericAxisSpec(
                        renderSpec: charts.SmallTickRendererSpec(
                          labelStyle: charts.TextStyleSpec(
                            color: charts.MaterialPalette.white
                          )
                        )
                      ),
                      behaviors: [
                        charts.ChartTitle('Top family by birth count',
                            subTitle: "Current month",
                            behaviorPosition:
charts.BehaviorPosition.top,
                            titleOutsideJustification:
charts.OutsideJustification.start,
                            titleStyleSpec: charts.TextStyleSpec(
                              color: charts.MaterialPalette.white
                            ),
                            subTitleStyleSpec: charts.TextStyleSpec(
                                color: charts.MaterialPalette.white,
                                fontSize: 15
                            ),
                            innerPadding: 18),
                      ],
```

```
                )
              ),
              elevation: 5,
            )
        );
      }

      Widget getJiggedCountStat(){
        if (jiggingCount == null)
          return getLoadingCard();

        return Container(
            child: Card(
              child: Padding(
                padding: EdgeInsets.all(10),
                child: ListTile(
                    leading: Text("Jigging count\n(current month): ", style: TextStyle(fontSize: 20,)),
                    trailing: Text(jiggingCount.toString(), style: TextStyle(fontSize: 20),)
                ),
              ),
              elevation: 5,
              )
        );
      }
    }

    class FamilyData{
      final String FamilyId;
      final int BornCount;

      FamilyData(this.FamilyId, this.BornCount);
    }
```