Міністерство освіти і науки України
Національний університет «Одеська політехніка»
Інститут інформаційної безпеки, радіоелектроніки та телекомунікацій
Кафедра кібербезпеки та програмного забезпечення

Осколкова Олена Русланівна,
студентка групи РУ-181

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

Розробка мобільного додатка для зберігання та захисту персональних даних на підприємстві

Спеціальність:
125 Кібербезпека

Спеціалізація, освітня програма:
Управління кібербезпекою

Керівник:
Зорило Вікторія Вікторівна,
ст. викладач

Одеса – 2022

Міністерство освіти і науки України

Національний університет «Одеська політехніка»

Інститут інформаційної безпеки, радіоелектроніки та телекомунікацій

Кафедра кібербезпеки та програмного забезпечення

Рівень вищої освіти перший (бакалаврський)

Спеціальність 125 – Кібербезпека

Освітня програма – Кібербезпека

ЗАТВЕРДЖУЮ

Завідувач кафедри КБПЗ

_____

д.т.н.,проф.А.А.Кобозєва

___ _____2022 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

Осколковій Олені Русланівни

1.Тема роботи: *Розробка мобільного додатка для зберігання та захисту персональних даних на підприємстві*, керівник роботи *Зоріло Вікторія Вікторівна, к. ф.-м. н., доцент,*

затверджені наказом ректора від 17. 05 2022 р. № 168-в

2.Зміст роботи*: розробка мобільного додатку для зберігання та захисту персональних даних на підприємстві, аналіз сучасних методів і рішень, охорона праці.*

3. Перелік ілюстративного матеріалу: *схеми взаємодії архітектури клієнт-сервер, схема бази даних та тестування мобільного дадатку.*

4. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
Охорона праці	Ярова І.А. к.т.н., доцент	Завдання видав	Завдання рийняв

5. Дата видачі завдання "_____"_____2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання	Примітка
1	*Аналіз джерел з теми випускної кваліфікаційної роботи*	*20.03.2022*	*виконано*
2	*Обробка отриманих даних*	*26-03-2022*	*виконано*
3	*Розробка алгоритму*	*10-04-2022*	*виконано*
4	*Аналіз і вибір засобу реалізації додатку*	*15-04-2022*	*виконано*
5	*Розробка мобільного додатка*	*15-05-2022*	*виконано*
6	*Підготовка тексту роботи*	*28-05-2022*	*виконано*
7	*Підготовка презентації та доповіді*	*29-05-2022*	*виконано*
8	*Попередній захист*	*03-06-2022*	*виконано*
9	*Нормоконтроль, рецензування*	*17-06-2022*	*виконано*
10	*Занесення роботи в електронний архів*		
11	*Допуск до захисту у завідувача кафедри*		

Здобувач вищої освіти _____ *Осколкова О.Р.*

Керівник роботи _____ *Зорило В.В.*

ЗАВДАННЯ

на розробку розділу «Охорона праці» у кваліфікаційній роботі бакалавра

студенту Осколковій Олені Русланівни, група РУ-181

Інститут інформаційної безпеки, радіоелектроніки та телекомунікацій
Кафедра кібербезпеки та програмного забезпечення

Дата отримання завдання 11.05.2022

Консультації 19.05.2022, 26.05.2022

Дата закінчення розділу 15.06.2022

Тема роботи *Розробка мобільного додатка для зберігання та захисту парольних даних на підприємстві*

Зміст розділу:

1. Аналіз умов праці і вибір основних заходів виробничої безпеки
2. Аналіз пожежної безпеки і вибір заходів і засобів пожежної безпеки

Керівник дипломної роботи

_____ Зоріло В.В.

(підпис)

15. 06. 2022 р.

Консультант з охорони праці

_____ Ярова І.А.

(підпис)

15. 06. 2022 р.

.

АНОТАЦІЯ

Кваліфікаційна робота на тему: «Розробка мобільного додатка для зберігання та захисту персональних даних на підприємстві» на здобуття бакалаврського рівня вищої освіти за спеціальністю 125 - Кібербезпека та за спеціальністю: Управління кібербезпекою. Дана робота включає 19 рисунків та 1 таблицю в основному тексті, 1 додаток та 17 джерел з переліку посилань. Кваліфікаційна робота була виконана на 83 сторінках загального тексту та 61 сторіках основного тексту.

Мета роботи полягає у розробці мобільного застосунку для захисту та зберігання персональних даних на підприємтсві та удосконалення сучасних рішень даного питання. У роботі був проведений аналіз сучасних подібних додатків. У результаті аналізу було виявлено, що кожне рішення має свої недоліки та жодне з них не відповідає повністю поставленому завданню.

У результаті роботи був реалізований програмний застосунок. Після проведення тестування даного застосунку, було виявлено, що розроблений додаток виконує всі завдання, які поставлені в даній кваліфікаційній роботі.

МОБІЛЬНИЙ ДОДАТОК, ДОДАТОК ДЛЯ ЗБЕРЕЖЕННЯ ТА ЗАХИСТУ ДАНИХ НА ПІДПРИЄМТСВІ, МОБІЛЬНИЙ ДОДАТОК ДЛЯ ЗБЕРЕЖЕННЯ ІНФОРМАЦІЇ.

ABSTRACT

This qualification work on the topic: "Development of a mobile application for storage and protection of personal data at the enterprise" for a bachelor's degree in higher education in the specialty 125 - Cybersecurity and in the specialty: Cybersecurity Management. Does this work include 19 figures and 1 table in the main text, 1 appendix and 17 references. Qualification work was performed on 83 general text pages and 61 pages of the main text.

The purpose of this work is to develop a mobile application for the protection and storage of personal data in the enterprise and improve modern solutions to this issue. The analysis of modern similar applications was carried out in the work. The analysis revealed that each decision has its shortcomings and none of them fully meets the task.

As a result, the software application was implemented. After testing this application, it was found that the developed application performs all the tasks set in this qualification work.

MOBILE APPLICATION, APPENDIX FOR STORAGE AND PROTECTION OF DATA AT THE ENTERPRISE, MOBILE APPENDIX FOR STORAGE OF INFORMATION.

ЗМІСТ

ВСТУП

У сучасному світі дуже часто постає питання, щодо збереження та захисту парольних даних. Існує чимало сервісів та інтернет-ресурсів, до яких потрібно мати певні доступи, аби потрапити до системи. На даний момент існують рішення, які закривають дане питання, але в них деякі недоліки.

Мета даної кваліфікаційної роботи полягає у розробці подібного рішення, яке буде виконане для підприємства, а також у вдосконалені існуючих рішень, враховуючи потреби підприємства. Завдання які вирішуватимуться в даній роботі полягають у наступному:

1) Огляд та аналіз сучасних рішень
2) Розробка алгоритму роботи програмного додатку
3) Реалізація програмного додатку

Результатом даної кваліфікаційної роботи є програмний застосунок, який виконує такі функції як:

- збереження та захист даних;
- можливість створення записів та ведення їх обліку;
- реалізація системи нагадування.

Усі отримані результати представлено в даній кваліфікаційній роботі.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Проблематика. Інформаційна безпека

Інформаційна безпека – це такий комплекс заходів, що спрямовується на забезпечення інформаційної безпеки. Головна проблема інформаційної безпеки включає в себе багато аспектів та через те її забезпечення є дуже складним процесом, який включає в себе ряд важливих питань та завдань. Такі проблеми інформаційної безпеки весь час ускладнюються процесами проникнення у всі сфери суспільства технічними засобами обробки та передачі даних, особливо комп'ютерними системами [1].

В кожній системі та програмі існують спеціальні заходи щодо їх забезпечення від несанкціонованого входу. Основною метою будь-якої системи інформаційної безпеки є забезпечення сталого функціонування об'єкта: запобігання загрозам його безпеці, захист законних інтересів власника інформації від протиправних порушень, у тому числі злочинних діянь у сфері відносин, забезпечення нормальної оперативної діяльності всіх підрозділів об'єкта.

За законом України існує поняття грифу секретності. Як зазначено в законодавстві гриф секретності - це реквізит матеріального носія секретної інформації, що засвідчує ступінь секретності даної інформації. Існують наступні ступені секретності: "особливої важливості", "цілком таємно", "таємно". Власне ступінь секретності - це категорія, яка характеризує важливість секретної інформації, ступінь обмеження доступу до неї та рівень її охорони державою [2].

У ході виконання поставленого завдання також враховувався технічний захист секретної інформації. Це такий вид захисту, що напрямлений на забезпечення інженерно-технічними заходами конфідеційності, цілісності та унеможливлення блокування інформації.

У даному випадку найважливішим є захист інформаційних даних на підприємстві, оскільки довіра клієнтів заснована насамперед на наданні їх персональних даних і, відповідно, на захист персональних даних співробітниками організації.

Тому метою роботи є розробка безпечної політики та забезпечення захисту інформації в компанії для її нормального функціонування.

1.2 Важливість збереження та захисту даних від несанкціонованого доступу на підприємстві

На сьогоднішній день інформаційні технології охопили майже всі сфери економіки та життєдіяльності. Саме тому для кожної сучасної компанії інформація є одним із головних чинників та ресурсів. Зберігання, захист і правильне використання інформацією є ключовим аспектом розвитком бізнесу, а також зниження рівню багатьох ризиків, з якими можливо зіштовхнутися. Отже, актуальною проблематикою для організацій та підприємств є забезпечення інформаційної безпеки.

Під інформаційною безпекою на підприємстві можна розуміти певний комплекс мір організаційного та технічного характеру, який потрібен для захисту та зберігання інформації та її ключових елементів. Під ключовими елементами інформації розуміють корпоративні системи, обладнання, які призначені для роботи з інформацією, а саме: її збереження, обробки та передачі.

Забезпечення інформаційної безпеки на підприємстві допомагає захистити конфідеційну інформацію та інформаційну інфраструктуру організації від негативних впливів та наслідків. Такі негативні впливи можуть бути випадковими, зроблені навмисно або вони також можуть мати зовнішній характер. Як результатом подібних втручань може стати втрата, несанкціоноване змінення або використання третіми особами важливих даних або конфідеційної інформації. Саме тому інформаційна безпека є дуже

важним аспектом для захисту підприємства та забезпечення його беперервної роботи.

Інформаційна структура організації постійно може піддаватися баготочисельним загрозам, які в свою чергу можуть поділятися на декілька видів [3]:

1) Природні загрози – такі загрози, як правило, можуть виникати незалежно від людини. До таких загроз відносять природні катаклізми.
2) Штучні загрози – подібні загрози виникають уразі створення комплексу загроз, які створила людина. Такі загрози, зазвичай, можна поділити за характером дії, а саме: ненавмисні та навмисні. До навмисних загроз входять дії, що зроблені людиною цілеспрямовано. Це може бути крадіжка, навмисне змінення та знищення конфіденційної інформації підприємства. До ненавмисних загроз відносять дії людини, які були вчинені за причиною їх недостатньої компетенції або за необережністю чи випадково.
3) Внутрішні загрози – загрози, які виникають у середині інформаційної інфроструктури підприємства.
4) Зовнішні загрози – дані загрози можуть виникати за межами інформаційної інфроструктури організації.

Також загрози інформаційної безпеки за залежністю від характера загроз поділяються на пасивні та активні. До активних загроз відносяться такі фактори впливу, що можуть змінювати структуру інформації або її зміст. У той час, як пасивні загрози не здатні вносити будь-які зміни. Насамперед, головну небезпеку представляють штучні загрози, адже через втрату або крадіжку конфідеційних даних, підприємство може мати негативні наслідки, а також у результаті понести фінансові втрати.

1.3 Огляд та аналіз існуючих мобільних додатків, що призначені для забезпечення та захисту парольних даних

Перед початком роботи над створенням нового мобільного додатку, необхідно розглянути вже існуючі рішення та проаналізувати їх.

На сьогоднішній день існує безліч систем, які мають певну систему авторизації. Важливою метою є зберігання та забезпечення захисту тих даних, що стосуються авторизації певної людини чи підприємства.

До основної мети забезпечення та зберігання даних відноситься наявність усіх доступів до систем, які має певна людина чи організація. Важливо зберігати усю інформацію, щодо авторизації до різних систем. Адже втрата конфіденційної інформації конкретного бізнесу або підприємтсва може призвести до дуже тяжких та іноді до незворотних наслідків. Для таких цілей вже розроблені певні рішення. Далі їх розглянемо.

Існують наступні програми, які надають можливість користувачам зберігати свої парольні дані [4]:

1) LastPass

Сумісніть: iOS , Mac, Linux, Windows, Android.

Основні функції та переваги додатку:

- 256-бітове шифрування AES
- PBKDF2 SHA-256 та солоні хеши
- Автозаповнення паролем
- Редагування пароля
- Генератор паролів
- Безпечний пароль і спільний доступ до нотаток
- Пошук паролем або сайтом
- Масове додавання та збереження паролей
- Система авторизації

Недоліки:

- Базові установки генератора паролів могли бути більш захищеними.

- висока вартість ліцензії;

LastPass зберігає паролі та особисту інформацію у безпечному сховищі. Крім запису паролей, сервіс допомагає згенерувати більш надійне кодове слово.

2) Dashlane

Сумісність: iOS , Mac, Linux, Windows, Android.

Основні функції та параметри додатку:

- Автоматичне заповнення паролей
- Масове додавання та збереження паролей
- 256-бітове шифрування AES
- Генератор паролів
- Можливість внесення додаткової інформації щодо збережених паролів
- Система авторизації
- Можливість редагування паролів
- Хмарний контейнер

Переваги:

- Кросплатформеність
- Автоматичний контроль термінів старіння паролів
- Багатофакторна автентифікація та автозахоплення
- Синхронізація паролів та їх генерація
- Надсилання повідомлень про необхідність змінити дані для доступу
- Можливість встановити новий персональний пароль

Недоліки:

- висока вартість ліцензії;
- "хмарний" контейнер доступний тільки у платній версії.

3) 1Password (інтерфейс краще, ніж у ластпасс)

Сумісність: iOS , Mac, Linux, Windows, Android.

Основні функції та параметри додатку:

- Висока надійність

- Наявність вбудованого генератора паролів
- Можливість створення резервних копій
- Гібридне AES-шифрування з 128 або 256-бітної маскою
- Зберігання та редагування паролів
- Зберігання додаткової важливої інформації

Програма зберігання паролів 1Password допомагає підключатися до комп'ютера віддалено. Синхронізує пристрої за допомогою єдиного облікового запису на смартфонах та ПК.

Недоліки:

- Додаток не є безкоштовним

4) Bitwarden – перефразировать пункты

Сумісність: iOS , Mac, Linux, Windows, Android.

Основні функції та параметри додатку:

- Bitwarden забезпечує багатофакторну автентифікацію шляхом інтеграції з Duo Security.
- Він забезпечує 1 ГБ зашифрованого файлового сховища з преміум-планами.
- Він може надійно синхронізуватися з хмарою, щоб дозволити вам отримати доступ до даних з будь-якого місця за допомогою будь-якого пристрою.
- Він використовує наскрізне шифрування AES-256 біт, солене хешування та PBKDF2 SHA-256.
- Використовуючи потужні інструменти командного рядка є можливість писати та виконувати сценарії у сховищі Bitwarden.

Недоліки:

- Додаток має відкрите джерело. Це означає, що будь-яка людина може проаналізувати цей код та виявити вразливості.

Програма Bitwarden допомагає зберігати та обмінюватися конфіденційними даними з будь-якого пристрою. Програмою користуються по всьому світу, тому що вона працює 40 мовами. У сервісі можна

створювати унікальні паролі, а постійний аудит безпеки робить Bitwarden надійним менеджером для зберігання паролів.

5) Keeper

Сумісніть: iOS , Mac, Linux, Windows, Android.

Основні функції та параметри додатку. Keeper надає персональне сховище кожному користувачу для того, щоб зберігати та управляти власними паролями, файлами, обліковими даними тощо. Дане сховище представлене у зашифрованому вигляді. Існує можливість автоматичного заповнення паролів. Кількість даних, які можна зберігати у додатку, не обмежуються. Він надає функції консолі адміністратора, контроль версій, рольовий доступ та історію записів та звітування.

Програма спрямована на те, щоб зберігати паролі та особисту інформацію. Keeper працює на будь-якому мобільному пристрої та ПК, де можливе створення власного зашифрованого сховища. Сервіс швидко вигадує нові та надійні паролі та одночасно застосовує їх до підключених облікових записів у додатках та на веб-сайтах. Крім паролів, у менеджері зберігають паспортні дані, посвідчення водія, важливі фотографії, документи та інші типи файлів.

У ході аналізу існуючих аналогів та у процесі опрацювання табл. 1.1 можна зробити висновок, що з обраних для аналізу мобільних додатків, які є одними з найкращих рішень на даний момент, не має жодного лідера, який мав би усі перелічені функції. А також немає функціоналу для можливості створення нагадувань власне користувачем. У такому разі, найбільш оптимальним рішенням є розробка власного мобільного додатку, що буде включати основні функції та завдання подібних додатків та впровадження удосконалення шляхом розробки системи нагадувань.

Таблиця 1.1 – Порівняльний аналіз існуючих аналогів

Функції	Назва мобільного додатку				
	LastPass	Dashlane	1Password	Bitwarden	Keeper
Шифрування даних	+	+	+	+	+
Система авторизації	+	+	+	+	+
Збереження та редагування паролей	+	+	+	+	+
Створення резервних копій	-	-	+	-	+
Можливість внесення додаткової інформації щодо збережених паролів	+	+	+	-	-
Генератор паролів	+	+	-	-	-
Можливість створювати нагадування	-	-	-	-	-
Зручний та простий інтерфейс	-	+	+	-	+
Сучасний дизайн	-	+	+	+	+
Автономність	-	+	-	+	+

1.4 Обґрунтування необхідності та мети створення додатка

Метою даної роботи є створення мобільного додатку, який надає можливість користувачу зберігати усі парольні дані та доступи до певних сервісів та систем. Для захисту даних клієнта буде використовуватися шифрування даних, яке буде створюватись на стороні бази даних. У першу чергу завдання полягає у захисті парольних даних користувача, які потрібні для входу у застосунок, де в подальшому буде зберігатись інформація щодо парольних даних. З цією метою буде виконуватися шифрування паролю для входу в аккаунт конкретного користувача. Таким чином буде неможливо виявити пароль для входу, навіть отримавши доступ до бази даних системи. У зв’язку з тим, що на даний момент вже існують подібні рішення, було вирішено впровадити функціонал, що удосконалить існуючі системи. Насамперед, цим удосконаленням буде виступати функціонал нагадування з

використанням додатково системи оповіщування. У такому випадку у користувача буде можливість створювати нагадування, які будуть приходити у вигляді push-оповіщень.

Даний функціонал потрібен, адже для того, щоб підтримувати онлайн інфраструктуру бізнесу потрібно і важливо вчасно регулювати усі заплановані процеси, які є у конкретного підприємства. Система нагадувань спрямована на те, щоб у користувача була можливість керувати та слідкувати за доступами та подіями в певних системах. Це може бути нагадування про зміну пароля, або про заплановане створення доступу, або нагадування про необхідність виконання певних дії в конкретній системі. У рамках підприємства даний функціонал також може використовуватися для передачі усіх запланованих нагадувань іншому співробітнику компанії засобом передачі даних авторизації у конкретний аккаунт співробітника. Усе це допомагає будувати якісну онлайн інфраструктуру підприємства та слідкувати за її порядком.

Дана проблема є актуальною, це можна побачити з того, що наявна велика кількість згадувань даного питання у відкритому доступі: на сайтах в Інтернеті, відкритому друці, тощо. Також при спілкуванні з підприємством, для якого розробляється додаток, була виявлена індивідуальна потреба у розробці даного застосунку та впровадження удосконалення у існуючі рішення. Саме тому було вирішено реалізувати дане завдання в кваліфікаційній роботі. В даному розділі було виконано завдання 1 з переліку завдань, поставлених в роботі для досягнення мети даної кваліфікаційної роботи.

2 ПОСТАНОВКА ЗАВДАННЯ І МЕТОДИ ДОСЛІДЖЕННЯ

2.1. Мета та завдання

Мета даної роботи полягає у тому, щоб розробити мобільний додаток для підприємства, який дозволятиме зберігати парольні дані, а також здійснюватиме створення нагадувань безпосередньо зареєстрованим у системі користувачем. Головною задачею є реалізація такого додатка. Для реалізації цієї задачі буде проведено аналіз існуючих технічних можливостей та розробка алгоритму роботи додатку. У першу чергу для цього потрібно з'ясувати які задачі повинен виконувати застосунок. До головних технічних завдань можна віднести:

- Створення бази даних, де зберігатиметься інформація користувачів. Створення таблиць та стовпців для зберігання даних. У базі даних є дві таблиці: «user», де зберігаються та записуються дані щодо авторизації, та таблиця «passwords», де знаходяться та записуються дані про записи користувача.
- Можливість авторизації та реєстрації користувачів у системі. Також для захисту даних для авторизації виконується шифрування пароля для входу в акаунт. Шифрування відбувається на стороні сервера, а дані зберігаються в базі даних у таблиці «user».
- Можливість створення та видалення записів користувача, що містять дані з доступами до різних систем, а саме: назва, логін, пароль та коментар щодо конкретного запису. Дані також мають зберігатись в одній базі даних.
- Ручне створення користувачем нагадувань. Можливість створення певного запису з інформацією про нагадування. Цей запис може створювати безпосередньо сам користувач. Для цього він повинен мати можливість занесення певної інформації до системи, а саме: назва нагадування, його опис та час, коли має прийти нагадування.

2.2 Алгоритм роботи програми

Маючи на увазі технічні завдання та мету створення додатку необхідно створити алгоритм та логіку, за якою додаток буде працювати.

Логіка роботи полягає у виконанні кожного з наступних етапів:

- Етап реєстрації нового користувача. Головна задача даного етапу – це створення такого функціоналу, за допомогою якого буде можливо реєструвати нові акаунти у системі. Для реалізації даної задачі буде використовуватись спеціально виділена база даних. На етапі реєстрації нового користувача повинна надаватись можливість заповнення користувачем спеціально створеної форми. Форма має містити кнопку та поля для вводу даних: логіна та пароля. Після того, як користувач заповнив необхідні поля та натиснув на кнопку, дані необхідно відправити на сервер. На етапі відправки даних на сервер відбувається шифрування пароля, який ввів користувач. Після цього відбувається перевірка даних спрямована на те, щоб визначити чи вже існують ті дані, що ввів користувач у форму, у базі даних. У випадку, якщо збіг значень не знайдено, потрібно створити новий запис у базу даних з інформацією, яку ввів користувач та його персональним ідентифікатором. У протилежному випадку, коли збіг введених значень знайден у базі даних, користувачу надходить повідомлення про те, що користувач із такими даними вже існує в системі та зареєструватися не можливо.
- Авторизація користувача у додатку. На даному етапі користувач повинен заповнити форму з логіном та паролем та натиснути на кнопку відправки даних на сервер. Під час відправки даних буде відбуватися шифрування паролю та вноситися новий запис у базу даних.
- Створення та обробка записів, які містять інформацію щодо доступів у різні системи. На даному етапі користувач знаходиться на сторінці, де йому надається можливість побачити усі створені ним записи та дві

кнопки: одна для додавання нового запису, інша – для виходу з акаунту. Якщо записів не має, то виводяться лише ці дві кнопки. Для виводу на сторінці існуючих записів конкретного користувача береться ідентифікатор юзера та відправляється запит до бази даних до таблиці «passwords» та відбувається вибірка тих даних, що належать користувачеві – такі дані визначаються за айді. Також можливо видалити запис за допомогою спеціальної кнопки. Під час натискання на кнопку відбувається запит до таблиці з бази даних на видалення запису, який обрав користувач.

- Функціонал нагадувань. Даний функціонал працює через можливості Android. Користувач має можливість перейти до сторінки, де знаходитимуться всі його раніше створені записи з нагадуваннями. Ті нагадування, які вже відпрацювали, тобто надійшли користувачеві у заданий час, вони відображуються як виконані –перекреслені. Ті нагадування, які заплановані ще до роботи, вони відображені як дійсні, в них вказаний час і дата, коли вони мають прийти та час і дата, коли вони були саме заплановані. Також в записі про нагадування вказана назва нагадування та коментар. Якщо у користувача немає ще нагадувань, то виводиться порожня сторінка. На даній сторінці з нагадуваннями користувачеві також надається можливість створити нагадування за допомогою кнопки, яка розташована в тулбарі. При натисканні на кнопку, користувач переходить до сторінки створення нагадування. На даній сторінці йому відображається поля для вводу назви, коментаря для майбутнього нагадування, а також є три кнопки, за допомогою яких можна обрати бажану дату (день, місяць, рік), час, хвилину та секунди. Далі, коли всі поля були введені та була обрана дата і час виконання нагадування, користувач може натиснути на кнопку «Зберегти» і перейти до сторінки з усіма нагадуваннями та побачити щойно створений запис та інші.

2.3 Архітектура мобільного клієнт-серверного додатку

На сьогоднішній день більшість мобільних додатків використовують клієнт-серверне рішення. В першу чергу слід визначити наступні терміни:

- Сервер – це віддалений комп’ютер, який призначений для запуску роботи програми. Це працює наступним чином: програма отримує запит від користувача, далі проходить обробка отриманих даних та після сервер відправляє відповідь та оновлює базу даних. Загалом можна сказати, що сервер накопичує та збирає інформацію, а також контролює та організує доступ до певної зареєстрованих користувачів.
- Клієнт – в даному випадку це додаток, що попередньо встановлений на мобільному пристрої користувача. Додаток передає та отримує інформацію з серверу за допомогою http запитів
- API (апі) – це такі запити до сервера. Ці запити визначені та прописані на етапі програмування. Завдяки API є можливість передавати або отримувати дані та записувати певну інформацію до бази даних.
- Клієнт-серверна архітектура – це така архітектура, що виконує зв’язок між додатком (тобто клієнт) та сервером за допомогою API.

У цілому дана архітектура забезпечую взаємодію між цими системами.

Архітектура "клієнт-сервер" регулює загальні принципи щодо організації взаємодії у мережі. Таким чином вона забезпечує взаємодію всіх елементів системи. Для того, щоб цю взаємодію здійснити необхідно виділити три групи функцій, що будуть орієнтовані на вирішення задачі [5]:

1) функції для вводу та відображення інформації - дані функції надають можливість користувачу взаємодіяти з програмою;
2) прикладні функції - спрямовані на реалізацію поставленої задачі;

3) та функції, що призначені для управління ресурсами (наприклад, з базою даних, файлами та інше).

Виконання перелічених функцій відбувається за допомогою програмних засобів. Їх можна представити у вигляді наступних взаємопов'язаних компонентів:

- компонент представлення - даний компонент належить до користувацького інтерфейсу;
- прикладний компонент - потрібен для реалізації алгоритму, що необхідний для вирішення поставленої задачі;
- компонент управління - необхідний для управляння ресурсом, що у результаті забезпечує доступ до потрібних ресурсів.

Якщо говорити про практичну реалізацію такої архітектури, то у такому разі дана архітектура називається клієнт-серверними технологіями [5]. Дана технологія може визначати власні правила взаємодії між клієнтом та сервером або використовувати загальні правила, що називаються протоколом обміну або проколом взаємодії.

Існують наступні види архітектури:

1) Дволанкова архітектура
2) Триланкова архітектура

Для дволанкової архітектури характерний розподіл трьох базових компонентів, перерахованих вище, між двома вузлами, а саме: між сервером та клієнтом (рис 2.1).

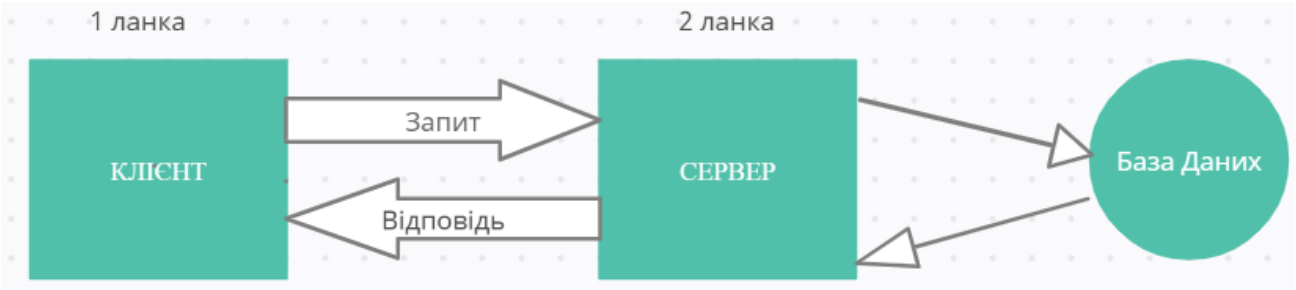

Рисунок 2.1 – Схема взаємодії дволанкової архітектури клієнт-сервер

Розташування цих компонентів на стороні сервера чи клієнта визначають основні моделі їх взаємодії. До цих основних моделей входить [6]:

- сервер терміналів - подання даних у розподіленій формі;
- файл-сервер - доступ до файлових ресурсів та бази даних;
- сервер бази даних - віддалене представлення даних;
- сервер додатку - віддалений додаток.

На сьогоднішній день, завдяки спеціально виділеним системам управління базами даних можлива реалізація такої моделі доступу до віддаленої бази даних, як модель сервера бази даних. В даному випадку ядро системи управління базами даних працює на сервері, прикладна програма на клієнті, а в свою чергу протокол обміну (або протокол взаємодії) функціонує за допомогою мови запитів SQL. У такому разі дана модель сприяє зменшенню навантаження мережі, адже вона спрямована на те, щоб швидко та оперативно виконувати певні запити до бази даних.

Також існує термін активного сервера БД. В такому випадку частина усіх функцій прикладного компонента реалізовуються у виді процедур, що зберігаються на стороні сервера. Інша частина прикладної логіки робиться на стороні клієнта. Усе це працює згідно протоколу взаємодії, що здійснюється за допомогою відповідної мови SQL.

Якщо говорити про переваги даного підходу, то вони полягають у наступному:

- існує можливість централізованого управління прикладними функціями;
- є можливість зниження вартості володіння системою, адже можливо орендувати сервер, а не купляти його;
- зниження навантаження на мережу.

Триланкова архітектура у свою чергу реалізується на базі моделі сервера додатків [7]. В даному випадку мережевий додаток поділяється на декілька чи більше частин. Виконування кожної з таких частин можливе на окремих комп'ютерах. Відмінність триланкової архітектури від дволанкової у

тому, що виділені частини додатку один з іншим утворюють взаємодію і таким чином вони створюють обмін повідомленнями у конкретному форматі (рис. 2.2).

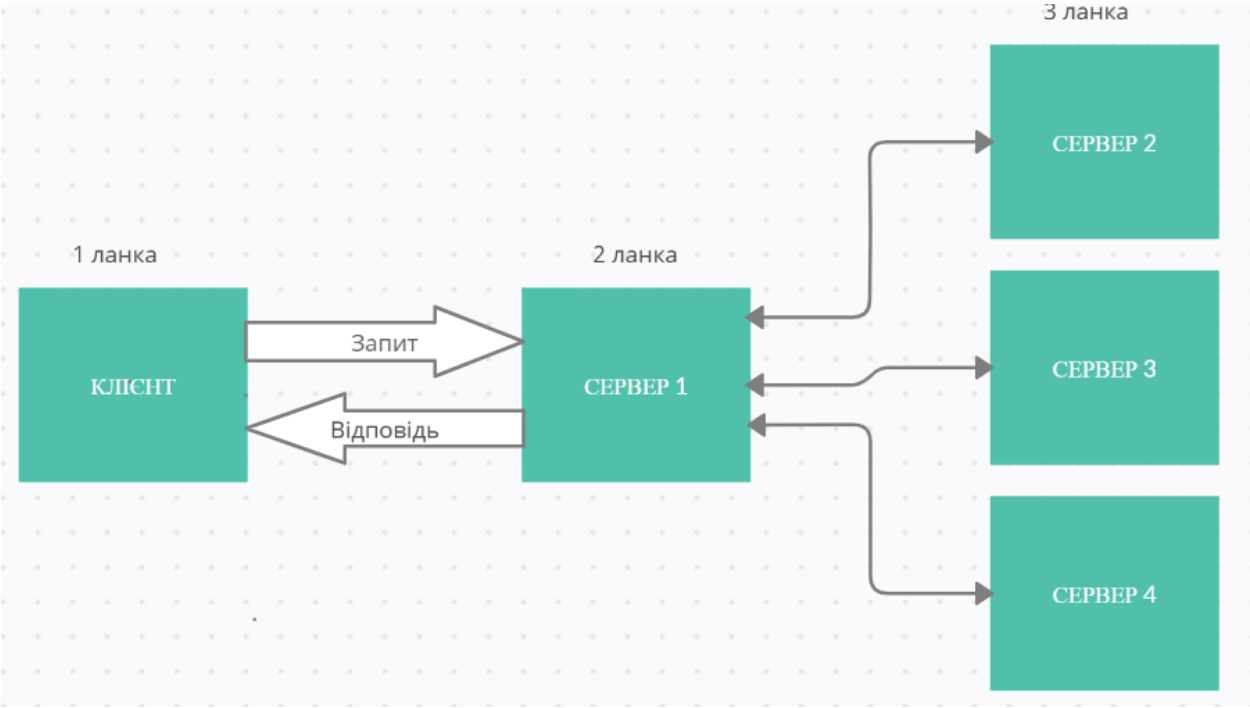

Рисунок 2.2 – Схема взаємодії триланкової архітектури клієнт-сервер

Враховуючи особливості триланкової архітектури, можна прийти до висновку, що подібна архітектура може бути розширена до багатоланкової. Це можливо у разі виділення додаткових серверів. Кожний з таких серверів може представити власні сервіси та користуватися послугами інших серверів буд-якого рівня.

У триланковій архітектурі усі компоненти поділяються наступним чином:

1) Представлення даних - працює на стороні клієнта.
2) Прикладний компонент - знаходиться на віддаленому сервері додатків;
3) Управління ресурсами - управління ресурсами відбувається на сервері бази даних. Даний сервер представляє запитуванні дані.

Якщо порівнювати ці дві архітектури (дволанкову та триланкову), то можна зробити наступні висновки:

1) Дволанкова архітектура набагато простіше за триланкову, адже усі запити в ній обробляються одним виділеним сервером.
2) Триланкова архітектура побудована більш складно, так як функції розподіляються між серверами другого та третього рівня. Саме це призводить до більшої гнучкості та маштабованості, а також до високого рівню безпеки, так як для кожного с сервісів та рівнів можна зазначити захист відповідно.

Клієнт-серверна архітектура застосовується у багатьох мережевих технологіях, що використовуються для доступу до різних сервісів (мережевих). Існують наступні види таких сервісів [7]:

- Web-сервери - підтримують розширені можливості, в тому числі й роботу з бінарними файлами.
- Сервери додатків - дані сервери слугують для централізованого вирішення прикладних завдань. У даному випадку користувачеві надається право запускати серверні програми для виконання. Використання серверів додатків надає можливість спростити управління мережею.
- Сервери баз даних - потрібні для обробки запитів користувача, запити здійснюються на мові SQL. Це працює наступним чином: клієнтський додаток підключається до сервера, на якому знаходиться система управління базами даних.
- Файл-сервери - в файл-серверах зберігаються дані у виді файлів, до яких користувач має доступ.
- Проксі-сервер - даний тип сервісу слугує для того, аби допомогти користувачеві отримати інформацію з Інтернету та одночасно сприяє захист мережі. Також проксі-сервер зберігає інформацію про запити, які відбуваються найчастіше, до кеш-пам'яті до локального диску. Це допомагає швидше відправити запитувану інформацію користувачу, використовуючи кеш-пам'ять, а не повторне звернення до Інтернету.

- Фаєрволи - це так звані міжмережеві екрани, що здатні аналізувати, а також фільтрувати мережевий трафік, що проходить. Головна його ціль полягає у забезпечені безпеки мережі.
- Почтові сервери - слугують для відправки та отримання поштових повідомлень в електронній формі.

Отже, головна мета архітектури "клієнт-сервер" полягає в тому, щоб розподілити мережевий додаток на декілька компонентів. Кожний з компонентів працює таким чином, що реалізує специфічний набір сервісів. А також компоненти такого додатку мають можливість виконуватися на різних пристроях, при цьому виконуючи клієнтські та серверні функції. Усе це допомагає збільшити рівень безпеки, надійності та продуктивність мережевих додатків, а також мережі.

В роботі буде представлена реалізація роботи додатка з використанням дволанкової архітектури клієнт-серверного додатку. Тобто, особисті дані користувача для авторизації та ведення обліку записів зберігатимуться на спеціально виділеному сервері у базі даних. За допомогою мови SQL здійснюватиметься взаємодія між клієнтом та сервером, щодо конкретної інформації. Правила, за якими буде здійснюватися спілкування клієнта з сервером, задаватимуться безпосередньо в коді програми.

Мобільний додаток – це таке програмне забезпечення, яке створюється спеціально для мобільної платформи. Серед найбільш популярних мобільних платформ існують такі: Android, iOS, Windows Phone. Мобільний додаток призначений для використання на мобільних пристроях, в залежності від мобільної платформи. Іншими словами можна сказати, що це інструмент, який розробляється для того, аби забезпечити прямий контакт з користувачем, що дозволяє користувачеві зручне та багаторазове використання застосунка.

Звичайно для мобільних додатків існують свої переваги та недоліки. До переваг можна віднести:

- інтерактивність - користувачеві надається можливість взаємодіяти з ним різними способами;
- доступність - користувач має доступ до всього необхідного функціоналу пристрою у будь-який час;
- персоналізація – користувач має можливість зберігати власний контент, дані та проводити складні обчислення;
- автономність – частину додатків можливо використовувати в автономному режимі, тобто не потрібно підключення до Інтернету.

До недоліків можна віднести:

- високі вимоги до сумісності. Для нормального функціонування програми, вона повинна відповідати вимогам конкретної ОС під управлінням якої працює пристрій;
- вартість розробки, підтримки і обслуговування буде вищою, ніж у випадку з мобільним сайтом;
- для створення мобільного додатку зазвичай потрібна більша кількість часу;
- щоб почати використовувати додаток, користувач повинен встановити його на свій пристрій.
- високі вимоги до сумісності – для того, щоб програма коректно працювала вона повинна відповідати вимогам певної операційної системи, під керуванням якої працює пристрій;
- щоб почати застосовувати програму, користувач повинен перш за все встановити її на своєму приладі.

Також існує класифікація мобільних додатків. Однак, більша кількість з них можна класифікувати як «гібридні». Це означає, що такі додатки передбачають користувацькі функції з різноманітними завданнями. Але все ж таки деякі типи мобільних додатків класифікуються однозначно, тобто вони спрямовані спеціально для вирішення конкретних завдань. Зазвичай, мобільні додатки класифікують наступним чином:

- Бізнес-додатки – даний тип додатків вважається затребуваним для бізнесу. Подібні додатки можуть прискорити та оптимізувати роботу на підприємстві. Основною складністю представляється перенесення бізнес-завдань до додатку.
- Додатки для контенту – серед користувачів такий вид додатку користується найбільшою популярністю, адже він дозволяє користувачеві взаємодіяти з контентом. Для такого типу додатків притаманні такі дії користувача, як: прослуховування музики, взаємодія з фото-, відео-контентом. Все це доступно та зручно для користувача, тому й цей тип користується найбільшою популярністю.
- Мобільні соціальні мережі – спеціальні додатки, що створюються для конкретної соціальної мережі.
- Мобільні ігри – даний тип додатків займають значну частину світового ринку. Вони закривають контент-розважальні завдання.

У даному випадку робота проводитиметься над бізнес-додатком, адже в першу чергу додаток повинен закривати питання, що допоможуть під час роботи на підприємстві.

Після побудови алгоритму та головних завдань для розробки додатку, необхідно визначити для якої платформи буде відбуватися реалізація завдання. Для цього необхідно розглянути існуючі операційні системи та проаналізувати їх.

На сьогоднішній день для мобільних пристроїв існує чимало операційних систем. Для різних мобільних пристроїв існують різні мови програмування. Вибір мови програмування залежить від операційної системи пристрою. В даній роботі було розглянуто та проведено аналіз серед наступних платформ: iPhone SDK та Android SDK.

1) iPhone SDK – для розробки мобільного додатку використовується ObjectiveC. Для написання програми можна використовувати C та C++. Відкладка виконується за допомогою xCode. Також у xCode наявний

емулятор, у якому можна розглянути приклад виконання написаної програми.

2) Android SDK – для розробки мобільного додатку на операційній системі Android можна використовувати середу розробки Eclipse або Android Studio. Eclipse є найпопулярнішою середою виконання серед розробників. Але середа Android Studio є більш надійною, так як ця середа є більш новою порівняно з попередньою, а також вона активно розвивається та підтримується компанією Google як офіційна середа розробки для Android-додатків. Також у Android Studio наявний емулятор.

Необхідно з'ясувати яка з систем користується найбільшою популярністю, а також якою платформою користуються в роботі співробітники на підприємстві, щоб обрати засіб розробки додатку.

На даний момент на арені світового ринку конкурують дві платформи для мобільних пристроїв, це: Android, IOS.

За даними дослідження StockApps, які провели аналіз світового ринку мобільних пристроїв, на 2022 рік статистика використання операційної системи Android становить 69 відсотків, а у операційної системи IOS - лише 25,49 відсотків. З цієї статистики можна зробити висновок, що операційна система Android є найбільш популярною у світі. Фінансовий експерт Едт Рідс визначив, що Android залишається найпопулярнішою операційною системою через наступні фактори [8]:

- вона має відкритий код;
- наявний дуже великий вибір пристроїв;
- та завдяки мінімальній вартості техніки, адже вона набагато нижче за IOS.

На сьогоднішній день мобільний застосунок має чимале значення для підприємств. Особливо для тих підприємств, які у своїй діяльності більше спрямовані на сферу онлайн бізнесу. Тому для такого типу підприємств дуже зручно використовувати додатки, що в першу чергу полегшують та

пришвидшують роботу, адже мобільні пристрої зазвичай знаходяться поруч із користувачем. За будь-яких обставин, наприклад якщо тимчасово не має можливості отримати доступ до робочого ПК, використовуючи мобільний пристрій, співробітник матиме змогу швидко отримати або зафіксувати потрібні йому дані у спеціальному додатку, що призначений для цього. Оскільки на підприємстві, для якого розробляється додаток, використовуються в роботі мобільні додатки на платформі Android, а також дана операційна система користується найбільшою популярністю у світі, було вирішено розробляти додаток саме під дану платформу.

В розділі була розроблена логіка роботи програмного додатку, а також було розглянути існуючі методи для реалізації даної роботи, у результаті чого були обрані засоби реалізації застосунку для подальшої реалізації роботи. Таким чином, було виконано завдання 2 з переліку завдань, поставлених для досягнення мети кваліфікаційної роботи.

3 ПРОЦЕС РОЗРОБКИ МОБІЛЬНОГО ДОДАТКУ

3.1Аналіз та вибір методу шифрування парольних даних

До завдання даної роботи входить процес шифрування парольних даних користувачів. Завдання полягає у тому, щоб шифрувати паролі для авторизації користувачів у системі. Принцип роботи шифрування має відбуватися наступним чином: при реєстрації нового користувача у системі необхідно створювати шифрування паролю таким чином, щоб до бази даних цей пароль був у зашифрованому вигляді. Тому процес шифрування потрібно здійснювати на етапі відправки даних до серверу. Після того, як дані коректно добавлені до бази даних, додається новий запис у спеціально виділену таблицю.

Розглянемо випадок з авторизації існуючого користувача у системі (це означає, що запис конкретного користувача існує у таблиці з парольними даними). У такому разі введений пароль користувача повинен шифруватися під час відправки на сервер, далі відбувається перевірка на існування такого самого запису у спеціальній таблиці. Якщо збіг значень зашифрованого паролю, який користувач щойно ввів на сторінці з авторизацією, та запису в таблиці знайдено, то відбувається вхід користувача у систему. Головне завдання алгоритму шифрування даних полягає у тому, щоб була можливість за допомогою нього використовувати перевірку даних.

Для того, щоб обрати алгоритм шифрування, який підходитиме до поставленого завдання, необхідно розглянути існуючі рішення [9]:

- Симетричне шифрування – даний алгоритм є найпростішим. Шифрування та дешифрування даних відбувається за допомогою використання одного й того самого ключа. Коли ми використовуємо симетричне шифрування, то це означає, що секретний цифровий ключ повинен мати і відправник, і отримувач, для того, щоб здійснювати обмін інформацією.

- Асиметричне шифрування – на відміну від симетричного шифрування, даний алгоритм використовую два ключа, один з яких відкритий, а інший закритий. Відкритий, в свою чергу, може бути відомим всім, адже дешифрувати інформацію за його допомогою не можливо. Закритий ключ є секретним, саме він і потрібен для того, аби дешифрувати повідомлення.
- Хешування або хеш-функція – даний алгоритм у певному сенсі не зобов'язує використання ключа. Такий засіб шифрування даних, зазвичай, називають одностороннім шифруванням або дайджестами повідомлень. Даний алгоритм хешування може перетворювати певний об'єм даних в строковий вигляд двійкових чисел (або бітів) певної довжини, а саме хеш. Хеш зазвичай використовується для цифрового підпису, а також для шифрування та зберігання паролів.
- Блоковий шифр – це можна сказати різновид симетричного шифрування. Принцип роботи алгоритму блокового шифру працює таким чином, що кожен блок даних шифрується або дешифрується окремо, кожен біт у вихідному блоці залежить напряму від кожного біта у відповідному вхідному блоці, проте не від інших бітів. При цьому розмір самого блоку визначається алгоритмом.
- Потоковий шифр – цей шифр також використовує симетричне шифрування. Даний шифр схожий на блоковий, але потоковий відрізняється тим, що він виконує шифрування по одному біту за один раз. Тоді, коли блочний робить шифрування одночасно. При потоковому шифрі відбувається перетворення символів відкритого повідомлення на шифровані працює залежно від їх розташування в потоці відкритого ключа та тексту.
- Цифровий підпис – частіше за все для такого виду шифрування інформації застосовується асиметрична криптографія, що має відкритий ключ. Також цифровий підпис має цифровий індекатор з урахуванням конкретного сертифіката, який видається спеціальним

акредитованим центром сертифікації. Такий підпис є частиною системи безпеки. Як правило, він використовується для електронних документів для того, щоб закривати такі функції як: підтвердження автентифікації відправника, збереження цілісності тих даних, що передаються.

Для того, аби виконати поставлене завдання необхідно використовувати хешування даних, адже саме хеш, як правило, використовується для шифрування та збереження паролю. Аби обрати за допомогою якої хеш-функції можливе шифрування паролів для авторизації користувачів у додатку, необхідно розглянути які саме алгоритми існують на даний час, а також біль детально розібрати що таке хеш-функція.

Криптографічна хеш-функція - це такий спеціальний математичний алгоритм, який відображає дані, що мають довільний розмір, у бітовий масив, який у свою чергу має чітко фіксований розмір. Результатом хеш-функції називають хеш-сумою або частіше за все хешем. Вхідні ж дані називають повідомленням. Для хеш-функцій повинні бути відтворені такі вимоги [10]:

- хеш-функція є детермінована – це означає, що одне й те саме повідомлення має приводити кожен раз до однакового хеш-значенню;
- значення хеш-функції визначається швидко для будь-якого повідомлення;
- неможливо знайти повідомлення, котре давало б задане хеш-значення;
- для одного повідомлення існує тільки одне хеш-значення;
- якщо хоч трохи змінюється повідомлення, то й саме хеш-значення змінюється повністю.

На даний момент хеш-функції використовують на практиці дуже широко. Хеш-функції допомагають у реалізації наступних дій:

1) Перевірка цілісності даних. Тобто, для того, щоб мати можливість перевірити чи були проведені якісь зміни у даних.
2) Використання в системах автентифікації для хешування паролів.
3) Перевірка та створення електронного цифрового підпису.

До списку хеш-функції, що частіше за все використовують входять такі [11]:

- SHA (Secure Hash Algorithm) - це 160-розрядний хэш-код або його ще називають дайджестом. він має 512-бітові блоки. Було виявлено, що дана функція не є стійкою до колізій. Колізії - це коли з'являється рівність у значеннях хеш-функцій для двох або більше різних блоків даних.
- SHA-1 (Secure Hash Algorithm 1) - це нова версія SHA, тобто його модифікація. В даному алгоритмі були виправлені недоліки його попередника, тобто дана хеш-функція вирішую проблему з колізіями. SHA-1 має дайджест повідомлення у 160 біт.
- MD2 (Message Digest #2) - дана хеш-функція має низьку швидкість відпрацювання. Вона створює 128-розрядне повідомлення будь-якого об'єму.
- MD4 -(Message Digest #4) - це більш нова версія хеш-функції MD2. Вона має 128-розрядний дайджест даних будь-якого об'єму. Але було виявлено, що вона має менш надійний алгоритм, адже перш за все цей алгоритм створювався для того, аби створити більш швидке шифрування даних, саме за цих причин, він є поганим рішенням у плані криптостійкості.
- MD5 (Message Digest #5) - дана хеш-функція є версією MD4, але з більшою надійністю порівняно з попередником. Даний алгоритм вкдючає 128-розрядний дайджест. Проте було виявлено, що даний алгоритм не є стіким до колізій, також він не використовується для електроно цифрового підпису.

Судячи з розглянутих рішень для реалізації поставленої задачі була використана хеш-функція SHA-1 (Secure Hash Algorithm 1), адже вона є швидкою та вирішую проблему з появою колізій.

3.2 Створення бази даних додатку

Для створення клієнт-серверної архітектури необхідна побудова віддаленого серверу та віддаленої бази даних. Для реалізації даної архітектури були проведені роботи зі створенням та налаштуванням хостингу, на якому знаходиться серверна частина завдання програми. Для цього була обрана українська компанія, що надає послуги з оренди хостинга та домену – це Ukraine. На базі даного сервісу був орендований якісний домен та хостинг-акаунт з підтримкою PHP та MySQL.

На даному хостингу є менеджер-фалів, де знаходяться основні функції додатку та база даних. Базою даних, з якою взаємодіє додаток, є phpMyAdmin. Для того, щоб створити базу даних були проведені роботи зі створення логіну та паролю, за допомогою яких можна потрапити до бази та провести усі необхідні налаштування. До таких налаштувань відноситься створення бази, де зберігаються усі таблиці, які необхідні для збереження інформації та ведення її обліку. Для цього необхідно створити назву бази даних та вибрати її кодування. У якості назви було використано – «app1_db» та у якості кодування було обрано – «cp1251_ukrainian_c». Кодування необхідне для того, аби звичайний користувач мав можливість прочитати та зрозуміти її вміст, тобто інформацію, яка буде зберігатися. Наступним етапом роботи було створення таблиць, що зберігатимуться у базі даних. За логікою додатка необхідно зберігати у базі даних інформацію про авторизацію користувачів, а також інформацію про записи, який користувач буде створювати у додатку. З цією метою було створено окремі таблиці для кожної з задач:

- Таблиця для ведення обліку користувачів системи. Дана таблиця потрібна для того, аби зберігати інформацію про авторизацію користувачів. Для цього була створена таблиця «user» з такими стовбцями як: «id» (унікальне позначення облікового запису для конкретного користувача), «login» (логін для входу у додаток), «pass» (пароль користувача у шифрованому вигляді, шифрування відбувається на стороні серверу та запис у таблицю відбувається вже у зашифрованому вигляді).
- Таблиця для збереження та ведення обліку інформації про записи, які користувач додає у систему. Для даної мети була створена таблиця «passwords» з такими стовбцями як: «id» (унікальне позначення для конкретного запису), «name» (зберігається назва запису), «login» (зберігається поле логіну), «password» (зберігається поле паролю), «coment» (зберігається додатковий коментар користувача щодо створюваного запису), «user_id»(унікальне позначення користувача).

Схема взаємодії таблиць у базі даних виглядає наступним чином: інформація із стовбця «id» з таблиці «user» належить до стовбця «user_id» у таблиці «passwords» відповідно (рис. 3.1).

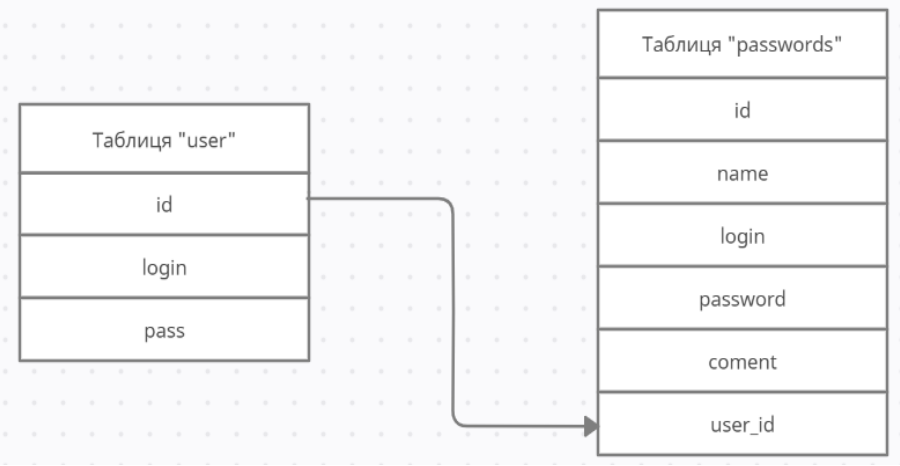

Рисунок 3.1 – Схема бази даних для серверної частини додатку

3.3 Реалізація програми

Для реалізації серверної частини додатку була використана середа розробки Visual Studio. Дана середа розробки дозволяє писати код на багатьох мовах програмування. Для реалізації поставленої задачі була використана мова програмування php, яка надає змогу прописувати функції для серверної частини додатку, html як мова розмітки, що використовується для візуальної розмітки сторінки та мова SQL-запитів для обміну інформацією з базою даних. В кожному файлі були прописані функції для кожного етапу відповідно. В першу чергу було зроблено підключення до бази даних за допомогою php та sql-запиту. З'єднання з базою даних відбувається в окремому файлі. Це потрібно для того, аби посилатися на цей файл під час виконання інших функцій у окремих файлах. Функції та відповідні файли, які були створені виглядають наступним чином:

- Реєстрація нового користувача. Для виконання даного етапу було створено файл «register.php». У даному файлі в блоці html наявна форма, яка відображує форма з поля для введення даних, а саме логін та пароль та наявна кнопка типу «submit». Після введення даних користувачем у відповідні поля та після натискання кнопки відбувається відправка даних на сервер. Відправку даних на сервер реалізовує блок php. У даному блоці відбувається підключення до нашої бази даних. Наступним кроком є відправка даних, які ввів користувач, на сервер за допомогою спеціального запиту POST. На данному етапі відбувається шифрування паролю за допомогою хеш-функції sha1. Також береться інформація з cooki для того, аби визначити сеанс користувача. Дана інформація обробляється сервером та перевіряє чи існує отримана інформація у базі даних. Для цього за допомогою мови sql відбувається запит до таблиці «user» з метою перевірки отриманих даних. Якщо відправлена інформація вже існує у базі даних, то користувачу надсилається

відповідь від сервера про те, що такий акаунт вже існує, у протилежному випадку – відбувається створення нового запиту за допомогою sql-запиту.

- Авторизація існуючого користувача відбувається відповідним чином у файлі «login.php». Але відмінність від попереднього етапу полягає у тому, що при заповнені html форми та натисканні на кнопку користувачем відбувається отримання даних за допомогою методу GET. На даному етапі відбувається шифрування введеного значення у поле «пароль». Після отримання даних, які ввів користувач, відбувається перевірка на існування такого самого запису у таблиці «user» за допомогою sql-запиту. Якщо відповідний запис знайдено, то користувачеві відправляється відповідь від сервера у вигляді входу у до його акаунту, у протилежному випадку – видається повідомлення, що такого користувача не існує і необхідно зареєструватися.
- Створення та управління записами користувача. Після авторизації користувач потрапляє до сторінки, де мають знаходитися його персональні записи. На даному етапі відпрацьовує файл «index.php». За допомогою блоку html відбувається візуальне відображення тих записів, які користувач вже створив, якщо таких записів немає, то відповідно записи не виводяться. За те, які конкретно дані необхідно вивести для користувача відповідає sql- запит, який бере id користувача та відповідно до нього виконує вибірку з бази даних з таблиці «passwords». Також на даній сторінці користувачу надається можливість створити новій запис за допомогою кнопки «ADD», вийти з акаунта за допомогою кнопки «Exit» та видалити будь з яких існуючих записів також за допомогою червоної кнопки, яка розташована біля кожного запису.
- Додавання нового запису. Якщо користувач натискає на кнопку «ADD», йому надається можливість заповнити форму з

конкретними полями для вводу: назва, логін, пароль, коментар та кнопка «Add». Після заповнення форми та при натисканні на кнопку відпрацьовує файл «add.php». В даному файлі відбувається отримання даних, які ввів користувач за допомогою методу GET. Також здійснюється перевірка отриманого id з тим, що виділений для користувача у базі даних. Після успішної перевірки виконується запит до бази даних, до таблиці «passwords» та відбувається створення нового запису для даного користувача.

- Видалення запису працює аналогічним чином, але замість створення запису, береться id існуючого запису в таблиці, що належить даному користувачеві та відбувається запит до таблиці за допомогою sql, який вказує на те, що необхідно видалити запис, який включає id користувача та id запису, який користувач обрав.
- Вихід користувача з акаунту. При натисканні користувачем на кнопку «Exit» відбувається виконання файлу «exit.php». Функція даного файлу полягає у тому, щоб видалити кукі та повернути користувача на сторінку авторизації. При цьому запис користувача не видаляється.

Для реалізації системи нагадувань у додатку було використано можливості Android. Адже для того, щоб нагадування надходили саме на пристрій користувача необхідно, щоб вони працювали згідно з правил операційної системи Android. Для реалізації даної задачі була використана середа розробки Android Studio та мова програмування, що характерна для даної операційної систему – kotline. Для початку роботи був створений файл формату apk, який включав у себе усі файли та функції додатка, що виконуються на стороні серверу. Після цього здійснювалося підключення додатка до серверу для коректної роботи. Наступним етапом було створення функціоналу нагадувань. Перш за все була створена база даних, де зберігається інформація про нагадування. Для цього була створена Database

під назвою AppDatabase. Для роботи з даною базою я використала бібліотеку Room. Дана бібліотека має три основні компоненти: Entity, Dao та Database.

Під час роботи з Room я використовувала SQL запити. Entity мені був необхідний, щоб помітити об'єкт, який ми хочемо зберігати у базі даних. І в об'єкті Dao я описую методи для роботи з базою даних.

За допомогою методу RecyclerView onClick я ставлю слухачі кліків.

У Ремайндер Модел - я прописую методи, які належать до нагадувань. У Ремайндер Активіті – за допомогою RecyclerView я прописую слухачі кліків і виводжу всі існуючі нагадування, якщо їх задавав користувач. Якщо користувач натискає на плюс у тулбарі, то спрацьовує Ремайндер Нот Активіті – у ньому відбувається створення нового нагадування. Тут я створюю календар із необхідними мені значеннями (рік, місяць, день, година, хвилини, секунди), а також фіксую час, коли нагадування створювалося.

Також використовується функція startTimePicker – вона потрібна для відліку відправлення повідомлення. Також наявний функціонал редагування нагадувань за допомогою функції onOptionsItemSelected. Тобто, коли користувач переходить на своє нагадування, він може його відредагувати та поставити інший час. У такому випадку, якщо користувач все-таки змінює час оповіщення, необхідно видалити старе оповіщення, щоб воно прийшло в новий вказаний час. Якщо час сповіщення користувач не змінив, то сповіщення залишається незмінним.

Отже, на виході ми маємо серверну частину додатку, яка спрямована на те, щоб зберігати інформацію на віддаленому сервері та забезпечувати спілкування з сервером для обміну інформацією між користувачем та сервером та частину із нагадуваннями, яка в свою чергу виконується за можливостями Android та усі дані про нагадування користувача зберігати в оперативній пам’яті власного пристрою користувача.

3.4 Тестування програми

Тестування програми необхідне проводити для того, аби перевірити наскільки коректно відпрацьовує програма. До першого етапу перевірки відноситься перше входження користувача до системи, а саме це буде перевірка функціоналу, що стосується авторизації та реєстрації у додатку. Почнемо з реєстрації користувача. На даному етапі відбувається заповнення форми користувачем. Форма виглядає наступним чином (рис. 3.4): наявні поля для введення даних, а саме: пароль та логін; також наявна кнопка «login» та «Registration».

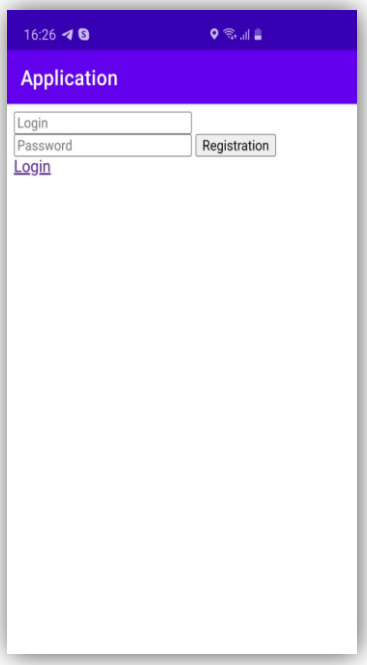

Рис. 3.4 – Сторінка реєстрації

За допомогою кнопки «login», користувачу надається можливість перейти до сторінки входу у додаток. При натисканні кнопки «Registration», відбувається відправка введеної користувачем інформації на сервер. В умовах завдання вказано, що на даному етапі, перед тим як створювати нового користувача, тобто новий запис у базі даних, необхідно, щоб відбулася перевірка на

існування такого запису, яка містить інформацію, що ввів користувач. Для цього реєструємо нового користувача (рис. 3.5) та після цього ще раз реєструємо такого самого користувача, з таким самим паролем та логіном.

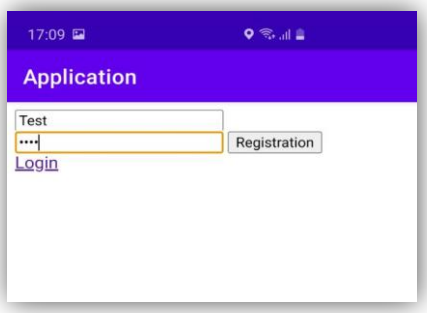

Рис. 3.5 – Реєстрація користувача

У результаті повторної спроби зареєструвати такого самого користувача, можна побачити, відповідь від сервера про те, що такий користувач вже є у системі та не можливо створити такий самий запис (рис. 3.6).

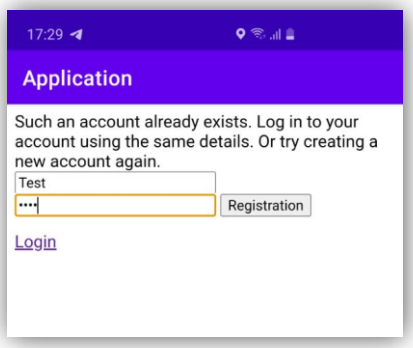

Рис. 3.6 – Помилка реєстрації

Наступним етапом є перевірка запису нового користувача у базі даних. Для цього необхідно перейти до бази даних, обрати таблицю «user» та знайти користувача за логіном «Test», який щойно був створений для перевірки реєстрації. Після переходу до таблиці, можна побачити щойно створений

запис з логіном «Test», паролем у зашифрованому вигляді та id користувача (рис. 3.7).

Рис. 3.7 – База даних для авторизації

Наступним етапом перевірки додатку є перевірка функціоналу авторизації. Інтерфейс сторінки авторизація виглядає так само, як і для сторінки реєстрації (рис. 3.8).

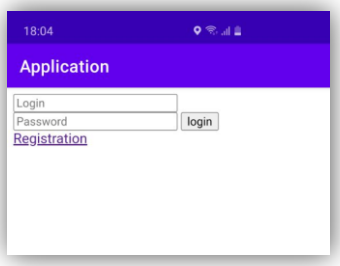

Рис. 3.8 – Сторінка авторизації

Після введення даних користувачем відбувається перевірка на наявність таких самих даних у таблиці «user». Якщо збіг даних було

виявлено, то користувач успішно потрапляє до власного акаунту. В іншому випадку, користувач повинен отримати повідомлення про те, що такого акаунту не існує (рис. 3.9).

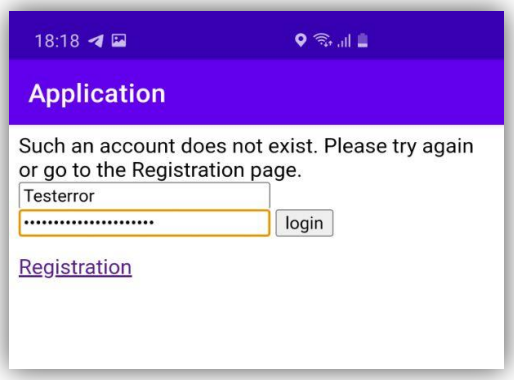

Рис. 3.9 – Помилка авторизації

Після успішного проходження авторизації користувач направляється автоматично до головної сторінки власного акаунта (рис. 3.10).

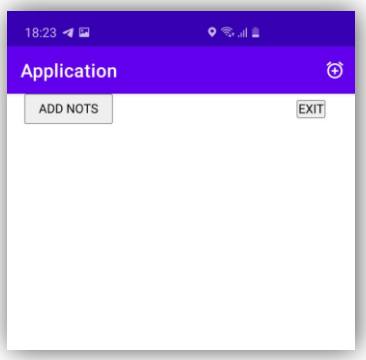

Рис. 3.10 – Головна сторінка

На даній сторінці відображуються усі записи поточного користувача, а також кнопки «ADD NOTS», «EXIT» та кнопка для додання нагадування у тулбарі. У тому випадку, якщо у користувача не має записів, виводяться лише ці дві кнопки, як на прикладі з рис. 3.10. Для того, щоб додати новий запис, необхідно натиснути на кнопку «ADD NOTS». Після натискання на

кнопку, з’являється форма з полями для вводу потрібної інформації, а саме: назва запису, логін, пароль, поле з додатковим коментарем та кнопка для збереження запису. (рис. 3.11).

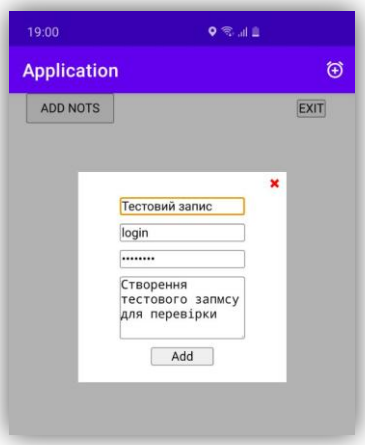

Рис. 3.11 – Форма для створення запису

Після натискання на кнопку «Add» на формі, створена запис відображується користувачеві на головній сторінці (рис. 3.12).

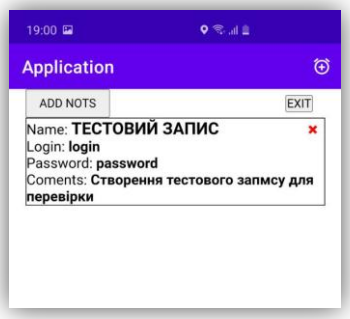

Рис. 3.12 – Створений запис

Для роботою з записами користувачеві надається можливість видаляти власні записи. Для цього біля кожного запису наявна кнопка у вигляді червоного хрестика, що призначена для видалення запису. Для того, аби перевірити роботу функціоналу видаленню запису, був створений тестовий запис, який позначений на рисунку 3.13.

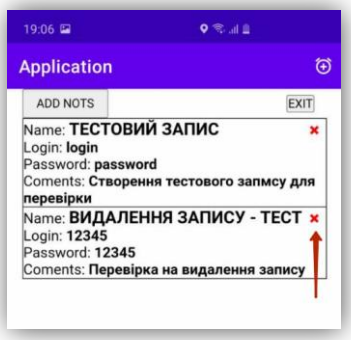

Рис. 3.13 – Видалення запису

Якщо натиснути на кнопку видалення запису, яка належить до щойно створеного тестового запису, то даний запис видаляється та відображаються користувачеві тільки ті записи, які залишилися (рис. 3.14).

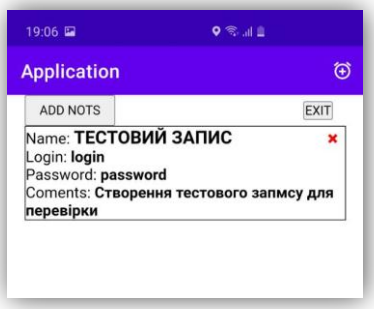

Рис. 3.14 – Результат видалення тестового запису

Наступним етапом перевірки виступає перевірка працездатності системи нагадувань. Для того, щоб перейти до нагадувань користувачеві необхідно натиснути на кнопку в тулбарі у вигляді годинника. Після натискання на таку кнопку, додаток перенаправляє користувача до сторінки з нагадуваннями. Якщо у користувача не має ніяких нагадувань, у тому числі не активних, то видається повідомлення на сторінці про відсутність нагадувань (рис. 3.15).

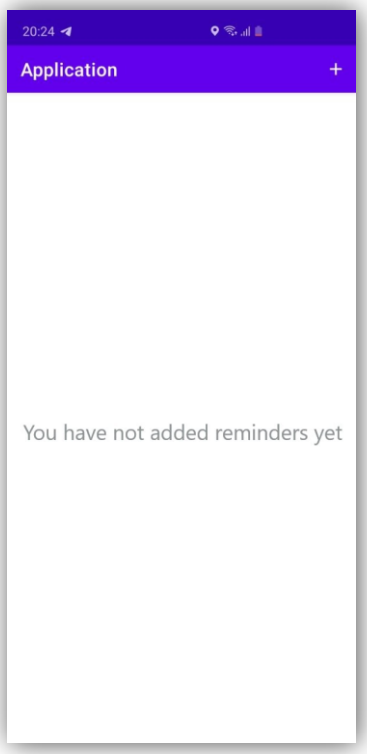

Рис. 3.15 – Сторінка для нагадувань

На даній сторінці користувачеві надається можливість створити нагадування за допомогою кнопки, яка знаходиться у тулбарі та має вигляд плюсу. При натисканні на дану кнопку користувач відправляється на сторінку для створення нагадування, яка має поля для введення даних (назва та опис нагадування), а також кнопки вибору дати та кнопку збереження нагадування (рис. 3.16).

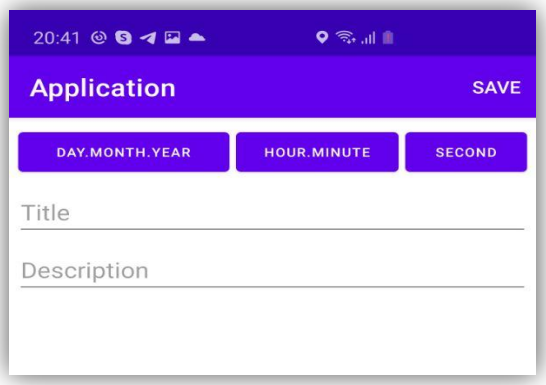

Рис. 3.16 – Створення нагадування

Для того, щоб протестувати роботу нагадувань необхідно виявити за яких умовах нагадування спрацьовуватимуть, а за яких – ні. В першу чергу необхідно перевірити чи виконується сповіщення про нагадування, якщо не виходити з додатку. Для цього створюємо тестове нагадування (рис. 3.17).

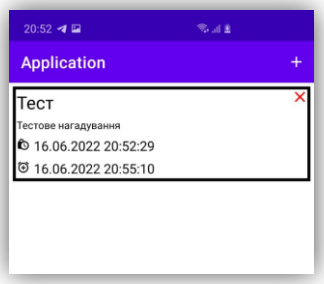

Рис. 3.17 – Тестове нагадування

Для цього необхідно заповнити дані про назву нагадування, опис нагадування та визначити дату і час, коли має прийти сповіщення з нашим нагадуванням. У результаті було отримано нагадування у вигляді push-повідомлення (рис. 3.18).

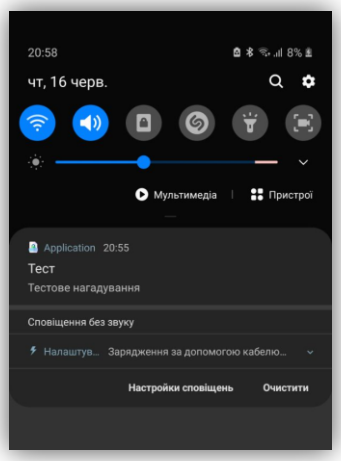

Рис. 3.18 – Результат роботи

Після спрацювання нагадування, воно позначається, як виконане (рис. 3.19).

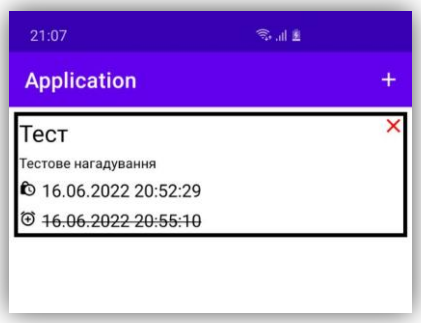

Рис. 3.19 – Виконане нагадування

Аналогічна робота була проведена для наступних тестів:

1) Перевірка на відпрацювання нагадування, коли користувач закрив додаток.
2) Перевірка на отримання нагадування при виході користувача з акаунту.
3) Перевірка на отримання сповіщення про нагадування при перезавантаженні пристрою.

В перших двох випадках сповіщення відпрацювали коректно, адже дані про нагадування зберігаються у оперативній пам’яті пристрою. Проте, якщо перезавантажити пристрій, то інформація про нагадування видаляється – це характерно для оперативної системи Android. Дану проблему можна вирішити наступним чином: потрібно відновити всі дані до перезавантаження пристрою через власне налаштування пристрою.

В даному розділі кваліфікаційної роботи була представлена реалізація програмного додатку, що вирішує такі питання, як: збереження інформації, створення системи нагадування та захист паролю користувачів. Також було проведене тестування додатку і результатами даного тестування було виявлено, що функції даного додатка працюють коректно, за винятком однієї умови, що зв’язана з функціоналом нагадування. Ці умови прописані в результатах тестування. В даному розділі було вирішено завдання 3 з переліку завдань, поставлених в роботі для досягнення мети даної кваліфікаційної роботи.

4 ОХОРОНА ПРАЦІ

У розділі передбачається розгляд робочого місця програміста, який буде виконувати розробку мобільного додатку, яке спрямовано на зберігання та захист парольних даних на підприємстві. У першу чергу буде розглянуто робоче місце працівника, а саме: робочій простір, що включає в себе наступні пункти:

- Окремо виділена для роботи кімната.
- Стіл, за яким буде проводитися уся робота. Розміщений у кімнаті навпроти вікна. Вікно слугує у якості джерела природнього світла.
- На столі знаходиться ноутбук – робоча машина, за якою буде відбуватися робота програміста. Також на столі наявна ємність з водою, щоб контролювати водний баланс в організмі, настольна лампа ліворуч від ноутбука та над ним полиці, на яких зберігаються книги.
- Під столом розташований ящик, який містить зошити та канцелярію.
- Навпроти столу знаходиться стілець зі спинкою. Даний стілець є підйомно-поворотним, легко регульованим за висотою та забезпечує належну підтримку і зручне положення спини і хребта програміста.
- У кімнаті наявний випарник – пристрій, що контролює вологість.
- Праворуч від вікна знаходяться батареї, що призначені для опалювання приміщення – централізоване опалення.
- Зліва від столу знаходяться розетки 220В із захисним щитком.
- Біля дверей, що знаходяться навпроти вікна, розташовані вимикачі загального світла на висоті 1,5 метри від підлоги.
- Зліва від дверей висить кондиціонер під стелею на висоті 2,5 м
- Зліва від вікна є система вентиляції.

Описане вище робоче місце має свої певні небезпечно шкідливі фактори виробництва. Такі фактори за ГОСТ 12.0.003-74 поділяються за природою дії на такі групи [12]:

- фізичні;

- хімічні;
- біологічні;
- психофізеологічні.

В даному випадку, враховуючи ті умови праці, які зазначені вище, можна визначити, що небезпечно шкідливі фактори для даного робочого місця за природою дії належать до групи - фізичних. Насамперед, такими факторами виступають:

- підвищена чи занижена температура повітря кімнати;
- підвищений рівень шуму у робочій кімнаті;
- понижена або підвищена вологість у приміщенні;
- підвищений рівень напруги в електричній системі, вірогідність виникнення замикання;
- надлишок або недостатній рівень освітленості кімнати.

З умов безпеки усі кабелі, що знаходяться у кімнаті, надійно сховані під захисними щитками, , які перешкоджатимуть можливому потраплянню робітника під напругу.

Для того, щоб забезпечити якомога кращі умови праці була запроваджена низка заходів. У першу чергу, для роботи програміста за ноутбуком були використані спеціальні окуляри з 7-шаровим покриттям антивідблиску на лінзах, що захищає від спазмів очей. Такі окуляри запобігають підвищенню внутрішньоочного тиску і перевантаженню очних м'язів. Завдяки таким окулярам працівник буде менше стомлюватися та шкода здоров’ю від опромінення екраном ноутбука зменшиться. Також для комфортної роботи були проведені роботи з налаштуваннями інтерфейсу ноутбука, а саме:

- Було зменшено яскравість екрану на 15% аби яскравість екрану буле не більше, ніж освітлення довкола.
- Збільшення шрифту на екрані на 10%. Це потрібно для того, щоб працівник не перенавантажував свій зір аби розгледіти відображуваний текст на екрані.

- Впровадження тимчасових перерв. Наразі це 10-15 хв кожні 2 години роботи за ноутбуком. А також введення у режим працівника щоденно 30 хв розминки або спорту. Це допомагає підвищити його фізичний стан та знизити рівень стресу.

Робоче місце програміста сягає 8,0 м2 за площиною та 21,0 м3 за об’ємом. Такі параметри задовольняють норму, так як площа робочого місця на одну людину більша ніж 6 м2, а об’єм більший ніж 20 м3.

Освітленість у кімнаті облаштована штучним та природним світлом. Природне світло надходить від вікна, що розташоване навпроти столу та коефіцієнт природнього світла становить 40%. Штучне освітлення має нараховує 2 типа:

1) Персональне освітлення. Його забезпечує настольна світодіодна (LED) лампа із світовим потоком 1080Лм та з потужністю 12Вт.
2) Загальне освітлення, розміщене по середині кімнати на стелі. Для даного типу освітлення використовується світодіодна (LED) лампа із світовим потоком 1080Лм та з потужністю 12Вт.

Для штучного освітлення кімнати були використані саме світодіодні лампи, а не лампи розжарювання, так як такі лампи використовувати не можна, бо вони чутливі до перебоїв та гріють приміщення.

Отже, дане освітлення відноситься до комбінованого і дані параметри освітлення є допустимими, так як коефіцієнт природнього світла > 15%, та окрім природнього наявне це й штучне освітлення.

Температура у кімнаті варіюється в межах 24-25 ° С в літній період і 20-24 ° С в зимовий при відносній вологості 50%. Наразі вологість у кімнаті регулює спеціальний пристрій – випарник, а температуру регулюють батареї взимку та кондиціонер влітку. Даний температурний режим є правильним за санітарними нормативами.

Для комфортних умов праці програміста було запроваджене щоденне вологе прибирання та провітрювання приміщення. Такий захід потрібен для того, аби у кімнаті не було пилу на поверхнях та завжди було свіже повітря.

Також у кімнаті наявна система вентиляції, яка призначена для циркуляції повітря у кімнаті.

Існують такі види вентиляційних систем за параметрами [13]:

- за способом циркуляції повітря у приміщені ця система вентиляції відноситься до механічної;
- за призначенням – це приточно-витяжна;
- за конструкцією вона є канальною.

У кімнаті є звукоізоляція, яка забезпечує рівень шуму, що вважається допустимим для даного робочого місця – 30дБ. Даний параметр досягається завдяки гіпсокартону на основних стінах робочого місця. Відповідно до санітарних норм виробничого шуму, ультразвуку та інфразвуку, допустимий рівень звуку в приміщені дорівнює 50 дБА. [14]

Внутрішнє оздоблення кімнати, де знаходиться робоче місце програміста включає має стіни і стелю, які пофарбовані в білий колір, також вікна закриті білими шторами – коефіцієнт відбиття дорівнює 70% . Підлога має паркетне покриття, яке пофарбоване матовою коричневою краскою – коефіцієнт відбиття дорівнює 40%. Такі параметри кімнати надають комфортні умови праці програміста. Жодна з поверхонь та предметів, які знаходяться у кімнаті не відзеркалюють, саме це не буде заважати працівнику, так як не буде його засліплювати.

Аналіз пожежної безпеки і вибір заходів і засобів пожежної безпеки.

Пожежа - це неконтрольоване горіння, яке розвивається у просторі та часі. Для того, щоб почався процес горіння необхідне виконання трьох умов: горюча речовина, джерело спалаху та окиснювач. У випадку, якщо виникає контакт між цими елементами, то виникає горюча система і навпаки, якщо відсутній хоча б один елемент, у такому разі горіння виключається.

Пожежна небезпека завжди буде актуальна, адже у будь-якому приміщені існують певні ризики, які можуть спровокувати появу пожежі. Тому надважливо забезпечувати належні умови для безпечної та комфортної праці програміста та проводити щонедільну профілактику у виді огляду наявних

предметів у приміщенні, які можуть спровокувати виникнення пожежної небезпеки. В першу чергу це стосується наступних ризиків [15]:

- наявність оголених електроприладів;
- поламка одного або всіх джерел елекроенергії;
- проблеми з проводкою або її несправність;
- неналежне використання горючої речовини;
- пролиття будь-якої рідини на джерело електроенергії: це може бути у даному випадку ноутбук, розетки;

Маючи на увазі можливі ризики, які можуть спровокувати пожежу, необхідно пам'ятати, та у разі небезпеки, використовувати засоби, які зможуть спричинити гасіння пожежі. Такими засобами є [15]:

- галоїдні вуглекислотні сполуки - вогнегасник;
- сухі порошки;
- системи пожежогасіння;
- інертні і негорючі гази;
- пісок, щільна тканина — повсть або азбест.

Також слід пам'ятати про головні правила поведінки під час виникнення пожежі [16]:

- Викликати пожежну службу "101".
- Вести себе зібрано та тверезо оцінювати ситуацію.
- Якщо це можливо, вимкнути усі джерела електроенергії.
- Спробувати загасити пожежу власноруч, використовуючи підручні засоби, якщо відсутні спеціально призначені (наприклад, вогнегасник).
- Якщо є вогнегасник або інші спеціальні засоби, які можуть загасити пожежу, то обов'язково почати гасити пожежу цими засобами.
- Важливо: гасити пожежу потрібно у тому разі, як її ступінь невеликий або локальний. У іншому випадку, якщо пожежа надто сильно - не намагатися її загасити.
- У разі виникнення сильної пожежі потрібно негайно покинути приміщення, будівлю.

- Якщо не вдається вийти з приміщення через звичайний вихід (двері), то необхідно використати пожежний або аварійний (запасний) вихід наззовні, використовуючи при цьому план евакуації.
- Під час покидання будівлі необхідно пригнутися якнайнижче, тому що дим зосереджується вгорі та вікористати, якщо можливо, вологу тканину, яка буде захищати від поглинання диму у легені.
- При пожежі не можна використовувати ліфт, так як він може зупинитися через відсутність елекроенергії у будівлі.
- Якщо потрібно подолати задимлену частину ділянки, то у такому разі необхідно затамувати подих, аби не вдихати дим. Вдихання такого диму є дуже небезпечним, адже через це можна втратити свідомість.
- У випадку, коли на людині палає одяг, потрібно якомога швидше зняти цей одяг, але ні в якому разі не можна бігти.
- На той випадок, коли у людини вже наявні опіки на тілі, то ні в якому разі не можна знімати одежу і якщо є можливість, то потрібно прикласти вологу тканину.

Отже, враховуючи те, що є конкретні інструкції що робити у разі виникнення пожежі та аналізуючи ті предмети, що знаходяться на робочому місці програміста, можна зробити висновок, що пожежна безпека є достатньою. У кімнаті є вогнегасник, усі кабелі та розетки покриті захисним щитком. Також для того, щоб попередити пожежу буде проводитися щонедільна перевірка робочого стану предметів та перевірка на несправність кожних елементів. У разі несправності будь-яких елементів необхідно припинити роботу та викликати спеціалістів, котрі будуть вживати заходи щодо таких елементів.

Якщо проаналізувати усі вище перераховані умови праці, то для даного розділу можливо зробити висновок, що усі умови праці є задовільними для комфортної роботи над поставленою задачею.

ВИСНОВКИ

В кваліфікаційній роботі виконані усі завдання з переліку тих завдань, які були поставлені.

Для виконання першого завдання була проведена робота над оглядом та аналізом сучасних рішень, яка описана у першому розділі – «Аналіз предметної області». У результаті виконання поставленого завдання, було виявлено, що на даний момент існує великий попит у даній тематиці, а також існують певні рішення, які закривають частину питань. Проте була виявлена необхідність створення нового рішення, яке закриває потреби підприємства та удосконалює існуючі рішення.

Наступною задачею була розробка алгоритму роботи програмного додатку. Дана задача є дуже важливою, адже від побудови алгоритму роботи застосунку, залежить подальша реалізація програмного застосунку, який показуватиме результати, які необхідні для досягнення мети. Дане завдання розглядається та вирішується у розділі – «Постановка задачі та методи дослідження».

Після виконання першого та другого завдання даної кваліфікаційної роботи була виконана реалізація програмного застосунку, що відповідає завданню 3 з переліку завдань, поставлених в роботі для досягнення мети. Для виконання даного завдання була описана та здійснена розробка мобільного додатка, а також його тестування для того, аби виявити чи коректно воно працює та чи виконує воно поставлену задачу. У результаті проведеної роботи, було отримано мобільний додаток, що відповідає усім поставленим завданням необхідним для його запуску.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Smart-Soft Team. Інформаційна безпека підприємства: ключові загрози та засоби захисту. URL: https://www.smart-soft.ru/blog/informatsionnaja-bezopasnost/
2. Відомості Верховної Ради України (ВВР), 1994, № 16, ст.93
3. Ковтун С. В. Інформаційна безпека: підручник. Харків: ХНЕУ, 2009. 368 с.
4. Лук'янов О. 10 програм для зберігання паролів: від пошти, Wi-Fi і навіть від шафки у спортзалі. URL: https://mc.today/10-programm-dlya-hraneniya-parolej-ot-pochty-wi-fi-i-dazhe-ot-shkafchika-v-sportzale/
5. Архітектура та технології клієнт-сервер. URL: https://en.ppt-online.org/730746 https://www.4stud.info/networking/lecture5.html
6. Анатольєв А.Г. Компоненти мережі. Клієнт-серверна взаємодія та ролі серверів. URL: https://www.4stud.info/networking/lecture5.html
7. Хусейн Ф. Клієнт-серверний мобільний додаток. Як це працює? URL: https://www.linkedin.com/pulse/клиент-серверное-мобильное-приложениекак-это-работает-hossein-fakhr?trk=public_profile_article_view
8. Баусов А. Дослідження: власники Android-смартфонів стали частіше переходити на iPhone у всьому світі URL: https://www.iphones.ru/iNotes/issledovanie-vladelcy-android-smartfonov-stali-chashche-perehodit-na-iphone-po-vsemu-miru-04-22-2022
9. Шифрування даних. URL: https://roi4cio.com/categories/category/shifrovanie-dannykh/
10. CorVVin. Хеш-функція, що це таке? URL: https://habr.com/ru/post/534596/
11. Лівак О. М. Функції хешування. Механізм хеш-функцій. URL: http://mf.grsu.by/UchProc/livak/b_protect/zok_7.htm

12. ДСТУ 12.0.003-74*. ССБТ. Небезпечні і шкідливі виробничі фактори Класифікація. [Чинний від 1999.01.01].
13. ДНБ В.2.5-67:2013. Опалення, вентиляція та кондиціонування. [Чинний від 2013]. Київ, 2013. 240с.
14. ДСН 3.3.6.037–99. Санітарні норми виробничого шуму, ультразвуку та інфразвуку. [Чинний від від 1999.01.12]. Київ, 1999. 10с.
15. ДБН В.1.1-7:2016. Пожежна безпека об’єктів будівництва. Загальні вимоги. [Чинний від від 1999.01.12]. Київ, 2016, 39с.
16. Рожков А.П. Пожежна безпека: Навчальний посібник для студентів вищих закладів освіти України. Київ, 1999. 256 с.

Додаток А. Лістинг програмного коду

Bd.php:

```
<?php
$bd =
mysqli_connect('app1.mysql.tools','app1_11','Lu0307','app1_db');
$bd->query("SET NAMES utf8");
$bd->query("SET CHARACTER SET utf8");
$bd->set_charset('utf8');
$bd->set_charset('utf-8');
?>
```

Registration.php:

```
<?php
if (isset($_COOKIE['id'])) {
header('location: index.php');
}
require 'bd.php';
if (isset($_POST['btn'])) {
$login = $_POST['login'];
$password = sha1($_POST['password']);
$query = "SELECT * FROM `user` WHERE login = '$login' AND pass =
'$password'";
$data = mysqli_query($bd,$query);
if (mysqli_num_rows($data)==1) {
echo "Такой аккаунт уже есть";
}else{
$qwery = "INSERT INTO `user`(`login`, `pass`) VALUES
('$login','$password')";
$data1 = mysqli_query($bd,$qwery);
$data = mysqli_query($bd,$query);
$row = mysqli_fetch_assoc($data);
setcookie('id',$row['id'],time()+(60*60*24*30),'/');
header('location: index.php');
}
}
?>
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-
scale=1.0">
<title>Document</title>
</head>
<body>
</body>
</html>
<!DOCTYPE html>
<html lang="en">
<head>
```

```
<meta charset="UTF-8">
<meta   name="viewport"   content="width=device-width,   initial-
scale=1.0">
<title>Document</title>
</head>
<body>
<form action="register.php" method="POST">
<input type="text" placeholder="Login" required="" name="login">
<input   type="password"   placeholder="Password"   required=""
name="password">
<input type="submit" value="Registration" name="btn">
</form>
<a href="login.php">login</a>
</body>
</html>
```

Login.php:

```
<?php
if (isset($_COOKIE['id'])) {
header('location: index.php');
}
require 'bd.php';
if (isset($_GET['btn'])) {
$login = $_GET['login'];
$password = sha1($_GET['password']);
$query = "SELECT * FROM `user` WHERE login = '$login' AND pass =
'$password'";
$data = mysqli_query($bd,$query);
if (mysqli_num_rows($data)==1) {
$row = mysqli_fetch_assoc($data);
setcookie('id',$row['id'],time()+(60*60*24*30),'/');
header('location: index.php');
}
}
?>
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta   name="viewport"   content="width=device-width,   initial-
scale=1.0">
<title>Document</title>
</head>
<body>
<form action="login.php" method="GET">
<input type="text" placeholder="Login" required="" name="login">
<input   type="password"   placeholder="Password"   required=""
name="password">
<input type="submit" value="login" name="btn">
</form>
<a href="register.php">Registration</a>
</body>
</html>
```

Add.php:

```
<?php
require 'bd.php';
$id = $_GET['id'];
$name = $_GET['name'];
$login = $_GET['login'];
$password = $_GET['password'];
$coment = $_GET['coment'];
$query = "SELECT * FROM `passwords` WHERE login = '$login' AND
user_id=".$id;
$data = mysqli_query($bd,$query);
if (mysqli_num_rows($data)==1) {
$a = array(
'status' => 'copy',
);
}else{
$query1 = "INSERT INTO `passwords`(`name`, `login`, `password`,
`coment`,                    `user_id`)                    VALUES
('$name','$login','$password','$coment',$id)";
$data = mysqli_query($bd,$query1);
$data = mysqli_query($bd,$query);
$row = mysqli_fetch_assoc($data);
$a = array(
'status' => 'success',
'id' => $row['id'],
);
}
echo json_encode($a);
?>
```

Del.php:

```
<?php
require 'bd.php';
if (isset($_GET['id'])) {
$query = "DELETE FROM `passwords` WHERE id=".$_GET['id'];
$data = mysqli_query($bd,$query);
}
```

Exit.php:

```
<?php
unset($_COOKIE['id']);
setcookie('id', '',-1,'/');
header('Location: login.php');
?>
```

AppDatabase.kt

```
@Database(entities = [Reminder::class], version = 1)
abstract class AppDatabase : RoomDatabase() {
    abstract fun userDao(): ReminderDao
    companion object {
```

```
        @Volatile
        private var INSTANCE: AppDatabase? = null

        fun getDatabase(context: Context): AppDatabase {
            val tempInstance = INSTANCE
            if(tempInstance != null) {
                return tempInstance
            }
            synchronized(this) {
                val db = Room.databaseBuilder(
                    context.applicationContext,
                    AppDatabase::class.java, "database_reminder"
                ).build()
                INSTANCE = db
                return db
            }
        }
    }
}
```

Reminder.kt:

```
@Entity
data class Reminder(
    @PrimaryKey(autoGenerate = true) var id: Int,
    @ColumnInfo(name = "title") val title: String?,
    @ColumnInfo(name = "desc") val desc: String?,
    @ColumnInfo(name = "date") val date: Long?,
    @ColumnInfo(name = "alarm_date") val alarmDate: Long?
)
```

ReminderDao.kt:

```
@Dao
interface ReminderDao {
    @Query("SELECT * FROM reminder")
    fun getAll(): List<Reminder>
    @Query("SELECT * FROM reminder WHERE id = (:id)")
    fun getReminderById(id: Int): Reminder?
```

```
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    fun insertAll(rem: Reminder): Long
    @Delete
    fun delete(rem: Reminder)
    @Update
    fun update(rem: Reminder)
}
ReminderRepository.kt:
class ReminderRepository(private val reminderDao: ReminderDao) {
    fun insertReminder(reminder: Reminder): Int {
        var lid = reminderDao.insertAll(reminder)
        return lid.toInt()
    }
    fun getAll(): List<Reminder> {
        return reminderDao.getAll()
    }
    fun getReminderById(id: Int): Reminder? {
        return reminderDao.getReminderById(id)
    }
    fun updateReminder(rem: Reminder) {
        reminderDao.update(rem)
    }
    fun deleteReminder(rem: Reminder) {
        reminderDao.delete(rem)
    }
}
MainActivity.kt:
class MainActivity : AppCompatActivity() {
private lateinit var binding: ActivityMainBinding
private var from: String =
"https://www.passmanager.icu/login.php"
var remindersItemMenu: MenuItem? = null
var isShowReminders: Boolean = false;
override fun onCreate(savedInstanceState: Bundle?) {
super.onCreate(savedInstanceState)
```

```
binding = DataBindingUtil.setContentView(this,
R.layout.activity_main)
showPrBar()
initWv()
binding.webView.loadUrl(from)
}
override fun onStart() {
super.onStart()
}
fun initWv() {
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.KITKAT) {
binding.webView.setLayerType(View.LAYER_TYPE_HARDWARE, null);
} else {
binding.webView.setLayerType(View.LAYER_TYPE_SOFTWARE, null);
}
val webSettings: WebSettings = binding.webView.getSettings()
webSettings.javaScriptEnabled = true
webSettings.setDomStorageEnabled(true);
webSettings.setAppCacheEnabled(true);
webSettings.setLayoutAlgorithm(WebSettings.LayoutAlgorithm.NARRO
W_COLUMNS);
webSettings.setUseWideViewPort(true);
webSettings.setSavePassword(true);
webSettings.setSaveFormData(true);
webSettings.setEnableSmoothTransition(true);
webSettings.setRenderPriority(WebSettings.RenderPriority.HIGH)
binding.webView.webChromeClient = MyWebChromeClient()
binding.webView.webViewClient = MyWebViewClient()
binding.webView.setDownloadListener(object : DownloadListener {
override fun onDownloadStart(
url: String,
userAgent: String,
contentDisposition: String,
mimeType: String,
contentLength: Long
```

```
) {
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
if
(checkSelfPermission(Manifest.permission.WRITE_EXTERNAL_STORAGE)
=== PackageManager.PERMISSION_DENIED) {
L.i("permission denied to WRITE_EXTERNAL_STORAGE - requesting
it")
val permissions =
arrayOf(Manifest.permission.WRITE_EXTERNAL_STORAGE)
requestPermissions(permissions, 1)
}
}
var request: DownloadManager.Request =
DownloadManager.Request(Uri.parse(url));
request.setMimeType(mimeType);
var cookies: String? =
CookieManager.getInstance().getCookie(url);
request.addRequestHeader("cookie", cookies);
request.addRequestHeader("User-Agent", userAgent);
request.setDescription("Downloading file....");
request.setTitle(URLUtil.guessFileName(url, contentDisposition,
mimeType));
request.allowScanningByMediaScanner();
request.setNotificationVisibility(DownloadManager.Request.VISIBI
LITY_VISIBLE_NOTIFY_COMPLETED);
request.setDestinationInExternalPublicDir(
Environment.DIRECTORY_DOWNLOADS,
URLUtil.guessFileName(url, contentDisposition, mimeType)
);
var dm: DownloadManager =
getSystemService(Context.DOWNLOAD_SERVICE) as DownloadManager
dm.enqueue(request);
}
});
}
```

```
inner class MyWebChromeClient : android.webkit.WebChromeClient()
{
override fun onProgressChanged(view: WebView?, newProgress: Int)
{
super.onProgressChanged(view, newProgress)
if (newProgress >= 100) {
hidePrBar()
}
}
override fun onShowFileChooser(
webView: WebView?,
filePathCallback: ValueCallback<Array<Uri>>?,
fileChooserParams: FileChooserParams?
): Boolean {
return true
}
}
private inner class MyWebViewClient : WebViewClient() {
@TargetApi(Build.VERSION_CODES.N)
override fun shouldOverrideUrlLoading(view: WebView, request:
WebResourceRequest): Boolean {
var url = request.url.toString()
var uri = request.url
if (URLUtil.isNetworkUrl(url)) {
view.loadUrl(url)
return false
}
var openLinkIntent: Intent = Intent(Intent.ACTION_VIEW, uri)
if (openLinkIntent.resolveActivity(packageManager) != null) {
startActivity(openLinkIntent)
} else {
L.i("Error")
}
return true
}
```

```
override fun shouldOverrideUrlLoading(view: WebView, url:
String): Boolean {
var uri = Uri.parse(url)
if (URLUtil.isNetworkUrl(url)) {
view.loadUrl(url)
return false
}
var openLinkIntent: Intent = Intent(Intent.ACTION_VIEW, uri)
if (openLinkIntent.resolveActivity(packageManager) != null) {
startActivity(openLinkIntent)
} else {
L.i("Error")
}
return true
}
override fun onPageStarted(view: WebView?, url: String?,
favicon: Bitmap?) {
super.onPageStarted(view, url, favicon)
}
override fun onPageFinished(view: WebView?, url: String?) {
super.onPageFinished(view, url)
binding.webView.setVisibility(View.VISIBLE)
var tmpurls = url!!.split("/")
if(tmpurls.size > 0) {
var index = tmpurls[tmpurls.size-1]
if(index.equals("index.php")) {
setRemindersMenuVisible(true)
} else {
setRemindersMenuVisible(false)
}
} else {
setRemindersMenuVisible(false)
}
hidePrBar()
}
```

```
override fun onReceivedError(
view: WebView?, errorCode: Int,
description: String?, failingUrl: String?
) {
super.onReceivedError(view, errorCode, description, failingUrl)
hidePrBar()
L.i("ERRROR")
}
}
fun setRemindersMenuVisible(isVisible: Boolean) {
if(remindersItemMenu != null) {
remindersItemMenu!!.isVisible = isVisible
} else {
CoroutineScope(Dispatchers.IO).launch {
while(remindersItemMenu == null) {
try {
Thread.sleep(300)
} catch (e:Exception) {}
}
withContext(Dispatchers.Main) {
remindersItemMenu!!.isVisible = isVisible
}
}
}
}
override fun onCreateOptionsMenu(menu: Menu?): Boolean {
val inflater = menuInflater
inflater.inflate(R.menu.menu, menu)
remindersItemMenu = menu?.findItem(R.id.reminders)
remindersItemMenu?.isVisible = false;
return super.onCreateOptionsMenu(menu)
}
override fun onOptionsItemSelected(item: MenuItem): Boolean {
// TODO Auto-generated method stub
val id = item.itemId
```

```
if (id == R.id.reminders) {
var intent = Intent(this, RemindersActivity::class.java)
startActivity(intent)
}
return super.onOptionsItemSelected(item)
}
fun showPrBar() {
if(binding.progressBar != null) {
binding.progressBar?.visibility = View.VISIBLE
}
}
fun hidePrBar() {
if(binding.progressBar != null) {
binding.progressBar?.visibility = View.GONE
}
}
}
```

ReminderActivity.kt

```
override fun onCreate(savedInstanceState: Bundle?) {
super.onCreate(savedInstanceState)
binding = DataBindingUtil.setContentView(this,
R.layout.activity_reminders)
var alarmUtils = AlarmUtils(this)
if(!alarmUtils.isPermissionHas(this)) {
AlertDialog.Builder(this).
setMessage("permission").
setCancelable(false).
setPositiveButton("Yes", object :
DialogInterface.OnClickListener {
override fun onClick(p0: DialogInterface?, p1: Int) {
val intent = Intent().apply {
action = Settings.ACTION_REQUEST_SCHEDULE_EXACT_ALARM
}
startActivity(intent)
}
```

```
}).setNegativeButton("No", object:
DialogInterface.OnClickListener {
override fun onClick(p0: DialogInterface?, p1: Int) {
p0!!.cancel()
}
}).create().show()
}
class RemindersRecycleAdapter(var remListener:
OnRemindersAdapterListener) :
RecyclerView.Adapter<RemindersRecycleAdapter.RemindersViewHolder
>() {
private var reminderList: List<Reminder?> = emptyList()
fun setList(list: List<Reminder?>) {
reminderList = list
notifyDataSetChanged()
}
override fun onCreateViewHolder(parent: ViewGroup, viewType:
Int): RemindersViewHolder {
val binding =
ItemReminderBinding.inflate(LayoutInflater.from(parent.context),
parent, false)
return RemindersViewHolder(binding)
}
override fun onBindViewHolder(holder: RemindersViewHolder,
position: Int) {
holder.bind(reminderList[position])
}
override fun getItemCount(): Int {
return reminderList.size
}
inner class RemindersViewHolder(private val binding:
ItemReminderBinding) : RecyclerView.ViewHolder(binding.root) {
fun bind(reminder: Reminder?) {
if(reminder != null) {
binding.idTitle.text = getLimitStr(reminder.title, 30)
```

```
binding.idDesc.text = getLimitStr(reminder.desc, 70)
binding.justDate.text = getDateFromMilli( reminder.date)
binding.alarmDate.text = getDateFromMilli(reminder.alarmDate)
var cal: Calendar = Calendar.getInstance()
if(reminder.alarmDate != null && reminder.alarmDate <
cal.timeInMillis ) {
binding.alarmDate.paintFlags =  binding.alarmDate.paintFlags or
Paint.STRIKE_THRU_TEXT_FLAG
} else {
binding.alarmDate.paintFlags =binding.alarmDate.paintFlags and
Paint.STRIKE_THRU_TEXT_FLAG.inv()
}
L.i("remdate = "+reminder.date + "|alarmdate
="+reminder.alarmDate )
binding.closeIv.setOnClickListener(View.OnClickListener {
remListener.clickRemoveItem(reminder)
})
binding.root.setOnClickListener(View.OnClickListener {
remListener.clickAllItem(reminder)
})
}
}
fun getDateFromMilli(milli: Long?): String {
if(milli == null || milli == -1L) return ""
var calendar: Calendar = Calendar.getInstance()
calendar.timeInMillis = milli
var sdf = SimpleDateFormat("dd.MM.yyyy HH:mm:ss")
var ret = sdf.format(calendar.time)
return ret
}
fun getLimitStr(str: String?, limit: Int): String {
if(str != null && str.length > limit) {
return str.substring(0, limit)
} else {
return str!!
```

```
}
}
}
}
}
```

ReminderNoteActivity.kt:

```
class ReminderNoteActivity : AppCompatActivity(),
TimePickerDialog.OnTimeSetListener,
    DatePickerDialog.OnDateSetListener, OnSecondSetListener {
    var mode: Int = Constatnts.MODE_ADD
    val reminderModel: ReminderModel by viewModels()
    private lateinit var binding: ActivityReminderNoteBinding
    var id: Int = -1
    private var mCalendar: Calendar = Calendar.getInstance()
    private var year: Int = -1
    private var month: Int = -1
    private var day: Int = -1
    private var hour: Int = -1
    private var minute: Int = -1
    private var second: Int = -1
    private var reminderGlobal: Reminder? = null
    private val alarmUtils = AlarmUtils(this)
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
       // reminderModel =
ViewModelProvider(this).get(ReminderModel::class.java)
        binding = DataBindingUtil.setContentView(this,
R.layout.activity_reminder_note)
        if(intent != null) {
            intent.getIntExtra(Constatnts.ID, -1)?.let {
                if(it != -1) {
                    mode = Constatnts.MODE_EDIT
                    id = it
                    reminderModel.getReminderByReminderId(it)
```

```
                    reminderModel.reminderLive.observe(this,
androidx.lifecycle.Observer { reminder ->
                        if(reminder != null) {
                            reminderGlobal = reminder

binding.etTitle.setText(Utils.getLimitStr(reminder.title,
Constatnts.LIMIT_TITLE))

binding.etDesc.setText(Utils.getLimitStr(reminder.desc,
Constatnts.LIMIT_DESC))

Utils.getOnlyDateFromMilli(reminder.alarmDate)?.let { date ->
                                if(date != null &&
!date.equals("")) {

binding.datePicker.setText(date)
                                }
                            }

Utils.getTimeFromMilli(reminder.alarmDate)?.let { time ->
                                if(time != null &&
!time.equals("")) {

binding.timePicker.setText(time)
                                }
                            }

Utils.getSecondFromMilli(reminder.alarmDate)?.let { mSecond ->
                                if(mSecond != null &&
!mSecond.equals("")) {

binding.timePickerSecond.setText(mSecond)
                                }
                            }
                        }
```

```
                    })
                }
            }
        }
        reminderModel.saveRes.observe(this,
androidx.lifecycle.Observer {
            if(it > 0) {
                if(mode == Constatnts.MODE_ADD) {
                    reminderGlobal?.id = it
                }
                if(reminderGlobal!!.alarmDate != null &&
reminderGlobal!!.alarmDate!! > 0L ) {

alarmUtils.setAlarm(reminderGlobal!!.alarmDate!!,
reminderGlobal!!)
                }
                setResult(Activity.RESULT_OK)
                finish()
            }
        })

binding.datePicker.setOnClickListener(View.OnClickListener {
            startDatePicker()
        })

binding.timePicker.setOnClickListener(View.OnClickListener {
            startTimePicker()
        })

binding.timePickerSecond.setOnClickListener(View.OnClickListener
{
            startSecondPicker()
        })
    }
    fun startTimePicker() {
```

```
        TimePickerFragment(this).show(supportFragmentManager,
"time")
    }
    fun startDatePicker() {
        DatePickerFragment(this).show(supportFragmentManager,
"date")
    }
    fun startSecondPicker() {
        SecondsPickerDialog(this).show(supportFragmentManager,
"seconds")
    }
    override fun onCreateOptionsMenu(menu: Menu?): Boolean {
        val inflater = menuInflater
        inflater.inflate(R.menu.menu3, menu)
        return super.onCreateOptionsMenu(menu)
    }
    override fun onOptionsItemSelected(item: MenuItem): Boolean
{
        val id = item.itemId
        if (id == R.id.save) {
            var nowCalendar: Calendar = Calendar.getInstance()
            var noteTime = nowCalendar.timeInMillis
            var alarmTime = getAlarmTime()
            if(alarmTime <= (noteTime+30000)) {
                alarmTime = -1
            }
            var reminder: Reminder? = null
            L.i("noteTime="+noteTime+"|alarmTime="+alarmTime)
            if(mode == Constatnts.MODE_ADD) {
                reminder = Reminder(0,
binding.etTitle.text.toString(),
                binding.etDesc.text.toString(), noteTime,
alarmTime)
                reminderGlobal = reminder
                reminderModel!!.insertReminder(reminder)
```

```
            } else {
                if(reminderGlobal != null &&
reminderGlobal!!.alarmDate != null
                    && reminderGlobal!!.alarmDate!! > 0L
                    && alarmTime != reminderGlobal!!.alarmDate)
{

alarmUtils.removeAlarm(reminderGlobal!!!!/*this.id*/)
                }
                reminder = Reminder(this.id,
binding.etTitle.text.toString(),
                    binding.etDesc.text.toString(), noteTime,
alarmTime)
                reminderGlobal = reminder
                reminderModel!!.updateReminder(reminder)
            }
        }
        return super.onOptionsItemSelected(item)
    }
    fun getAlarmTime() : Long {
        mCalendar = Calendar.getInstance()
        var finalTime: Long = mCalendar.timeInMillis
        var result = finalTime
        var res = finalTime
        if(year != -1) {
            mCalendar.set(Calendar.YEAR, year)
        }
        finalTime = mCalendar.timeInMillis
        res = finalTime - res
        res = result
        if(month != -1) {
            mCalendar.set(Calendar.MONTH, month)
        }
        finalTime = mCalendar.timeInMillis
        res = finalTime - res
```

```
        res = result
        if(day != -1) {
            mCalendar.set(Calendar.DAY_OF_MONTH, day)
        }
        finalTime = mCalendar.timeInMillis
        res = finalTime - res
        res = result
        if(hour != -1) {
            mCalendar.set(Calendar.HOUR_OF_DAY, hour)
        }
        finalTime = mCalendar.timeInMillis
        res = finalTime - res
        res = result
        if(minute != -1) {
            mCalendar.set(Calendar.MINUTE, minute)
        }
        finalTime = mCalendar.timeInMillis
        if(second != -1) {
            mCalendar.set(Calendar.SECOND, second)
        }
        var ppc = Utils.getDateFromMilli(mCalendar.timeInMillis)
        return mCalendar.timeInMillis
    }
    class TimePickerFragment(var timerListener:
TimePickerDialog.OnTimeSetListener) :
        DialogFragment(){
        override fun onCreateDialog(savedInstanceState:
Bundle?): Dialog {
            val c = Calendar.getInstance()
            val hour = c.get(Calendar.HOUR_OF_DAY)
            val minute = c.get(Calendar.MINUTE)
            return TimePickerDialog(activity, timerListener,
hour, minute, DateFormat.is24HourFormat(activity))
        }
    }
```

```
    class DatePickerFragment(var dateListener:
DatePickerDialog.OnDateSetListener ) :
        DialogFragment() {
        override fun onCreateDialog(savedInstanceState:
Bundle?): Dialog {
            val c = Calendar.getInstance()
            val year = c.get(Calendar.YEAR)
            val month = c.get(Calendar.MONTH)
            val day = c.get(Calendar.DAY_OF_MONTH)
            return DatePickerDialog(requireActivity(),
dateListener, year, month, day)
        }
    }
    class SecondsPickerDialog(var secondListener:
OnSecondSetListener) : DialogFragment() {
        var selectSecond = 0
        override fun onCreateDialog(savedInstanceState:
Bundle?): Dialog {
            val lstNums: ArrayList<String> = ArrayList()
            for(i: Int in 1..60) {
                if(i==60) lstNums.add("0")
                else lstNums.add(""+i)
            }
            val inflater = layoutInflater
            val alertLayout: View =
inflater.inflate(R.layout.dialog_second, null)
            var spinner = alertLayout.findViewById(R.id.spinner)
as MaterialSpinner
            spinner.setItems(lstNums)
            spinner.setOnItemSelectedListener { view, position,
id, item ->
                var itemInt = item as String
                selectSecond = itemInt.toInt()
            }
            return AlertDialog.Builder(requireContext()).
```

```
            setView(alertLayout).
            setPositiveButton("Ok", object :
DialogInterface.OnClickListener {
                override fun onClick(p0: DialogInterface?, p1:
Int) {
                    secondListener.onSecondSet(selectSecond)
                }
            }).create()
        }
    }
    override fun onTimeSet(view: TimePicker, hourOfDay: Int,
minute: Int) {
        L.i("hourOfDay="+hourOfDay+"|minute="+minute)
        hour = hourOfDay
        this.minute = minute
        var hourStr = if(hourOfDay < 10) "0"+hourOfDay else
""+hourOfDay
        var minuteStr = if(minute < 10) "0"+minute else
""+minute
        binding.timePicker.text = "" + hourStr + " : " +
minuteStr
    }
    override fun onDateSet(view: DatePicker, year: Int, month:
Int, day: Int) {
        this.year = year
        this.month = month
        this.day = day
        var monthStr = if(month < 10) "0"+(month+1) else
""+month
        var dayStr = if(day < 10) "0"+day else ""+day
        binding.datePicker.text = "" + dayStr + "." + monthStr +
"." + year
    }
    override fun onSecondSet(second: Int) {
        this.second = second
```

```
        var secondStr = if(second < 10) "0"+second else
""+second
        binding.timePickerSecond.text = ""+secondStr
    }
    fun test() {
        var calendar: Calendar = Calendar.getInstance()
        calendar.add(Calendar.MILLISECOND, 43556888)
    }
}
```