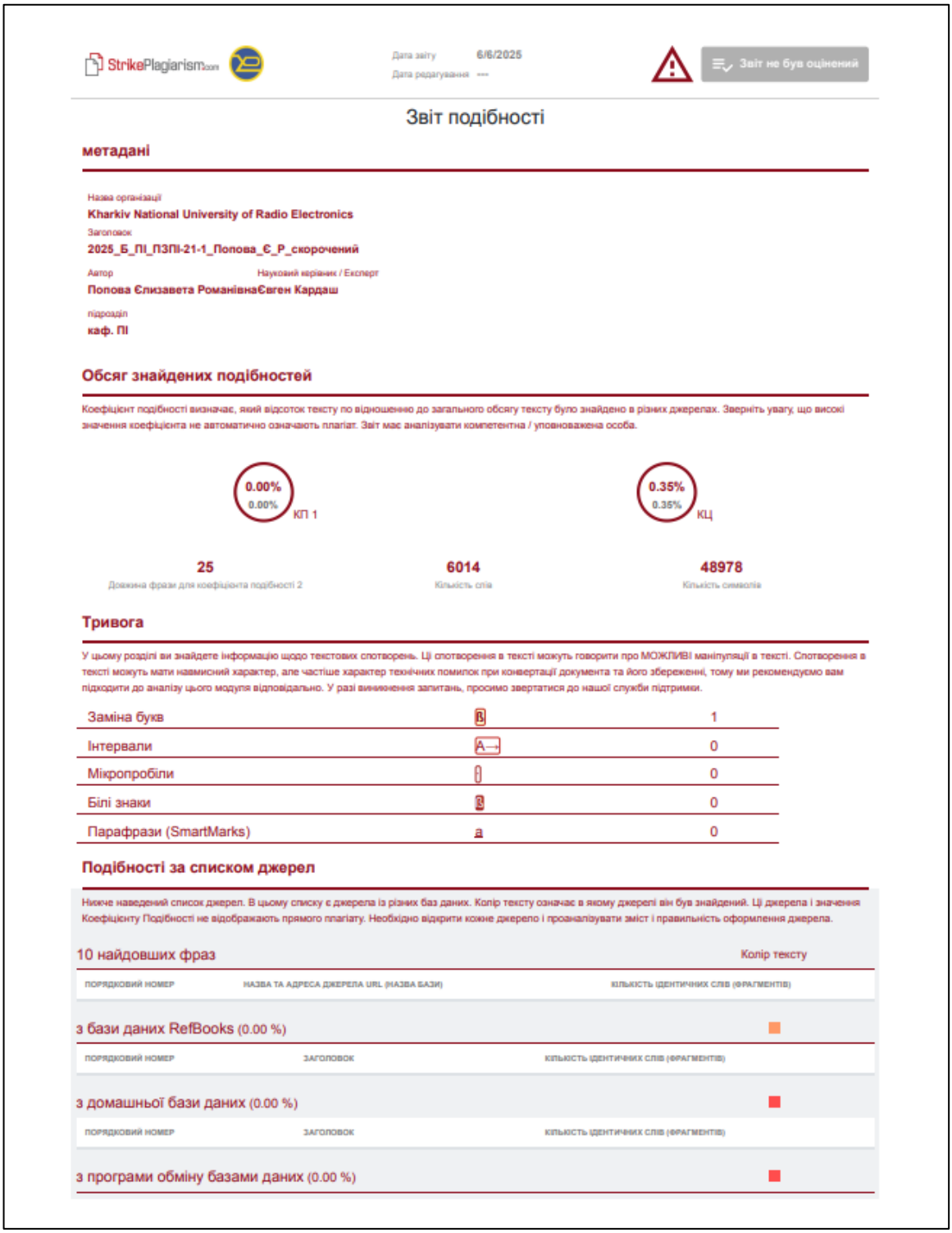
Додаток А

Звіт результатів перевірки на унікальність тексту в базі хнуре



StrikePlagiarism.com

Дата звіту 6/6/2025
Дата редагування ---

Звіт не був оцінений

Звіт подібності

метадані

Назва організації
Kharkiv National University of Radio Electronics
Заголовок
2025_Б_ПІ_ПЗПІ-21-1_Попова_Є_Р_скорочений
Автор
Попова Єлизавета Романівна
Науковий керівник / Експерт
Євген Кардаш
підрозділ
каф. ПІ

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

0.00% 0.00% КП 1

0.35% 0.35% КЦ

25	6014	48978
Довжина фрази для коефіцієнта подібності 2	Кількість слів	Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв	В	1
Інтервали	A⟶	0
Мікропробіли	¶	0
Білі знаки	␣	0
Парафрази (SmartMarks)	a	0

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)

з бази даних RefBooks (0.00 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)

з домашньої бази даних (0.00 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)

з програми обміну базами даних (0.00 %)

Рисунок А.1 – Результат перевірки на унікальність тексту

ДОДАТОК Б
Слайди презентації

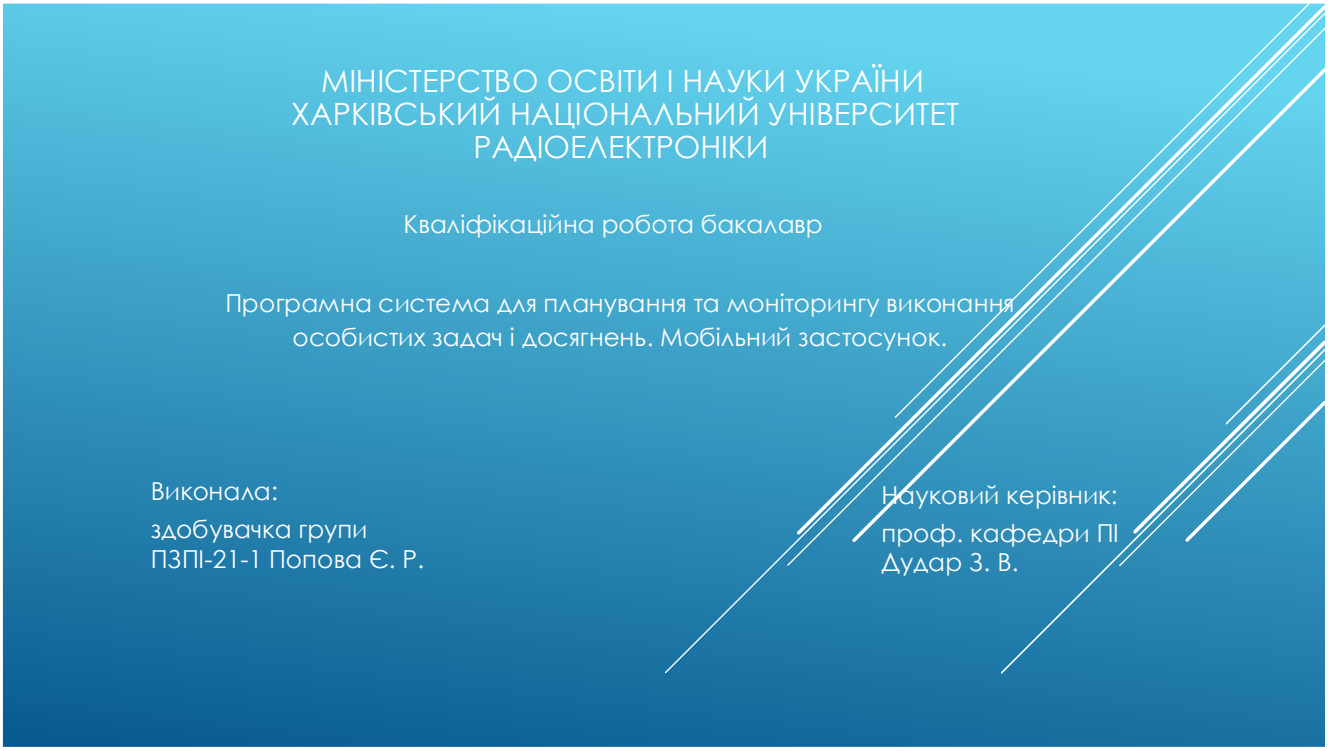


Рисунок Б.1 – Слайд презентації №1

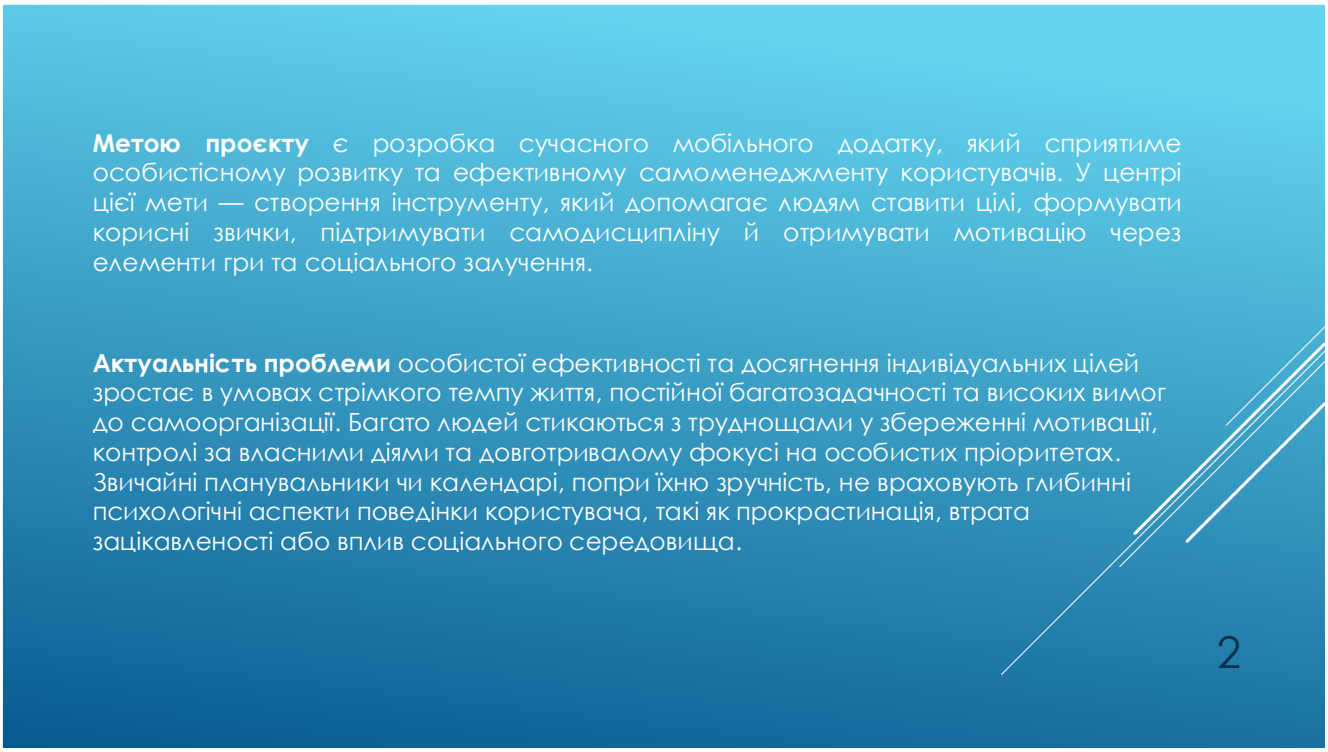


Рисунок Б.2 – Слайд презентації №2

ПЕРЕЛІК ДОСЛІДЖЕНИХ КОНКУРЕНТІВ

Назва	Сильні сторони	Слабкі сторони
Habitica	- Гейміфікація та рольова механіка мотивує - Соціальні функції (групи, квести)	- Складний інтерфейс для новачків - Може здатися «іграшковим» для дорослих
Any.do	- Мінімалістичний дизайн - Інтеграція з календарем і нагадуваннями	- Мало функцій для розвитку звичок - Більше орієнтований на завдання
Coach.me	- Підтримка коучів - Спільнота та фідбек	- Багато функцій платні - Менше інструментів для глибокого трекінгу
Loop	- Повністю безкоштовний - Простий, інтуїтивний інтерфейс	- Відсутність хмарної синхронізації - Немає підтримки чи спільноти
Respawn	- Система стекування звичок - Фокус на глибокій роботі та відновленні	- Обмежена доступність (не всюди поширена) - Новий продукт, мало відгуків

3

Рисунок Б.3 – Слайд презентації №3

ПОСТАНОВКА ЗАДАЧІ ТА ОПИС СИСТЕМИ

Ключовою задачею є забезпечення інтуїтивно зрозумілого інтерфейсу для:

- автентифікації та реєстрації користувачів;
- управління задачами та цілями (створення, редагування, нагадування, дедлайни);
- візуалізації аналітики щодо виконання завдань;
- реалізації системи гейміфікації (нагороди, рівні, рейтинг);
- інтеграції соціальних функцій (друзі, повідомлення);
- використання персонального AI-помічника для порад та автоматизованого планування.

4

Рисунок Б.4 – Слайд презентації №4

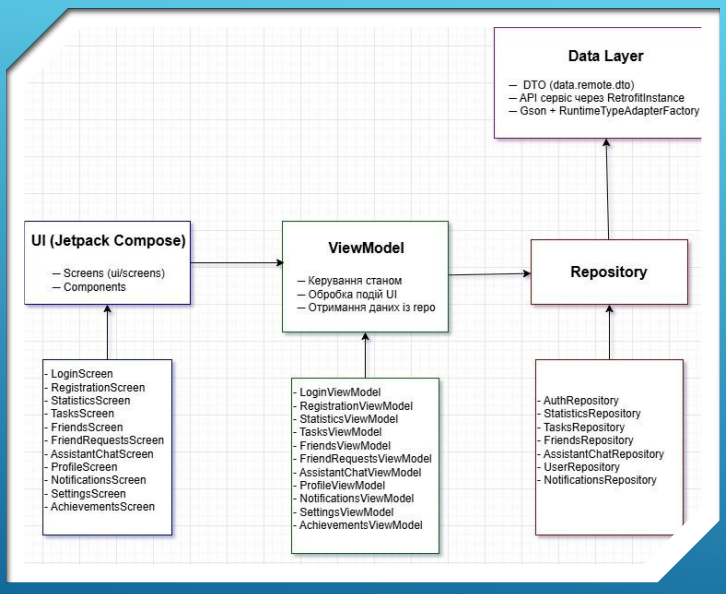
ПОСТАНОВКА ЗАДАЧІ ТА ОПИС СИСТЕМИ

Результатом розробки має стати мобільний застосунок із наступними характеристиками:

- Застосунок має реалізовувати повний набір інструментів для управління задачами, аналітики, гейміфікації та соціальної взаємодії.
- Інтерфейс застосунку повинен мати сучасний та адаптивний дизайн.
- Застосунок повинен забезпечувати захищені автентифікацію, передачу та зберігання персональних даних.
- Застосунок повинен бути оптимізованим для пристроїв середнього рівня, мати швидкий запуск та економія енергії.
- Повинен бути зпроєктований так щоб в майбутніх релізах архітектура можливість додавання нових платформ, функцій, інтеграції з ML-моделями, офлайн-режиму.AI-функціональність:
- Має бути реалізовага базова взаємодія з персональним AI-помічником через чат для мотивації, генерації задач та рекомендацій.

5

Рисунок Б.5 – Слайд презентації №5



АРХІТЕКТУРА СТВОРЕННОГО МОБІЛЬНОГО ЗАСТОСУНКА

6

Рисунок Б.6 – Слайд презентації №6

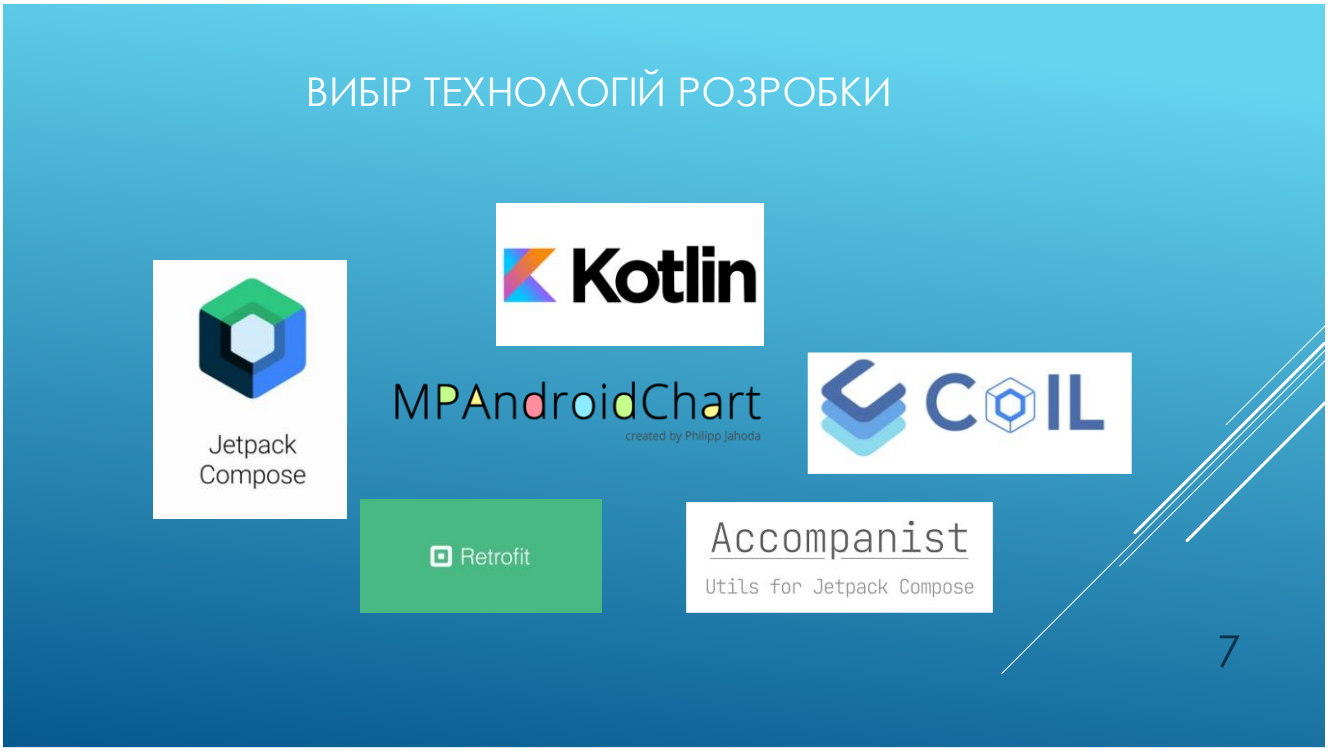


Рисунок Б.7 – Слайд презентації №7

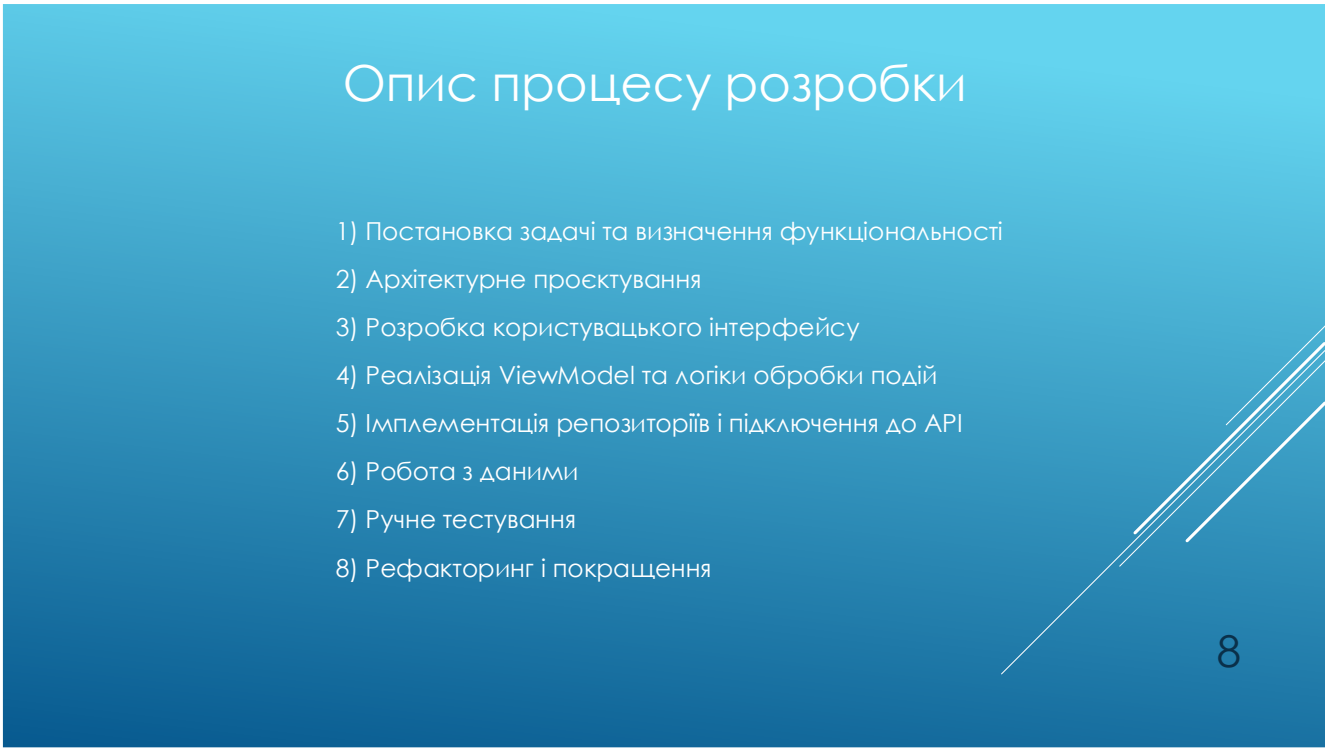


Рисунок Б.8 – Слайд презентації №8

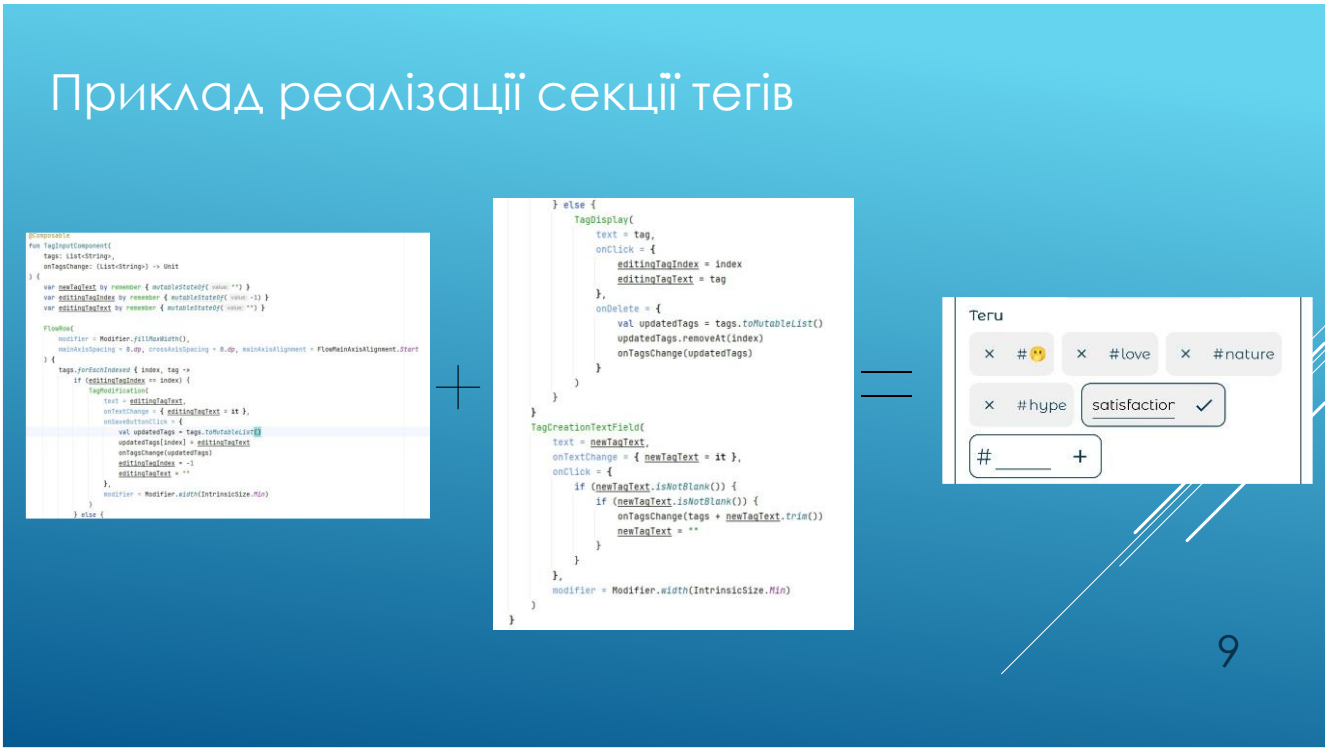


Рисунок Б.9 – Слайд презентації №9

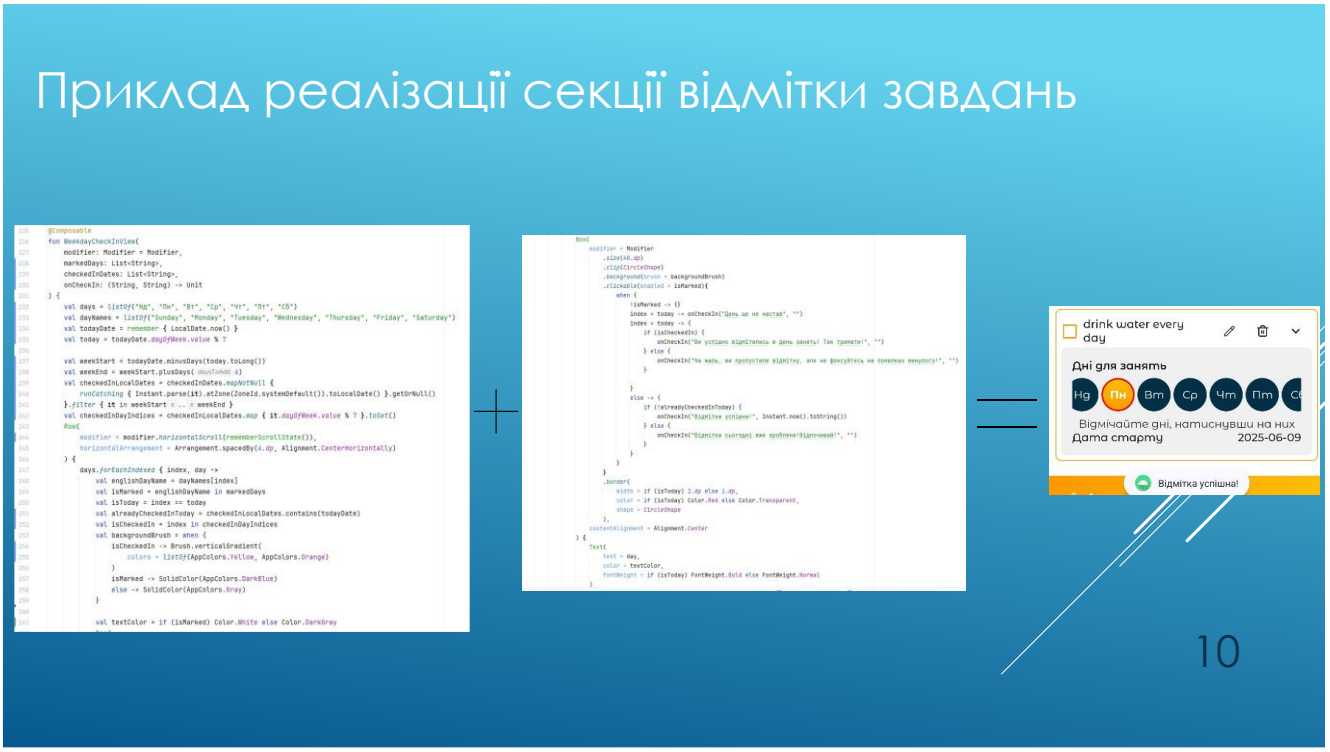


Рисунок Б.10 – Слайд презентації №10

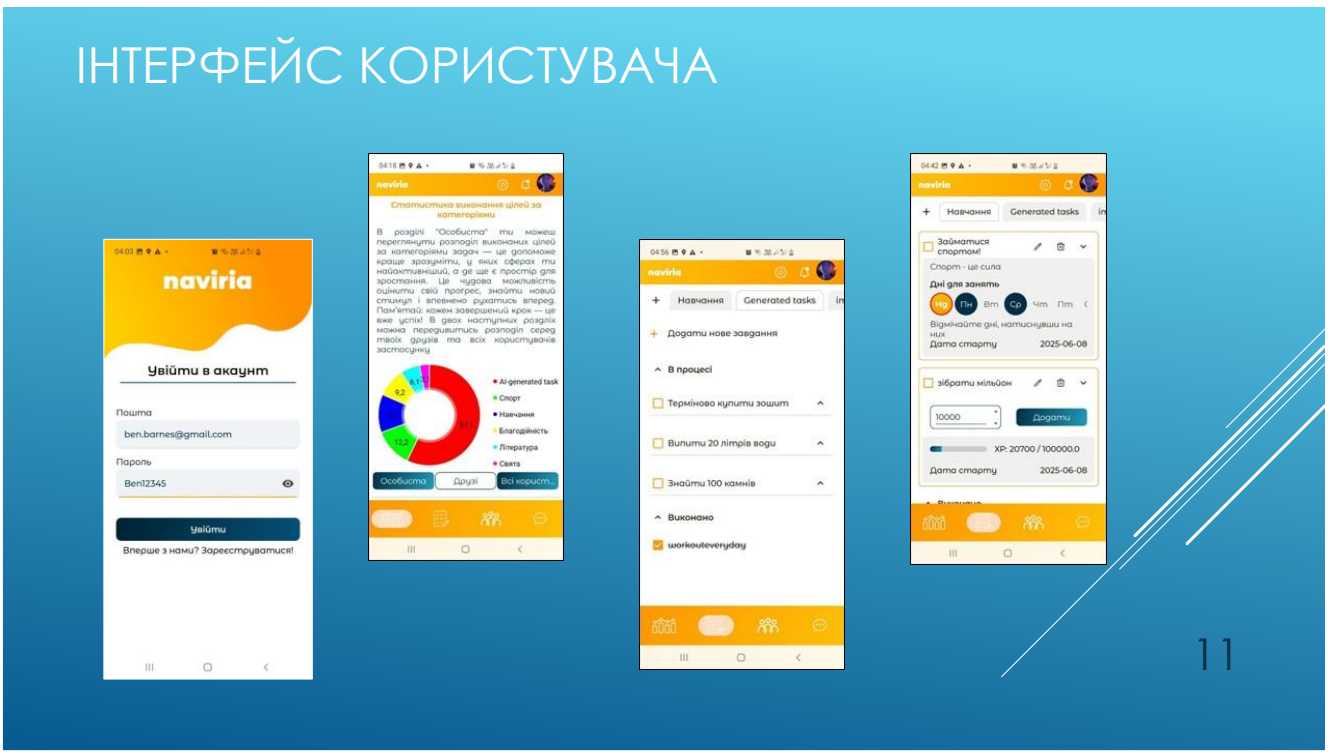


Рисунок Б.11 – Слайд презентації №11

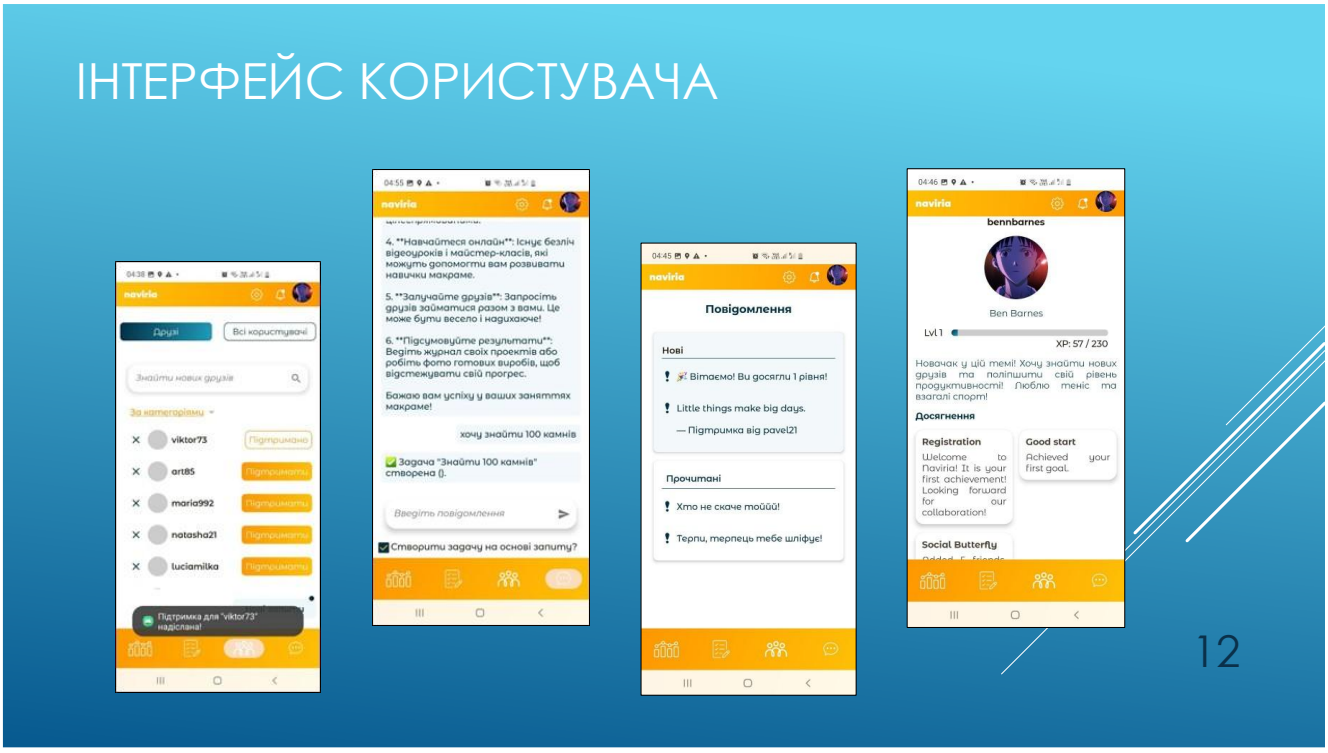


Рисунок Б.12 – Слайд презентації №12

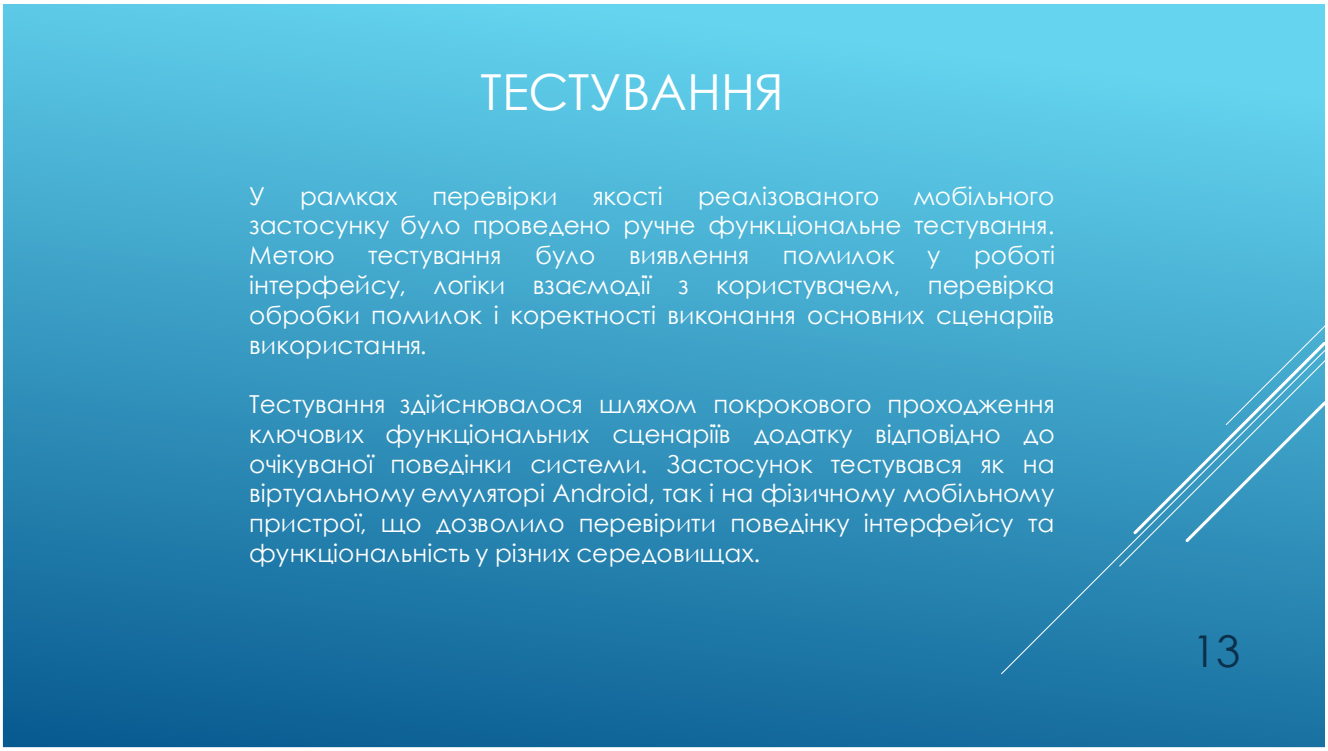


Рисунок Б.13 – Слайд презентації №13

Рисунок Б.14 – Слайд презентації №14

ДОДАТОК В

Діаграма варіантів використання

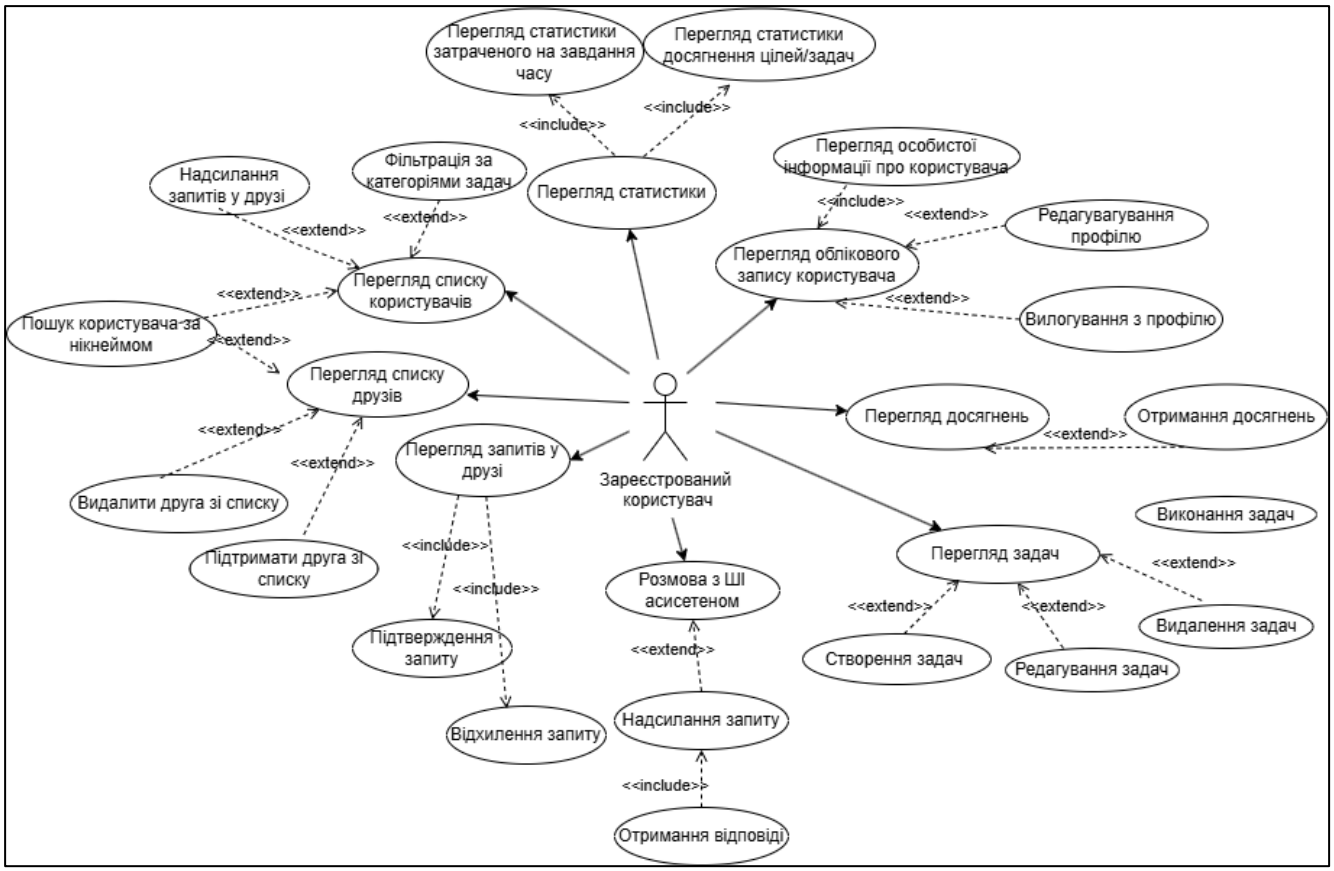


Рисунок В.1 – Діаграма варіантів використання для зареєстрованого користувача

ДОДАТОК Г

Результат ui-ux розробки

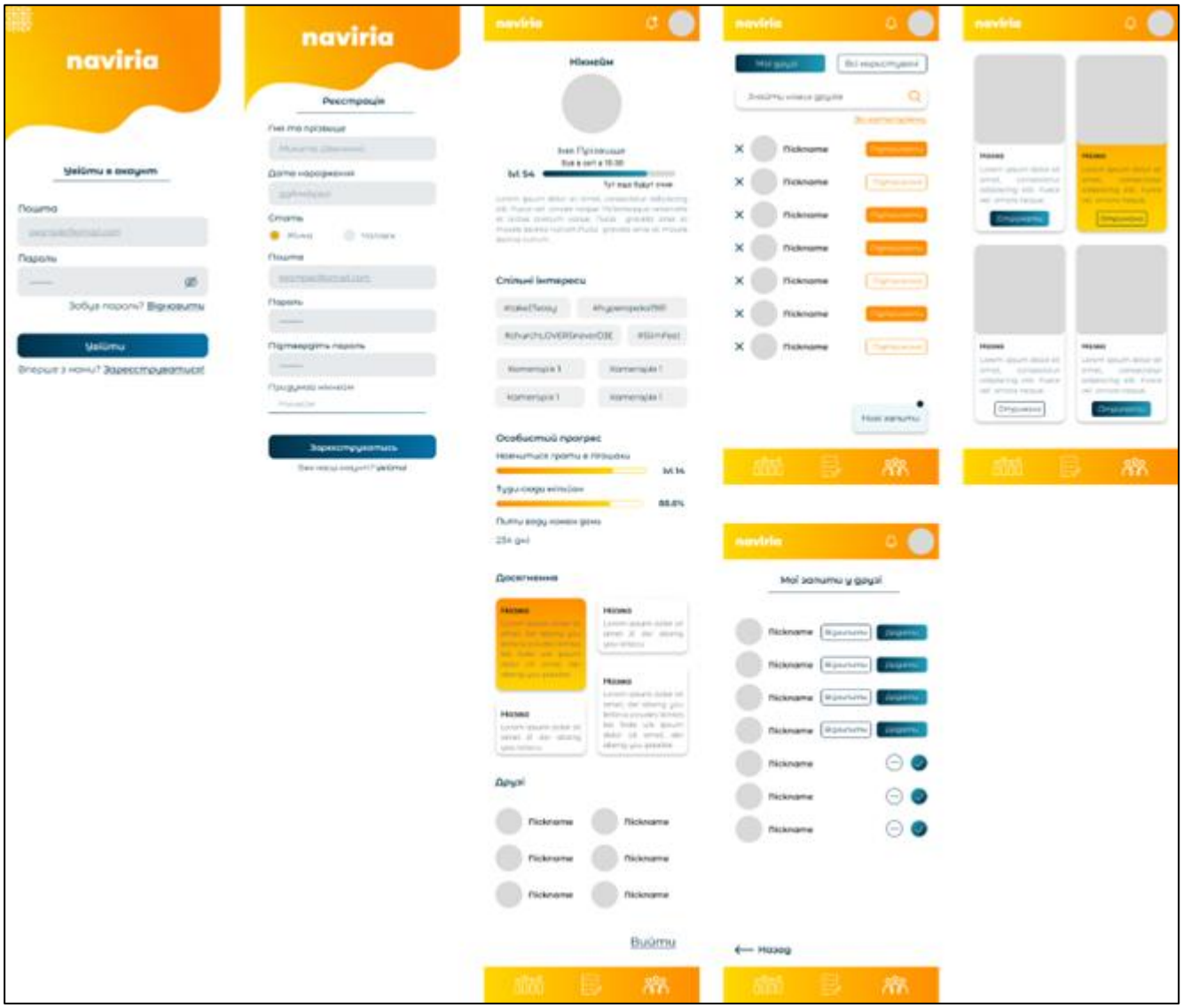


Рисунок Г.1 – Розробка дизайну екранів авторизації, реєстрація, профіля користувача, друзі, запити у друзі

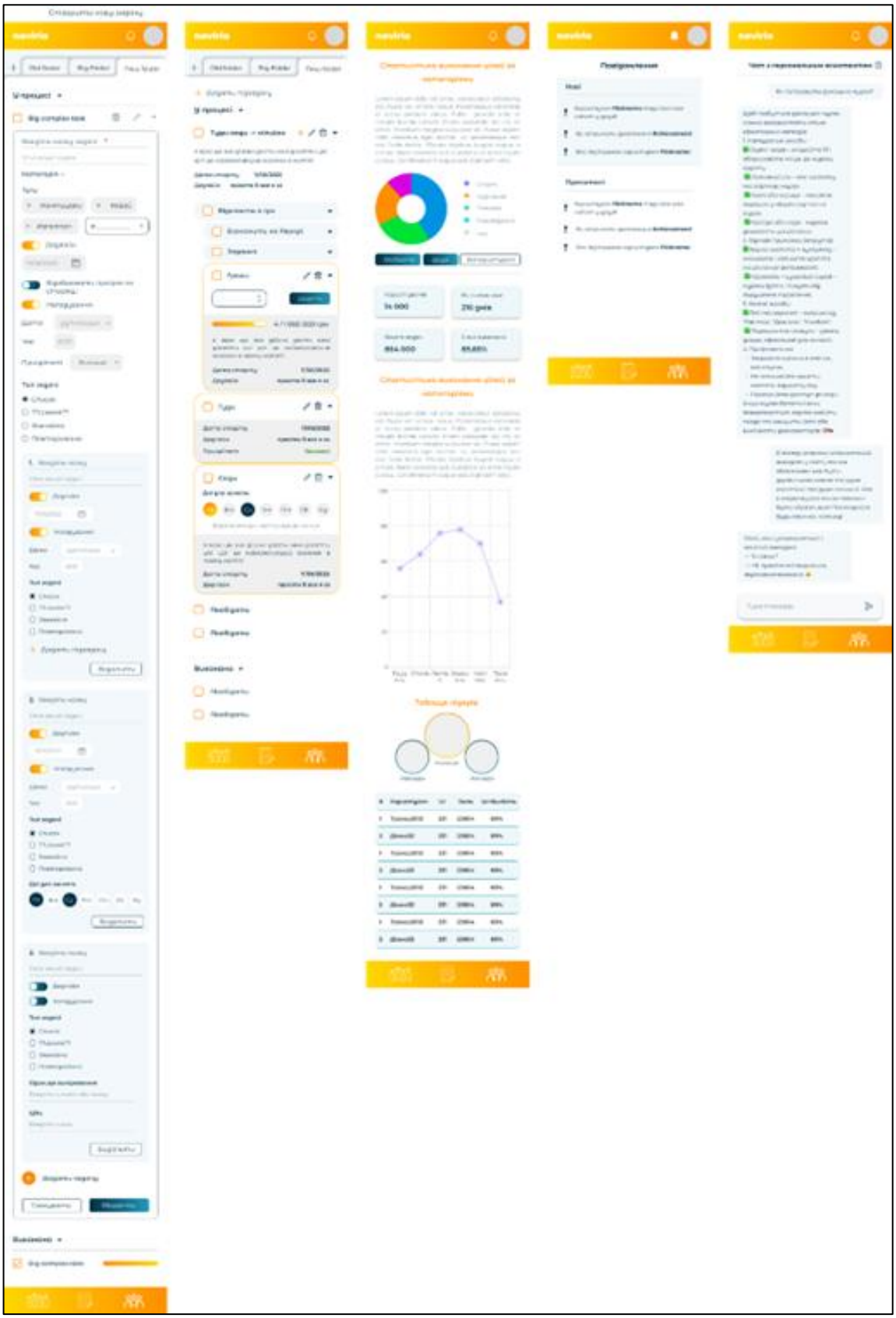


Рисунок Г.1 – Розробка дизайну екранів задачі, статистики, повідомлення, чат з помічником

ДОДАТОК Д
Скріншоти програмного застосунку

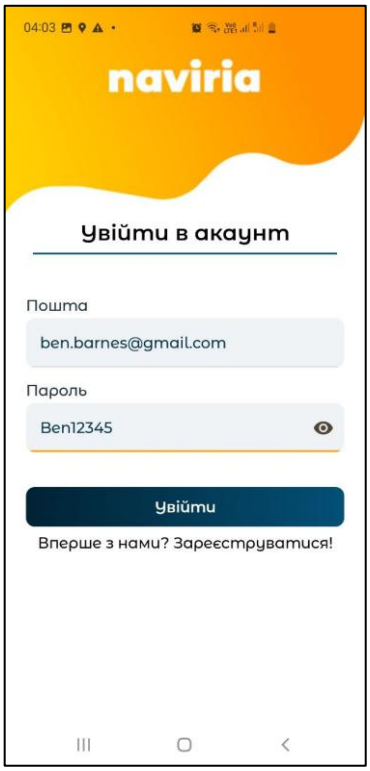


Рисунок Д.1 – Екран авторизації

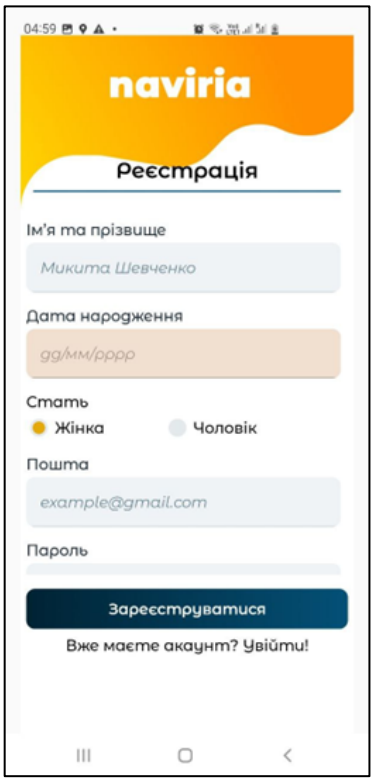


Рисунок Д.2 – Екран регістрації

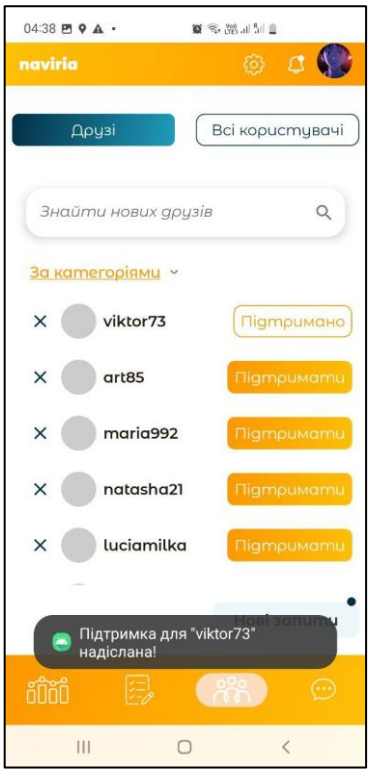


Рисунок Д.3 – Екран «друзі» під час здійснення функціоналу підтримки друга

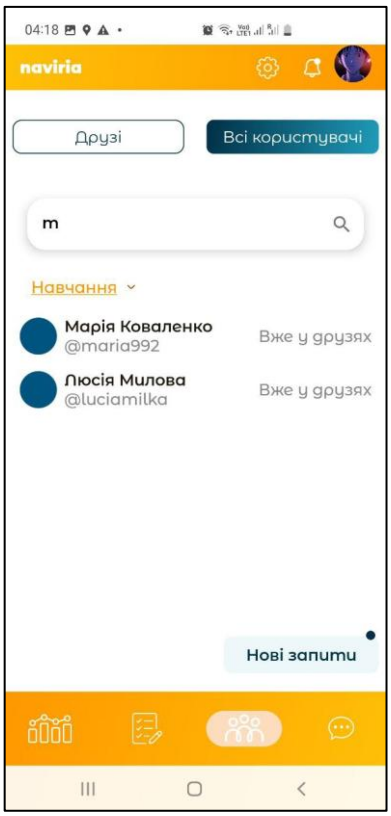


Рисунок Д.4 – Екран «друзі» під час здійснення функціоналу пошуку за вводом «m», категорію «Навчання» у вкладці всі користувачі

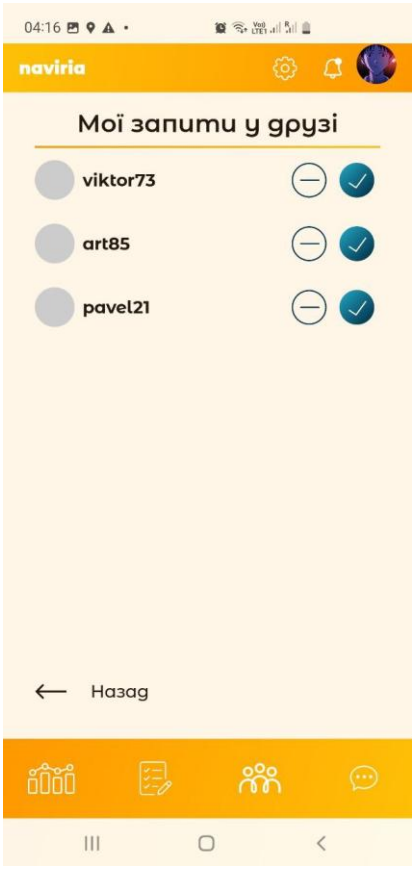


Рисунок Д.5 – Екран «запити у друзів»

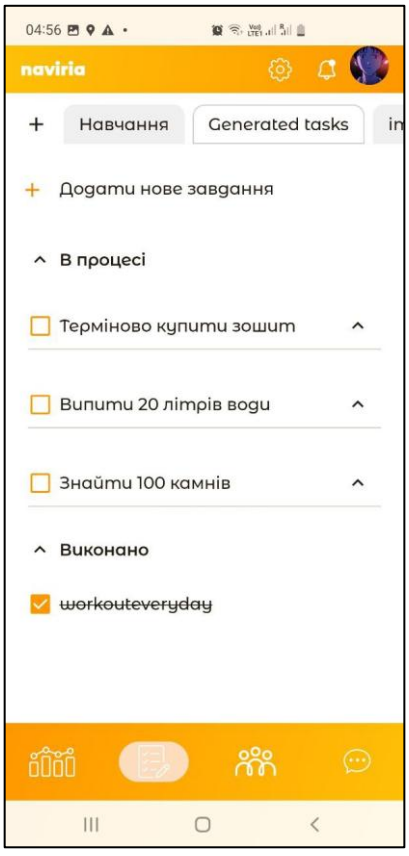


Рисунок Д.6 – Екран «задачі»

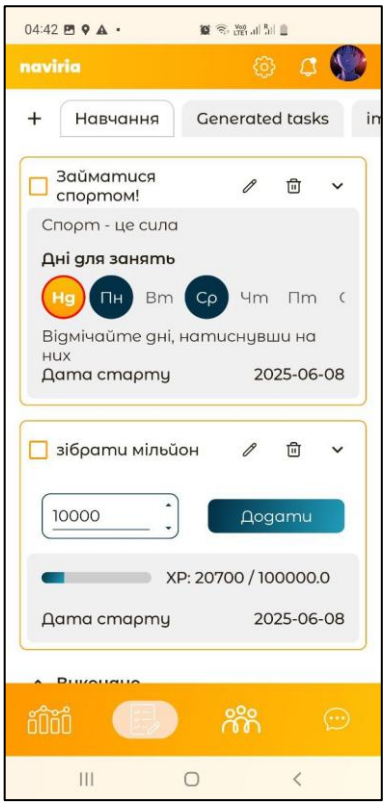


Рисунок Д.7 – Екран «задачі» демонстрація задач типу «повторювальна» та «зі шкалою»

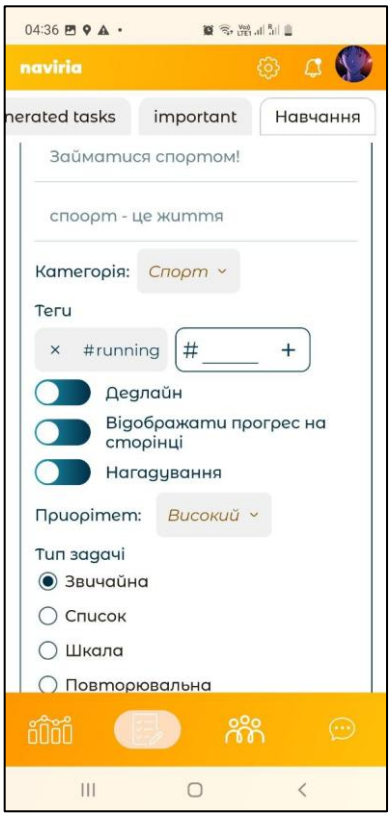


Рисунок Д.8 – Екран «задачі», форма створення нової задачі

Рисунок Д.9 – Екран «задачі», форма для підзадачі у формі створення задачі

Рисунок Д.10 – Екран «статистики» діаграма розподілу категорій

Рисунок Д.11 – Екран «статистики» картки зі зведеними показниками

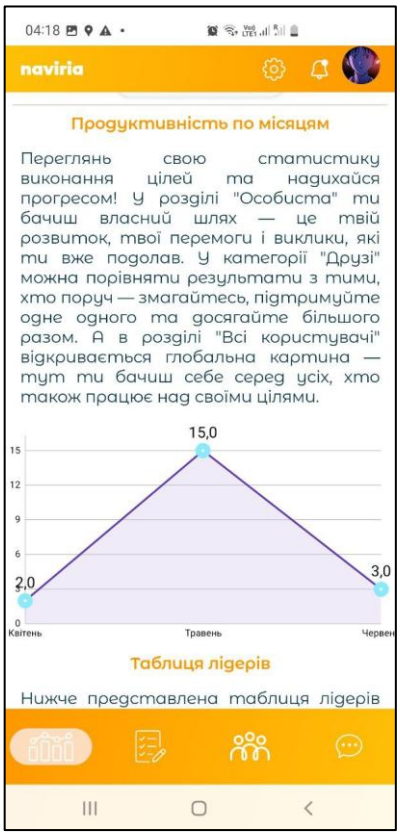


Рисунок Д.12 – Екран «статистики» лінійна діаграма продуктивності

Рисунок Д.13 – Екран «статистики» таблиця лідерів

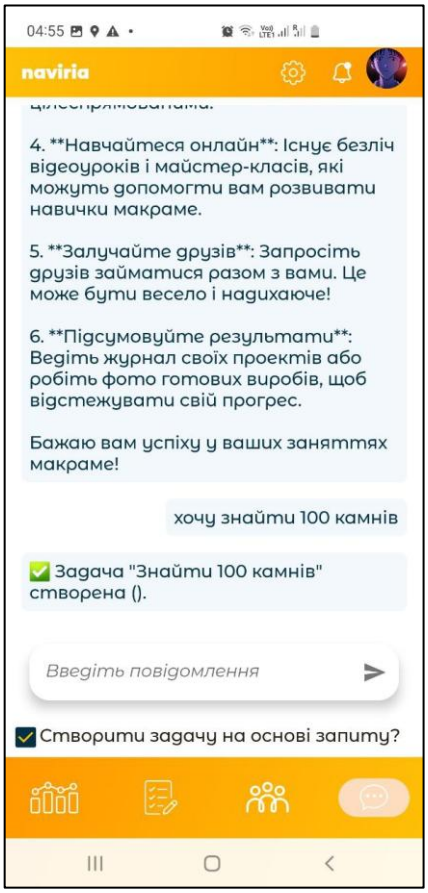


Рисунок Д.14 – Екран «чат з помічником»

Рисунок Д.15 – Екран «повідомлення»

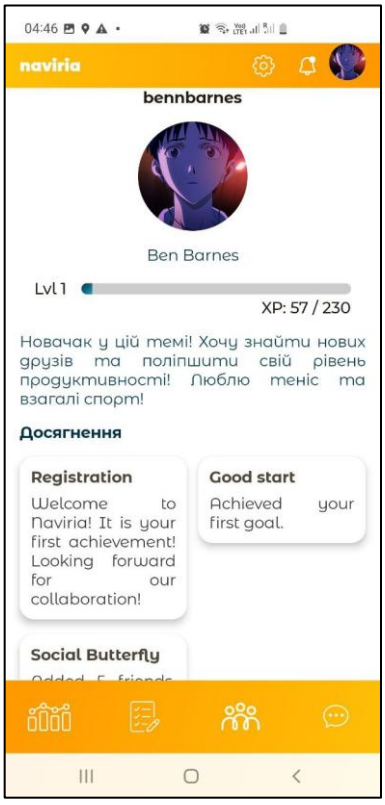


Рисунок Д.16 – Екран «профіля користувача» верхня частина

Рисунок Д.17 – Екран «профіля користувача» нижння частина

Рисунок Д.18 – Екран «досягнення користувача»

ДОДАТОК Є
Програмний код

Є.1 Код файлу RetrofitInstance.kt

```
package com.example.dyplomproject.utils

object RetrofitInstance {
    private const val BASE_URL = "http://192.168.1.9:5186/"

    private val loggingInterceptor = HttpLoggingInterceptor().apply {
        level = HttpLoggingInterceptor.Level.BODY
    }

    private val httpClient = OkHttpClient.Builder()
        .addInterceptor(loggingInterceptor)
        .build()


    private val taskCreateAdapterFactory = RuntimeTypeAdapterFactory.of(
        TaskCreateDto::class.java,
        "type"
    )
        .registerSubtype(TaskStandardCreateDto::class.java, "standard")
        .registerSubtype(TaskRepeatableCreateDto::class.java, "repeatable")
        .registerSubtype(TaskScaleCreateDto::class.java, "scale")
        .registerSubtype(TaskWithSubtasksCreateDto::class.java,
"with_subtasks")

    val taskDtoAdapterFactory =
RuntimeTypeAdapterFactory.of(TaskDto::class.java, "type")
        .registerSubtype(TaskStandardDto::class.java, "standard")
        .registerSubtype(TaskRepeatableDto::class.java, "repeatable")
        .registerSubtype(TaskScaleDto::class.java, "scale")
        .registerSubtype(TaskWithSubtasksDto::class.java, "with_subtasks")

    val subtaskCreateDtoAdapterFactory = RuntimeTypeAdapterFactory.of(
        SubtaskCreateDto::class.java,
        "subtask_type"
    )
        .registerSubtype(SubtaskStandardCreateDto::class.java, "standard")
        .registerSubtype(SubtaskRepeatableCreateDto::class.java,
"repeatable")
        .registerSubtype(SubtaskScaleCreateDto::class.java, "scale")

    val subtaskDtoAdapterFactory =
RuntimeTypeAdapterFactory.of(SubtaskDto::class.java, "type")
        .registerSubtype(SubtaskStandardDto::class.java, "standard")
        .registerSubtype(SubtaskRepeatableDto::class.java, "repeatable")
        .registerSubtype(SubtaskScaleDto::class.java, "scale")


    private val gson: Gson = GsonBuilder()
        .registerTypeAdapterFactory(taskCreateAdapterFactory)
        .registerTypeAdapterFactory(taskDtoAdapterFactory)
        .registerTypeAdapterFactory(subtaskCreateDtoAdapterFactory)
```

```
            .registerTypeAdapterFactory(subtaskDtoAdapterFactory)
            .create()

        val api: ApiService by lazy {
            Retrofit.Builder()
                .baseUrl(BASE_URL)
                .client(httpClient)
                .addConverterFactory(GsonConverterFactory.create(gson))
                .build()
                .create(ApiService::class.java)
        }
}
```

Є.2 Код функції MyApp() файла MainActivity

```
@Composable
fun MyApp(authViewModel: AuthViewModel, loginViewModel: LoginViewModel) {
    val navController = rememberNavController()

    val isAuthenticated by authViewModel.isAuthenticated.collectAsState()
    val userId by authViewModel.userId.collectAsState()
    val notificationViewModel: NotificationViewModel? = userId?.let {
        val repo = NotificationRepository(RetrofitInstance.api)
        viewModel(factory = NotificationViewModelFactory(repo, it))
    }

    NavHost(
        navController = navController,
        startDestination = if (isAuthenticated && userId != null) "main"
else "login"
    ) {
        composable("login") { LoginScreen(navController, loginViewModel,
authViewModel) }
        composable("register") { RegistrationScreen(navController,
authViewModel) }

        navigation(startDestination = "friends", route = "main") {
            composable("friends") {
                MainScaffold(navController, notificationViewModel) {
padding ->
                    RequireUserId(userId, padding) { id ->
                        FriendsScreen(navController, id, padding)
                    }
                }
```

```
            }
            composable("statistics") {
                MainScaffold(navController, notificationViewModel) {
padding ->
                    RequireUserId(userId, padding) { id ->
                        StatisticsScreen(navController, id, padding)
                    }
                }
            }

            composable("tasks") {
                MainScaffold(navController, notificationViewModel) {
padding ->
                    RequireUserId(userId, padding) { id ->
                        TaskScreen(navController, id, padding)
                    }
                }
            }

            composable("assistant_chat") {
                MainScaffold(navController, notificationViewModel) {
padding ->
                    RequireUserId(userId, padding) { id ->
                        AssistantChatScreen(navController, id, padding)
                    }
                }
            }

            navigation(route = "secondary", startDestination = "profile") {
                composable("profile") {
                    MainScaffold(navController, notificationViewModel) {
padding ->
                        RequireUserId(userId, padding) { id ->
                            ProfileScreen(navController,authViewModel, id,
padding)
                        }
                    }
                }
                composable("friend_requests") {
```

```
                    MainScaffold(navController, notificationViewModel) {
padding ->
                        RequireUserId(userId, padding) { id ->
                            FriendRequestsScreen(navController, id,
padding)
                        }
                    }
                }
                composable("notifications") {
                    MainScaffold(navController, notificationViewModel) {
padding ->
                        RequireUserId(userId, padding) { id ->
                            NotificationScreen(navController, id, padding)
                        }
                    }
                }
                composable("achievements") {
                    MainScaffold(navController, notificationViewModel) {
padding ->
                        RequireUserId(userId, padding) { id ->
                            AchievementsScreen(navController, id, padding)
                        }
                    }
                }

                composable("settings") {
                    MainScaffold(navController, notificationViewModel) {
padding ->
                        RequireUserId(userId, padding) { id ->
                            SettingsScreen(navController, id, padding)
                        }
                    }
                }
            }
        }
    }
}
```

Є.3 Код файлу AuthRepository

```
package com.example.dyplomproject.data.remote.repository


class AuthRepository(private val apiService: ApiService) {
    suspend fun login(request: LoginRequestDto): Result<AuthResponseDto> {
        return try {
            val response = apiService.login(request)
            if (response.isSuccessful) {
                Result.success(response.body()!!)
            } else {
                val userMessage = when (response.code()) {
                    400 -> "Некоректні дані. Перевірте заповнення полів."
                    401 -> "Невірний email або пароль."
                    403 -> "Доступ заборонено."
                    404 -> "Сервер не знайдено."
                    500 -> "Помилка сервера. Спробуйте пізніше."
                    else -> "Невідома помилка: ${response.code()}"
                }
                Result.failure(Exception(userMessage))
            }
        } catch (e: Exception) {
            Result.failure(Exception("Не вдалось під'єднатися до сервера,
перевірте підключення!"))
        }
    }

    suspend fun register(request: RegistrationRequestDto):
Result<AuthResponseDto> {
        return try {
            val response = apiService.register(request)
            if (response.isSuccessful) {
                Result.success(response.body()!!)
            } else {
                val userMessage = when (response.code()) {
                    409 -> "Пошта вже використовується! Ввійдіть в вже
створений акаунт або використайте іншу пошту для реєстрації"
                    422 -> "Нікнейм зайнятий, вигадайте інший!"
                    500 -> "Пошта або нікнейм вже використовуються!
Використовуйте іншу пошту або змініть нік"
                    else -> "Невідома помилка: ${response.code()}"
                }
                Result.failure(Exception(userMessage))
            }
        } catch (e: Exception) {
            Result.failure(Exception("Не вдалось під'єднатися до сервера,
перевірте підключення!"))
        }
    }
}
```

Є.4 Код файлу FriendsViewModel

```
package com.example.dyplomproject.ui.viewmodel

class FriendsViewModel(
    private val repository: UserRepository =
UserRepository(RetrofitInstance.api),
```

```
    private val userId: String
) : ViewModel() {
    private val _uiState = MutableStateFlow(FriendsUiState())
    val uiState: StateFlow<FriendsUiState> = _uiState.asStateFlow()

    private val _messageFlow = MutableSharedFlow<String>()
    val messageFlow = _messageFlow.asSharedFlow()

    fun onTabSelected(tab: TabType) {
        resetSearch()
        _uiState.value = _uiState.value.copy(selectedTab = tab)
        when (tab) {
            TabType.FRIENDS -> if (_uiState.value.friends.isEmpty())
loadFriends()
            TabType.ALL_USERS -> if (_uiState.value.allUsers.isEmpty())
loadAllUsers()
        }
    }

    fun onRefresh(){
        if (_uiState.value.selectedTab == TabType.FRIENDS) {
            loadFriends()
        } else {
            loadAllUsers()
        }
    }

    private fun loadFriends() = launchWithStateUpdate {
        val result = repository.getFriends(userId)
        _uiState.value = if (result.isSuccess) {
            _uiState.value.copy(friends = result.getOrThrow(), error =
null)
        } else {
            _uiState.value.copy(error = result.exceptionOrNull()?.message)
        }
    }

    fun loadAllUsers() = launchWithStateUpdate {
        val result = repository.getAllUsers(userId)
        _uiState.value = if (result.isSuccess) {
            _uiState.value.copy(allUsers = result.getOrThrow(), error =
null)
        } else {
            _uiState.value.copy(error = result.exceptionOrNull()?.message)
        }
    }

    fun loadCategories() = launchWithStateUpdate {
        val result = repository.getCategories()
        _uiState.value = if (result.isSuccess) {
            _uiState.value.copy(categories = result.getOrThrow(), error =
null)
        } else {
            _uiState.value.copy(error = result.exceptionOrNull()?.message)
        }
    }

    private fun launchWithStateUpdate(block: suspend () -> Unit) {
```

```
        viewModelScope.launch {
            _uiState.value = _uiState.value.copy(isLoading = true)
            try {
                block()
            } finally {
                _uiState.value = _uiState.value.copy(isLoading = false)
            }
        }
    }

    enum class TabType {
        FRIENDS, ALL_USERS
    }

    fun checkNewFriendRequests(userId: String) = launchWithStateUpdate {
        val result = repository.getIncomingUserRequests(userId)
        _uiState.value = result.fold(
            onSuccess = { list ->
                if (list.isEmpty()) {
                    _uiState.value.copy(hasNewFriendRequests = false)
                } else {
                    _uiState.value.copy(hasNewFriendRequests = true)
                }
            },
            onFailure = { exception ->
                _uiState.value.copy(error = exception.message)
            }
        )
    }

    fun sendFriendRequest(user: UserShortUiModel) = launchWithStateUpdate {
        val request = FriendRequest(userId, user.id)
        val result = repository.sendFriendRequest(request)
        if (result.isSuccess) {
            val updatedUser = user.copy(isRequestSent = true)
            val updatedList = _uiState.value.allUsers.map {
                if (it.id == updatedUser.id) updatedUser else it
            }
            _uiState.value = _uiState.value.copy(allUsers = updatedList)
            _messageFlow.emit("Запит до користувача \"${user.nickname}\"
надіслано!")
        } else {
            _messageFlow.emit("Сталася помилка!")
        }
    }

    fun sendFriendSupport(user: UserShortUiModel) = launchWithStateUpdate {
        val result = repository.sendFriendSupport(userId, user.id)
        if (result.isSuccess) {
            val updatedUser = user.copy(isRequestSent = true)
            val updatedList = _uiState.value.friends.map {
                if (it.id == updatedUser.id) updatedUser else it
            }
            _uiState.value = _uiState.value.copy(friends = updatedList)
            _messageFlow.emit("Підтримка для \"${user.nickname}\"
надіслана!")
        } else {
            result.exceptionOrNull()?.message?.let { _messageFlow.emit(it +
```

```
"\"${user.nickname}\" вже підтриманий!") }
        }
    }

    fun removeUserFromFriends(user: UserShortUiModel) =
launchWithStateUpdate {
        val result = repository.removeUserFromFriends(userId, user.id)
        if (result.isSuccess) {
            val updatedList = _uiState.value.friends.filterNot { it.id ==
user.id }
            _uiState.value = _uiState.value.copy(friends = updatedList)
            _messageFlow.emit("Користувач \"${user.nickname}\" успішно
видалений з друзів")
        } else {
            _messageFlow.emit("Не вдалось видалити \"${user.nickname}\" з
друзів! Спробуйте ще раз!")
        }
    }

    fun onSearchQueryChanged(query: String) {
        _uiState.update { it.copy(searchQuery = query) }
    }

    fun resetSearch() {
        _uiState.update {
            it.copy(
                searchQuery = "",
                searchResults = null,
                selectedCategory = Category("", "За категоріями")
            )
        }
    }

    fun onSelectedCategoryChanged(category: Category) {
        _uiState.update { it.copy(selectedCategory = category) }
        searchUsers()
    }

    fun searchUsers() = launchWithStateUpdate {
        val query = _uiState.value.searchQuery.trim()
        val categoryId = _uiState.value.selectedCategory.id
        if (query.isEmpty() && categoryId.isEmpty()) {
            _uiState.update { it.copy(searchResults = null) }
            return@launchWithStateUpdate
        }

        val result = when (_uiState.value.selectedTab) {
            TabType.FRIENDS -> repository.searchAmongFriends(userId,
categoryId, query)
            TabType.ALL_USERS -> repository.searchAmongAllUsers(userId,
categoryId, query)
        }

        _uiState.update {
            it.copy(
                searchResults = if (result.isSuccess) {
                    result.getOrNull() ?: emptyList()
                } else {
```

```
                    emptyList()
                },
                error = result.exceptionOrNull()?.message ?: "Unknown
error"
            )
        }
    }
}
```

Є.5 Код файлу TaskDisplay.kt

```
@Composable
fun TaskDisplay(task: TaskDto, viewModel: TasksViewModel) {
    var taskInfoVisible by remember { mutableStateOf(false)
    var isChecked by remember { mutableStateOf(false) }
    var isEnabled by remember { mutableStateOf(true) }
    var isEditing by remember { mutableStateOf(false) }

    val createdAtDate = ZonedDateTime.parse(task.createdAt).toLocalDate()
    val formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd")

    if(isEditing) {
        TaskModificationForm(
            task = task,
            onSave = { updatedTask ->
                viewModel.updateTask(updatedTask)
                isEditing = false
            },
            onCancel = {
                isEditing = false
            },
            viewModel
        )
    } else {
        val borderModifier = if (taskInfoVisible) {
            Modifier.border(1.dp, AppColors.Orange,
RoundedCornerShape(8.dp))
        } else {
            Modifier.border(0.dp, Color.Companion.Transparent,
RoundedCornerShape(8.dp))
        }
        Column(
            modifier = Modifier
                .fillMaxWidth()
                .padding(8.dp)
                .background(AppColors.White, RoundedCornerShape(8.dp))
                .then(borderModifier)
                .padding(8.dp)
        ) {
            Row(
                modifier = Modifier
                    .fillMaxWidth(),
                verticalAlignment = Alignment.CenterVertically
            ) {
                Checkbox(
                    checked = isChecked,
                    onCheckedChange = { checked ->
```

```
                        if (checked) {
                            isChecked = true
                            isEnabled = false
                        }
                        viewModel.updateTaskStatus(task)
                    },
                    enabled = isEnabled,
                    modifier = Modifier
                        .size(24.dp)
                        .align(Alignment.CenterVertically),
                    colors = CheckboxDefaults.colors(
                        checkedColor = AppColors.DarkBlue,
                        uncheckedColor = AppColors.Orange,
                        checkmarkColor = AppColors.Yellow
                    )
                )
                Text(
                    text = task.title,
                    style = MaterialTheme.typography.bodyLarge,
                    maxLines = 2,
                    overflow = TextOverflow.Ellipsis,
                    modifier = Modifier
                        .weight(1f)
                        .padding(start = 8.dp, end = 8.dp)
                )
                Row (
                    horizontalArrangement = Arrangement.spacedBy(0.dp),
                    verticalAlignment = Alignment.CenterVertically
                ){
                    if (taskInfoVisible) {
                        IconButton(onClick = { isEditing = true }) {
                            Icon(
                                painter = painterResource(id =
R.drawable.ic_pencil),
                                contentDescription = "Олівець для
модифікації"
                            )
                        }
                        IconButton(onClick = {
viewModel.deleteTask(task.id) }) {
                            Icon(
                                painter = painterResource(id =
R.drawable.ic_trash_can),
                                contentDescription = ""
                            )
                        }
                    }

                    IconButton(onClick = { taskInfoVisible =
!taskInfoVisible}) { subtasksVisible = !subtasksVisible }) {
                        Icon(
                            imageVector = if(!taskInfoVisible)
Icons.Default.ExpandLess else Icons.Default.ExpandMore,
                            contentDescription = if(!taskInfoVisible)
                        )
                    }
                }
            }
```

```
            if (task is TaskScaleDto && taskInfoVisible) {
                var valueToAdd by remember { mutableStateOf("") }
                Spacer(Modifier.height(8.dp))
                ValueInput(valueToAdd,
                    { valueToAdd = it },
                    {
                        val current = valueToAdd.toIntOrNull() ?: 0
                        valueToAdd = (current + 1).toString()
                    },
                    {
                        val current = valueToAdd.toIntOrNull() ?: 0
                        val newValue = if (current > 0) current - 1 else 0
                        valueToAdd = newValue.toString()
                    },
                    { viewModel.updateTaskCurrentValue(task,
valueToAdd.toInt())}
                )
            }
            if (taskInfoVisible) {
                Column(modifier = Modifier
                    .background(AppColors.Gray, RoundedCornerShape(8.dp))
                    .padding(start = 16.dp, end = 16.dp, bottom = 16.dp)) {
                    if (task is TaskScaleDto) {
                        Spacer(Modifier.height(16.dp))
                        Row (
                            verticalAlignment = Alignment.CenterVertically
                        ){

                            GradientProgressBar(
                                progress =
(task.currentValue/task.targetValue).toFloat(),
                                modifier = Modifier.weight(1f).padding(end
= 16.dp),
                                height = 12.dp,
                                gradientColors = listOf(Color(0xFF023047),
Color(0xFF219DBB))
                            )
                            Text(
                                "XP: ${task.currentValue.toInt()} /
${task.targetValue}", ${user.levelInfo.progress}",
                                modifier = Modifier
                                    .wrapContentWidth(Alignment.End)
                                    .padding(end = 16.dp),
                                style = additionalTypography.mediumText
                            )
                        }
                        Spacer(Modifier.height(16.dp))
                    }
                    if (task.description != null && task.description != "")
{
                        Spacer(Modifier.height(8.dp))
                        Text(
                            task.description!!,
                            style = additionalTypography.regularText,
                            textAlign = TextAlign.Justify
                        )
                        Spacer(Modifier.height(8.dp))
                    } else {
```

```
                        Spacer(Modifier.height(8.dp))
                    }
                    if (task is TaskRepeatableDto && taskInfoVisible) {
                        Spacer(Modifier.height(8.dp))
                        Text("Дні для занять", style =
additionalTypography.semiboldText, modifier =
Modifier.align(Alignment.Start))
                        Spacer(Modifier.height(8.dp))
                        val context = LocalContext.current
                        WeekdayCheckInView(
                            markedDays = task.repeatDays,
                            checkedInDates = task.checkedInDays,
                            onCheckIn = { message, date ->
                                if (message == "Відмітка успішна!") {
                                    viewModel.checkInTask(task.id, date)
                                }
                                Toast.makeText(context, message,
Toast.LENGTH_SHORT).show()
                            }
                        )

Spacer(Modifier.height(8.dp).align(Alignment.CenterHorizontally))
                        Column(
                            modifier = Modifier.fillMaxWidth(),
                            horizontalAlignment =
Alignment.CenterHorizontally
                        ) {
                            Text(
                                "Відмічайте дні, натиснувши на них",
                                style = additionalTypography.regularText
                            )
                        }
                    }
                    Row (
                        modifier = Modifier.fillMaxWidth(),
                        horizontalArrangement = Arrangement.SpaceBetween
                    ){
                        Text("Дата старту", style =
additionalTypography.mediumText)
                        Text(createdAtDate.format(formatter), style =
additionalTypography.mediumText)
                    }
                    if (task.deadline != null) {
                        val deadlineDate =
ZonedDateTime.parse(task.deadline).toLocalDate()
                        Spacer(Modifier.height(8.dp))
                        Row (
                            modifier = Modifier.fillMaxWidth(),
                            horizontalArrangement =
Arrangement.SpaceBetween
                        ) {
                            Text("Дедлайн", style =
additionalTypography.mediumText)
                            Text(
                                deadlineDate.format(formatter),
                                style = additionalTypography.mediumText
                            )
                        }
```

```
                    }
                }
            } else {
                HorizontalDivider()
            }
            if (task is TaskWithSubtasksDto
&&!task.subtasks.isNullOrEmpty()) {
                Column(modifier = Modifier.padding(start = 16.dp)) {
                    task.subtasks.forEach { subtask ->
                        Spacer(Modifier.height(8.dp))
                        SubtaskDisplay(task.id, subtask, viewModel,
task.createdAt, task.deadline)
                    }
                }
            }
        }
    }
}
```

Є.6 Код файлу PieChart.kt

```
package com.example.dyplomproject.ui.features.statistics


fun categoryToPieEntries(data: List<CategoryPieCharSliceDto>):
List<PieEntry> {
    return data.map { PieEntry(it.value.toFloat(), it.categoryName) }
}

@Composable
fun PieChartView(data: List<PieEntry>) {
    AndroidView(
        factory = { context ->
            PieChart(context).apply {
                layoutParams = ViewGroup.LayoutParams(
                    ViewGroup.LayoutParams.MATCH_PARENT,
                    600
                )
                setPieChartProperties(this, data)
            }
        },
        update = { chart ->
            setPieChartProperties(chart, data)
        }
    )
}

private fun setPieChartProperties(chart: PieChart, data: List<PieEntry>) {
    val customColors = listOf(
        0xFFFF0000.toInt(),
        0xFF00FF00.toInt(),
        0xFF0000FF.toInt(),
        0xFFFFFF00.toInt(),
        0xFF00FFFF.toInt(),
        0xFFFF00FF.toInt(),
        0xFFD3D3D3.toInt(),
        0xFF444444.toInt(),
```

```
            0xFFFF5722.toInt(),
            0xFF00BCD4.toInt(),
            0xFF8BC34A.toInt(),
            0xFFE91E63.toInt()
        )

        val dataSet = PieDataSet(data, "").apply {
            colors = customColors
            valueTextSize = 16f
            sliceSpace = 2f
            setDrawValues(true)
        }

        chart.data = PieData(dataSet)
        chart.setUsePercentValues(true)
        chart.setDrawEntryLabels(false)
        chart.description.isEnabled = false
        chart.legend.apply {
            isEnabled = true
            textSize = 16f
            form = Legend.LegendForm.CIRCLE
            verticalAlignment = Legend.LegendVerticalAlignment.CENTER
            horizontalAlignment = Legend.LegendHorizontalAlignment.RIGHT
            orientation = Legend.LegendOrientation.VERTICAL
            setDrawInside(false)
            xEntrySpace = 10f
            yEntrySpace = 10f
        }
        chart.invalidate()
    }
```

Є.7 Код файлу LineChart.kt

```
package com.example.dyplomproject.ui.features.statistics

val ukrMonthNames = listOf("", "Січень", "Лютий", "Березень", "Квітень",
"Травень", "Червень", "Липень", "Серпень", "Вересень", "Жовтень",
"Листопад", "Грудень")

fun completedTasksToLineEntries(data:
List<CompletedTasksLineChartElementDto>): Pair<List<Entry>, List<String>> {
    val sorted = data.sortedWith(compareBy({ it.year }, { it.month }))
    val entries = sorted.mapIndexed { index, item -> Entry(index.toFloat(),
item.completedCount.toFloat()) }
    val labels = sorted.map { ukrMonthNames[it.month] }
    return entries to labels
}
@Composable
fun LineChartView(data: List<Entry>, xLabels: List<String>) {
    AndroidView(
        factory = { context ->
            LineChart(context).apply {
                layoutParams = ViewGroup.LayoutParams(
                    ViewGroup.LayoutParams.MATCH_PARENT,
                    600
                )
```

```
                setChartProperties(this, data, xLabels)
            }
        },
        update = { chart ->
            setChartProperties(chart, data, xLabels)
        }
    )
}
private fun setChartProperties(chart: LineChart, data: List<Entry>,
xLabels: List<String>) {
    val dataSet = LineDataSet(data, "Продуктивність").apply {
        color = ColorTemplate.rgb("#5E35B1")
        valueTextSize = 16f
        setDrawFilled(true)
        fillColor = ColorTemplate.rgb("#D1C4E9")
        setDrawCircles(true)
        circleRadius = 8f
        lineWidth = 2f
    }
    chart.data = LineData(dataSet)

    chart.xAxis.apply {
        valueFormatter = IndexAxisValueFormatter(xLabels)
        granularity = 1f
        position = XAxis.XAxisPosition.BOTTOM
        setDrawGridLines(false)
    }
    chart.axisLeft.apply {
        axisMinimum = 0f
        setDrawGridLines(true)
    }
    chart.axisRight.isEnabled = false

    chart.description.isEnabled = false
    chart.legend.isEnabled = false
    chart.notifyDataSetChanged()
    chart.invalidate()
}
```