Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мережних технологій факультету інформатики

Розробка мобільного застосунку для платформи IOS для організації централізованої волонтерської допомоги літнім людям

Текстова частина до курсової роботи
за спеціальністю „Інженерія програмного забезпечення ” 121

Керівник курсової роботи
кандидат фізико-математичних наук, старший викладач
Гречко А. В.

_____

(Підпис)

“___” _____ 2021 року

Виконала студентка ІПЗ-БПЗ
Ксенофонтова С.В.

“___” _____ 2021 року

Київ 2021

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мережних технологій факультету інформатики

ЗАТВЕРДЖУЮ
Зав. Кафедри мережних технологій,
професор, д.ф-м.н.
_____ Г.І. Малашонок
(підпис)
“____” _____ 2021 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

Студентці Ксенофонтовій Софії Вячеславівні факультету інформатики 3 курсу

ТЕМА: Розробка мобільного застосунку для платформи IOS для організації централізованої волонтерської допомоги
літнім людям

Зміст ТЧ до курсової роботи:

Календарний план

Вступ

Аналіз предметної області та постановка завдання курсової роботи

Теоретичні відомості (про задачу, методи і підходи до її розв’язку тощо)

Опис реалізації програмного продукту

Висновки

Список використаної літератури

Додатки

Дата видачі “____” _____ 2021 р.

Керівник _____

(підпис)

Завдання отримав _____

(підпис)

Календарний план виконання роботи

Тема: Розробка мобільного застосунку для платформи IOS для організації централізованої волонтерської допомоги літнім людям

№	Назва етапу	Термін виконання	Примітка
1.	Отримання теми курсової	29.09.2020	
2.	Ознайомлення з предметною областю	1.10.2020-14.10.2020	
3.	Пошук та ознайомлення з корисною літературою	15.10.2020-1.11.2020	
5.	Проведення дослідження та опитувань серед студентів	2.11.2020-7.11.2020	
6.	Аналіз отриманих результатів	8.11.2020-10.11.2020	
7.	Планування структури та архітектури практичної частини роботи	10.11.2020-15.11.2020	
8.	Розробка макету в Figma	16.11.2020-16.12.2020	

9.	Ознайомлення з мовою програмування Swift	Січень 2021	
10.	Перенесення макету	Лютий-березень 2021	
11.	Обрання бази даних та інших інструментів розробки	Березень 2021	
12.	Програмування застосунку	Березень-квітень 2021	
13.	Написання текстової частини роботи	10.05.2021	
14.	Створення презентації та доповіді до неї	15.05.2021	
15.	Здача курсової роботи на перевірку	17.05.2021	

Студентка Ксенофонтова С.В.

Керівник Гречко А. В. “____” _____

ЗМІСТ

Вступ

Завдяки поліпшенню санітарних та житлових умов, розвитку сфер освіти і охорони здоров'я тривалість життя сучасної людини помітно збільшилася протягом XXI століття. За прогнозами вчених вже за 15 років кожна сьома людина на Землі буде старшою за 70 років [1,2].

Однією з проблем людей літнього віку є повальна діджиталізація усіх сфер сьогоденного життя, оскільки їм достатньо важко опановувати кардинально нові способи взаємодії та складні у розумінні технології. З масовим розповсюдженням інформаційних технологій, а тепер ще й з приходом світової пандемії, все більше звичних для людей сервісів та бізнесів переходять виключно або частково у онлайн режим. Тому так важливо саме зараз допомогти людям похилого віку навчитися взаємодіяти з ґаджетами і без перешкод виконувати свої звичні докарантинні дії так, щоб отримання пенсії, замовлення ліків, запис до лікаря, оплата комунальних послуг, поповнення мобільного тощо не здавалися надскладними завданнями.

У часи пандемії питання опанування літніми людьми цифрового середовища постало ще більш гостро, адже кількість контактів з літніми родичами різко зменшилася, спілкування в основному перейшло у телефонний режим. Якщо раніше молоді нащадки, котрі проживають в одному місті зі своїми родичами пенсійного віку, могли спокійно та без перешкод допомогти їм у будь-якому питанні щодо використання ґаджетів, то тепер на заваді став карантин, мінімізувавши усі соціально-близькі стосунки. Літня людина, яка погано орієнтується в технологіях та всесвітній павутині, дуже скоро може стати викинутою за межі соціального суспільства та бути безпорадною у сфері отримання будь-яких послуг. Саме тому дослідження підходів до створення зручного для пенсіонерів користувацького інтерфейсу, що враховує фізіологічні особливості літніх людей, та розробка системи що допомогла б у вирішенні повсякденних питань є таким актуальним завданням.

Метою та завданням даної курсової роботи є розробка застосування, що дозволить спростити життя літнім людям, котрі користуються планшетом або смартфоном, а також опосередковано допомогти волонтерським організаціям, що

займаються проблемами пенсіонерів, налагодити та централізувати свою взаємодію з підопічними літного віку. Будь-який користувач розроблюваної системи матиме можливість викликати на дім вільного волонтера, попередньо обравши певне завдання для виконання з категорії, наприклад, здоров'я, доставка, хатня робота, рахунки тощо. Волонтер обирається зі списку волонтерських організацій-партнерів, актуальних для міста України: в якому проживає пенсіонер. Волонтер у свою чергу матиме змогу чітко зрозуміти потреби пенсіонера та завчасно отримати інформацію про те, що саме йому необхідно зробити.

Об'єктом дослідження в даній курсовій роботі є розробка інтерфейсу: орієнтованого на літніх людей.

Предметом даного дослідження є розробка користувацького інтерфейсу (UI) під платформу iOS.

Основним завданням даної роботи є створення застосування, однаково зручного як для волонтерів, так і для пенсіонерів. Врахувавши всі особливості та побажання обох сторін, було вирішено розробити два сценарії поведінки застосування: сценарій Волонтер та сценарій Пенсіонер, які включатимуть функціональні можливості приблизного однакового рівня складності, але матимуть певні відмінності у інтерфейсі.

При створенні системи було використано:

- Мову програмування Swift (фреймворки UIKit, Alamofire);
- NoSQL База даних Mongo DB;
- Середовище Node js з використанням Express фреймворку для реалізації серверної бекенд частини
- Хмарна PaaS-платформа Heroku для хостингу серверу бекенд частини застосунку
- Редактор Figma для побудови дизайн макету застосунку;
- Adobe Photoshop для для промальовки логотипу та відображення іконки застосунку на робочому столі.

РОЗДІЛ 1 Аналіз предметної області. Постановка завдання курсової роботи

1.1 Аналіз сучасного стану питання та обґрунтування теми

З настанням певного віку патерни взаємодії з програмним продуктом у кожної людини змінюються. Наприклад, згідно з результатами світових досліджень [2], більш літні користувачі:

- Мають тенденцію довше та ретельніше вивчати нові програми та пристрої.
- Довше опановують функціонал із завершенням будь-якого завдання.
- Використовують різні стратегії пошуку.
- Гірше справляються з завданнями, які вимагають запам'ятовування.
- Найчастіше відволікаються.
- З великими труднощами виправляють помилки.
- Роблять більше хаотичних і випадкових рухів мишею та пальцями.
- Найчастіше припускаються помилок при введенні тексту.
- Насилу потрапляють в цілі на екрані.

Окрім, описаних вище проблем, в Україні основними також залишаються проблеми з зором у літніх людей, тож розробка користувацького інтерфейсу має містити більш контрастні та інтуїтивно зрозумілі кольори, як от зелений для відповіді – "так", червоний для "ні" відповідно. Також будь які елементи інтерфейсу та шрифт по можливості мають бути збільшені, та містити максимально прості анімації. Також повальною проблемою літніх українців є деменція (тобто людині все важче і важче взаємодіяти з девайсами, адже навіть для навчання літньої людини взаємодіяти зі своїм смартфоном або планшетом, необхідні мінімальні навички запам’ятовування інформації, що вдається далеко не кожному пенсіонеру), через це повсюди у вашому застосунку мають бути підказки. Тому так важливо було розробити застосунок який би враховував всі ці особливості.

1.2 Огляд існуючих аналогів розробки

В результаті аналізу поточного стану ринку систем-аналогів було виявлено, що наразі не існує подібних застосувань, котрі покривають та вичерпно вирішують описані у попередньому розділі роботи проблеми пенсіонерів. Основними конкурентами реалізованому у даній роботі застосунку безперечно могли б стати будь які сайти або ж застосунки, котрі надають різного роду послуги, такі як: доставка товарів та їжі, виконання хатніх обов'язків та інше. Проте значними недоліками є те, що ці платформи не є націленими та адаптованими під фізіологічні особливості пенсіонерів та зазвичай досить не однозадачні. Проте, досліди показують, що для пенсіонерів дуже важливим є те щоб один застосунок міг вирішувати якомога більше їх життєвих проблем та труднощів, адже багатьом стареньким з віком досить важко стає тримати щось у пам'яті, а тим паче щоразу відкривши свій смартфон чи планшет згадувати та з'ясовувати який застосунок за що відповідає.

На ринку з 2012 року є платформа "Кабанчик" [3] у вигляді сайту та однойменного застосунку, яка допомагає людям з України вирішувати свої проблеми. Вигуляти собаку, допомогти з прибиранням, полагодити комп'ютер — усе це стає доступним для користувачів цього ресурсу. Ресурс має два види функціоналу для користувача з роллю "Замовник" та для користувача з роллю "Виконавець", що є гарною аналогією з представленим у даній роботі застосунком. Проте на відміну від розробленого застосування, де все виконується виключно на волонтерських засадах і є безкоштовним для пенсіонера, в "Кабанчик"`у Замовники платять за роботу Виконавців. Також істотним мінусом є те що сайт та застосунок "Кабанчик"`а містить досить громіздкі функціонал та алгоритми роботи, що насамперед робить його досить складним в опануванні, адже містить досить багато нюансів роботи, котрі не видно при першому ж відвідуванні застосунку (див. додаток А).

На противагу йому, українські сервіси доставки [4] “Glovo”, “Raketa”, мають порівняно досить простий інтерфейс (див. додаток Б) проте, нажаль, виконують лише одну задачу доставки, та не являються мультифункціональними.

Тож, проаналізувавши це все, робимо висновок, що наразі не існує простих в опануванні для пенсіонера застосунків, котрі мали б зручний та зрозумілий інтерфейс та вирішували б усі нагальні проблеми пенсіонерів просто в одному застосуванні. Також, досліджуючи український ринок на наявність застосунків, котрі полегшували б роботу волонтерам, довелося стикнутися з їх відсутністю, як на рівні волонтерських монополістів так і на рівні держави. Ні перші, ні другі чомусь зовсім не зацікавлені в розробці застосувань, які містили б вичерпну інформацію про послуги тієї чи іншої волонтерської організації, що опікується питаннями пенсіонерів.

1.3 Результати проведеного соціологічного опитування

В рамках даної роботи було проведене соціологічне опитування, яке охопило 128 членів могилянської спільноти. В результаті вдалося зробити такі висновки (основна частина статистичних даних наведена у додатку В):

- Левову частку (89%) тих, хто прийняв участь у даному опитуванні, склали молоді люди віком від 18-21 року, тож орієнтуватися слід саме на них.
- 58,6% опитуваних, тобто 75 чоловік, мають літніх родичів віком від 70 до 79 років, наступною найрозповсюдженішою групою літніх родичів виявилися родичі віком від 60 до 69 років, котрих мають 56,3% опитуваних.
- Більше половини респондентів (68%) проживають зі своїми літніми родичами в одному місті.
- 75% літніх родичів, постійно мають проблеми з розумінням та опануванням власних мобільних пристроїв.
- Близько половини молодих опитуваних (47,9%) постійно вимушені допомагати своїм родичам та консультувати їх з приводу користування пристроями та мережею, проте з радістю доручили це комусь іншому.

- Більше половини людей пенсійного віку використовують всесвітню павутину для перегляду фільмів та серіалів, другими рівноправними по витрачанню часу є оплата комунальних послуг та рахунків, запис до лікаря та онлайн шоппінг. Деякі літні родичі молодих респондентів досить часто використовують послуги доставки, замовляють різні послуги та спілкуються в популярних на сьогодні месенджерах.
- Нажаль, лише мала частина(31,6%) молодих респондентів довірила б догляд та опіку за своїми літніми родичами волонтерам з перевірених організацій.
- Так само невтішні для мого застосунку новини з приводу найзручнішої ОС для літніх людей, адже лише 35,6 % опитуваних вважають ОС iOS більш зручнішою для літніх користувачів ніж Android.
- Серед запропонованих в опитуванні волонтерських організацій [5], найбільш відомими серед респондентів виявилися : “Життєлюб”(33,3%), “Дім милосердя ”(18,8 %), “Молодь за мир” (14,6 %), тож було прийнято рішення включити в застосунок для демонстрації роботи саме їх.

Ці дані наочно демонструють, що проблема складності опанування літніми людьми своїх мобільних пристроїв існує, і що достатня кількість молодих людей охоче делегували б допомогу котра необхідна їх літнім родичам волонтерським організаціям.

1.4 Постановка задачі

Основною задачею було створити однаково зручний клієнт-серверний iOS застосунок як для волонтерів так і для пенсіонерів. Врахувавши всі особливості та бажання тих та інших, було вирішено розробити два сценарії поведінки застосунку сценарій Волонтер та сценарій Пенсіонер. Які матимуть схожу складність функціоналу але різнитимуться інтерфейсом. Пенсіонер матиме змогу:

- Обирати завдання з однієї з п’яти зарезервованих категорій, таких як: “Здоров’я”, “Рахунки”, “Доставка”, “Хатня-робота”, “Ремонт”, кожна з яких міститиме до десяти найпопулярніших серед пенсіонерів завдань для виконання волонтером;

- Отримати консультацію по телефону, просто набравши вказаний номер консультанта, з будь-яких питань у будь-який зручний для пенсіонера час;
- Шукати волонтера, готового до виконання замовленого завдання в реальному часі;
- Позначити завдання як виконане, коли волонтер на думку пенсіонера закінчив свою роботу;
- Переглядати список виконаних завдань;
- Редагувати інформацію про себе у профілі.

Волонтер у свою чергу матиме можливість:

- Приймати/відхиляти завдання пенсіонера в реальному часі;
- Переглядати вхідні завдання в реальному часі;
- Переглядати список виконаних завдань;
- Редагувати інформацію про себе у профілі;
- Виходити в режим готовності отримання завдань від пенсіонера натисканням на кнопку.

РОЗДІЛ 2 Теоретичні відомості (про задачу, методи і підходи до її розв'язку тощо)

2.1 Основні поняття про розробку клієнт-серверних застосунків

Наразі переважна більшість веб сайтів, мобільних застосунків та систем, з котрими доводиться працювати щодня, побудовані на основі клієнт-серверної архітектури. Адже, будь який застосунок в основу котрого входить взаємодія з багатьма клієнтами скоріш за все буде побудований на основі цього підходу. Основним принципом такого архітектурного рішення являється те, що користувач маючи при собі будь яке апаратне забезпечення з доступом до мережі надсилає запит на певну інформацію на віддалений сервер, а той у свою чергу, обробивши вхідні дані від клієнта певним чином, надсилає йому назад відповідь, яка відображається на екрані комп'ютера або смартфона клієнта. Клієнт серверна архітектурна модель в першу чергу характеризується розподілом обов'язків між клієнтом та сервером. За цією характеристикою виділяють дворівневі та трирівневі клієнт-серверні архітектури.

Дворівнева клієнт-серверна архітектура передбачає взаємодію виключно двох рівнів: рівня клієнтського представлення та рівня серверу. Така модель може бути поділена на дві категорії, в залежності від того який функціонал бере на себе рівень представлення клієнту [6]:

- Модель *тонкого клієнта(thin client)*, де основна бізнес логіка взаємодії з даними відбувається на сервері. А клієнтська сторона забезпечує виключно представлення даних в програмному інтерфейсі, що робить його функціонал меншим, тонким;
- Модель *товстого клієнта(thick client)*, передбачає що сервер відповідальний лише за керування даними. А вся обробка інформації та представлення даних відбувається на стороні клієнта, через що його функціонал перевантажується, стає ширшою, товстішою.

Трирівнева клієнт серверна архітектура складається з [7] (див. рис. 2.1.1):

- Представлення клієнту: це front-end частина програмного забезпечення, яка напряму взаємодіє з користувачами, навіть якщо вони знаходяться на різних платформах використовуючи різні технології. Цей рівень містить екрани входу, реєстрації, меню, екрани представлення даних, спливаючі вікна які надають та приймають інформацію клієнтів. Наприклад, більшість засобів розробки дозволяють створити одну адаптивну версію програми, яка однаково добре масштабуватиметься і працюватиме, як для настільних ПК, так і для менших пристроїв таких як планшети та смартфони;
- Основний сервер: Це віддалений або ж локально розташований сервер, на якому встановлені програмні модулі необхідні для існуючого застосування і виконується основна бізнес логіка взаємодії з даними що надходять від користувача. Він напряму підключається до серверу бази даних та надсилає їй запити в зрозумілому для неї форматі (JSON, BSON, SQL запит). Також основний сервер взаємодіє з користувачами, виконуючи їх запити та надсилаючи відповіді на рівень представлення (front-end). Таким чином основний сервер виконує роль посередника між рівнями представлення клієнту та сервером бази даних;
- Сервер бази даних: Цей сервер містить таблиці, індекси та дані, керовані застосунком. Тут виконуються операції пошуку та вставки/видалення/оновлення об’єктів та атрибутів за запитом що надходить від основного серверу. Сервер баз даних взаємодіє лише з основним сервером.

В реалізованому в даній роботі застосуванні було використано трирівневу модель в якій, front-end клієнтське представлення написане на мові Swift, віддалений сервер хоститься на Heroku та написаний на Node.js Express, а робота з базою даних Mongo DB відбувається через cloud сервіс Mongo DB Atlas. Оскільки вся логіка взаємодії з даними відбувається на серверній та БД частинах, можна також сказати що в розробленій архітектурі застосунку клієнт є тонким.

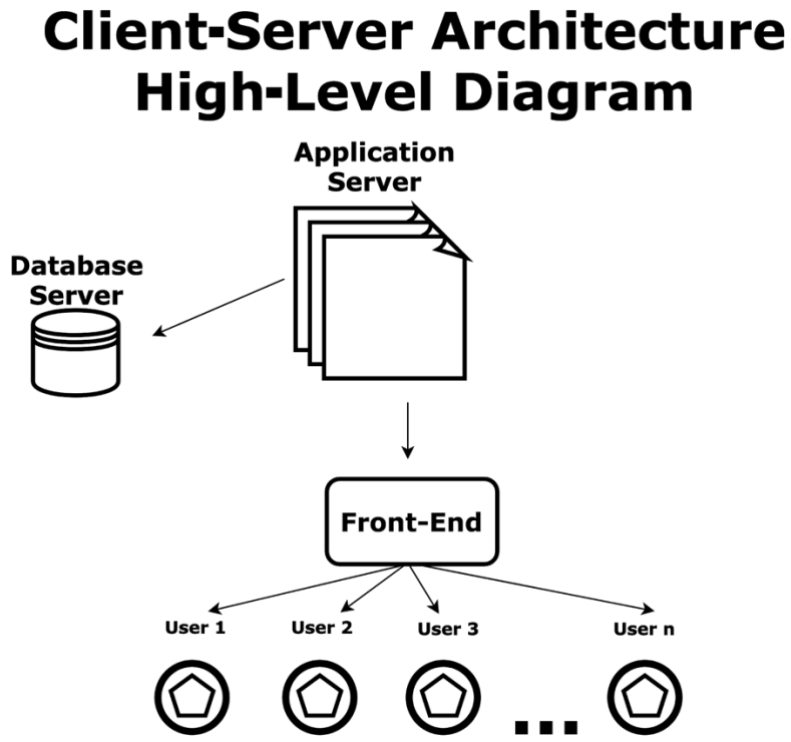


Рисунок 2.1.1 Трирівнева клієнт серверна архітектура [7]

2.2 Відомі архітектурні патерни для мобільної розробки під iOS

Оскільки представлений застосунок побудований за концепцією клієнт серверної архітектури він містить як front-end (рівень представлення клієнту) так і back-end (рівні основного та БД серверів) частини, кожна з яких вимагає структуризації внутрішньої побудови задля більшої ефективності виконання відведеної ролі в мобільному застосунку. Також розглянемо найвідоміші архітектурні патерни, які можна застосовувати як для front-end, так і для back-end частин розробки мобільного застосування.

Серед найактуальніших архітектурних патернів, не тільки для iOS застосувань, що наразі використовуються в мобільній розробці безумовними лідерами є класичні архітектурні моделі MVC, MVP, MVVM [8,9,10] а також VIPER, котрий є достатньо новим на ринку патернів. Варто більш детальніше розповісти про кожен з них.

Патерн MVC найвідоміший з трьох перелічених вище патернів, та один з найбільш рекомендованих та використовуваних серед iOS розробників. Він уособлює собою розділення архітектури на 3 рівні:

- Model – відповідає за дані та рівень доступу до них
- View – відповідає за графічне представлення даних у вигляді користувацького інтерфейсу
- Controller – відповідає за взаємозв'язок між Model та View

Основними недоліками даного архітектурного рішення є те, що дане рішення не підходить для великомасштабних застосувань та те, що часті зміни будь якого з компонентів архітектури призводять до постійних запитів на оновлення представлення цих компонентів, що досить уповільнює процес розробки. Також особливо в iOS розробці делегування всієї логіки взаємодії між Model та View Controller'у може призвести до суттєвого перевантаження останнього, що призведе до відомої серед розробників проблеми Massive View Controller'а [10].

Патерн MVP, що являє собою покращену реалізацію MVC патерну, у свою чергу складається з

- Model – керує всією бізнес логікою застосунку;
- View (Passive) – котрий складається з View та View Interface. Перший відповідає за представлення даних користувачу. Другий у свою чергу керує зв'язком між Model та View лише на рівні інтерфейсу;
- Presenter – є зв'язуючим компонентом між View та Model, що відповідає саме за дані;

Основними відмінностями між MVC та MVP є те, що перший делегує всю логіку частин View в Controller (відображення в тому числі) і відповідно виходить, що частина View відповідає лише за інтерфейсне представлення. MVP у свою чергу розділяє чисто логіку відображення та логіку взаємодії даних з відображенням. Помістивши першу частину в View Interface компоненти Passive View , а другу делегувавши компоненту Presenter.

Патерн MVVM наразі являється найуживанішим серед iOS розробників, з переліку класичних патернів, адже в основному, коли модель вимагає відокремлення графічного інтерфейсу від логіки розвитку бізнесу, розробники

віддають перевагу використанню саме його. Цей архітектурний шаблон як і два його попередники послуговується розділенням застосування на 3 важливі частини:

- Model – структура котра відповідає за управління даними;
- View – відповідає за рівень представлення даних;
- View-Model – працює зв'язуючим компонентом між частинами Model та View;

Така популярність MVVM зумовлена тим що, цей архітектурний патерн не викликає перевантаження таких посередників взаємодії між частинами застосунку як Controller та View.

Всі ці 3 патерни по своїй суті є досить схожими адже, мають такі однакові компоненти як Model та View а також Controller/Presenter/ ViewModel котрі являються посередниками між попередніми двома складовими та відповідають за їхню взаємодію без прямого звернення (див. рис. 2.1.1).

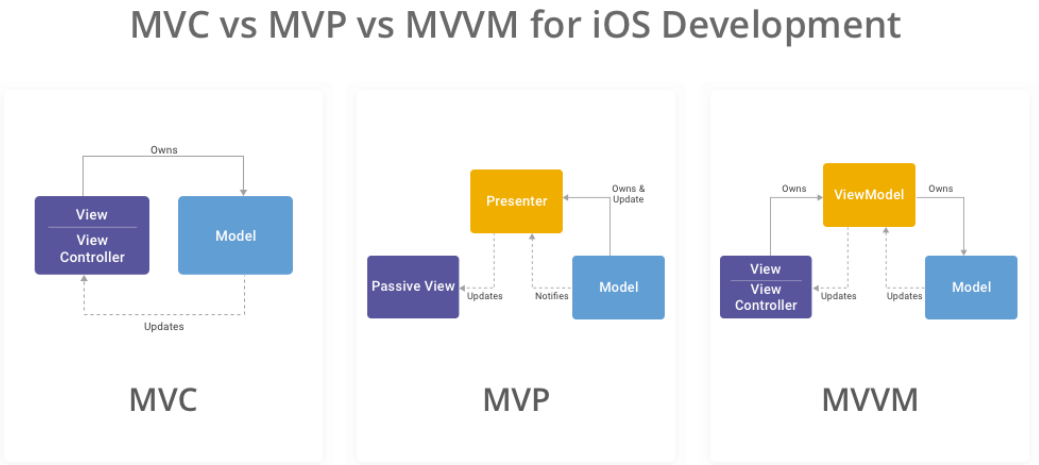


Рисунок 2.2.1 Візуальне представлення патернів MVC, MVP, MVVM [11]

Також таке розділення на патерни значно спрощує та прискорює тестування, адже дані моделі і дані представлення інтерфейсу можуть бути протестовані ізольовано

Останнім наразі часто вживаним патерном саме у сфері iOS розробки являється VIPER [12], котрий з'явився на ринку досить недавно і одразу завоював прихильність серед розробників. Абревіатура VIPER розшифровується як View, Interactor, Presenter, Entity та Router, це модель, яка містить п'ять елементів (див. рис. 2.2.2), кожний з яких виконує свою функцію.

- View – відповідає за обмін подіями(actions) між користувачами і Presenter'ом;
- Interactor – містить у собі всю бізнес логіку;
- Presenter – отримує доступ до даних Interactor'а і надсилає їх на View;
- Entity –містить у собі базове представлення об'єктної моделі, що використовується Interactor'ом;
- Router – відповідає за логіку пересилання (навігації);

Основними перевагами Viper які виділяють розробники є :

- Спрощення складних проєктів, за рахунок деталізованої оперативної архітектури, що підходить для одночасної співпраці великих команд;
- Полегшена процедура тестування та перевикористання та злиття коду, за рахунок повної декомпозиції на окремі незалежні частини;
- Прозорість розуміння за рахунок створення інтерфейсів з чітко визначеним окремим функціоналом;

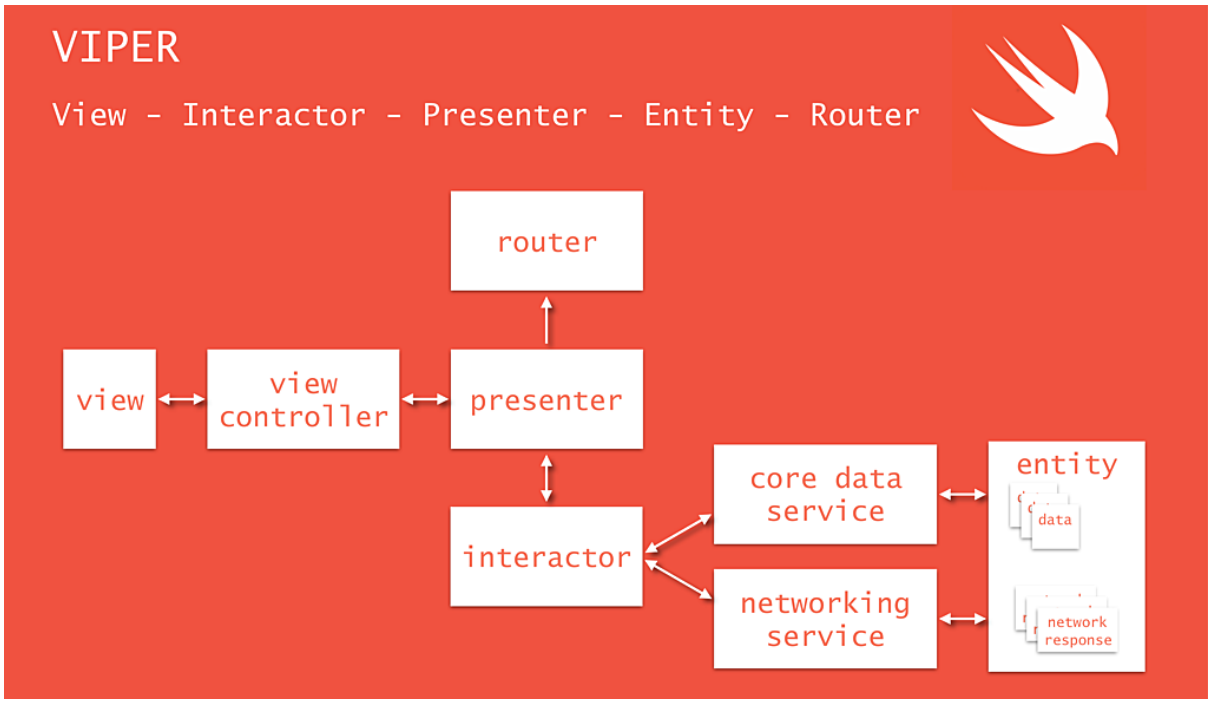


Рисунок 2.2.2 Візуальне представлення патерну VIPER з уточненнями для мови Swift [12]

Оскільки розроблений застосунок є трирівневим клієнт-серверним застосунком, що фактично містить тонкого клієнта, адже вся логіка взаємодії з даними відбувається на back-end частині, гарним рішенням по упорядкуванню частин застосунку на рівні відображення клієнту буде MVC патерн. Оскільки

клієнт тонкий та реалізований застосунок є дрібномасштабним проєктом, то задіюваність View-Controller’а для обробки даних користувача та відправку на серверну частину буде незначною, що не призведе до його перевантаження, та вищезгаданої проблеми “Massive View Controller’а”.

2.3 Особливості розробки користувацького інтерфейсу для літніх людей під платформу iOS

Основна задача будь-якої мобільної програми чи застосування полягає в тому, щоб допомогти вирішити проблему користувача якомога швидше. Згідно з Human Interface Guidelines [13], єдиного зводу правил написаних компанією Apple, котрих рекомендовано дотримуватись щоб розробляти софт під їх ОС, треба дотримуватись певних принципів та інструкцій щоб зробити інтерфейс зручним для всіх користувачів. Деякі з принципів звучать так:

- Естетична цілісність — відображає, наскільки зовнішній вигляд та поведінка програми інтегруються з її функцією;
- Послідовність — застосування реалізує звичні стандарти та парадигми, використовуючи надані системою елементи інтерфейсу, стандартні стилі тексту та єдину термінологію. Застосування функціонує і поводиться так, як очікують люди;
- Безпосереднє маніпулювання — вмістом на екрані залучає людей та полегшує розуміння;
- Зворотній зв'язок — підтверджує дії користувача та показує результати для інформування людей. Інтерактивні елементи коротко виділяються при натисканні, індикатори прогресу повідомляють про стан тривалих операцій, а анімація та звук допомагають уточнити результати дій;

Оскільки даний застосунок розрахований на дві такі різні вікові категорії як, молодь та літні люди, було важливо було пов’язати ці принципи з особливостями розробки ПЗ для літніх людей

Конкуренція на ринку програмного забезпечення змушує розробників щоразу представляти на ринку все новіші й новіші версії своїх програмних продуктів. Реліз нової версії застосунку, як правило, пропонує нові функції або зручність використання, але збільшення кількості функцій може також вплинути на складність програмного забезпечення, що може відштовхнути лояльних літніх користувачів. Дотримуючись наведених нижче рекомендацій [13] ваш застосунок зможе залучити та утримати велику частку людей похилого віку:

- Будь які інформаційні повідомлення мають бути якомога коротшими та лаконічнішими;
- Зменште вибір, де це можливо, пропонуйте користувачам від двох до трьох варіантів;
- Вставляйте підтверджуючі запитання, де це можливо;
- Функціонал застосунку що зовсім не використовується його користувачами підлягає видаленню;
- Уникайте використання складних взаємодій. Дотримуйтесь принципу єдність функції сторінки (одна сторінка - одне завдання) та зменште кількість елементів на робочій сторінці до 3-4 елементів. Тримайте основну робочу зону в центрі робочої сторінки. Дизайнери повинні бути переконані, що увага літніх людей не розділяється кількома завданнями чи частинами екрану [14], коли випускають у світ програмний продукт;
- Надавайте візуальні інструкції якомога частіше. Навіть знайомі дії можна зробити більш зручними, включивши нагадування та підказки;
- Проблеми з пам’яттю можна скоригувати за допомогою таких речей, як надання чітких відгуків про прогрес та нагадування користувачам про кінцеву мету [14]. Забезпечуйте швидкий та чіткий відгук на запит чи дотик користувача;
- Стежте за високою відновлюваністю даних та прагніть мінімізації помилок, (велика кількість літніх користувачів можуть з необережності стерти всі важливі дані про себе;

У міру того, як люди старіють, їхній зір зазнає ряд негативних змін. Багато людей похилого віку використовують окуляри для читання або обирають набагато більші розміри шрифту, коли їм надається така можливість. За рекомендаціями компетентних дизайнерів вважається [15], що кольорову контрастність слід збільшити на веб-сайтах та мобільних застосуваннях, якими користуються люди віком за 60. Проте, велика контрастність застосунку в першу чергу вплине на втомлюваність очей літніх користувачів, адже з віком концентрація та зосередження на елементах інтерфейсу потребують ще більшої роботи зорового апарату, тож очі швидко втомлюватимуться. Тож, на мою думку правильне рішення лежить десь посередині: основні на важливі елементи інтерфейсу треба робити акцентними (збільшуючи їх контрастність), а ті елементи що ніякого функціоналу не несуть, використовуються лише задля покращення візуального сприйняття дизайну, робити в більш спокійних відтінках. Нижче я узагальнюю та наводжу ще декілька корисних порад [15]:

- Використовуйте компоненти інтерфейсу належного розміру(сенсорна зона / Розмір кнопки повинен бути від 16,5 мм до 19,05 мм; Розмір інтервалу між кнопкою / чутливою до дотику 3,15 мм до 12,7 мм);
- Розмір тексту та кнопок обов’язково повинен бути збільшеним в порівнянні з іншими елементами. По суті, все, що призначено для читання або натискання, має бути масштабовано. Шрифт повинен бути не менше 16 пікселів (чим більше тим краще). Загалом, де б не було вказано “рекомендований” розмір або відстань, дизайнери повинні розглядати це як абсолютний мінімум для будь-якого інтерфейсу, орієнтованого на людей похилого віку;
- Використовуйте чорні шрифти без засічок на білому тлі та уникайте використання вигадливого тексту (рух, не горизонтальна орієнтація, сплеск тощо);
- Якщо в вашому застосунку присутній відео чи аудіо вміст котрий необхідний для подальшої взаємодії з застосунком, важливо включити субтитри;

- Іконки , як елементи мобільного інтерфейсу, ще одна важлива сфера. Кожна з них має бути підписана або позначена текстом для кращого тлумачення їх функціоналу літніми людьми. Тих, кому корисний текст не потрібен, швидше за все, він не образить, але ті, кому він потрібен, можуть загубитися, якщо його не відображати;
- Жести - це ще одна сфера, де люди похилого віку іноді можуть натрапити, особливо коли вони новачки в технології сенсорного екрану. Існує ряд моделей взаємодії старших людей, які не є поширеними у молодих поколінь. Сюди входять такі речі, як друк однією рукою, особливо на мобільному пристрої (не слід вважати, що старші покоління не знають, як друкувати на звичайній клавіатурі; багато хто друкував ще на друкарських машинах). При розробці дизайну для людей похилого віку, особливо для людей старше 70 років, дотримуйтесь простих жестів. Забудьте про складні жести, для яких потрібно більше двох пальців Прості горизонтальні, вертикальні або діагональні рухи - це чудово, оскільки це все природні рухи. Але уникайте поєднання жестів із швидкими рухами, важким позиціонуванням або кількома жестами, які вимагають використання обох рук або більше двох пальців. Все це може викликати розчарування навіть у технічно підкованих літніх користувачів, оскільки рухова функція знижується;

Поступові впровадження характеристик товару, розкриття інформації та мінімалістичний дизайн може допомогти запобігти когнітивним перевантаженням у літніх людей, вплине на їх залученість та покращить враження про ваш застосунок.

РОЗДІЛ 3 Опис реалізації програмного продукту

3.1 Загальна інформація про Swift як мову програмування

Не зважаючи на те, що Swift досить молода мова програмування, вона вже встигла завоювати прихильність iOS розробників по всьому світу. Swift це об’єктно-орієнтовна статично-компільована мова програмування з суворою типізацією розроблена компанією Apple, та представлена на одній зі своїх щорічних презентацій WWDC в 2014 році [16]. За даними досліджень відомого серед усіх програмістів порталу StackOverflow Swift займає 16-те місце по популярності серед програмістів світу [17]. Основним застосуванням Swift є розробка користувацьких застосувань під операційні системи компанії Apple: iOS, macOS, watchOS, tvOS з використанням API Cocoa та фреймворку Cocoa Touch.

Окрім всіх особливостей притаманних ООП мовам, таких як наслідування, поліморфізм, інкапсуляція та абстракція, Swift успадковує найкращі елементи таких мов як, C та Objective-C, тому розробники що вже опанували будь-яку з цих двох мов, зможуть швидко вивчити Swift. Проте існують деякі відмінності. До прикладу Swift використовує засоби автоматичного розподілу пам'яті і контролю переповнення змінних і масивів, що значно збільшує надійність і безпеку коду написаного саме на цій мові [18].

За заявою Apple [19], код Swift виконується в 2.6 рази швидше коду на Objective-C і в 8,4 рази швидше, ніж код написаний на Python 2.7 (див рис. 3.1.1, 3.1.2). Замість garbage collector`a яким користується Objective-C Swift використовує засоби підрахунку посилань на об'єкти, що дає змогу автоматично видаляти об’єкти за відсутності використання.

Мова також пропонує низку сучасних методів програмування, таких як замикання(closure), узагальнене програмування, лямбда-вирази, та елементи функціонального програмування. Містить у собі такі структури даних як кортежі та словникові типи, може здійснювати швидкі операції над колекціями об’єктів.

Також радістю для розробників є те, що код написаний мовою Swift може з легкістю комбінуватися та бути сумісним з кодом на C і Objective-C та використовуватися в одному проєкті.

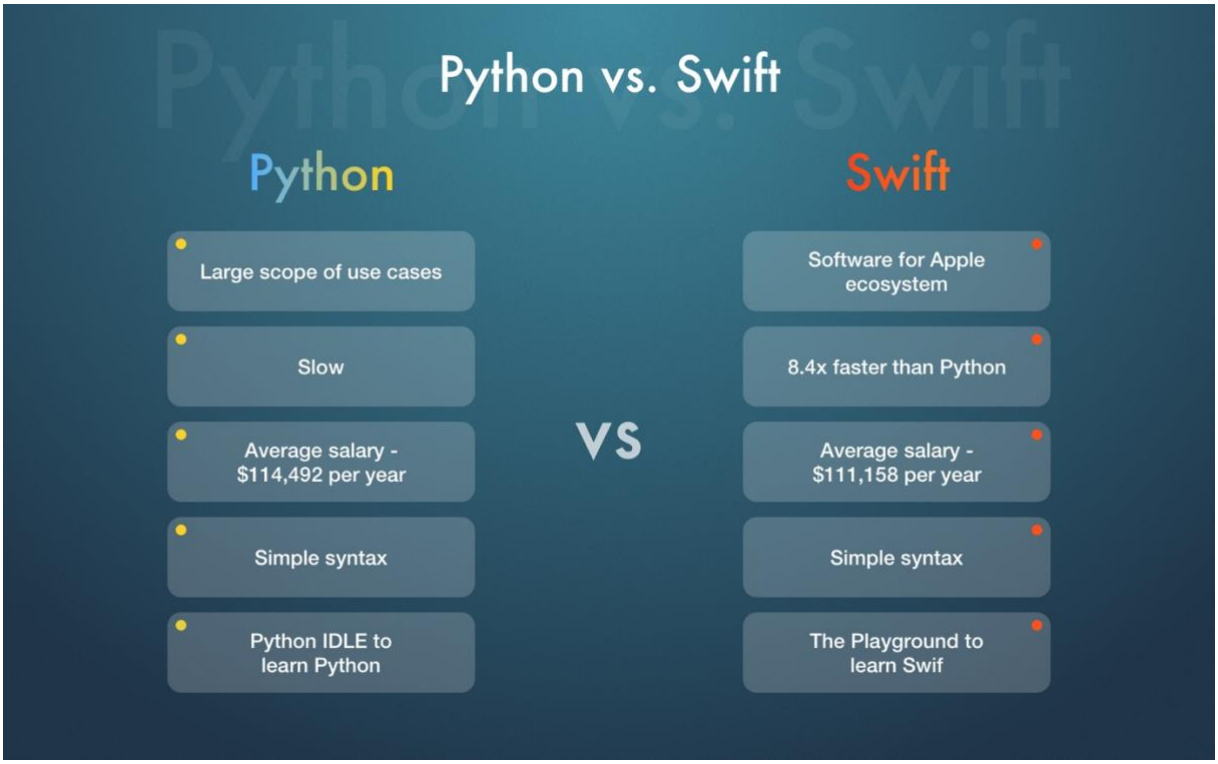


Рисунок 3.1.1 Порівняння мов Swift та Python [20]

Objective-C	Swift
Types are checked, but not enforced in compile time	Types are enforced in compile and runtime
Out of bounds and stack overflows are possible	Out of bounds and stack overflows are prevented
Lightweight generics are informational for compile time	Generics are enforced in compile time and runtime
Header files and implementation files	One file that contains both interface and implementation
Manual memory management for CoreFoundation	Automatic memory management for CoreFoundation
Message sending(dynamic dispatch)	Static method dispatch for value types
Support for Objective-C++	No support for Objective-C++

Рисунок 3.1.2 Порівняння мов Swift та Objective-C [21]

3.2 Обґрунтування вибору засобів розробки

Оскільки необхідно розробити застосунок під ОС iOS, то єдиним лідером серед мов програмування, що підходять для розробки ПЗ саме під цю операційну систему безумовно був та залишається Swift, переваги якого вичерпно описані в попередньому розділі. Крім того, не мало важливим фактором, котрий вплинув на обрання саме розробки під iOS, стало особисте бажання опанувати мову програмування Swift та доторкнутися до розробки саме під цю ОС. Середовище розробки це IDE — Xcode [22], котра була розроблена Apple виключно для розробки саме під свої нативні ОС як: macOS, iOS, watchOS, tvOS. Найбільшою перевагою Xcode є те, що він містить функціонал для легкої побудови та редагування користувацьких інтерфейсів всередині IDE, такі редактори інтерфейсів називаються StoryBoard`ами (див. рис. 3.2.1).

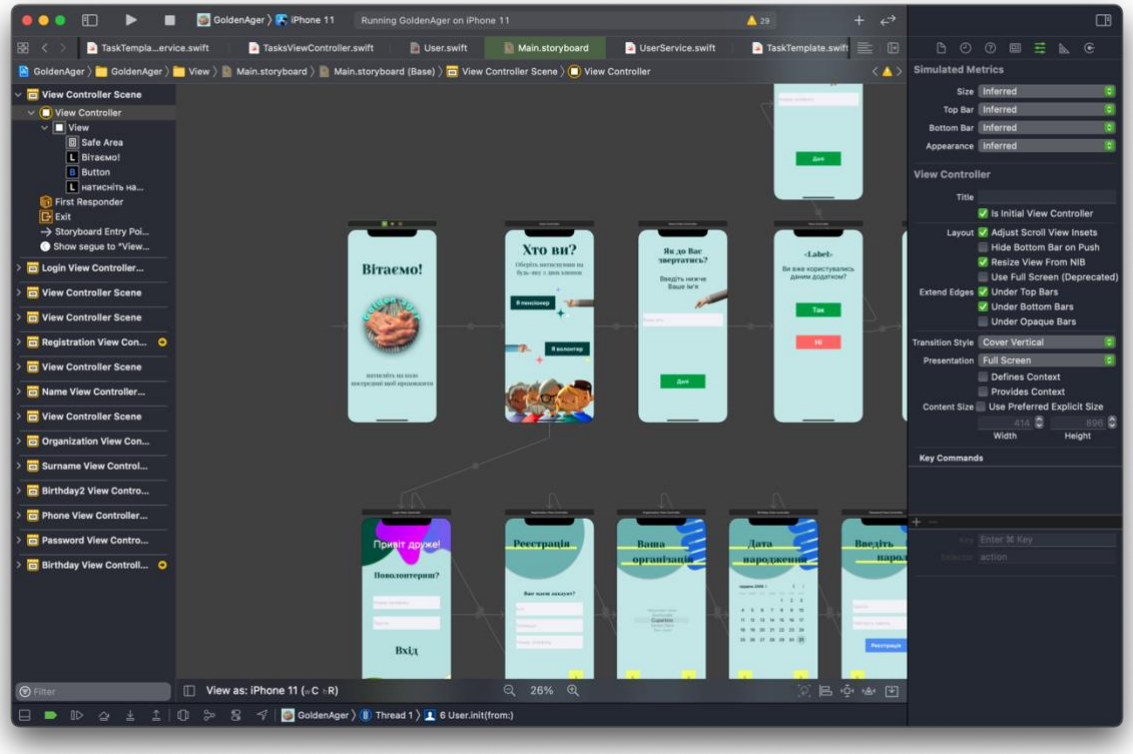


Рисунок 3.2.1 Редактор інтерфейсу Main.storyboard

Для розробки серверної back-end частини мого застосунку було обране середовище Node JS для виконання коду написаного на мові javascript в серверній частині, Express фреймворк, середовище розробки WebStorm. У якості бази даних

була обрана NoSQL база даних MongoDB, яка була була розміщена на хмарному сервісі Heroku. Для розробки дизайну макету логотипу та ware-frame мого застосунку були обрані Adobe Photoshop та Figma [23] відповідно.

Обрання Node Js Express як основного фреймворку для розробки сервера на back-end`і було зумовлено тим, що фреймворк являється досить простим в освоєнні та досить швидким засобом розробки робочого серверу. Також на вибір Node Js Express як основного стеку для back-end розробки застосування вплинула наявність в ньому широкого інструментарію для розробки саме клієнт-серверних застосувань. Також не мало важливу роль відіграв попередній досвід роботи з цим фреймворком та популярність на ринку України цього засобу розробки. У вигляді допоміжних бібліотек були використані node пакети: mongoose, bcrypt, jwt, для роботи з БД, реалізації функції хешування паролів та авторизації/аунтентифікації відповідно.

Базою даних для мого проєкту була обрана noSQL база даних Mongo DB. На таке рішення вплинув в першу чергу попередній досвід у роботі з нею, а також деякі її суттєві переваги в порівнянні з іншими реляційним та не реляційними базами даних [24,25,26].

- В MongoDB “колекції” та “документи” – аналоги “таблиць” та “кортежів” в реляційних БД;
- Гнучкі схеми документів, що дозволяють легко створювати та маніпулювати даними;
- Зручний доступ до власних даних з коду багатьох мов програмування;
- Дизайн, зручний для змін, що дає можливість змінювати структуру та схеми ваших колекцій без суттєвих труднощів;
- Потужні запити та аналітика, за допомогою мови запитів MongoDB (MQL) можна глибоко досліджувати документи та виконувати складну аналітику;
- Просте горизонтальне масштабування;
- Простий формат індексів;

Всі ці критерії роблять базу даних Mongo DB легкою та дружньою в освоєнні новачками.

Оскільки, формат розроблюваного застосунку передбачав клієнт серверну взаємодію, тож перед початком роботи постало питання де саме розмістити сервер. Звісно, це все могло відбуватися на localhost просте значно кращою та більш розповсюдженою практикою є розміщення серверу у віддаленому хмарному сховищі. Тож, був обраний хостинговий сервіс Heroku з котрим раніше доводилось працювати. Також суттєву роль відіграло те, що процедура розміщення серверу Node js Express на Heroku э дуже швидкою та не вимагає додаткових зусиль.

3.3 Опис структури програми

Структура програми побудована на основі клієнт-серверного підходу, що включає реалізацію віддаленого серверу за допомогою Node.js Express (див. рис. 3.3.1) та відображення клієнтської front-end частини за допомогою мови Swift.

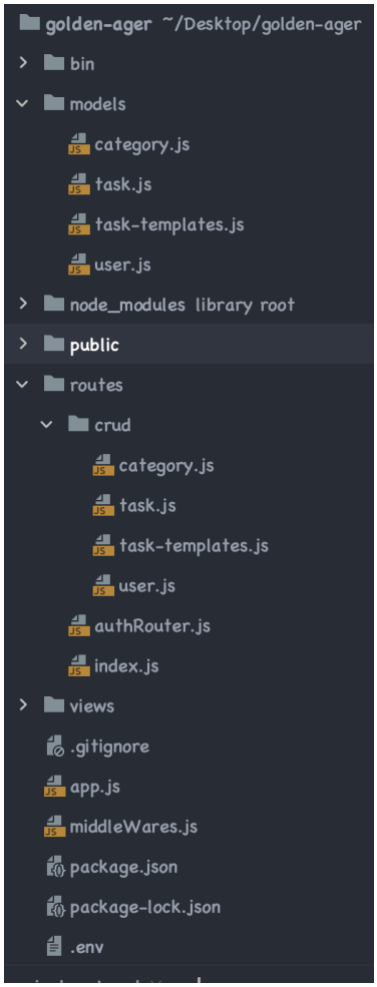


Рисунок 3.3.1 Структура серверу Node.js Express

Обрана трирівнева клієнт-серверна архітектура, а побудова та структуризація front-end частини відбувається за допомогою патерну MVC. (див. рис. 3.3.2). Де папка View містить всі користувацькі інтерфейси, як для волонтера так і для пенсіонера, папка View-Controllers, фактично уособлює в собі роль Контролера з патерну MVC, для керування взаємодією між інтерфейсами з папки View та моделями з папки Model відповідно. У папці Model містяться файли що відповідають за взаємодію з сервером та відправку запитів до БД для кожної сутності (Категорії, Завдання, Шаблони завдань, Користувачі).

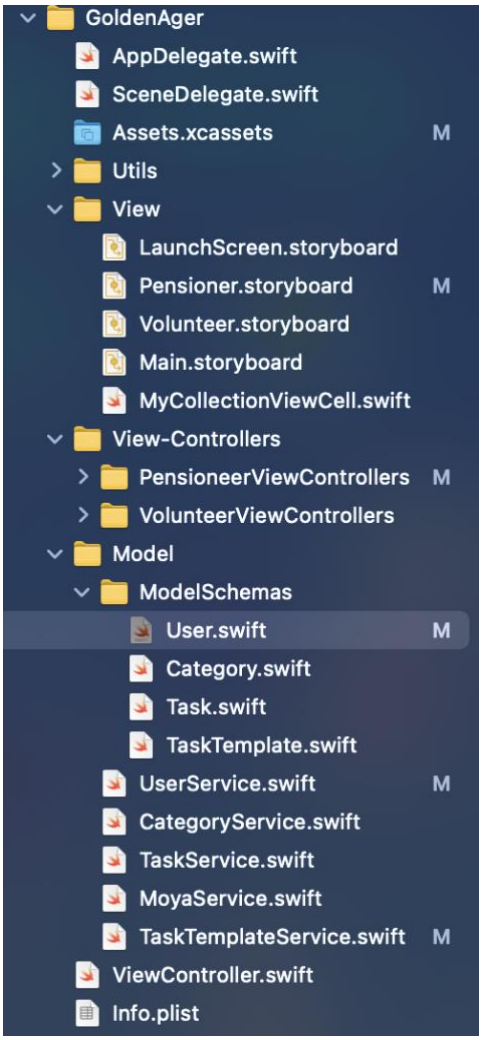


Рисунок 3.3.2 Front-end чистина застосування побудована за MVC патерном

3.4 Опис розробки програми

Робота над застосунком почалася з продумування влучної назви, логотипу та підбору кольорової гами, яка влаштовувала б як пенсіонерів так і волонтерів. Адже, так важливо було створити цікавий та зручний у використанні застосунок який сподобався б таким різним віковим групам.

Були розглянуті три варіанти назв застосунку.

Перший варіант – “senior cityZen”. Слово senior таке знайоме серед усіх програмістів, означає досвід та стаж, також означає доросла, зріла людина. На додачу, планувалося погратися з словом “cityZen”, що в перекладі буквально означає містянин, той що живе в місті, та складається з слова “Zen” – що означає спокій, який може подарувати даний застосунок, адже він направлений саме на вирішення проблем літніх(“senior”) користувачів. Проте цю назву було відкинуто через надуману складність та перенасиченість прихованими та незрозумілими з першого погляду сенсами, тому що головною поставленою метою була прозорість та доступність, навіть у назві застосунку, як для молоді так і для літніх користувачів.

Другою назвою, що розглядалася був “Old-Cap”, що в перекладі означає старий друг. Ідея цієї назви зародилася з бажання створити такого собі друга-помічника що завжди буде у кожного пенсіонера буквально під рукою, знаходячись у планшеті чи смартфоні. Також у цій назві присутнє слово “old”, яке хотілось якось виділити на логотипі, щоб продемонструвати майбутнім користувачам, що це не лише програма помічник, а програма що розроблена саме для літніх людей.

І наостанок, назва що в результаті і стала фінальною це – “Golden Ager”, що в перекладі буквально означає людину пенсійного віку, проте хотілось донести і інше бачення. Адже, люди літнього віку це ті, хто в першу чергу має величезний життєвий досвід, що є дуже цінним для молодих людей, котрі можуть цей досвід перейняти. Цінність уособлюють різні коштовності та дорогоцінні метали, тож поміркувавши, було обрано золото, що в комбінації зі словом “ager” утворює

неймовірно влучні комбінації перекладу як наприклад “людина золотого віку” або “людина з досвідом”.

Отже, фінальна версія назви вийшла дуже точною та доцільною, адже при дослівному перекладі означає “пенсіонер”, в свою чергу окремо взяте слово “golden”, відображає саме той меседж який би я хотіла передати своїм застосунком.

З логотипом я не хотіла довго гратися та продумувати такі популярні наразі в AppStore ігри з першими буквами назви в логотипі. В першу чергу, я хотіла щоб логотип містив назву в тонах майбутньої кольорової гами та зображення, що демонструє роль та призначення мого застосунку, а саме допомогу літнім людям. Таким чином була обрана фотографія рук старенької бабусі, що символізує щось таке дороге та рідне для кожного з нас, те про що хочеться піклуватися. Згодом це зображення було доведене до ладу та перетворене в повноцінний символ та логотип застосунку, в редакторі Photoshop (див. рис. 3.4.1).

Рисунок 3.4.1 Розробка логотипу застосування “Golden Ager”

Наступним важливим етапом розробки стало створення макету (ware-frame) майбутнього застосунку в редакторі Figma (див. додаток Г). Для пенсіонерів була обрана пастельна кольорова гама з простими акцентними кольорами, такими як зелений червоний та жовтий, на підказках та на кнопках. На противагу цьому, для сценарію Волонтера було обрано яскраві кольори в

інтерфейсі в усіх тенденціях сучасних варіантів розробки застосунків для молоді (див. рис. 3.4.2, 3.4.3).

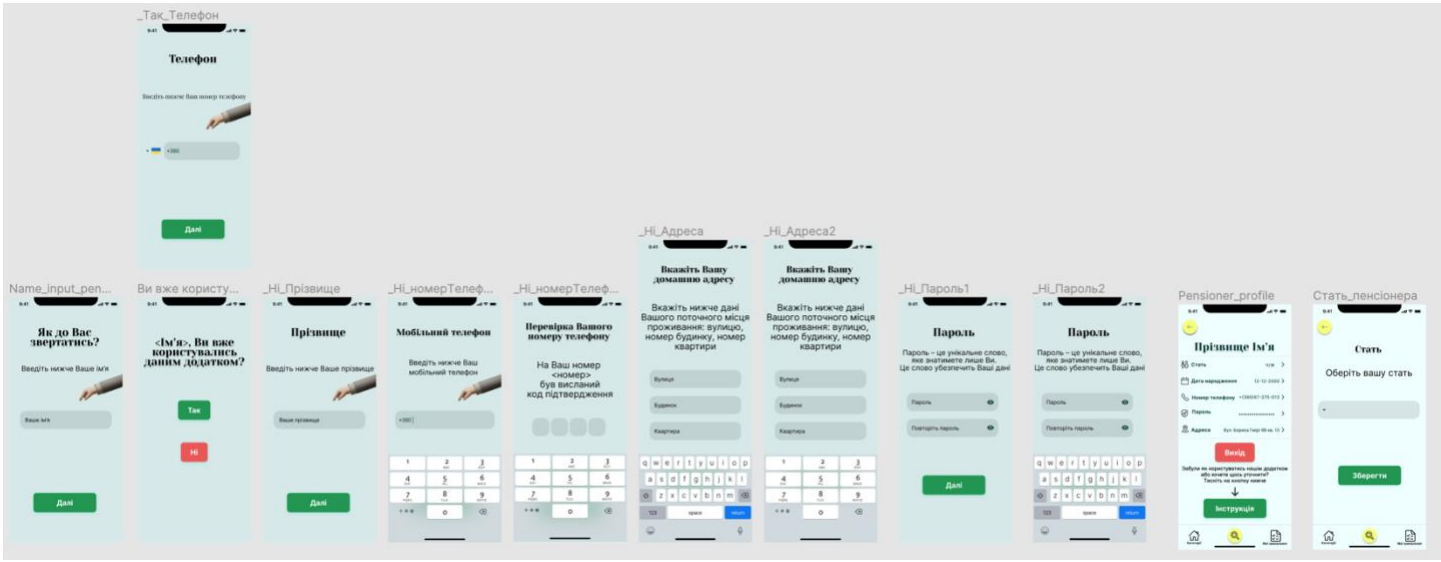


Рисунок 3.4.2 Сценарій "Пенсіонер" — UI для реєстрації пенсіонера

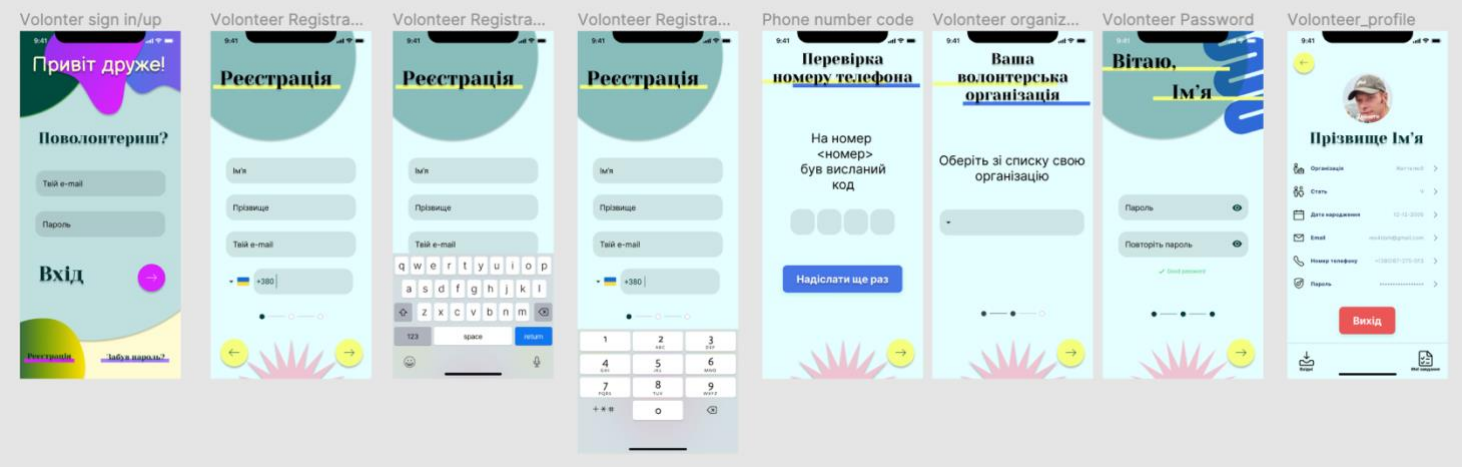
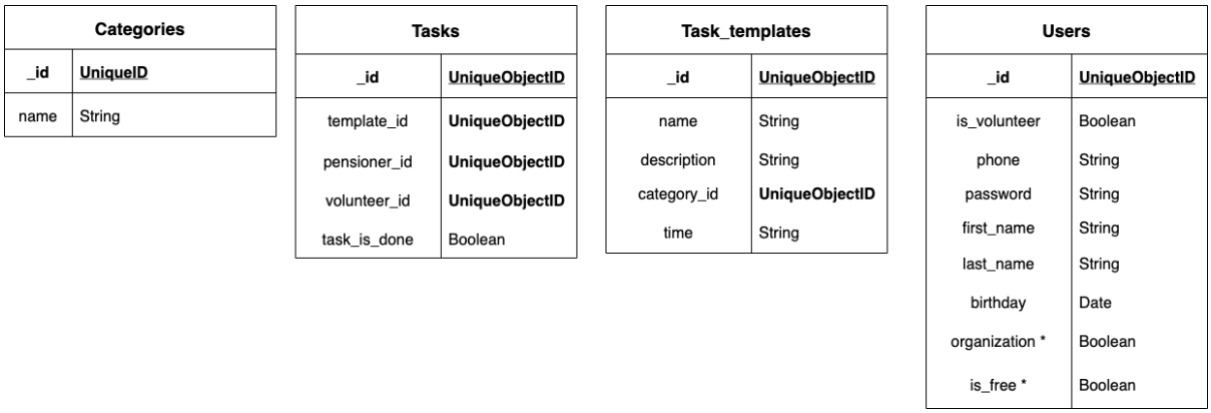


Рисунок 3.4.3 Сценарій "Волонтер" — UI для реєстрації волонтера

Наступним необхідним етапом створення робочого мобільного застосування є формування та розробка бази даних котра включала б всі зазначені властивості та сутності (див. рис. 3.4.5). Отже, розроблена для даного застосування база складається з таких сутностей як: Користувачі(Users), Завдання(Tasks), Шаблони завдань(Task templates), Категорії(Categories).



Categories	
_id	UniqueID
name	String

Tasks	
_id	UniqueObjectID
template_id	UniqueObjectID
pensioner_id	UniqueObjectID
volunteer_id	UniqueObjectID
task_is_done	Boolean

Task_templates	
_id	UniqueObjectID
name	String
description	String
category_id	UniqueObjectID
time	String

Users	
_id	UniqueObjectID
is_volunteer	Boolean
phone	String
password	String
first_name	String
last_name	String
birthday	Date
organization *	Boolean
is_free *	Boolean

Рисунок 3.4.5 Представлення сутностей БД в вигляді таблиць (жирним виділені ключі)

Сутність Користувачі(Users) є спільною для користувачів Волонтерів та Пенсіонерів та містить у собі такі атрибути:

- _id — Первинний ключ сутності Користувачі типу ObjectID, унікальний ідентифікатор
- Is_volunteer — атрибут що фактично вказує ким являється даний користувач, волонтером чи пенсіонером, типу Boolean
- Phone — мобільний телефон користувача типу String
- Password — захешований пароль користувача типу String
- First_name — ім’я користувача типу String
- Last_name — прізвище користувача типу String
- Birthday — дата народження користувача типу Date
- Organization — опціональний атрибут типу String, якщо Is_volunteer=false, то атрибут відсутній, містить у собі назву волонтерської організації до якої належить даний волонтер якщо
- Is_free — опціональний атрибут типу Boolean, якщо Is_volunteer=false, то атрибут відсутній, містить у собі інформацію чи зайнятий наразі даний волонтер якимось завданням, чи він вільний і готовий прийняти нове завдання від пенсіонера

Сутність Шаблони завдань(Task templates) складається з полів котрі будуть статичними в завданні і можуть повторюватись, містить у собі такі атрибути:

- _id — Первинний ключ сутності Шаблони завдань типу ObjectID, унікальний ідентифікатор
- Name — назва завдання, типу String
- Description — опис завдання, типу String
- Category_id — зовнішній ключ, що зв’язує сутність Шаблони Завдань з сутністю Категорії
- Time — атрибут типу String, що вказує час, котрий піде на виконання відповідного завдання

Сутність Завдання(Tasks) уособлює всю логіку котра буде міститися в одному екземплярі завдання, котре зможе замовити пенсіонер. Ця сутність розширює сутність Шаблони завдань та містить динамічно-змінювані атрибути Pensioner_id, Volunteer_id, Task_is_done котрі не будуть повторюватись. Містить у собі такі атрибути:

- _id — Первинний ключ сутності Завдання типу ObjectID, унікальний ідентифікатор
- template_id — зовнішній ключ, що зв’язує сутність Завдання з сутністю Шаблони завдань
- Pensioner_id — зовнішній ключ, атрибут що зв’язує сутність Завдання з сутністю Користувач
- Volunteer_id — зовнішній ключ, атрибут що зв’язує сутність Завдання з сутністю Користувач
- Task_is_done — атрибут типу Boolean, що вказує виконане завдання чи ні

Сутність Категорії(Categories) містить у собі такі атрибути:

- _id — Первинний ключ сутності Категорії типу ObjectID, унікальний ідентифікатор
- Name — назва категорії (будуть доступні 7 категорій: “Здоров’я”, “Рахунки”, “Доставка”, “Ремонт”, “Хатня робота”, “Інше ”, “Консультація ”), типу String

За допомогою ресурсу Mongo DB Atlas була створена віддалена база, розміщення якої відбувається на серверах цього ресурсу. Далі вся попередня взаємодія з базою, до моменту розробки серверу відбувалася через СКБД Mongo DB Compass (див. рис. 3.4.6)

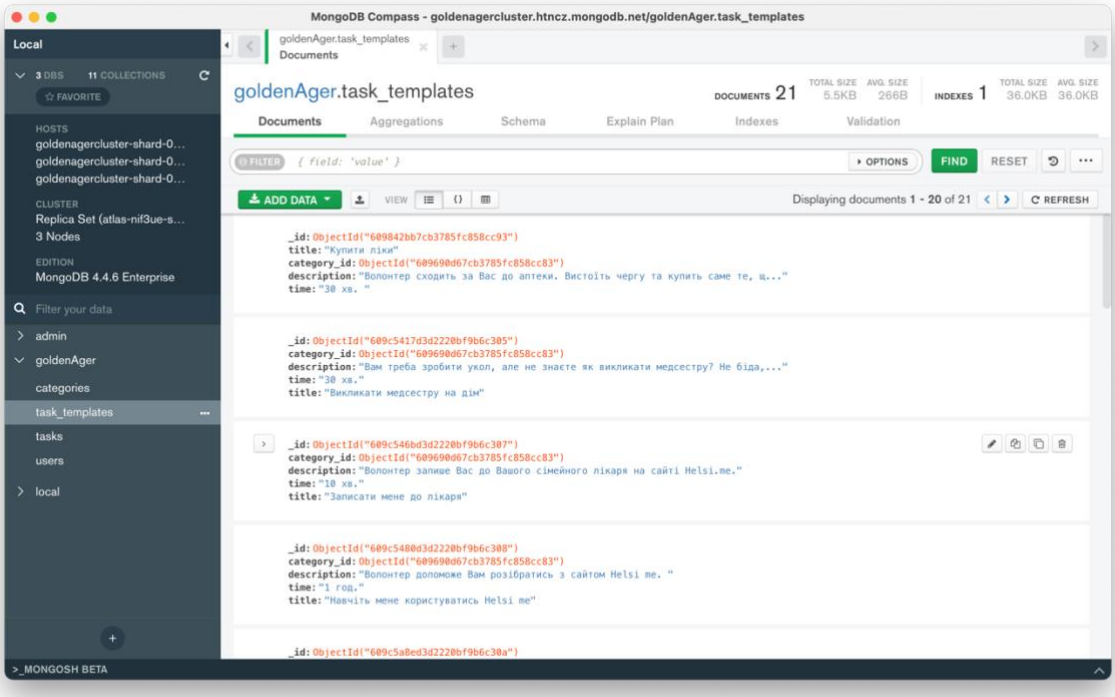


Рисунок 3.4.6 Список користувачів в СКБД Mongo DB Compass

3.5 Створення об'єктів і розробка головної програми

Першочерговою задачею було створити сервер для зв’язування з віддаленою БД. Було обрано стек технологій Node.js Express фреймворк, який є досить швидким в імплементації, адже при створенні проєкту містить автоматично згенерований код, що допомагає в подальшому зменшити час на встановлення підключення. Наступним етапом є додавання node пакету moongoose, котра являє собою ORM бібліотеку доступу до даних mongodb. ORM [27] — це технологія, котра дозволяє відправляти запити та взаємодіяти з БД за допомогою ООП підходу. Далі необхідно підключити віддалену БД до moongose (див. додаток Ґ).

Після успішного створення серверу на localhost, необхідно захостити сервер на будь яке хмарне середовище, у випадку даної роботи було обрано хостинговий сервіс Heroku (див. додаток Д).

Наступним етапом є створення моделей (папка models) (див. рис. 3.5.1) схем для кожної сутності наявної в БД, для подальшої взаємодії з нею (див. додаток Е) як об’єктом бібліотеки moongoose. Також у автоматично згенерованій Express’ом папці routes було створено папку crud, у якій необхідно створити файли javascript роутерів для кожної сутності, яка потребуватиме надсилання запиту на сервер. Ці роутери визначатимуть функції, які будуть виконуватись у відповідь на певний запит надісланий сервером. Запит на сервер надсилатимуть за певним, відповідним до сутності що надсилає запит, URL маршрутом. Окрім унікальних та складних запитів всередині кожного роутера необхідно реалізувати “CRUD” [28] операції-запити до кожної сутності наявної в БД (див. рис. 3.5.1). Також для роутерів кожної сутності у файлі app.js було створено відповідний зв’язок, що забезпечуватиме відправлення запитів на сервер по відповідному URL шляху (див. додаток Є пункт 1).

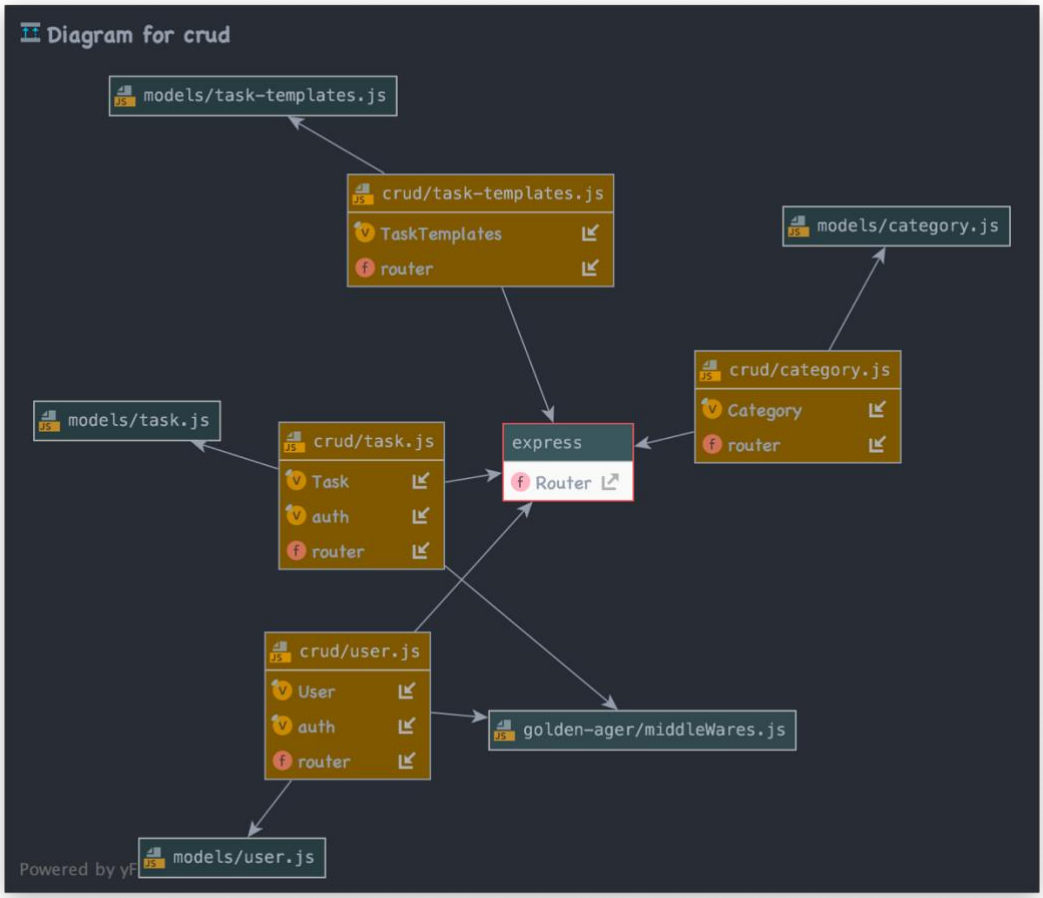


Рисунок 3.5.1 Зв’язок роутерів сутностей(crud/) з їхніми моделями(models/)

У папці routes було створено файл authRouter у котрому відбувається взаємодія з БД для процесів реєстрації та логіну користувача. В цьому файлі я послуговуюсь бібліотекою bcrypt котра допомагає хешувати паролі перед відправкою їх до БД (див. додаток Є пункт 2).

Також, для розбиття логіки мого застосунку на сценарії волонтер та пенсіонер вкрай необхідно було реалізувати авторизацію та автентифікацію. Автентифікація — це безпосередній вхід користувача в систему під відомими для неї, до прикладу, логіном і паролем. Авторизація — це надання користувачу певних прав та повноважень в цій системі після процесу автентифікації. Для реалізації цих двох процесів я скористалась бібліотекою jsonwebtoken, котра надає користувачу певний токен під час першої автентифікації. Цей токен буде передаватися в заголовку запиту під час наступних звернень до сервера. Отримавши заголовок, сервер, спочатку, декодує токен за секретним ключем і порівняє з даними користувача, та якщо операція була успішною – користувача буде авторизовано (див. додаток Є пункт 3).

Для авторизації користувача було створено окремий файл middleWares.js. Роль користувача в системі я визначаю умовою, чи дорівнює параметр is_volunteer, користувача що увійшов, значенню true – якщо він волонтер та false – якщо він пенсіонер відповідно (див. додаток Є пункт 4). (Для ознайомлення з загальною структурою серверної back-end частини див. додаток Ж.)

Наступним етапом було створення інтерфейсу користувача в IDE Xcode, за попередньо створеним в Figma макетом, та зв’язування його з серверною back-end частиною.

Відповідно до описаної раніше концепції MVC, в новоствореній папці View було розроблено 4 файли (див. рис. 3.5.2) візуального представлення користувачу, що містять відповідну до назв логіку:

- Main.storyboard – інтерфейс що містить екрани для логіну/реєстрації (спільний файл для логіки пенсіонера і волонтера)
- Pensioner.storyboard – місить екрани категорій, відображення завдань по категоріям та список замовлень завдань

- Volunteer.storyboard – містить екрани вхідних завдань та архіву виконаних завдань, також екран з кнопкою що виводить волонтера в режим готовності до виконання завдань (is_free==true) замовлених пенсіонером.
- Launch.storyboard – містить початковий екран завантаження застосування

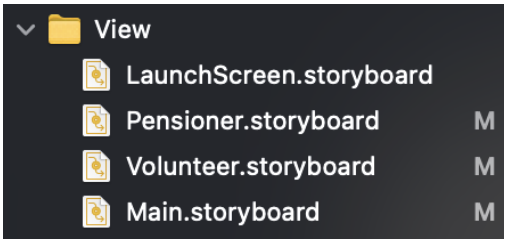


Рисунок 3.5.2 Структура папки View

У папці Model були створені файли сервісів та папка Shemas (див. рис. 3.5.3), в котрій містяться-файли обгортки над існуючими сутностями (Категорії, Шаблони завдань, Завдання, Користувачі), для коректного запакування даних по цим сутностям і подальшого відправлення їх на сервер в тілі необхідного запиту. Також по всім переліченим сутностям були створені файли сервісів (User, Category, Task, TaskTemplate), кожний з яких відповідає за відправлення запитів, пов'язаних конкретно зі своєю сутністю відповідною до назви файлу, на сервер. Також у папці Model був розміщений файл FieldsValidator.swift, що відповідно до назви валідує, тобто відповідає за правильність заповнення текстових полів вводу на стороні користувача, для введення ним коректних даних в програму.

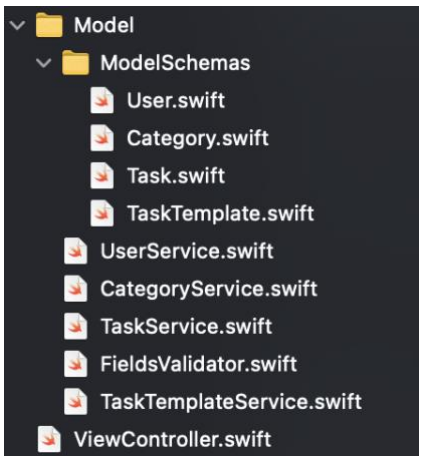


Рисунок 3.5.3 Структура Model(частини MVC патерну), що містить файли сервісів, та схем сутностей

Наступним найважливішим етапом розробки front-end частини було зв’язування файлів візуального представлення користувачу з файлами сервісів, котрі, як було описано раніше відправляють запити на сервер. Досягти цього можна за допомогою створення Controllr`a, що коректно визначав би місце або дію користувача після якої необхідно відправити запит на сервер і чекати відповідь від нього. Отримавши відповідь від серверу Controller знав би де, як та в якому форматі відобразити дані з цієї відповіді, щоб найкращим чином донести необхідну інформацію до користувача, котрий її очікує від застосування. У мові swift роль Controllr`a виконують файли типу UIViewController, котрі пов’язують графічні екрани користувача з необхідною логікою, котра має виконатися на цьому екрані. Тож для реалізації цього зв’язку було створено папку View-Controllers (див. рис. 3.5.4), у якій було розміщено папки PensioneerViewControllers, VolunteerViewControllers, кожна з яких містить велику кількість різних ViewController`ів відповідно до певного екрану користувача у сценаріях взаємодії з застосунком Пенсіонера і Волонтера відповідно.

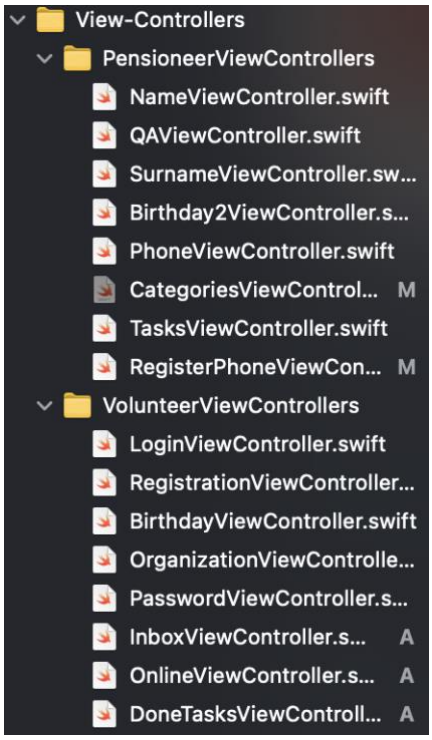


Рисунок 3.5.4 структура Controller`a (з патерну MVC)

3.6 Опис файлів даних front-end частини та інтерфейсу програми

Оскільки про файли даних на back-end частині було описано у попередніх розділах перейдемо до опису файлів що містяться на front-end частині. Файли даних front-end частини в першу чергу включають в себе схеми сутностей, для коректного відправлення даних по цим сутностям на сервер (див. рис. 3.6.1). У свою чергу, файли сервісів про які було описано в попередньому розділі мають досить просту структуру: містять у собі шаблони GET та POST запитів для відповідною сутності(див. рис. 3.6.2), що відправляються за визначеним в router файлах back-end`у URL шляхом. Відправляння цих запитів здійснюється за допомогою бібліотеки Alamofire [29], що являється обгорткою над типовим для відправлення HTTP запитів класом URLSession у мові Swift.

Дії користувача результатом виконання яких буде відправлення цих запитів та отримання відповіді, визначають файли ViewController`ів про які також було описано в попередньому розділі.

```
import Foundation



struct User: Decodable {
    var _id: String
    var is_volunteer: Bool
    var phone: String
    var password : String
    var first_name: String
    var last_name: String
    var birthday: Date
    var organization: String
    var is_free: Bool


    enum CodingKeys : String, CodingKey{
        case _id = "_id"
        case is_volunteer = "is_volunteer"
        case phone = "phone"
        case password = "password"
        case first_name = "first_name"
        case last_name = "last_name"
        case birthday = "birthday"
        case organization = "organization"
        case is_free = "is_free"


    }

}
```

Рисунок 3.6.1 Схема даних для сутності Користувач(User)

```
import Foundation
import Alamofire

class UserService{
    //https://golden-ager.herokuapp.com/users

    fileprivate var baseUrl = ""

    init(baseUrl:String) {
        self.baseUrl = baseUrl
    }


    //MARK:- getAllUsers отримати всіх юзерів
    func getAllUsers(endPoint:String){
        AF.request(self.baseUrl + endPoint, method: .get, parameters: nil, encoding:
            URLEncoding.default, headers: nil, interceptor: nil).response {
            DataResponse in
            guard let data = DataResponse.data else {return}
            do{


                let users = try JSONDecoder().decode([User].self,from: data)
                print("users == \(users)")
            } catch{
                print("Error decoding == \(error)")
            }
        }
    }


    //MARK:- getUserById отримати юзера по ід
    func getUserById(endPoint:String, userId: String){
        AF.request(self.baseUrl + endPoint + userId, method: .get, parameters: nil, encoding:
            URLEncoding.default, headers: nil, interceptor: nil).response {
            DataResponse in
            guard let data = DataResponse.data else {return}
            do{
                self.decodeDate()
                let user = try JSONDecoder().decode([User].self,from: data)
                print("user by id == \(user)")
            } catch{
                print("Error decoding == \(error)")
            }
        }
    }
```

Рисунок 3.6.2 Приклади GET запитів сутності Користувача до серверу

Перейдемо до опису інтерфейсу. При першому вході в застосування інтерфейс користувача передбачає реєстрацію нового користувача (волонтера або пенсіонера) (див. рис. 3.6.3) або ж логін за вже зареєстрованими в системі паролем та номером мобільного телефону, якщо користувач хоче ввійти як волонтер, або лише номер телефону для логіну як пенсіонер (див. рис. 3.6.4).

Рисунок 3.6.3 Перші екрани входу в застосунок (Волонтер чи Пенсіонер)

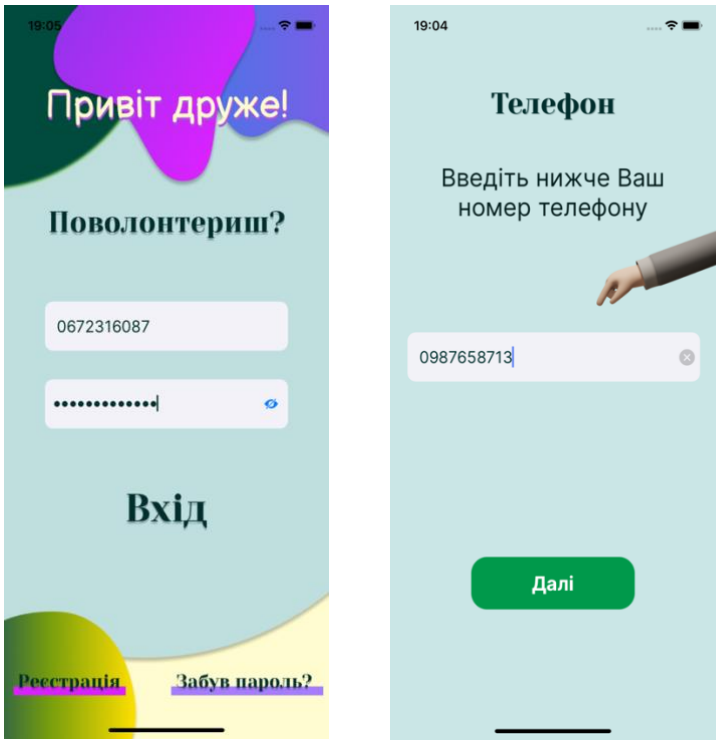


Рисунок 3.6.4 Процедура логіну для волонтера та для пенсіонера (зліва направо)

Після процесів автентифікації та авторизації, що відбуваються для користувача непомітно, кожний користувач потрапляє у основну програму та відповідно до того під ким він увійшов, під пенсіонером чи під волонтером, бачить

різні елементи на меню знизу екрану. У волонтера ці елементи включають (див. рис. 3.6.5):

- Іконку ”Вхідні”, натиснувши на яку волонтер перейде на екран актуальних вхідних завдань.
- Іконку перемикача, натиснувши на яку, волонтер перейде на екран з тумблером, перемкнувши який він вийде в режим онлайн і зможе приймати завдання від пенсіонерів в реальному часі.
- Іконку “Мої завдання”, натискання якої дасть змогу волонтерові побачити список виконаних завдань на новому екрані.

У пенсіонера елементи меню складаються з (див. рис. 3.6.6):

- Елементу “Категорії”, натискання на який дає змогу пенсіонеру обрати завдання з представленого на новому екрані списку категорій.
- Елементу “Інструкція”, де пенсіонер зможе знайти всі необхідні інструкції по користуванню застосунком використовуючи scroll, якщо раптом забуде як працює той чи інший елемент інтерфейсу.
- Елемент “Мої замовлення”, де на новому екрані міститимуться всі замовлені пенсіонером завдання

Рисунок 3.6.5 інтерфейс Волонтера (переключання тумблеру посередині зверху і знизу)

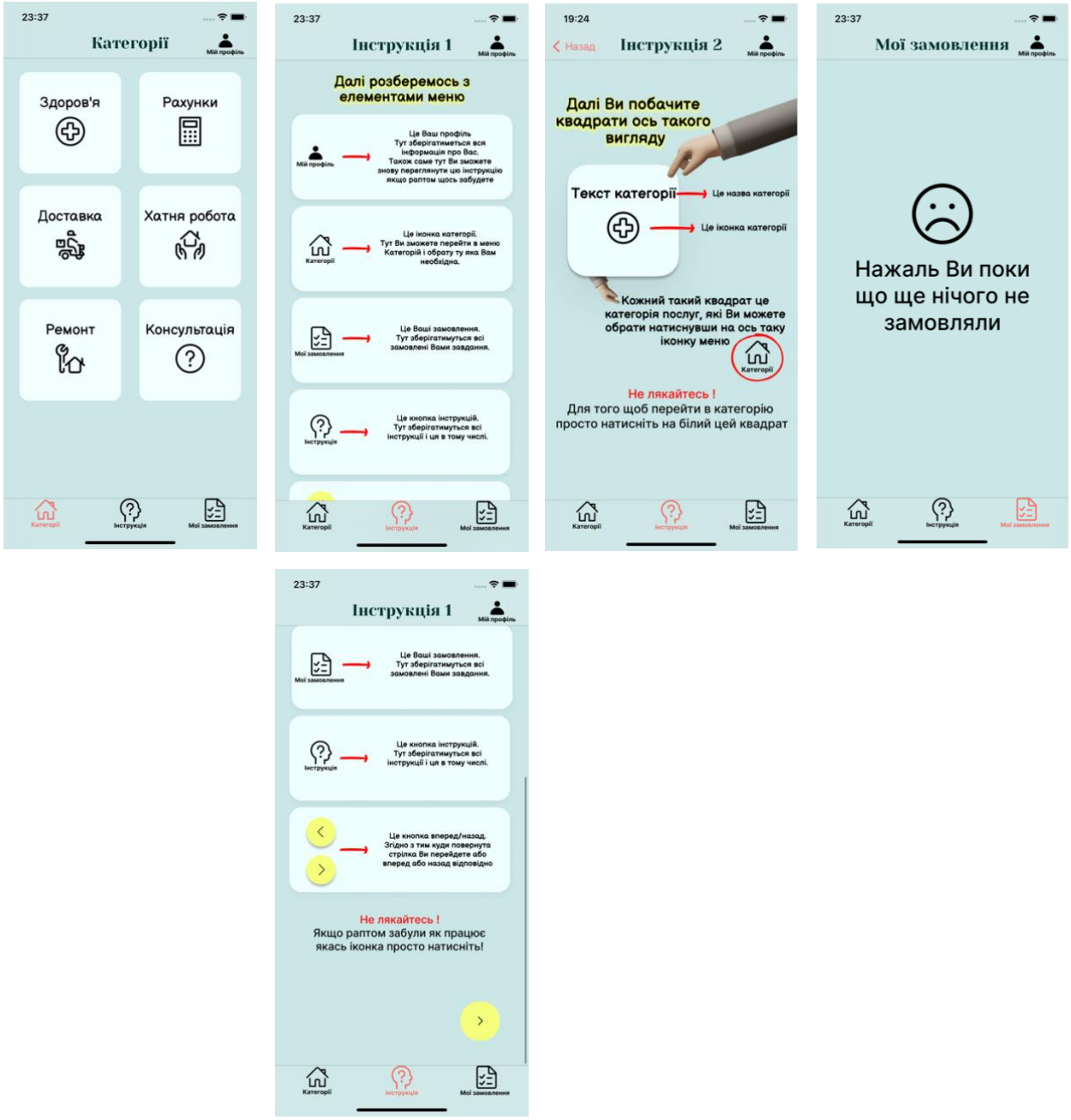


Рисунок 3.6.6 інтерфейс Пенсіонера

3.7 Інструкція користувача

Для запуску застосунку “goldenAger” на своєму ПК:

- Необхідно мати встановлені на комп’ютері Xcode версії 10 та вище з можливістю запуску симулятора iPhone 11 з ОС iOS 12 та вище.
- Треба завантажити відповідний архів front-end застосунку з GitHub або скористатися командою “git clone”, вставивши відповідне посилання на

репозиторій після якої можна клонувати його собі на ПК та вносити зміни в віддалений репозиторій GitHub (див. посилання на проєкт на GitHub в додатку З пункт 1).

- В терміналі за допомогою команди “cd” перейдіть в репозиторій клонованого проєкту.
- Введіть в терміналі команду “pod init,” в папці проєкту має створитися файл Podfile.
- Скопіюйте код наведений в додатку К в новостворений Podfile.
- Введіть в терміналі команду “pod install”.
- В папці завантаженого проєкту відкрийте файл GoldenAger.xcworkspace, як результат проєкт “Golden Ager” має відкритись в Xcode
- Запустіть проєкт в Xcode натиснувши на кнопку “Run” та зачекайте, проєкт може білдитись декілька хвилин.

При подальшій роботі з застосунком можливі два сценарії взаємодії користувача: Волонтер та Пенсіонер.

При необхідності працювати з застосунком як волонтер :

- Натисніть на кнопку “Волонтер” на початковому екрані з запитанням “Хто ви?”
- Зареєструйтесь в застосунку слідуючи покроковій інструкції.
- Після реєстрації перейдіть на екран перемикача статусу, натиснувши на середній елемент нижнього меню на екрані.
- Увімкніть тумблер провівши зліва направо, зробивші це ви перейдете в режим онлайн.
- Чекайте на надходження завдань, нові завдання з’являтимуться в розділі меню “Вхідні”(крайній лівий елемент меню).
- Якщо висвітилось нове завдання, то у Вас є можливість прийняти або відхилити його.
- Список виконаних вами завдань Ви можете переглядати в пункті меню “Мої завдання”(крайній правий).

Для роботи за сценарієм пенсіонер:

- Натисніть на кнопку “Пенсіонер ” на початковому екрані з запитанням “Хто ви?”
- Зареєструйтесь в затосунку слідуючи покроковій інструкції.
- Перейдіть в розділ “Категорії” ”(крайній лівий елемент нижнього меню на екрані).
- Оберіть будь яку з представлених категорій (“Здоров’я”, “Рахунки”, “Хатня робота”, “Ремонт”, “Доставка”) або клацніть на пункт “Консультація”, якщо у Вас виникли питання щоб зателефонувати за номером що висвітиться(див. рис. 3.7.1).
- В обраній категорії оберіть будь яке завдання натисніть “Замовити” та чекайте поки вільний волонтер прийме замовлення.
- Присутня можливість переглядати інструкцію по роботі з застосуванням, натиснувши на середній елемент нижнього меню “Інструкція”.
- Є можливість переглядати замовлені завдання натиснувши на “Мої замовлення” в меню.

Рисунок 3.7.1 Номер телефону для консультації

Висновки

Результатом виконання даної роботи є мобільне застосування GoldenAger для організації волонтерської допомоги літнім людям, яке охоплює два сценарії роботи та два різновиди інтерфейсів, призначених для людей з різних вікових груп та різного рівня навичками застосування мобільних пристроїв (літні люди та молодь). Реалізований проєкт складається з двох частин - back-end та front-end, кожна з яких міститься на GitHub (див. додаток З пункт 1,2).

Найголовнішою метою в подальшій роботі над проектом є розробка повноцінного аналогу даного застосування для платформи Android.

Застосування “goldenAger” має багато варіантів для подальшого розвитку. Оскільки популярною практикою закордоном при прийомі в університет та на першу роботу є аналіз позаучбової та волонтерської діяльності, я розглядаю для волонтера можливість отримання сертифікату, який би відображав скільки годин він провів допомагаючи пенсіонерам. Кожен хто має застосування і зареєстрований в ньому як волонтер мав би в особистому кабінеті лічильник годин, який би зараховував години за завдання, відразу ж як пенсіонер позначив би його як виконане. Щомісяця “goldenAger” сповіщав би волонтера про нараховані години та пропонував би завантажити сертифікат у форматі PDF. Це відкрило б широкі можливості для молодих українців як для майбутніх абітурієнтів закордонних університетів та можливо, та можливо, Україна почала б, подібно до інших європейських країн, враховувати такого типу сертифікати при прийомі на роботу або при зарахуванні в університет.

Також у майбутньому розглядаю опцію пенсіонерів завантажувати своє пенсійне посвідчення, а волонтерів свій паспорт в застосунок, задля уникнення шахрайства.

Подальші варіанти розвитку мого застосунку можуть включати вихід його на ринок як повноцінного стартапу, з власною волонтерською організацією та штабом співробітників. Також я розглядаю співпрацю з порталом, “ITbabushka” [30], що пропонує широкий вибір онлайн-курсів для літніх людей для їхнього опанування свої ґаджетів та інтернет світу.

Список використаної літератури

1. United Nations World Population Prospects [Електронний ресурс] / UNWPP. – 2017. https://www.un.org/development/desa/publications/world-population-prospects-the-2017-revision.html.

2. Designing User Interfaces for an Aging Population / J. Johnson, K. Finn., 2017. – 258 с. – (9780128044674).

3. Як засновникам стартапу “Метнись кабанчиком” вдалось перетворити хобі на прибутковий бізнес [Електронний ресурс] // Investory news. – 2019. https://investory.news/zasnovnikam-startapu-metnis-kabanchikom-vdalos-peretvoriti-xobi-na-pributkovij-biznes/.

4. Glovo, Raketa: як служби доставки використовують свій зоряний час [Електронний ресурс] // BBC News Україна. – 2020. https://www.bbc.com/ukrainian/features-52208958.

5. Допомога літнім людям ліками та їжею, 27 перевірених фондів та організацій RU [Електронний ресурс] // mc.today. – 2020. https://mc.today/pomogite-pozhilym-lyudyam-edoj-i-lekarstvami-uzhe-sejchas-vot-kak-eto-sdelat-ne-vyhodya-iz-doma/.

6. Клієнт-серверна архітектура та ролі серверів. [Електронний ресурс] // Medium Ivan Zmerzlyi. – 2017. https://medium.com/@IvanZmerzlyi/клієнт-серверна-архітектура-та-ролі-серверів-9893d8048229.

7. Client-Server Architecture [Електронний ресурс] // openclassroom: https://openclassrooms.com/en/courses/6397806-design-your-software-architecture-using-industry-standard-patterns/6896156-client-server-architecture.

8. MVC vs MVP vs MVVM vs VIPER: Best iOS Architecture Patterns to Choose From [Електронний ресурс] // Suryansh Sarawat. – 2021. https://www.appventurez.com/blog/ios-architecture-patterns/.

9. MVP vs MVC vs MVVM vs VIPER. What is Better For iOS Development? [Електронний ресурс] // Mind Studios. – 2017. https://themindstudios.com/blog/mvp-vs-mvc-vs-mvvm-vs-viper/.

10. "Massive" View Controllers or bad coding style? [Електронний ресурс] // Medium Ivan Zmerzlyi. – 2019. https://medium.com/flawless-app-stories/massive-view-controllers-or-bad-coding-style-bf2b0d57c268.

11. Ending the debate: MVC vs MVP vs MVVM for iOS application development [Електронний ресурс] // Simform. – 2018. https://www.simform.com/mvc-mvp-mvvm-ios-app-development/.

12. VIPER-Architecture for iOS project [Електронний ресурс] // Medium. – 2018. https://medium.com/cr8resume/viper-architecture-for-ios-project-with-simple-demo-example-7a07321dbd29.

13. Design Principles [Електронний ресурс] // Apple https://developer.apple.com/design/human-interface-guidelines/ios/overview/themes/.

14. Phiriyapokanon T. Is a big button interface enough for elderly users? Towards user interface guidelines for elderly users : маг.роб. на здобуття ступеня магістра"Computer Engineer" / Phiriyapokanon Tanid – Вестерос, 2011. – 54 с.

https://www.diva-portal.org/smash/get/diva2:416488/FULLTEXT01.pdf

15. Age Before Beauty – A Guide to Interface Design for Older Adults [Електронний ресурс] // Sergey Polyuk https://www.toptal.com/designers/ui/ui-design-for-older-adults.

16. About Swift [Електронний ресурс] // Apple Inc.. – 2021.

https://swift.org/about/#swiftorg-and-open-source.

17. Most Popular Technologies [Електронний ресурс] // StackOverflow. – 2020. https://insights.stackoverflow.com/survey/2020#most-popular-technologies.

18. Welcome to Swift.org [Електронний ресурс] // Apple Inc.. – 2021.

https://swift.org.

19. Swift. A powerful open language that lets everyone build amazing apps. [Електронний ресурс] // Apple Inc.. – 2021. https://www.apple.com/swift/.

20. Swift vs Python: Which of Them is More Promising in 2019? [Електронний ресурс] // Cleveroad. – 2021. https://www.cleveroad.com/blog/python-vs-swift.

21. Swift vs Objective-C. Which iOS Language To Choose [Електронний ресурс] // MLSDev. – 2021. https://mlsdev.com/blog/swift-vs-objective-c.

22. Xcode Documentation [Електронний ресурс] // Apple Inc.. – 2021. https://developer.apple.com/documentation/xcode/.

23. Олександр О. Керівництво по Figma v. 1.3 Beta (рос. версія) / Окунєв Олександр., 2018. – 256 с.

24. 5 причин використовувати Монго (MongoDB) [Електронний ресурс] https://echo.lviv.ua/dev/9693.

25. Advantages of MongoDB [Електронний ресурс] // 2021 https://www.mongodb.com/advantages-of-mongodb.

26. Вступ до Mongoose для MongoDB [Електронний ресурс] // hackit-ukraine. – 2021. https://hackit-ukraine.com/177-introduction-to-mongoose-for-mongodb.

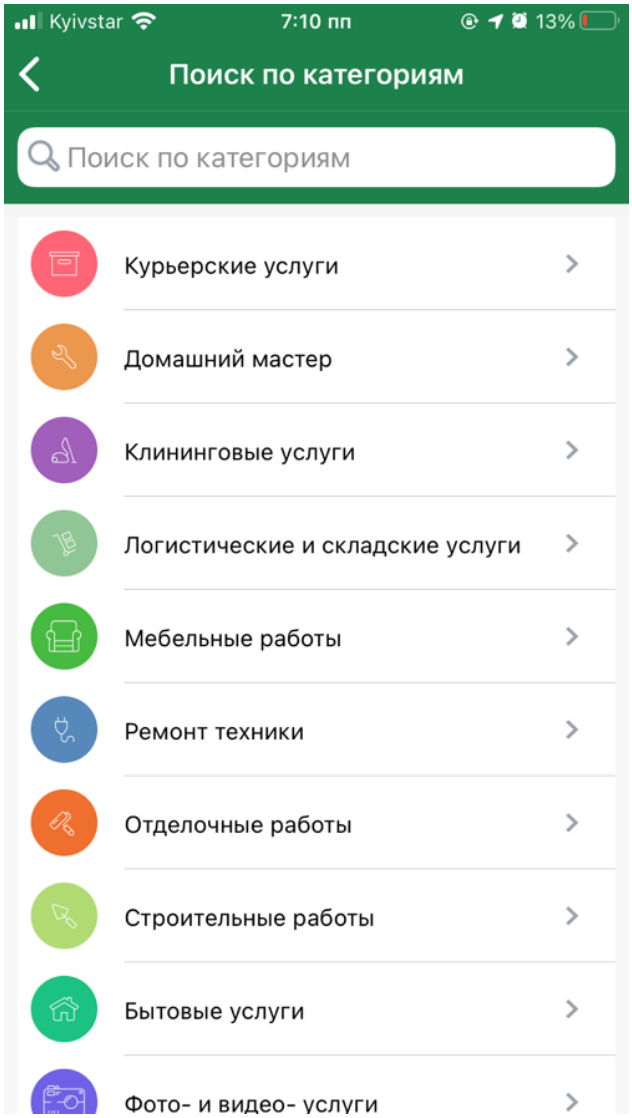
27. WHAT IS ORM? [Електронний ресурс] // Brian Cline. – 2018. https://www.brcline.com/blog/what-is-orm.

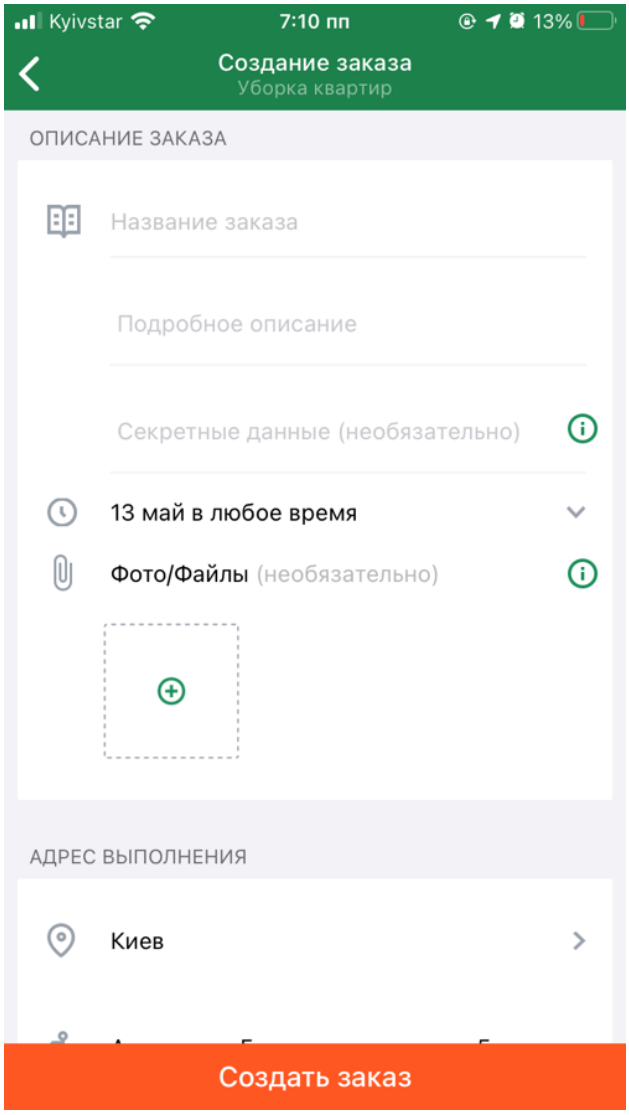
28. What is CRUD? [Електронний ресурс] // code academy https://www.codecademy.com/articles/what-is-crud.

29. Corey D. Alamofire 5 Tutorial for iOS: Getting Started [Електронний ресурс] / Davis Corey. – 2020. https://www.raywenderlich.com/6587213-alamofire-5-tutorial-for-ios-getting-started.

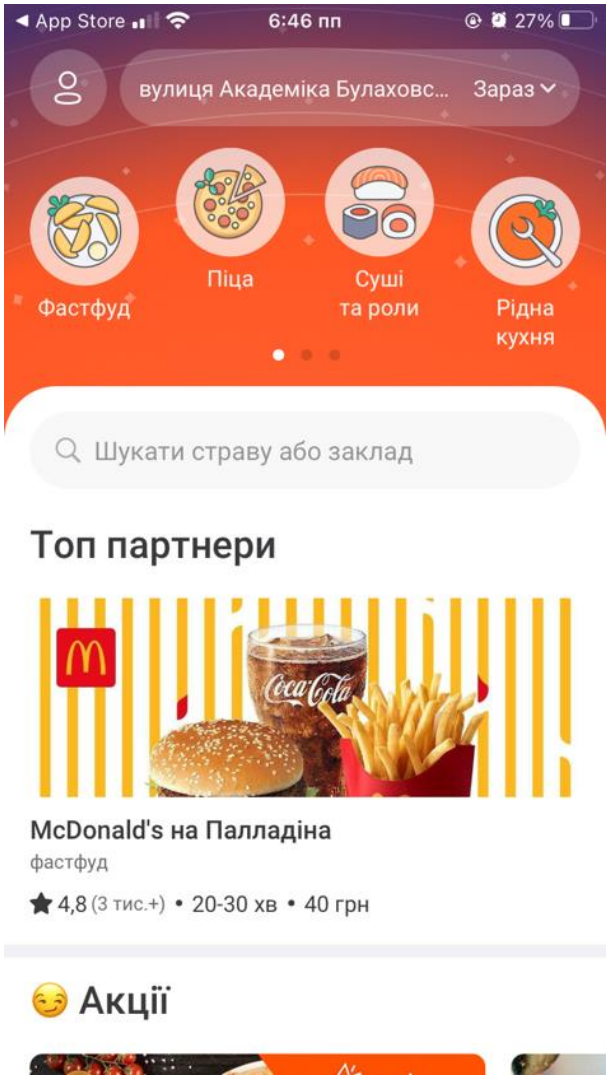
30. IT курси для пенсіонерів [Електронний ресурс] // ITbabushka. – 2020. https://www.itbabushka.com.

Додаток А. Скріншоти інтерфейсу мобільного застосування для замовлення послуг “Кабанчик”





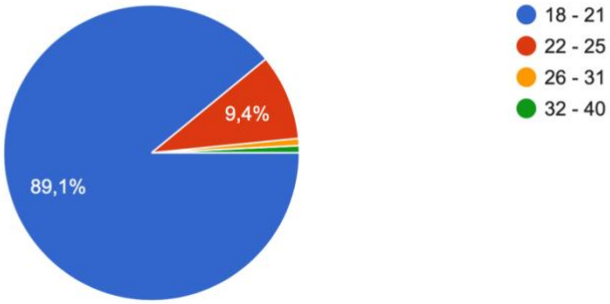
Додаток Б. Скріншоти інтерфейсів мобільних застосувань, що виконують доставку “Glovo”, “Raketa” (зліва направо відповідно



Додаток В. Проведене онлайн-опитування серед студентів Могилянки про актуальність даного застосунку

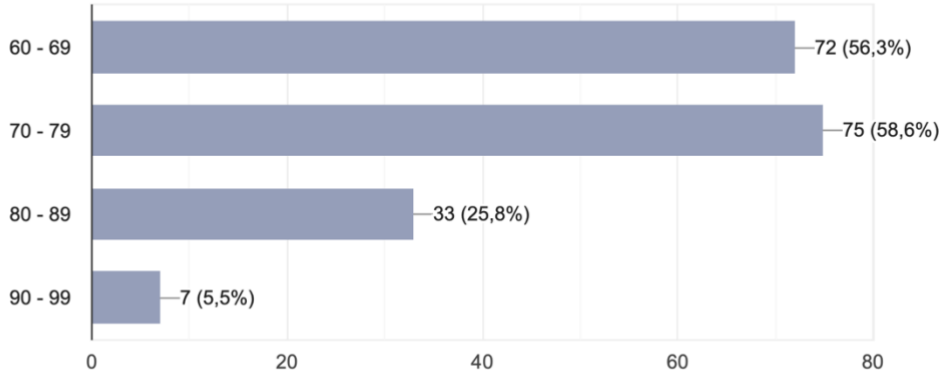
Вкажіть ваш вік?

128 відповідей



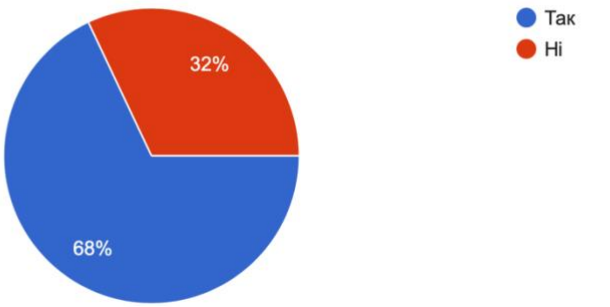
Скільки років Вашим літнім родичам?

128 відповідей

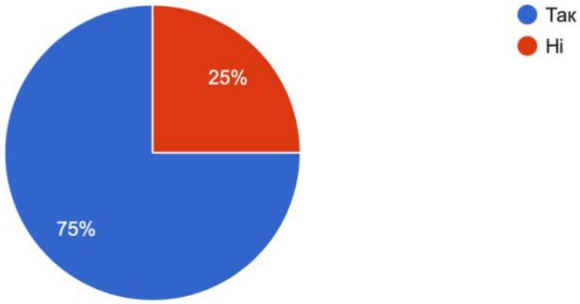


Чи живете Ви зі своїми родичами літнього віку в одному місті/населенному пункті ?

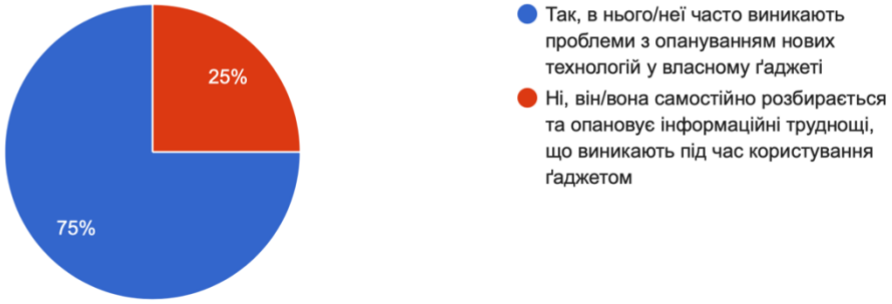
128 відповідей



Чи користуються вони сматрфонами/планшетами для будь-яких власних потреб?
128 відповідей



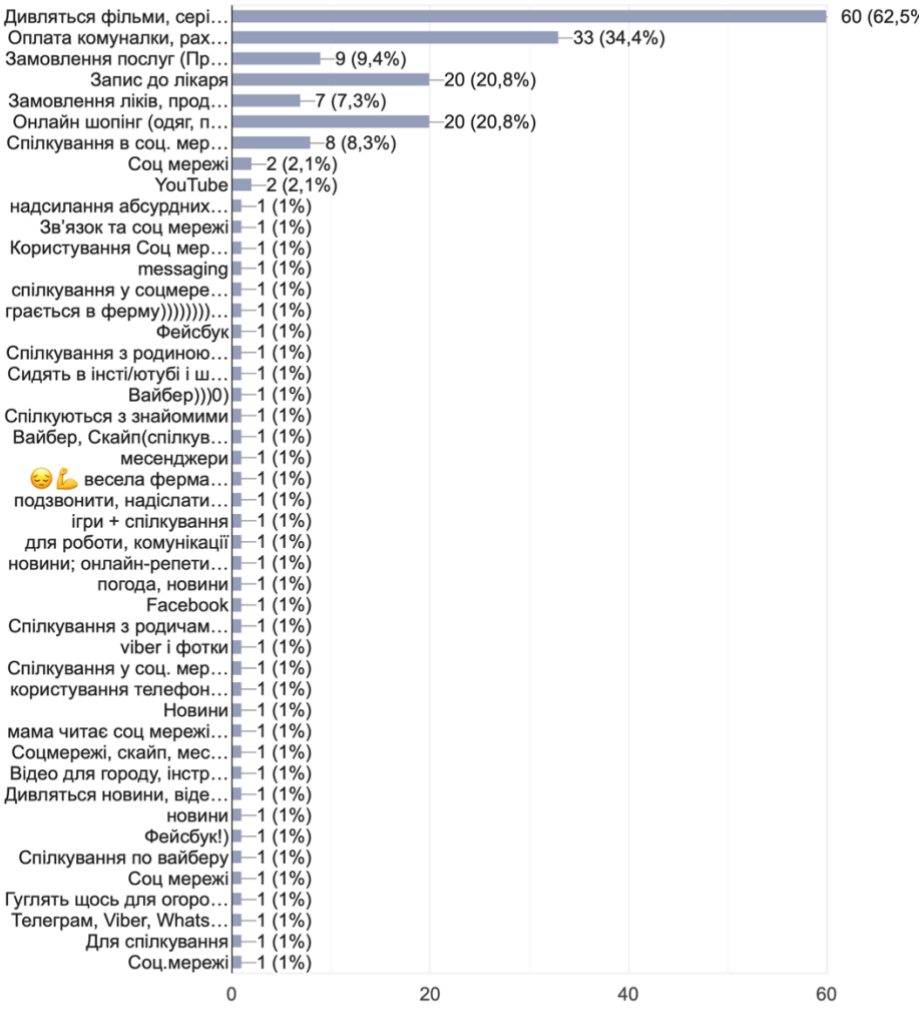
Чи хоч хтось з Ваших родичів похилого віку жалівся Вам, що не знає/не розуміє як взаємодіяти зі своїм ґаджетом ?
96 відповідей



Чи допомагаєте Ви їм з планшетом/смартфоном при виникненні таких непорозумінь ? Консультуєте при виникненні питань на цю тему ? І якщо так, то чи подобається це Вам?
96 відповідей

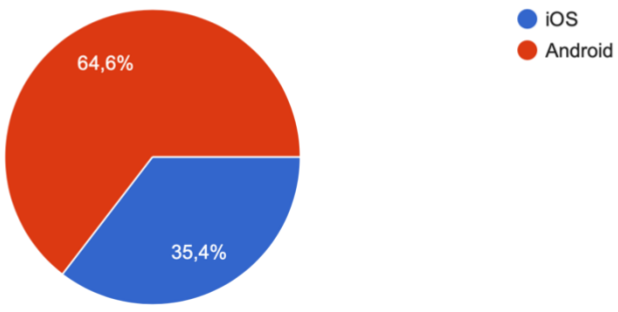
Для чого найчастіше Ваші літні родичі використовують інтернет у своїх ґаджетах?

96 відповідей



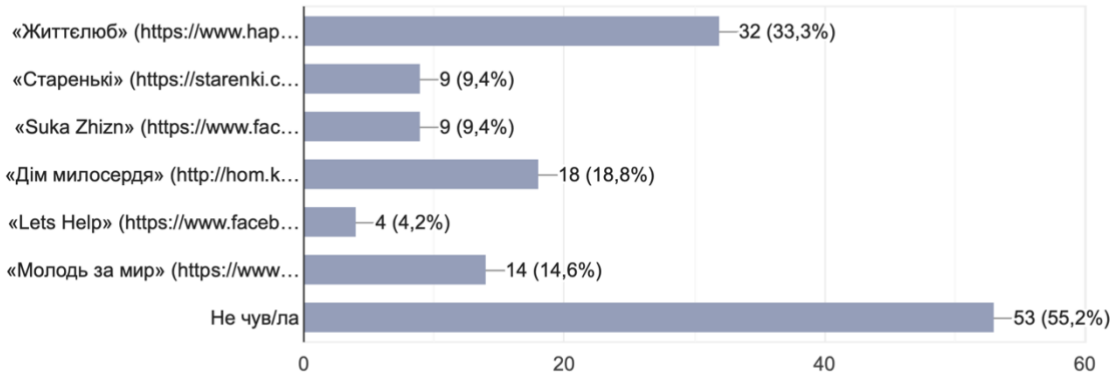
На Вашу думку смартфоном/планшетом на базі якої ОС літнім людям користуватися легше (більш інтуїтивно, зрозуміліше) ?

96 відповідей



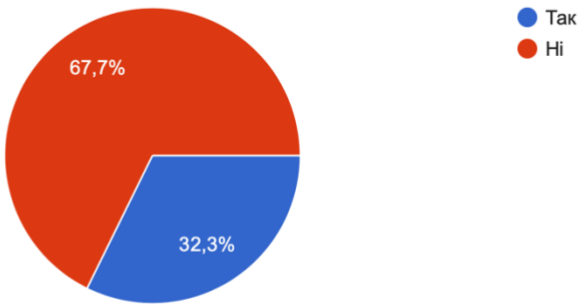
Про яку з перелічених громадських організацій ви чули/знаєте ?

96 відповідей



Чи довірили б Ви волонтеру з перевіреної громадської організації догляд за своїм літнім родичем/родичкою? Оплату комуналки, рахункі...нання будь-яких замовлених ним/нею послуг ?

96 відповідей



Додаток Г. Посилання на макет проєкту в Figma

https://www.figma.com/file/U2bCxntyXj50YZsIMQAlHM/Golden-Ager?node-id=0%3A1

Додаток Ґ. Робота з mongoose на серверній частині в Node.js

DB_URL – лінк за яким відбувається підключення до бази

```
 const mongoose = require('mongoose');

mongoose.connect(process.env.DB_URL,{
  useNewUrlParser: true,
  useUnifiedTopology: true

}).then(()=>{console.log("Successfully connected to DB")}).catch (console.error);
```

Додаток Д. Посилання на хостинг серверу на Heroku; посилання для підключення віддаленої БД Mongo DB

- Heroku: https://golden-ager.herokuapp.com/
- MongoDB: mongodb+srv://sofia:12345@goldenagercluster.htncz.mongodb.net/goldenAger?retryWrites=true&w=majority

Додаток Е. Код схем моделей сутностей з БД на back-end частині

```
const categorySchema = new mongoose.Schema( definition: {

    title: {type: String, required: true},

});
```

```
const taskSchema = new mongoose.Schema( definition: {

    title: {type: String, required: true},
    description: {type: String, required: true},
    category_id: {type: ObjectId, required: true, default: "60983edb7cb3785fc858cc91"},
    time: {type: Timestamp, required: true},

    volunteer_id: {type: ObjectId, required: true},
    pensioner_id: {type: ObjectId, required: true},
    task_is_done: {type: Boolean, required: true, default: false}

});
```

```
const userSchema = new mongoose.Schema( definition: {

    is_volunteer: {type: Boolean, required: true, default: true},
    phone: {type: String, required: true, unique: true},
    password: {type: String, required: true},
    first_name: {type: String, required: true},
    last_name: {type: String, required: true},
    birthday: {type: Date, required: true},
    organization: {type: String, required: false},
    is_free: {type: Boolean, required: false, default:null}

});
```

```
const taskTemplatesSchema = new mongoose.Schema( definition: {

    title: {type: String, required: true},
    description: {type: String, required: true},
    category_id: {type: ObjectId, required: true, default: "60983edb7cb3785fc858cc91"},
    time: {type: Timestamp, required: true},

});
```

Додаток Є. Зв’язок роутерів в фалі app.js, робота з bcrypt (хешування паролів користувачів), автентифікація (логін), реалізація авторизації в файлі middleWares.js

1. Зв’язок роутерів в фалі app.js

```
const authRouter = require('./routes/authRouter');


const userRouter = require('./routes/crud/user');
const categoryRouter = require('./routes/crud/category');
const taskRouter = require('./routes/crud/task');
const taskTemplateRouter = require('./routes/crud/task-templates');


const app = express();


app.use('/auth', authRouter);

app.use('/users', userRouter);
app.use('/task', taskRouter);
app.use('/category', categoryRouter);
app.use('/template', taskTemplateRouter);
```

2. Хешування bcrypt та реєстрація нового користувача

```
const bcrypt = require('bcrypt');
const saltRounds = 12;

async function registration(req, res) {
   try {
      const errors = validationResult(req)
      if (!errors.isEmpty()) {
         return res.status(400).json({message: "Помилка при реєстрації", errors})
      }

      const {
         is_volunteer,
         phone,
         password,
         first_name,
         last_name,
         birthday,
         organization
      } = req.body

      const candidate = await User.findOne({phone})
      if (candidate) {
         return res.status(400).json({message: "Користувач з таким номером телефону уже існує"})
      }

      const salt = bcrypt.genSaltSync(saltRounds);
      const hashPassword = bcrypt.hashSync(password, salt);
```

```
        const user = new User({
            is_volunteer,
            phone,
            password: hashPassword,
            first_name,
            last_name,
            birthday,
            organization

        })

        if (user.is_volunteer) {
            user.is_free = true
            if (!user.organization) {
                return res.status(400).json({message: `Волонтер ${user.first_name} ${user.last_name} не належить до жодної організації`})
            }
        } else {
            user.is_free = null
            user.organization = null
        }

        await user.save()
        return res.json({message: "Користувач був успішно зареєстрований"})

    } catch (e) {
        console.log(e)
        res.status(400).json({message: "Registration error"})

    }
}
```

3. Автентифікація

```
const jwt = require('jsonwebtoken')

async function login(req, res) {

    try {

        const {
            phone,
            password,
        } = req.body

        const user = await User.findOne({phone})
        if (!user) {
            return res.status(400).json({message: `Користувача з номером телефону ${phone} не знайдено`})
        }

        const validPassword = bcrypt.compareSync(password, user.password)
        if (!validPassword) {
            return res.status(400).json({message: `Невірний пароль`})
        }
        const token = generateAccessToken(user._id, user.is_volunteer)
        return res.json({token})

    } catch (e) {
        console.log(e)
        res.status(400).json({message: "Login error"})
```

```
    }
}


const generateAccessToken = (id, is_volunteer) => {
    const payload = {
        id,
        is_volunteer
    }

    return jwt.sign(payload, process.env.SECRET_KEY, {expiresIn: "24h"});
}
```

4. Авторизація користувача, файл middleWare.js

```
const jwt = require('jsonwebtoken')

function authMiddleWare(req, res, next) {
    if (req.method === "OPTIONS") {
        next()
    }

    try {
        const token = req.headers.authorization.split(' ')[1]
        if (!token) {
            return res.status(403).json({message: "Користувач не аутентифікований"})
        }

        const decodedToken = jwt.verify(token, process.env.SECRET_KEY)
        req.user = decodedToken
        console.log("decoded token:", decodedToken)
        next()

    } catch (e) {
        console.log(e)
        return res.status(403).json({message: "Користувач не аутентифікований"})
    }
}

function roleMiddleWare(is_volunteer) {
    return function (req, res, next) {
        if (is_volunteer !== req.user.is_volunteer) {
            return res.status(403).json({message: "У вас немає доступу"})
        }
        next()
    }
}
module.exports = {
    authMiddleWare, checkPensioner: roleMiddleWare(false), checkVolunteer: roleMiddleWare(true)
}
```

Додаток Ж. Структура основних файлів на серверній back-end частині Node.js Express

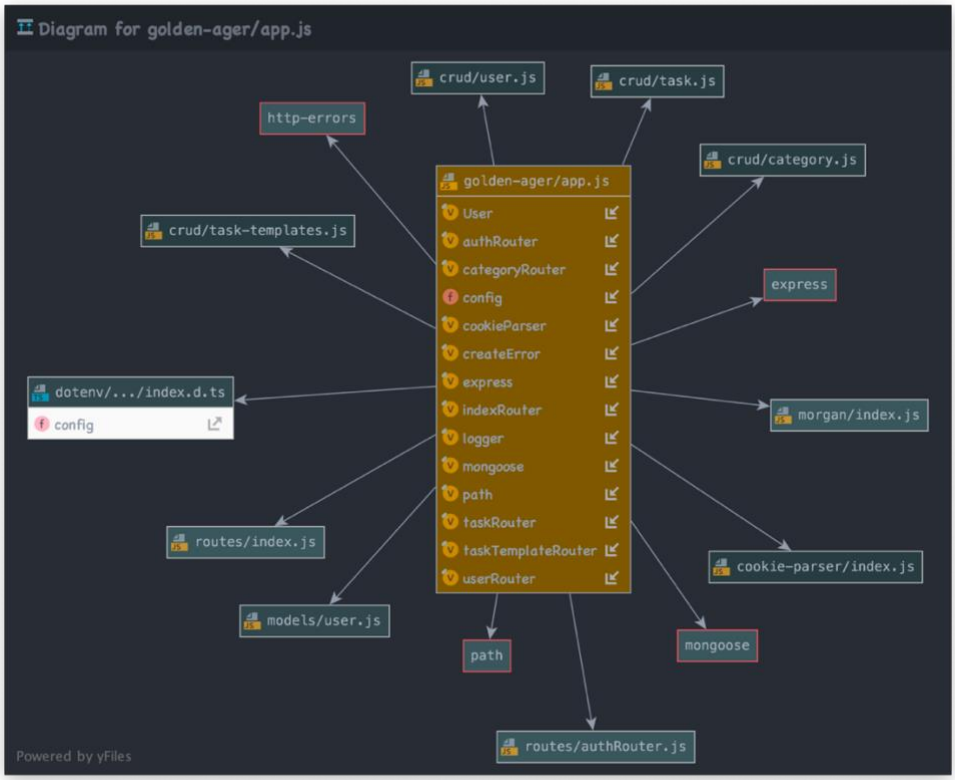


Рисунок 1 Структура файлу app.js

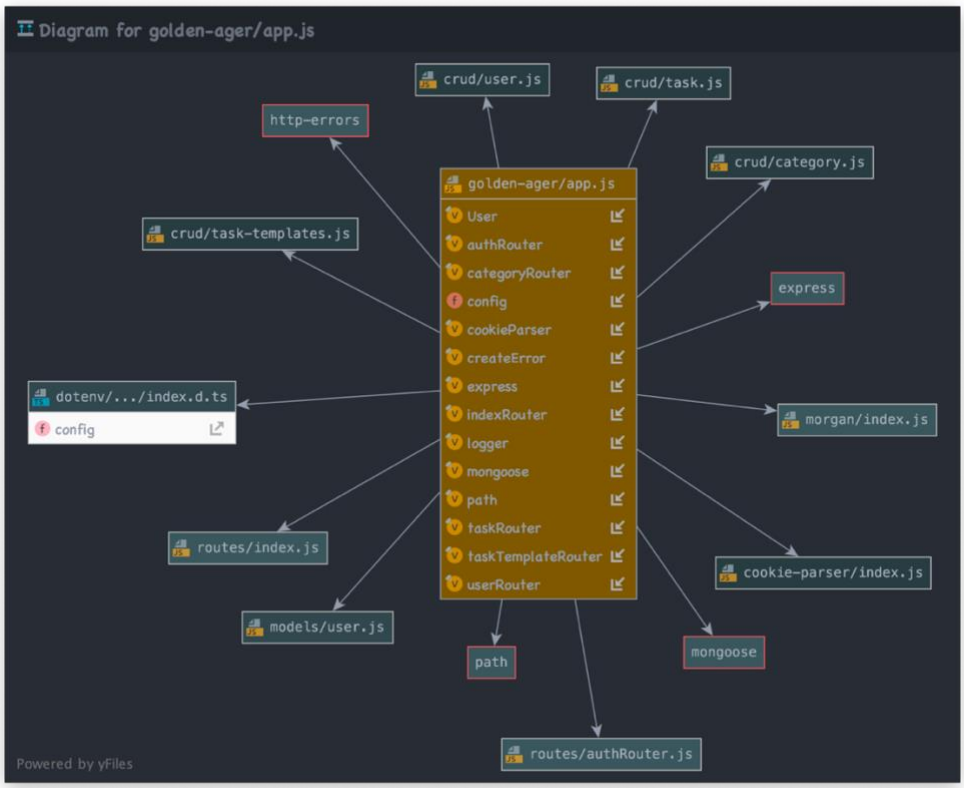


Рисунок 2 Структура папки routes

Додаток З. Посилання на репозиторій проєкту на GitHub

1. Front-end: https://github.com/SofiXeno/goldenAger
2. Back-end: https://github.com/SofiXeno/golden-ager

Додаток К. Код Podfile необхідний для запуску застосунку

```
# Uncomment the next line to define a global platform for your project
# platform :ios, '9.0'

target 'GoldenAger' do
  # Comment the next line if you don't want to use dynamic frameworks
  use_frameworks!

  # Pods for GoldenAger

  target 'GoldenAgerTests' do
    inherit! :search_paths
    # Pods for testing
  end

  target 'GoldenAgerUITests' do
    # Pods for testing
  end

pod 'HideShowPasswordTextField', :git => 'https://github.com/Guidebook/HideShowPasswordTextField'
pod 'Alamofire', '~> 5.2'
pod 'SkeletonView'
pod 'NVActivityIndicatorView'
pod ''ViewAnimator''
pod 'CSS3ColorsSwift'
pod 'Hue'
pod 'Moya', '~> 14.0'

end
```