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

Розробка дизайну та прототипу iOS-застосунку «KMAscheduler»

Текстова частина до курсової роботи
за спеціальністю «Інженерія програмного забезпечення» - 121

Керівниця курсової роботи
Бітаєва О. В.
_____
(Підпис)
“___” _____ 2024 року
Виконала студентка БП ІПЗ-3
Грисюк А. О.
“___” _____ 2024 року

Київ 2024

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

ЗАТВЕРДЖУЮ
Викладач кафедри інформатики ,
канд. фіз-мат. наук, доц. _____ Гороховський С.С.
(підпис)
„____”_____2024р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу
студентці Грисюк Анастасії Олександрівні
факультету інформатики 3 курсу бакалаврської програми
ТЕМА: Розробка дизайну та прототипу iOS-застосунку «KMAscheduler»

Зміст ТЧ до курсової роботи:

Індивідуальне завдання
Вступ
Теоретичні основи
Теоретичні відомості (про задачу, методи й підходи до її розв’язку тощо)
Розробка та опис реалізації програмного продукту
Висновки
Список використаних джерел
Додатки

Дата видачі „___” _____ 2023 р. Керівниця _____
(підпис)

Завдання отримала _____
(підпис)

Тема: **Розробка дизайну та прототипу iOS-застосунку «KMAscheduler»**

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	23.10.2023	
2.	Огляд літератури за темою роботи.	23.11.2023	
3.	Проведення досліджень.	24.11.2023-24.01.20234	
3.	Опис результатів дослідженн	28.02.2024	
4.	Аналіз отриманих результатів з керівницею.	10.03.2024	
6.	Корегування роботи.	10.04.2024	
7.	Створення презентації та написання доповіді.	10.05.2024	
8.	Остаточне оформлення пояснювальної роботи та слайдів.	10.05.2024	
9.	Захист курсової роботи.	23.05.2024	

Студентка Грисюк А. О.

Керівниця _____

“_____”_____

Зміст

Table des matières

Анотація

У цій роботі досліджено засоби створення дизайну та прототипу iOS-застосунку для зручного перегляду розкладу занять в університеті.

У роботі розглянуто процес створення UI та UX дизайну та самого застосунку. Також розглянуто проблеми, що виникають у ході розробки застосунку та їхні рішення. Готовий застосунок складається з двох основних екранів: "Розклад" та "Налаштування" для зручного вибору потрібних дисциплін.

Вступ

Станом на 2024 рік операційною системою iOS користується більше ніж 1.46 мільярда осіб. iPhone має 28,46% ринку у світовій індустрії смартфонів станом на 2024 рік. За останнє десятиріччя кількість користувачів iPhone збільшилася на 230.31% [7]. Продажі iPhone зростають разом із прихильністю користувачів до операційної системи. Станом на кінець 2023 року в AppStore нараховується 2.29 мільйони застосунків [8].

Навчання в університеті є водночас цікавим і потрібним, проте може також вимагати від студентів та студенток чимало докладених зусиль для виконання всіх необхідних завдань вчасно. Громіздкий графік пар нерідко стає справжнім викликом для тих, хто навчається. Вимоги відвідування лекцій, семінарів та практичних занять існують майже на всіх спеціальностях, адже заняття з викладачами й викладачками є одним із найефективніших способів навчання. Кожне покоління бере за основу вже наявний рівень розвитку, саме тому так важливо переймати досвід старших поколінь.

З плином часу дедалі більше студентів та студенток потребують якіснішого планування свого часу. Ледь не кожна людина хоч раз у житті стикається з проблемами, що постають внаслідок завеликої кількості обов’язків і планів та відсутности явної системи тайм-менеджменту. Зокрема виникають наступні негаразди: забування про важливі справи, неякісно виконані завдання, втрачені можливості. Усе це поступово веде до погіршення емоційного та фізичного стану: постійний стрес чинить руйнівний вплив на здоров’я людини.

Для реалізації проєкту було використано сучасні інструменти розробки під iOS. Рекомендації щодо написання коду та використання кожного інструменту було взято з документацій, наукових статей тощо.

Робота складається з трьох окремих розділів. Кожен розділ містить підрозділи задля зручнішого сприйняття інформації.

Перший розділ, «Дослідження та аналіз предметної області. Формулювання завдання», містить усю інформацію про актуальність та конкурентів роботи. У ньому також зазначена постановка задачі.

У другому розділі під назвою «Аналіз доступних інструментів» описано сучасні засоби створення дизайну та застосунків під iOS, їхні переваги й недоліки, а також порівняння в контексті використання в цій роботі.

Третій розділ, «Розробка та опис реалізації програмного продукту», повністю описує процес створення застосунку від початкового дизайну до фінальних покращень коду.

Основна мета курсової роботи — дослідити інструменти створення застосунку під платформу iOS для зручного перегляду розкладу занять в Академії, що дозволить легко й швидко отримувати інформацію про зайнятість студента чи студентки в конкретний день/тиждень.

РОЗДІЛ 1: Теоретичні основи

1.1. Актуальність та аналіз предметної області

Нині ґаджети є невіддільною складовою життя майже кожної людини. Якщо раніше телефони створювали виключно з метою забезпечити стільниковий зв’язок, аби мати змогу тримати контакт навіть на відстані в тисячі кілометрів, то зараз до цієї мети додався незліченний ряд людських потреб, які можна імплементувати в досить компактний, проте потужний пристрій. На відміну від комп’ютерів, ноутбуків чи інших громіздких та важких пристроїв, потужність та пам’ять смартфонів значно менша, на що варто зважати при проєктуванні та розробці мобільного застосунку.

У сучасному світі існує потреба в постійному розвитку, аби залишатися потрібним/-ою як працівник/-ця та бути цінним кадром. Протягом останніх кількох років у мережі дедалі популярнішими стають прагнення саморозвиватися та будувати стрімку кар’єру ще з юних літ, аби мати впевненість у майбутньому.

Дедалі більше компаній починають розглядати співробітників не лише як тих, хто виконує ту чи іншу роботу за будь-які кошти. Поступово бізнеси вибудовують щиру та свідому спільноту всередині, адже оточення колег суттєво впливає на становлення людини та її мрії, цілі та можливості. Компанії намагаються привабити до себе найкращих із найкращих: що більше переваг буде в компанії, то більша ймовірність того, що людина з високим рівнем знань і навичок обере саме їхній бізнес як місце для роботи. Така людина буде неповторною у своїй сфері. Вона допомагатиме бізнесу заробляти ще більше коштів, залучати інвесторів та отримувати позитивні відгуки клієнтів. Надзвичайно важливо вибудовувати спільноту, учасники якої прагнутимуть розвиватись та розвивати одне одного.

Компанія може досягти цього завдяки відбору людей не лише за hard skills, а й за soft skills. Їх можна перелічувати годинами, втім одним із ключових є саме тайм-менеджмент.

Тайм-менеджмент — вміння керувати власним часом та розподіляти власні плани так, аби виконувати все вчасно та якісно. Навички тайм-менеджменту часто є критичними та вирішальними як у професійній, так і в особистій діяльності.

Споконвіку люди намагаються отримати якомога більший контроль над власним життям. Таке прагнення зумовлене відчуттям відповідальности за власне життя та усвідомленням того, що кожна людина сама в змозі обирати своє майбутнє.

Найбільший брак самоконтролю приходить разом із надмірною кількістю обов'язків, що їх важко структуровано тримати в пам'яті. Одним із прикладів таких випадків є навчання в університеті — періоду становлення людини як особистости, який є чи не найбільш насиченим усілякими подіями. Саме тому для студентів та студенток так важливо набути навичок опанування свого часу.

Розглянувши систему навчання в Києво-Могилянській Академії, можна зробити наступні висновки:

1. Усі студенти мають різний розклад занять, формати яких також різняться. Це говорить про важливість кастомізації під потреби кожної людини. Застосунок повинен бути гнучким для інформації, підлаштовуючись під індивідуальні потреби.

2. Потреба в автоматизації. Передбачається, що розклад занять повинен передаватися ззовні від адміністрації університету. Робота з вхідними даними в коді повинна бути чітко відділена від інших частин програми, аби мати змогу отримувати дані не лише з вказаного у цій роботі формату даних.

3. Можливість оновлення даних. Розклад занять може змінюватися в перші тижні навчання через непередбачувані обставини. Саме тому надзвичайно важливою є коректна робота з базами даних. Дані не повинні дублюватися при повторному фетчингу.

Актуальність цієї роботи є очевидною в контексті сучасних потреб в ефективності та продуктивності суспільства. Студенти та студентки постійно мають справу з великим навантаженням, що нерідко призводить до стресу, емоційного вигорання та інших серйозних проблем, довга тривалість яких має надзвичайно деструктивний вплив на подальше життя. Саме застосунок для відстеження розкладу занять стає корисним інструментом, який дає студентам змогу слідкувати за своєю успішністю в університеті.

1.2. Аналіз наявних рішень та конкурентів

Аби створити застосунок, який матиме популярність, необхідно враховувати не лише наявні потреби користувачів/-ок, а й попередній досвід інших продуктів на ідентичну тематику. Цього можна досягти завдяки аналізу ринку й знаходження та дослідження конкурентів. Це дає змогу уникнути помилок, що їх припустились інші розробники та врахувати хороші практики, адже що більше думок та поглядів буде розглянуто, то вищою є ймовірність створення ідеального продукту, використання якого викликатиме лише приємні емоції. При аналізі наявних застосунків на цю тематику особливий наголос було зроблено на зручності планування саме навчального процесу.

Наразі існує лише один повноцінний аналог, створений студентами й студентками Києво-Могилянської Академії. Йдеться про Telegram-бот, що має назву KMAScheduler.

Розгляньмо його основні переваги:

- **Зручність обраного формату та платформи.** Telegram широко використовується студентами й студентками як в особистих, так і в професійних цілях. Завдяки цьому розклад майже завжди знаходиться «під рукою».
- **Зручний для користувача/-ки інтерфейс перегляду.** Серед них можливості переглянути наступну пару, пари на сьогодні, пари на завтра, пари на обраний день тижня, пари на весь тиждень та ще чимало функцій.

Попри популярність бота, він містить певні недоліки:

- **Ручний парсинг розкладу.** Розробники повинні щотриместра самостійно передавати всі дані в бот. Це є незручним та часто може ставати причиною помилок у розкладі. Зважаючи на те, що бот необхідний для навчання, такі помилки призводять до втрати балів, якщо, до прикладу, інформація про заняття не буде відображатися в студента/-ки через внутрішню помилку розробників/-иць.
- **Не зовсім зручний інтерфейс для введення даних.** Цей процес може зайняти чимало часу в разі наявності значної кількості вибіркових предметів. Для кожного предмета потрібно окремими повідомленнями вказувати назву й групу, після чого повертатись до попереднього пункту та повторити дію для кожної дисципліни. Це займає певний час та може бути більш оптимізованим.

Основна відмінність розробленого застосунку від Telegram-бота полягає в

тому, що передбачається, що розклад буде переданий як вхідний файл, аби уникнути помилок із боку розробників.

Окрім описаного вище проєкту також було детально розібрано застосунки на тематику тайм-менеджменту: Google Calendar та Notion Calendar.

Google Calendar — безплатний вебзастосунок, розроблений компанією Google у 2006 році. Його рейтинг в App Store — 4.5. Детально розглянувши відгуки в App Store, можна виокремити такі зауваження від користувачів/-ок:

- **Необхідність реєстрації.** Існують користувачі, що надто турбуються про свою приватність. Безумовно важливою є змога зберігати дані та передавати їх автоматично між пристроями, проте в цьому не завжди виникає потреба.
- **Неінтуїтивний інтерфейс.** Не маючи досвіду роботи з онлайн-календарями, користувачам може бути складно почати роботу з Google Calendar.

Завдяки цьому застосунку було вирішено додати функцію нотаток до кожного заняття, адже Google Calendar відомий широким вибором інструментів для додавання інформації про конкретну подію.

Notion Calendar — новий застосунок, що його побачив світ на початку 2023 року. Його рейтинг в App Store 3.9, що є хорошим результатом, зважаючи на те, що продукт знаходиться на стадії початку свого розвитку. Найбільше його користувачі/-ки скаржаться на такі недоліки: вузький вибір кольорів та можливість створювати задачі лише в 15-хвилинних інтервалах, що значно спантеличувало б цільову авдиторію застосунку, створеного як результат дослідження для цієї курсової, адже в Академії проводять заняття, що починаються, наприклад, об 11:40 чи закінчуються о 19:20. Клієнтам помітно імпонує загальна система дизайну, що відповідає зовнішньому вигляду застосунку Notion та є давно знайомим для тих, хто вже мав із ним справу. Завдяки цим відгукам було прийнято рішення обрати дизайн-систему, що вже відома студентам та студенткам Академії, а саме Скорочене керівництво з

використання елементів візуальної комунікації НаУКМА. Це сприяє уніфікуванню візуального стилю всіх сервісів університету.

Як було сказано вище, ключова відмінність цієї роботи від описаних раніше застосунків полягає у вузькій авдиторії клієнтів. Саме тому застосунок було адаптовано під їхні потреби.

У такий спосіб вдалося створити застосунок, що буде золотою серединою між вже наявними сервісами для покращення тайм-менеджменту. Детальна аналітика відгуків реальних осіб дала змогу поглянути на застосунки з їхньої перспективи та втілити їхні побажання в реальність.

1.3. Формулювання завдання курсової роботи

Основною ціллю цієї курсової роботи є створення прототипу застосунку KMAscheduler для перегляду індивідуального розкладу занять.

Додаткові задачі:

- Аналіз ринку та наявних на ньому застосунків та інших інструментів для створення розкладу.
- Виокремлення переваг та недоліків обраних проєктів.
- Створення правильного та зручного UI та UX дизайну відповідно до офіційного стилю НаУКМА.

У застосунку повинні бути втілені наступні можливості:

1. Вибір дня за допомогою календаря та перегляд заняття на обраний день.
2. Створення нотаток до конкретного заняття.
3. Вибір спеціальности.
4. Вибір року навчання
5. Вибір професійно-орієнтованих дисциплін та дисциплін вільного вибору.
6. Вибір груп для всіх дисциплін.

7. Отримання розкладу з даних, що передаються ззовні в застосунок.

8. фільтрація предметів та занять.

9. Коректне оновлення екранів після зміни даних користувачем/-кою.

10. Перегляд місця розташування корпусу, в якому відбуватиметься заняття на карті в реальному часі.

Додаткові можливості, що необхідні для плавної та злагодженої роботи застосунку:

1. Адаптивний інтерфейс за допомогою AutoLayout, що матиме однаковий правильний вигляд на всіх пристроях незалежно від їхніх розмірів.
2. Колірна палітра та шрифтова гарнітура відповідно до Гайдлайну з візуальної комунікації НаУКМА.
3. Відповідність Human Interface Guideline.
4. Уникнення надмірного споживання ресурсів ґаджета.

Отже, для створення ідеального застосунку необхідно дослідити можливості наявних інструментів та платформи iOS.

Також важливим кроком є вибір архітектурного шаблону та інших патернів, що будуть втілені в застосунку для чистішого коду.

Останній крок дослідження — вибір фреймворків для користувацького інтерфейсу, роботи з базами даних, мапами та засобу перевірки коду на відповідність загальноприйнятому спільнотою стилю його написання.

1.4. Висновки до розділу 1

У першому розділі йдеться про актуальність теми розробки застосунку під платформу iOS для перегляду розкладу дисциплін в НаУКМА.

З метою деталізації вимог до застосунку було проаналізовано ринок наявних сервісів для планування. Цей етап дав змогу побачити погляд користувачів/-ок на наявні продукти, вдатися до найкращих практик, які

заслужили на похвалу та уникнути негараздів, з якими стикалися люди при роботі з застосунком.

Останній крок, описаний у цьому розділі, — чітке формулювання завдання за пунктами, аби мати готовий план роботи та врахувати всі раніше згадані особливості.

Основний висновок, виведений з цього розділу, полягає в тому, що застосунок повинен відрізнятися від інших наявних продуктів функціоналом, що підходить саме для студентів та студенток Києво-Могилянської Академії, його цільової авдиторії.

РОЗДІЛ 2. Теоретичні відомості (про задачу, методи й підходи до її розв’язку тощо)

2.1. Основні поняття про розробку застосунків під iOS

Задля забезепечення приємного користувацького досвіду існує набір рекомендацій, стандартів та правил побудови UI та UX застосунків для пристроїв від Apple (iPhone, iPad, MacBook тощо), що визначають, як користувацький інтерфейс повинен функціонувати для максимальної зручности й ефективности. Вони носять загальну назву Human Interface Guidelines (скорочено — HIG). Їхнє існування зумовлене значною кількістю потреб, а саме:

- **Забезпечення узгодженості застосунків одне з одним та наявности спільних рис.** Така спільність у користувацькому досвіді сприяє полегшенню переходу від одного продукту Apple до іншого, що, власне, і є основною рисою якісної екосистеми.
- **Правильна взаємодія з користувачем.** Настанови містять найкращі практики, перевірені тестуванням на мільйонах осіб. Таким способом можна створити інтуїтивно зрозумілі інтерфейси.
- **Висока якість продуктів.** Використання HIG гарантує уникнення багатьох помилок та недоліків, із якими часто стикаються молодші розробники через брак досвіду та загального розуміння процесу написання застосунку.
- **Інклюзивність та доступність.** До набору правил також входять вимоги, завдяки яким застосунком можуть користуватися також особи з інвалідністю, порушенням слуху чи зору, нейродивергентністю, діти тощо. До прикладу, варто уникати

ейблізмів та повідомляти про сенситивний контент чи контент для дорослих.

- **Відсутність дискримінації та ненависті.** Apple не підтримує жодного виду неповаги до будь-якої категорії населення. Рівність та толерантність — основні ознаки освіченого суспільства.
- **Приватність.** Варто просити доступ до даних користувача лише в разі дійсної потреби, а також чітко повідомляти, в яких цілях може бути використана отримана інформація.
- **Метафори в інтерфейсі.** Іконки, що мають вигляд предметів, що існують у реальності, дозволяють користувачеві моментально проводити аналогію й розуміти суть та функцію кожного елемента. До прикладу, символ збільшувальної лупи уособлює пошук, у той час як значок батарейки в дизайні є уособленням рівня заряду акумулятора.

2.2. Патерн MVC

MVC — абревіатура для назви Model-View-Controller, одного з найчастіше використовуваних архітектурних патернів у розробці під iOS.

MVC розділяє бізнес-логіку й операції застосунку від презентаційної логіки, при цьому використовуючи Controller як посередник для спілкування між View та Model.

Суть патерна полягає в тому, аби розділити застосунок на три основні логічні компоненти, кожен із яких повинен обробляти специфічні аспекти застосунку:

- **Model** відповідальна за зберігання даних, проте не повинна маніпулювати ними для потреб View, оскільки вона жодним чином не є обізнаною про View.

- **View** має дві відповідальності: піклування про передачу даних користувачеві і отримання дій, учинених користувачем. Controller не має уявлення про наявні Model'i, вона оновлюється обробленими даними завдяки своєму власникові.
- **Controller** володіє View та Model та комунікує з ними. Він відповідальний за їхнє оновлення обробленими даними. Controller зазвичай є делегатом для View, що дає змогу View передавати дані та інформацію.

Розробники під iOS обирають MVC для своїх застосунків із кількох причин:

- Чіткий розподіл обов’язків.
- Простота використання.
- Перевикористання (Reuse).
- Чимала кількість інформації та прикладів використання.

Аналогом MVC є MVVM (Model-View-ViewModel) — патерн, сенс якого полягає в розділенні графічного користувацького інтерфейсу від бізнес- та бекенд-логіки.

MVVM є новішим, ніж MVC, проте щороку кількість проєктів, що використовують його, зростає. Його основною перевагою є можливість тестування ViewModel та Model без використання View.

Патерн чудово підходить для складних проєктів, проте для невеликих продуктів він може бути занадто комплексним. Зважаючи на це основним архітектурним патерном було обрано MVC.

2.3. Мова програмування Swift

Swift — основна мова програмування під операційну систему iOS на сьогодні. Компанія Apple презентувала Swift у 2014 році як сучасну, швидку та інноваційну мову.

Переваги:

- **Синтаксис та читабельність.** Swift має простий та лаконічний синтаксис, створений для того, аби код був більше читабельним та легшим для розуміння. Такий синтаксис є доступнішим зокрема для розробників-початківців.
- **Продуктивність і швидкість.** Swift використовує сучасні компілятор та runtime, завдяки чому використовується значно менше ресурсів. У Swift також є такі особливості, як опційне зв’язування та 4 способи розгортання опційних значень, що сприяє розробці ефективного та безпечного коду.

Другою мовою для розробки застосунків під iOS є Objective-C. Створена на початку 1980-х, вона є розширенням мови C. Оскільки Objective-C існує досить давно, вона має велику базу застарілого коду. Через це в разі роботи над додатком, що існує давно, ця мова може бути кращим вибором.

Станом на сьогодні Objective-C має чимало недоліків, через яку вибір розробників часто падає саме на Swift.

Основні проблеми:

- Складність у вивченні.
- Брак сучасних особливостей та конструкцій.
- Комплексний синтаксис.

З огляду на наведені вище причини для розробки застосунку було обрано Swift.

2.4. Core Data: фреймворк для керування базами даних

Core Data — спільний фреймворк для всіх OS від Apple для зручної та ефективної роботи з базами даних. Це граф об’єктів і система збереження даних, що в класичному випадку працює на базі SQLite. Підтримує також XML(macOS), бінарний запис та in-memory (в оперативній пам’яті). Іншими словами Core Data це прошарок між програмою та базою даних, що дає змогу працювати з нею, використовуючи зручні абстракції.

Переваги:

- Інструменти для роботи з транзакціями.
- Можна робити операції undo/redo.
- Робота у фоні і досить легка інтеграція з контейнерами iCloud для синхронізації.
- Зручна інтеграція в XCode.
- Відсутність потреби в підключенні до мережі.

З альтернатив існують: Realm, Firebase, SwiftData. Втім для цієї роботи найбільше підходить саме CoreData.

Realm — одна з найновіших реактивних баз даних із хорошою швидкодією. Найбільший її мінус полягає в обмеженості підтримки сторонніх інструментів та бібліотек.

Firebase — хмарна NoSQL база даних. Дані зберігаютьсь у JSON-форматі. Втім, для повноцінної роботи вона потребує стабільного зв’язку з мережею, а також використання додаткових функцій оплачується додатково. Найбільшим недоліком є обмежені можливості запитів, тобто, робота зі складними запитами та операціями з даними не завжди буде доступною.

SwiftData — новий фреймворк від Apple. Основним недоліком SwiftData в контексті цієї роботи є вимога використання SwiftUI, оскільки замість нього для основної частини проєкту було використано UIKit.

Тож вибір CoreData для цього проєкту є зваженим та обґрунтованим. Завдяки роботі на нативному рівні робота з ним є швидкою та ефективною.

2.5. UIKit: фреймворк для створення складного користувацького інтерфейсу

UIKit — фреймворк для побудови графічного користувацього інтерфейсу під iOS, iPadOS та tvOS.

Переваги:

- Підтримка якісної та привабливої анімації.
- Наявність можливостей малювання.
- Широкі можливості кастомізації інтерфейсу.
- Підтримка більшости фреймворків від Apple.
- Зручна та легка робота з розміткою інтерфейсу.
- Адаптивність для різних пристроїв.

Альтернатива — SwiftUI, відносно новий фреймворк для декларативної побудови користувацького інтерфейсу. Його було використано для відображення певних частин застосунку, що були простими в реалізації та не вимагали особливих налаштувань. Основна частина проєкту написана на UIKit через необхідність кастомізації інтерфейсу.

2.6. SwiftUI: Фреймворк для декларативної побудови UI простих екранів

SwiftUI — фреймворк для кросплатформеної побудови користувацького інтерфейсу.

Переваги:

- Декларативний синтаксис.
- Автоматичне оновлення інтерфейсу.
- Наявність передогляду застосунку.

- Хороше поєднання з іншими технологіями від Apple.

У цій роботі SwiftUI було використано для роботи з мапами, на яких відображається корпус НаУКМА, у якому відбудеться заняття.

2.7. Figma: застосунок для роботи з віжуалами

Figma — сучасний інструмент для створення дизайнів.

Переваги:

- Зручний інтерфейс.
- Безлімітна кількість файлів у безкоштовній версії.
- Наявність гарячих клавіш, що суттєво пришвидшує роботу.
- Можливість зберігати файли в різних форматах.

Існують такі альтернативи: Canva, Adobe Illustrator.

Canva — хороший варіант для створення презентації та нескладних односторінкових дизайнів (логотипів, банерів тощо)

Illustrator — найпопулярніший інструмент серед дизайнерів, втім має суттєвий недолік — висока вартість підписки на пакет Adobe. Також інструмент не зовсім зручний через обмеженість розмірів фрейму.

Figma ж найбільш пристосована саме для вебдизайну та UI/UX дизайну завдяки можливості створювати інтерактивні прототипи та будь-яку кількість екранів в одному файлі.

У такий спосіб було обрано Figma, враховуючи її гнучкість та зручність у користуванні.

2.8. MapKit: Фреймворк для роботи з мапами

MapKit — інструмент для відображення мап чи фото зі супутників в застосунку.

Переваги:

- Можливість вставити мапу напряму в будь-яку View.

- Додавання анотацій до мапи.
- Інтерактивність: підтримка жестів збільшення та повороту.

Існують наступні альтернативи:

Google Maps SDK — SDK для інтеграції з iOS та Android. Його основний недолік — залежність від сервісів Google, що може призвести до певних обмежень або ж плати за його використання.

Leaflet — бібліотека JavaScript для створення вебкарт. Має обмежений функціонал порівняно з іншими варіантами, а також деякі функції можуть бути обмежені чи недоступні через обмеження конкретноого браузера.

Отже, MapKit — найбільш вдалий вибір для цієї роботи. Фреймворк вражає своїм функціоналом та зручністю в використанні: він надає змогу інтегрувати інтерактивні мапи в застосунки відповідно до стилю екосистеми Apple.

2.9. SwiftLint: інструмент для перевірки стилю коду

SwiftLint — інструмент для забезпечення дотримання стилю й конвенцій мови програмування Swift. SwiftLint дотримується загальноприйнятих правил, укладених спільнотою Swift. Ці правила описано в популярних посібниках зі стилю, до прикладу, у Swift Style Guide від Kodeco.

Переваги:

- Можливість вимкнути виправлення певних правил стилю.
- Зручна інтеграція з XCode та іншими середовищами розробки.
- Автоматична перевірка коду.

Існує кілька альтернатив для перевірки коректности написання коду:

Swift-format — технологія форматування від Apple. Зручним є його використання як Swift Package Manager dependency. Основний недолік

цієї технології — дещо складніше встановлення та більш комплексна конфігурація.

Tailor — багатоплатформний аналізатор та лінтер коду. Утім він має обмежений набір правила для перевірки стилю коду.

Існують також онлайн-форматери для швидкої компіляції коду, проте такі інструменти виявляють значно меншу кількість помилок.

Зважаючи на інформацію, надану вище, для перевірки коду було обрано SwiftLint.

2.10. Висновки до розділу 2

Основний фреймворк, використаний у роботі, — UIKit, оскільки він дає значно ширші можливості для кастомізації користувацького інтерфейсу. Так вдалося зробити застосунок, що відповідає усім вимогам Гайдлайну з візуальної комунікації НаУКМА.

Для роботи з даними про розклад занять та перелік доступних предметів університету було використано CoreData — сучасний фреймворк для роботи з базою даних.

Останній крок — перевірка коду. Оптимальним вибором є SwiftLint завдяки простоті його використання.

Розділ 3. Розробка та опис реалізації програмного продукту

3.1. Створення UI дизайну

UI дизайн — графічна структура програми. До нього належать всі елементи, які може побачити користувач: кнопки, тексти, зображення, поля для введення тексту тощо. До UI дизайну також належать: макет екрана, переходи між екранами, анімація інтерфейсу та будь-які взаємодії користувача з елементами. Навіть найдрібніші деталі повинні бути враховані на цьому етапі розробки. В загальному цей процес можна назвати створенням зовнішнього вигляду застосунку.

Першим кроком для створення красивого та правильного UI дизайну був вибір загального стилю дизайну: колірної палітри та типографії. Для цієї задачі було використано Скорочене керівництво з використання елементів візуальної комунікації НаУКМА (далі — Гайдлайн).

Основними кольорами було обрано Біла блакить (#DEE8FF) та Академічний синій (#062159) для фону та акцентного відтінку основних елементів відповідно. Для кнопок та покликань було використано Королівський кобальт (#0D52BF), оскільки в користувачів блакитний найчастіше асоціюється з клікабельними елементами. Як додаткові кольори для фонів певних елементів було обрано: Alice blue (#EBF2FF) та Arc Light (#C9D9FF). Для зовнішнього вигляду обраного елементу TabBar було використано Dutch Blue (#4E6391).

Варто звертати увагу на найдрібніші деталі, зокрема на відтінки елементи при різних станах. У елементів застосунку переважно існує два стани: normal та selected. Важливо слідкувати за тим, аби всі відтінки в дизайні відповідали встановленим раніше нормам.

Основною та єдиною шрифтовою гарнітурою було обрано Proba Pro в Medium та Bold нарисах, адже це є прямою вимогою Гайдлайну.

Для більшості текстових елементів було використано Medium нарис. Bold використано для назви спеціальности та заголовків екранів.

Фінальним завданням цього етапу було створення іконки та LaunchScreen застосунку. LaunchScreen — екран, що відображається під час запуску застосунку.

Іконка — перше враження від погляду на застосунок та те, з чим щоразу стикатиметься користувач, коли захоче зайти в застосунок.

Головне в цій задачі — впізнаваність та очевидність. Найбільше з розкладом асоціюється календар, тому саме його було обрано як основний елемент. Одним із обов'язкових елементів є лого КМА. В результаті вийшла лаконічна та красива іконка застосунку.

LaunchScreen — екран, який відображається під час того, як завантажується застосунок. Кожен застосунок повинен мати його, а в разі відсутності ним буде стандартний білий екран. Для LaunchScreen цієї роботи було використано логотип застосунку, що розміщений у центрі екрана.

3.2. Створення UX дизайну

UX дизайн — користувацький досвід, тобто, те, що відчуває користувач, під час взаємодії з інтерфейсом. UX дизайн вирішує розташування кожного елемента у такий спосіб, аби в користувача не виникало питань із приводу його функціоналу. Іншими словами основною задачею UX дизайну є створення user-friendly користувацького інтерфейсу. Це також рівень того, наскільки складно чи просто користувачу буде виконати заплановані дії. Підсумовуючи, вдалий UX дизайн можна назвати запорукою хороших вражень користувача від взаємодії із застосунком.

Початковим кроком до розв’язання цієї проблеми було визначення кількости необхідних вкладок. Було обрано два основні екрани: екран із розкладом та екран з налаштуваннями.

Екран із розкладом повинен містити в собі три основні елементи. Зображення екрану 1 знаходяться в Додатку Ґ.

Перший — заголовок із номером тижня, що обраховується автоматично на основі дати та оновлюється після кожної взаємодії з календарем у разі її зміни.

Другий — елемент для вибору дати, при натиску на який можна побачити інтерактивний календар. Дані повинні оновлюватися після кожного вибору дати.

Останній та найважливіший — таблиця з переліком пар, відсортованих за часом заняття.

Кожна комірка таблиці складається з 3 елементів: назва предмету, його час, корпус Академії і його авдиторія. Натиск на комірку ініціює перехід на додатковий екран (Додаток Д) — детальний розгляд конкретної пари, де можна залишити та прочитати нотатки стосовно певного заняття. Це буде корисним як у повсякденному навчанні, тобто, запис домашнього завдання чи літератури до прочитання, так і для триместрових оцінювань: колоквіумів, самостійних та контрольних робіт, тестів тощо. Комірка заняття, що має нотатку, має також додатковий ідентифікатор, а саме зафарбоване коло праворуч від тексту, що здебільшого асоціюється з непрочитаним повідомленням у месенджерах, завдяки чому майже неможливо візуально пропустити важливу нотатку. Також на екрані є кнопка переходу на перегляд точного розташування корпусу, в якому відбудеться пара (Додаток Д).

Другий екран необхідний для налаштувань: вибору власної спеціальности, року навчання, професійно-орієнтованих предметів та предметів вільного вибору. Він складається з таблиці з 4 комірок: "Моя

спеціальність", "Нормативні предмети", "Професійні дисципліни" та "Дисципліни вільного вибору". Зображення знаходиться в Додатку А.

Додатковий екран вибору спеціальности та року навчання містить напис, який показує назву спеціальності в разі, якщо користувач вже обрав її. У протилежному випадку текстом напису є рядок "Спеціальність не обрано". Приклади всіх можливих виглядів цього екрану знаходяться в Додатку Б.

Під написом із назвою спеціальности знаходиться кнопка, при натиску на яку презентується модальне вікно, де можна обрати спеціальність з переліку наявних. Перелік можна фільтрувати, ввівши одну чи кілька літер, або ж код спеціальности. При натиску на комірку з назвою спеціальности відбувається збереження даних та повернення на попередній екран.

Нижче знаходиться напис з інформацією про рік навчання, а під ним — кнопка, при натиску на яку з’являється поп-ап із цифрами від 1 до 4, що дає змогу швидко обрати необхідний рік навчання.

Для зручности користувачів також було додано автоматичне підвантаження нормативних предметів після вибору спеціальности та року навчання. Таким способом для нормативних предметів користувачу необхідно обрати лише свої групи з дисциплін. Переглянути вигляд екрану з нормативними дисциплінами можна в Додатку В.

Аби обрати вибіркові дисципліни та їхні групи, користувач повинен натиснути на "+", перебуваючи на екрані "Професійні дисципліни" або "Дисципліни вільного вибору". Після натиску відбувається презентація модального вікна, що складається з пошукової строки та таблиці з доступними предметами. Вибір дисципліни відбувається внаслідок натиску на конкретну комірку таблиці, що містить назву предмету. Пошук по таблиці повинен бути динамічним, фільтруючи дані та залишаючи лише потрібні користувачу дисципліни. Вище описані екрани для вибору професійних дисциплін та їхніх груп знаходяться в Додатку Г.

Після опису детального плану застосунку можна переходити до наступного кроку.

3.3. Створення вхідних даних

Третій етап — вибір формату вхідних даних та їхнє створення.

У цій роботі вихідними даними є розклад занять та перелік предметів.

Об’єкт переліку предметів містить наступні атрибути: faculty (типу String), group (типу Integer), id (типу Integer), isRegistered (типу Boolean), name (типу String), numberOfGroups (типу Integer), specialty (типу String), trimester (типу Integer), type (типу String), year (типу Integer).

Об’єкт розкладу занять складається з наступних атрибутів: auditorium (типу String), lessonDate (типу String), lessonTime (типу String), name (типу String), notes (типу String), group (типу Integer), id (типу Integer), longitude (типу Integer) та latitude (типу Integer).

Найзручнішим форматом даних для цієї роботи є JSON. Мова програмування Swift дає змогу швидко та зручно декодувати JSON-файл. Для цього потрібно спершу створити модель даних — структуру, що містить ідентичні до об’єктів JSON-файлів поля. Наступний крок — процес декодування за допомогою Decoder протоколу Decodable. Останній крок — стандартна обробка помилок. Усього було створено 2 JSON-файли: schedule.json та subjects.json. Об’єктами, що містяться в першому файлі, є заняття, а в другому — предмети. Дані було створено власноруч, взявши за приклад розклад занять та список предметів Академії.

3.4. Створення бази даних

Четвертий етап — створення, наповнення баз даних та робота з ними.

Усього було створено 3 бази даних: Lesson, Subject та MySpecialty. Структура об’єктів Lesson та Subject повністю повторюють структуру

об’єктів, що розташовані у JSON-файлах Schedule.json та Subjects.json відповідно. Третя база даних, MySpecialty, необхідна для даних про спеціальність, на якій навчається студент/-ка (типу String) та рік його/її навчання (типу Integer64).

Для зручного керування базами даних було створено клас CoreDataProcessor, що містить набір методів та змінних, які дозволяють уніфіковано працювати з наявними даними. Так вдалося уникнути дублювання коду та його надлишку в інших файлах.

Для безпечної роботи з базами даних було використано патерн Singleton. Патерн розв’язує дві проблеми, що постають під час розробки:

1. Наявність кількох екземплярів класу. Для більшости випадків це правильне рішення, проте у цій роботі існує потреба у безпечному доступі до спільного ресурсу — баз даних.

2. Відсутність глобальної точки доступу до єдиного екземпляра. Іноді розробники-початківці помилково використовують глобальні змінні. Попри зручність, вони є небезпечними, оскільки будь-який код може переписати вміст таких змінних. Патерн Singleton дає змогу мати доступ до об’єкта з будь-якої точки програми.

Втілення патерну відбувається у два кроки, які є розв’язанням проблем відповідно до вказаного вище переліку. Перший крок — створення статичної змінної, яка надалі використовуватиметься як єдина точка доступу до баз даних. Другий крок — інкапсуляція конструктора, тобто, надання йому private рівня доступу. Таким способом вдалося впровадити безпечний спосіб роботи з базами даних.

3.5 Створення екрану з мапою та інтеграція частини з мапою у SwiftUI у проєкт, написаний на UIKit

Для створення мапи було використано MapKit та SwiftUI. Було визначено потрібні параметри та передано необхідні дані в структуру. Також важливо протестувати мапу на різних даних: варто впевнитись у тому, що мапа відображається коректно для всіх потенційно можливих координат, аби в користувача/-ки не виникало проблем з пошуком корпусу.

Наступний крок — інтеграція наявного View у проєкт.

У Swift взаємодія UIKit та SwiftUI відбувається досить просто. Для того, щоб інтегрувати екран, створений за допомогою SwiftUI, у проєкт, написаний на UIKit, необхідно передати View у UIHostingController. Після цього необхідно налаштувати позиціонування мапи на екрані за допомогою constraints в AutoLayout. Останній крок цього етапу — тестування коректности передачі даних від UIKit до SwiftUI.

У такий спосіб вдалося зекономити час, витрачений на розробку екрану з мапою, оскільки за допомогою SwiftUI побудова інтерфейсу відбувається швидше та вимагає меншої уваги до деталей завдяки автоматичній розмітці елементів.

3.6. Створення механізму оновлення даних

Фінальний етап розробки — забезпечення правильного та вчасного оновлення даних.

Застосунок пропонує кілька варіантів взаємодії, внаслідок яких необхідно виконати зміну даних.

Перший варіант — вибір/зміна інформації про назву спеціальности чи року навчання в Академії. У такому разі відбувається, перш за все, зміна переліку нормативних дисциплін та відповідних даних у MySpecialty. Вибір

професійно-орієнтованих дисциплін та дисциплін вільного вибору повинен також підлаштовуватися під поточні дані. Останнє — фільтрація розкладу відповідно до обраних даних. Вона відбувається за допомогою переліку булеанів та логічних операцій над ними. Це допомагає швидко та ефективно складати розклад.

Другий варіант — зміна однієї чи кількох груп предмету (-ів). У такому випадку необхідно змінювати дві бази даних: Lesson та Subject, а також заново відфільтрувати розклад занять.

Коректне оновлення даних можливе завдяки патерну Delegate. Він використовується, коли є потреба в тому, аби одна сутність делегувала завдання або ж обов'язки на іншу сутність. У Swift такий механізм працює за допомогою протоколу, що містить вимогу певних методів та змінних. У цьому випадку делегатом є контролер вигляду, а виконавцем — контролер, у якому відбувається зміна даних. Перед зникненням з екрана виконавець викликає метод оновлення баз даних у делегата. Така архітектура є найбільш оптимізованою, оскільки оновлення даних відбувається лише після виходу з екрана вибору груп, спеціальности та року навчання, що цілком свідчить про остаточність рішення користувача/-ки.

3.7. Перевірка коду на відповідність стандартам

Важливим етапом після розробки застосунку є перевірка стилю коду. Зі SwiftLint це можна зробити швидко та зручно. Необхідно інтегрувати SwiftLint у XCode project за допомогою додавання Script Phase, а також додати XML-файл до папки проєкту.

Після цього в інтегрованому середовищі розробки з’являться позначки попереджень та помилок. Зручною є можливість вимкнути конкретні перевірки, якщо розробник/-ця цього потребує.

Фінальний крок — виправлення помилок після знаходження невідповідностей у коді автоматично або вручну. SwiftLint надає інструкції про те, які правила порушені та як саме їх можна виправити.

Цей етап дозволив упевнитись у коректності написання коду та в простій підтримці проєкту у майбутньому, а також забезпечити однорідний стиль коду в усьому проєкті.

3.8. Застосунок KMAscheduler як результат дослідження

Результатом цього дослідження є розробка дизайну та прототипу застосунку KMAscheduler, для створення якого було детально проаналізовано Гайдлайн із візуальної комунікації НаУКМА . Застосунок містить у собі два основні контролери вигляду: "Розклад занять" і "Налаштування". Застосунок використовує UIKit для створення користувацького інтерфейсу, Core Data для локального збереження даних та SwiftLint для остаточної перевірки коректности написання коду.

Екран "Розклад занять" — головний екран для користувача/-ки. Тут відбувається перегляд занять, що відбудуться у конкретну дату Вибір дати дає змогу швидше скласти плани на наступні дні.

Ключовою особливістю екрана є можливість відображення модального вікна, що дає змогу переглянути та залишити нотатки з приводу конкретного заняття.

Заняття, що містять нотатки, матимуть блакитну крапку біля стрілки у комірці як нативний індикатор наявности додаткової інформації.

Екран "Налаштування" містить 4 опції переходу на наступний екран: вибір спеціальности та року навчання, груп нормативних дисциплін, професійних дисциплін та дисциплін вільного вибору.

Три останні комірки переводять на екран із переліком відповідних предметів та можливістю вибору їхніх груп.

Інформація про назву спеціальности, рік навчання та обрані предмети групи міститься локально на пристрої, використовуючи Core Data — фреймворк для керування базами даних від Apple. Так користувач/-ка може мати доступ до даних без потреби в мережі. Оновлення даних відбувається оптимізовано та сприяє зменшенню навантаження на пам’ять та процесор пристрою.

Застосунок є практичним застосуванням проведеного дослідження щодо створення user-friendly User Interface- та User Experience-дизайну, що відповідає вимогам Академії щодо використання виключно корпоративних кольорів та гарнітури, а також є продуманим до найдрібніших деталей, аби користувачу/-ці було приємно використовувати застосунок та надавати йому перевагу серед наявних аналогів. Застосунок демонструє потенціал використання даних про розклад у нестандартному форматі та з використанням сучасних технологій. Подальші покращення можуть бути спрямована на створення більш зручного формату даних про розклад занять та перелік предметів та ще більш оптимізованого оновлення даних, а також створення інших потрібних для студентства особливостей.

Висновки

У курсовій роботі було досліджено інструменти для розробки прототипу та застосунку iOS застосунку, що слугує своєрідним календарем для зручного перегляду пар та нотаток до них. Дослідження взяло початок з вичерпного аналізу наявних фреймворків для розробки застосунків під iOS та статей про патерни розробки. Завдяки цьому дослідженню вдалося дізнатися більше про хороші практики створення дизайну й написання коду та обрати необхідні бібліотеки для створення застосунку.

Завдяки цьому можна було створити дизайн, що відповідав офіційному Гайдлайну з візуальної комунікації НаУКМА та найкращим практикам UI та UX дизайну.

Наступний крок — вибір формату файлів для зберігання необхідних даних. Для цієї мети було обрано JSON-формат. Опісля було створено вихідні файли, з яких отримується інформація при першому запуску чи оновленні застосунку.

За цим слідувала повна розробка застосунку, що включала в себе ряд етапів: створення UI, моделей бази даних, а також вчасного оновлення UI та бази даних.

Фінальний етап — перевірка коду на відповідність загальноприйнятим стандартам. Цей крок є важливим зокрема для подальшого розвитку продукту, аби інші розробники, що працюватимуть над ним, могли краще та швидше розуміти й покращувати наявний код.

Ця робота презентує можливості покращення наявного телеґрам-бота та сервісу автоматизованого запису Києво-Могилянської Академії. Проведене дослідження вказує на широкі перспективи мови Swift для розробки iOS-застосунків. У роботі також описано виклики, що постають внаслідок спроби розробки UI та UX дизайну застосунку.

Ця курсова робота може стати сильним підґрунтям для подальшої розробки повноцінного сервісу, що надалі може бути інтегрованим з усіма наявними сервісами Києво-Могилянської Академії. Розробка подібного застосунку та його дизайну сприяє розвитку загального мислення та навичок проведення досліджень, результати яких можна використати в реальних комерційних проєктах.

Попри вдалий результат дослідження, ця робота, як і будь-яка інша, містить певні недоліки: відсутність сповіщень та нагадувань про подальші заняття й підготовку до них, а також брак можливості переглянути корпус та авдиторію, в якій відбудеться заняття, на мапі.

Сенс подальшої роботи може полягати в прямому опитуванні студентів та студенток, тестуванні зручности застосунку, зокрема його дизайну. Неодмінно важливим також було б додавання нового та розширення вже наявного функціоналу.

Отже, в роботі було продемонстровано повний процес створення сервісу для перегляду розкладу занять та обґрунтування потреби в ньому.

Вдала розробка проєкту свідчить про якісно проведене дослідження та коректно вибрані інструменти для застосунку. Проєкт знаходиться у вільному доступі та є відкритим для доповнень і покращень. Переглянути проєкт можна за покликанням: https://github.com/nhrysiuk/KMAScheduler

Список використаних джерел

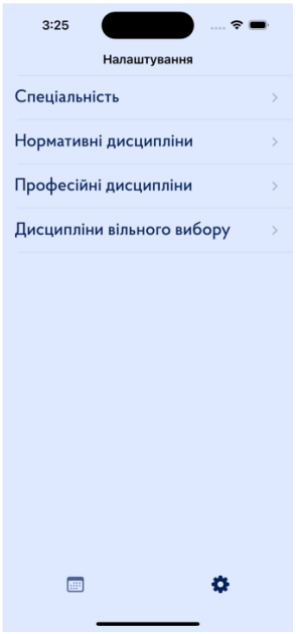
1. Apple Inc. Core Data Documentation [Електронний ресурс] / Apple Inc. — Режим доступу до ресурсу: https://developer.apple.com/documentation/coredata/ (дата звернення: 10.05.2024)
2. Скорочене керівництво з використання елементів візуальної комунікації НаУКМА [Електронний ресурс] / НаУКМА — Режим доступу до ресурсу: https://ekmair.ukma.edu.ua/items/d1e1906a-298a-40f9-abb8-a404758bd8a8 (дата звернення: 10.05.2024)
3. Apple Inc. UI Kit Documentation [Електронний ресурс] / Apple Inc. — Режим доступу до ресурсу: https://developer.apple.com/documentation/uikit/ (дата звернення: 10.05.2024)
4. Apple Inc. Human Interface Guidelines Documentation [Електронний ресурс] / Apple Inc. — Режим доступу до ресурсу: https://developer.apple.com/design/human-interface-guidelines (дата звернення: 10.05.2024)
5. Mastering MVC in iOS Swift Development: From Basics to Best Practices [Електронний ресурс] / Joshua Hart — Режим доступу до ресурсу: https://medium.com/@emt.joshhart/mastering-mvc-in-ios-swift-development-from-basics-to-best-practices-489217e9a3b9 (дата звернення: 10.05.2024)
6. SwiftLint Documentation [Електронний ресурс] / Realm — Режим доступу до ресурсу: https://github.com/realm/SwiftLint (дата звернення: 10.05.2024)
7. iPhone User & Sales Statistics for 2024 [Електронний ресурс] / Shubham Singh — Режим доступу до ресурсу:

https://www.demandsage.com/iphone-user-statistics/ (дата звернення: 10.05.2024)

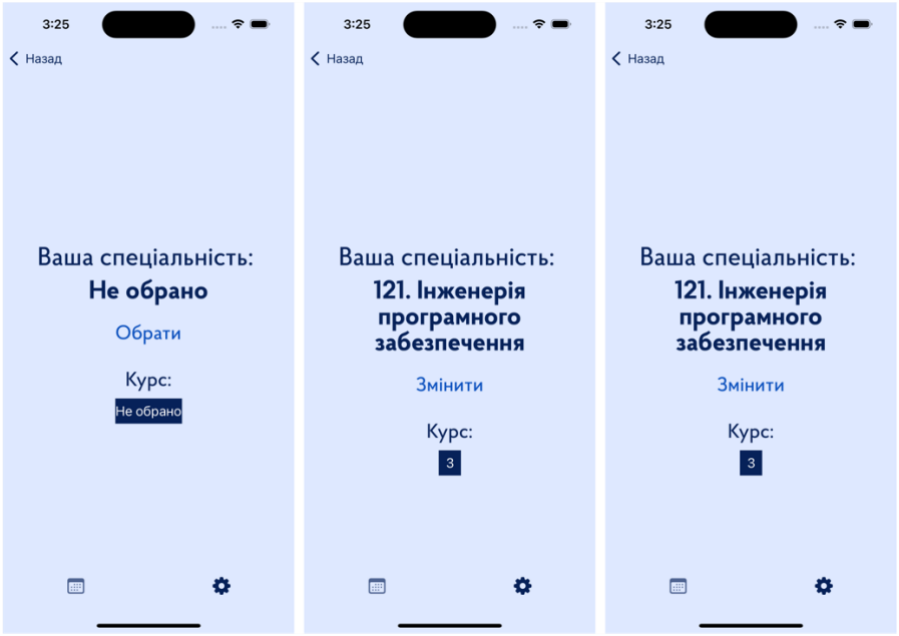
8. Apple App Store Statistics 2024 — Trends And Figures [Електронний ресурс] / By Harman Arora — Режим доступу до ресурсу: https://bigohtech.com/apple-app-store-statistics/ (дата звернення: 10.05.2024)
9. Google Calendar App [Електронний ресурс] — Режим доступу до ресурсу: https://apps.apple.com/ua/app/google-calendar-get-organized/id909319292?l=uk (дата звернення: 10.05.2024)
10. Notion Calendar App Figures [Електронний ресурс] — Режим доступу до ресурсу: https://apps.apple.com/ua/app/notion-calendar/id1607562761?l=uk (дата звернення: 10.05.2024)
11. KMAScheduler Telegram Bot [Електронний ресурс] — Режим доступу до ресурсу: https://t.me/KMASchedulerBot (дата звернення: 10.05.2024)
12. A Step-by-Step Guide to the Delegate Pattern in Swift [Електронний ресурс] / By Afonso Lucas — Режим доступу до ресурсу: https://medium.com/@afonso.script.sol/a-step-by-step-guide-to-the-delegate-pattern-in-swift-91a28de1baf8
13. Singleton [Електронний ресурс] / By — Oleksandr Shvets — Режим доступу до ресурсу: https://refactoring.guru/design-patterns/singleton

Додатки

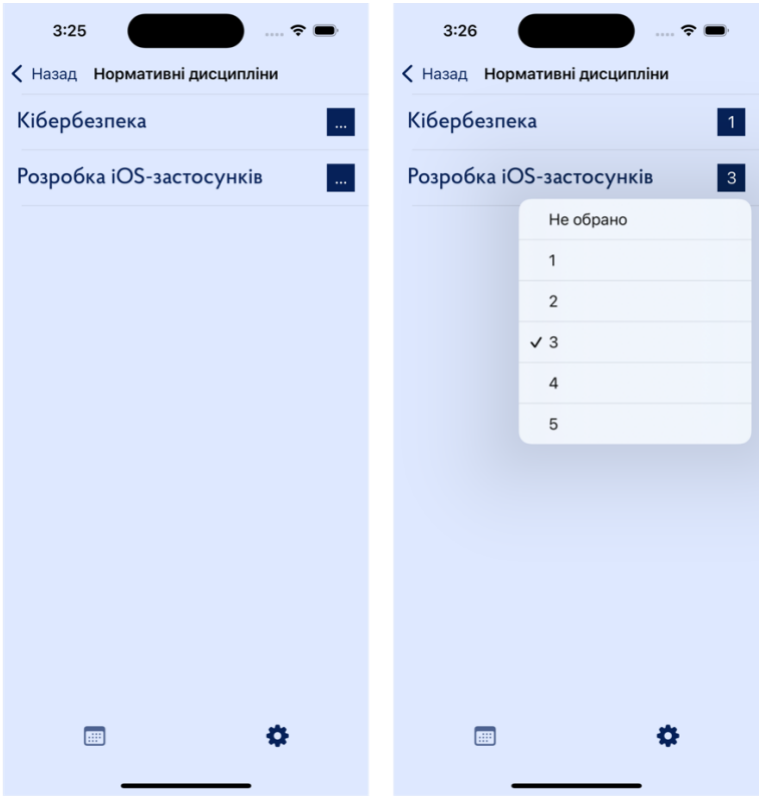
Додаток А. Другий екран для налаштувань



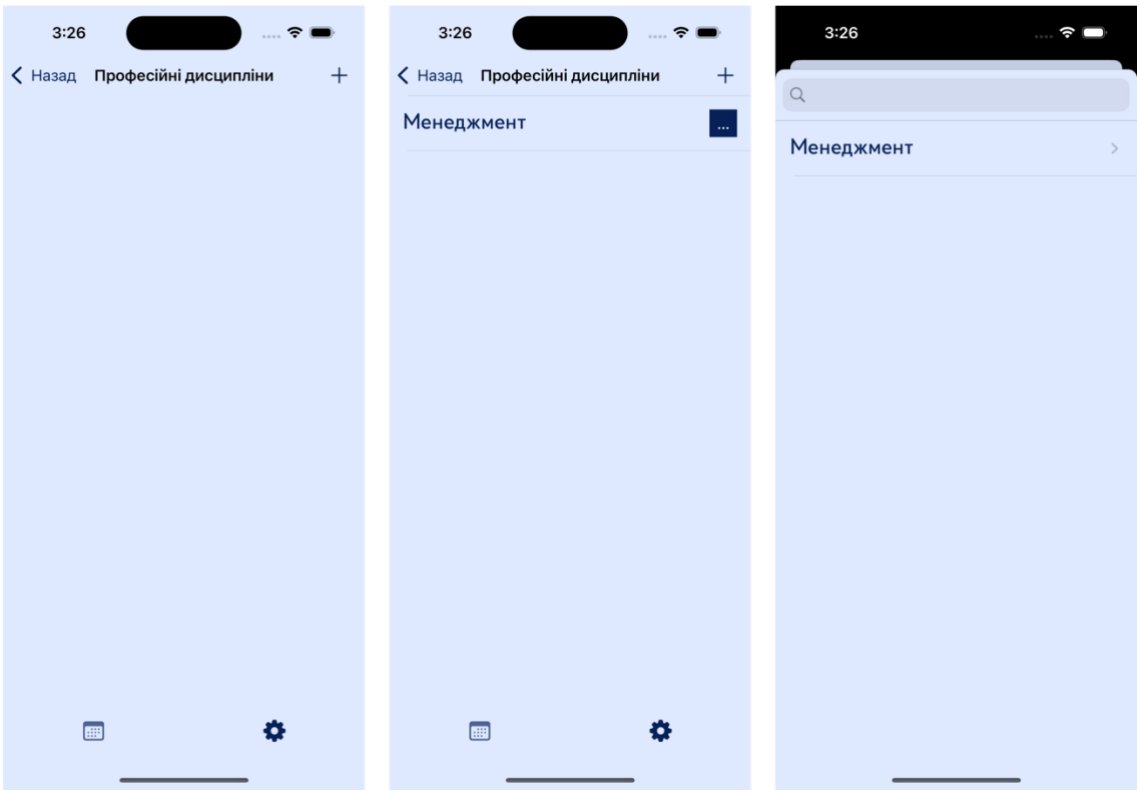
Додаток Б. Екран вибору спеціальности та року навчання



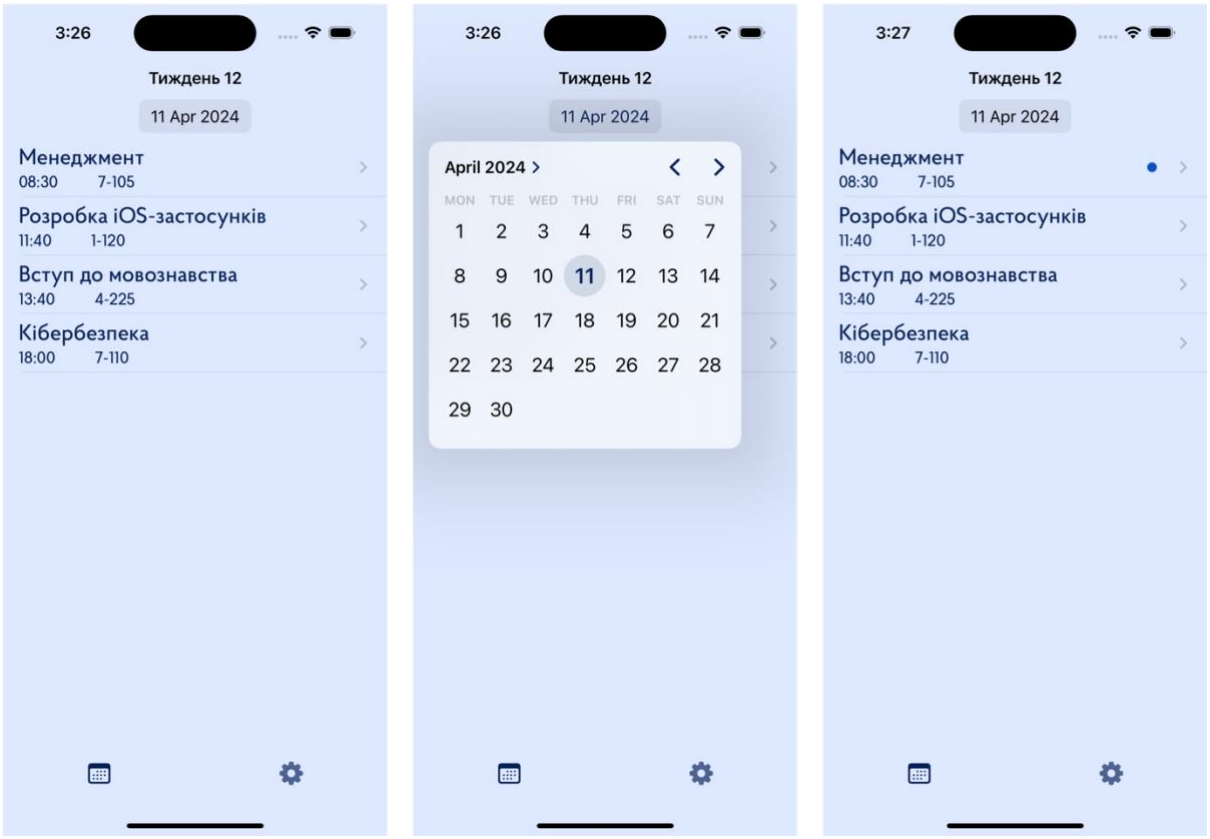
Додаток В. Екран вибору груп нормативних дисциплін



Додаток Г. Екран вибору професійних дисциплін та їхніх груп



Додаток Ґ. Перегляд розкладу дисциплін



Додаток Д. Перегляд екранів з нотатками та мапою