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

Розробка мобільного застосунку на базі iOS

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

Керівник курсової роботи
ст. викл. Борозенний С.О.
_____
(підпис)
“____” _____ 2021 р.

Виконав студент 4-го курсу
Зубко Д.А.
“____” _____ 2021 р.

Київ 2021

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ

Зав. кафедри мультимедійних систем,

доцент, к.ф.-м.н.

_____ О.П. Жежерун

(підпис)

“____”_____ 2020 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту Зубку Дмитру

4-го курсу факультету інформатики

ТЕМА: Розробка мобільного застосунку на базі iOS

Вихідні дані:

- Клієнт-серверне застосування для взаємної резервації подарунків

Зміст ТЧ до курсової роботи:

Вступ
1. Аналіз предметної області. Постановка завдання курсової роботи
2. Теоретичні відомості
3. Опис реалізації програмного продукту
Висновки
Список джерел
Додатки (за необхідністю)

Дата видачі “___” _____ 2020 р.

Керівник _____ Завдання отримано _____

Календарний план виконання курсової роботи

Тема: Розробка мобільного застосунку на базі iOS

Календарний план виконання роботи:

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу		
2.	Вивчення та аналіз задачі		
3.	Розробка архітектури та загальної структури програми		
4.	Створення застосунку		
5.	Написання текстової частини		
6.	Перегляд курсової роботи науковим керівником		
7.	Створення презентації		
8.	Захист курсової роботи		

Студент Зубко Д. А. _____

Керівник Борозенний С. О. _____

“_____” _____ 2020 р.

Зміст

Перелік умовних позначень

REST – Representational State Transfer (архітектурний стиль взаємодії компонентів певного застосування в мережі)

API – Application Programming Interface (набір методів для взаємодії різних компонентів).

СКБД – система керування базами даних.

iOS – мобільна операційна система, що створена компанією Apple для своїх пристроїв

Xcode – інтегроване середовище розробки

URI – уніфікований ідентифікатор ресурсу

DAO – Data Access Object (об’єкт доступу до даних)

Анотація

Робота присвячена створенню мобільного додатку на базі операційної системи iOS задля взаємної резервації та обміну подарунками серед користувачів.

У наведеній роботі описано основні принципи створення клієнтського мобільного додатку, побудову RESTful API для цього застосунку з використанням веб-сервера.

Клієнтський застосунок використовує платформу iOS, та написаний за допомогою Swift. Для комунікації з сервером, використовується бібліотека з відкритим вихідним кодом Alamofire. Серверна частина базується на Java та Spring фреймворці. В якості СКБД обрано PostgreSQL.

Вступ

Створення мобільних додатків для різного роду задач чи потреб нині є дуже поширеною справою. Мобільні пристрої стали чи не найзручнішим методом отримання, обробки та обміну інформацією між користувачами.

Метою даної курсової роботи є створення мобільного додатку, що дозволить кінцевим користувачам ділитися вподобаннями між собою та використовувати це застосування в особистих цілях. Розглянемо також принципи розробки веб-сервісу, що дозволить клієнтам взаємодіяти один з одним та доступатись до збережених даних самих користувачів.

Основною мовою програмування клієнтського застосунку є Swift. На сьогодні, майже кожен мобільний додаток написаний для iOS використовує Swift. Вибір даної мови обумовлений досить щільною інтеграцією по відношенню до мобільних пристроїв операційної системи iOS, а також, майже повною відсутністю аналогів розробки рішень для вищезгаданої системи. Серверна частина розроблена за допомогою Java та Spring Framework. Java є досить популярною мовою програмування та надає досить гнучкий інструментарій для проектування і створення серверних застосувань. Використання надбудови Spring дозволяє реалізувати взаємодію з різними СКБД, створити сервіси авторизації та автентифікації, і як результат, надати клієнту відповідне API.

1. Аналіз предметної області. Постановка завдання курсової роботи.

1.1. Огляд існуючих аналогів розробки.

Наразі існує чимала кількість мобільних застосунків, що пропонує для завантаження два ключових дистриб'ютора у цій галузі: станом на 2018 рік, онлайн-магазин Google Play пропонував своїм користувачам понад 2.8 мільйонів додатків різних типів і категорій [1]. В свою чергу, Apple App Store у 2015 році заявив, що кількість застосунків доступних для завантаження наближається до 1.5 мільйонів одиниць [2]. Сьогодні, очевидно, ця кількість є значно більшою. Розглядаючи тематику організації подарунків та формування списків побажань, можна віднайти певні додатки в вищезгаданих онлайн-магазинах. Нижче наведено кілька таких застосувань.

Christmas Gift List – мобільний застосунок, створений для користувачів операційної системи Android, та розміщений в онлайн магазині Google Play [3]. Додаток фактично являється утилітою для формування списків, які за потреби можуть бути доповнені або відредаговані. Кожен елемент списку присвоюється окремій особі, що може бути створена власноруч у додатку. Приклади функціональних можливостей додатку *Christmas Gift List*:

- Групування списків побажань за певною особою.
- Формування статистики щодо особистих витрат та кількості подарунків.
- Меню налаштувань, з функціями вибору грошової одиниці / методу сортування / захисту паролем.

Сильною стороною цього додатку є простота у використанні: користувач досить швидко та зручно може створити новий список та додати певні записи. В той же час, вся інформація користувача зберігається на пристрої локально – це може призвести до втрати даних при будь-якій проблемі у

застосунку чи з самим фізичним пристроєм. Варто також зазначити, що вагомим мінусом цього застосунку є проблема з синхронізацією. При зміні пристрою, або ж використанні одночасно двох клієнтів, списки потрібно оновлювати вручну.

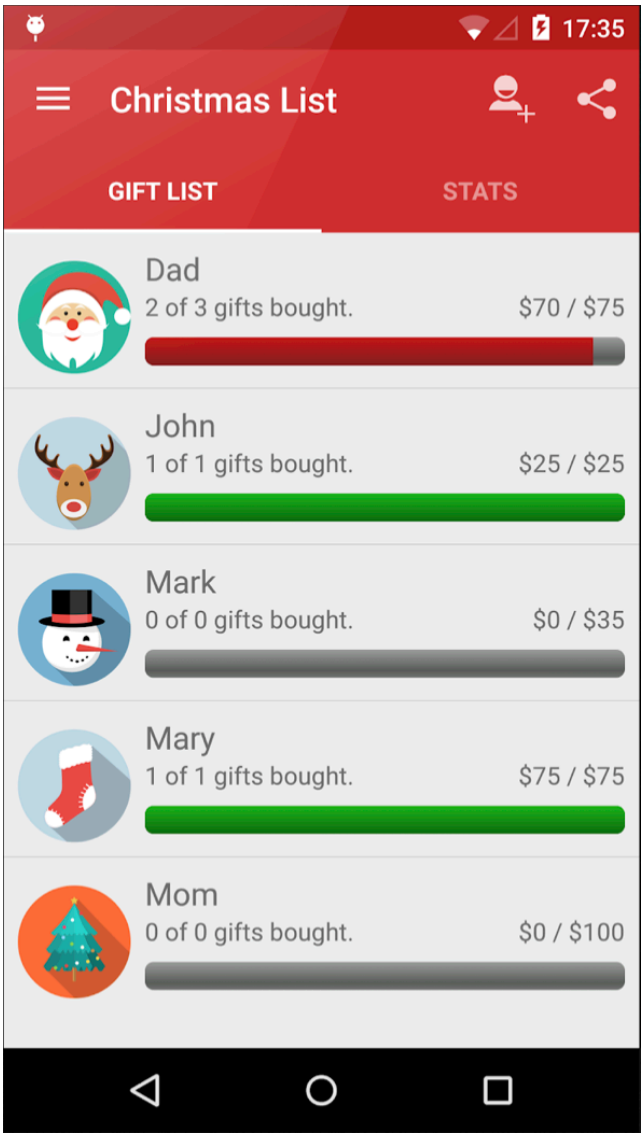

Рисунок 1 – приклад списків, створених користувачем в *Christmas Gift List*

GiftPlanner – застосунок для організації купівель в якості подарунків на базі iOS. [4]. Має великий список функцій, що виявляться корисними для кожного з користувачів: розширення для браузера дозволяє в мить додати певний продукт в список подарунків, можливість створення подій, людей, та закріплення подарунків за ними, редагування параметрів подарунків. В

додатку також реалізовано імпортування людей з телефонної книги. У випадку додавання нової подарункової позиції, необхідно створити нову особу або ж використати існуючу.

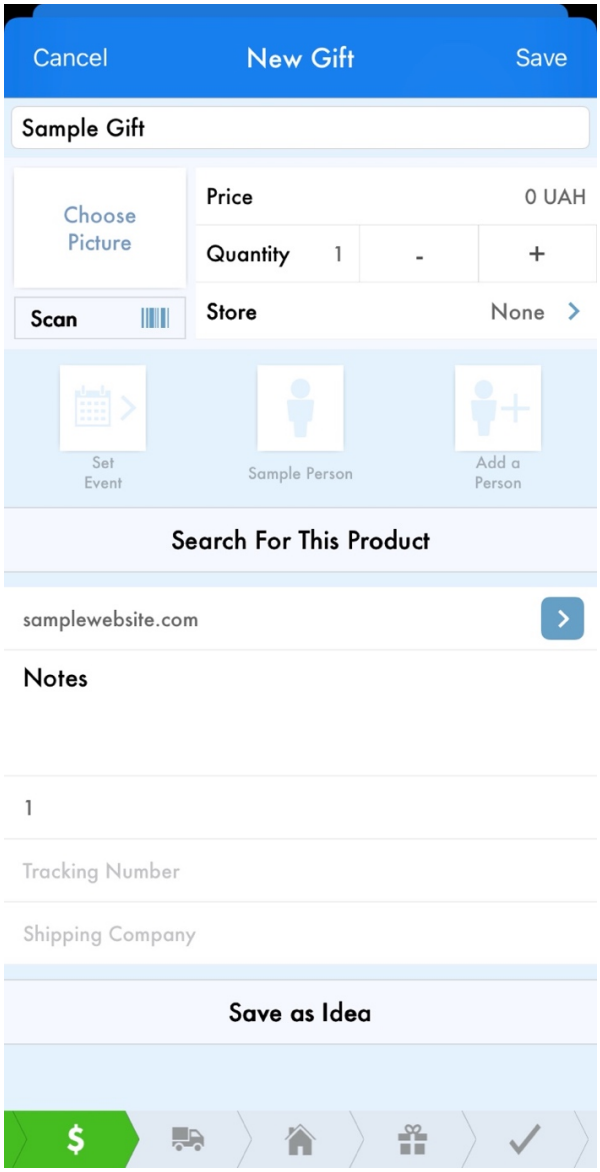

Рисунок 2 – створення нового подарунку в *GiftPlanner*

1.2. Постановка завдання

Необхідно розробити мобільний застосунок, для взаємного обміну та резервації вподобань користувачів.

Ключові модулі системи:

1) Клієнтський мобільний додаток для окремого користувача.

   Основні вимоги до клієнтської частини:

   1.1) Створення зручного графічного інтерфейсу користувача.

   1.2) Налаштування взаємодії клієнтського додатку з серверною частиною застосування.

2) Серверне застосування для організації взаємодії користувачів між собою та доступу до бази даних.

   Основні вимоги до серверного застосування:

   2.1) Реалізація прикладного програмного інтерфейсу задля можливості доступу окремого клієнта до внутрішнього функціоналу застосунку.

   2.2) Взаємодія з СКБД (збереження та отримання даних користувачів).

   2.3) Створення сервісів автентифікації та авторизації.

Функціональні можливості:

1) Реєстрація та авторизація окремого користувача.
2) Створення та подання інформації щодо окремого користувача (профіль користувача).
3) Створення та редагування побажань. Керування списком побажань.
4) Резервація побажань окремих користувачів.
5) Пошук і фільтрація даних про користувачів.
6) Створення рейтингу побажань на основі їх кількості.

2. Теоретичні відомості.

2.1. Swift – як інструмент для розробки мобільних додатків

Swift – сучасна мова програмування, що створена компанією Apple на заміну застарілій нині Objective-C [5]. Основною причиною для створення Swift була нестача функціональних можливостей, що були притаманні Objective-C на той час. На сьогодні, Swift є багатопарадигмовою мовою програмування, що використовується у різних задачах. Останні версії Swift підтримують низку платформ, а саме: операційні системи Apple (macOS, iOS, iPadOS, tvOS, watchOS), а також Windows, Linux та Android.

Swift є досить потужним засобом для створення мобільних застосунків, велика кількість внутрішніх компонентів дозволяє швидко та ефективно вести розробку додатків різної складності: так наприклад для побудови графічного інтерфейсу та розміщення елементів керування можна скористатись фреймворком SwiftUI [6]. В цьому випадку, розробник описує структуру графічного інтерфейсу за декларативним стилем прямо в програмному коді. З іншого боку, для візуалізації додатків можуть використовуватись зовнішні файли розмітки – Storyboards. Для керування залежностями на рівні додатку використовується менеджер CocoaPods. Він створений для того, щоб співіснувати з Swift та Objective-C, та надавати можливість підключення сторонніх бібліотек та розширень. Розглядаючи ключові особливості Swift, можна відмітити швидкодію в порівнянні з іншими мовами програмування, а також можливість написання стійкого до помилок коду.

2.1.1. Структура проекту з UIKit

Компонент UIKit являє собою частину одного великого фреймворку – CocoaTouch. Він надає основні шаблони та об’єкти, які використовуються для створення додатків під операційні системи Apple. Вищезгадані об'єкти мають різне призначення: одні об’єкти беруть участь у відображенні

графічного вмісту на екрані, інші - співпрацюють з цим вмістом, запускають основний цикл подій програми та керують взаємодією з системою. Кожен додаток, що створений за допомогою UIKit використовує його базову поведінку, проте існує багато способів налаштувати цю поведінку відповідно до певних задач чи потреб.

Використовуючи Xcode в парі з UIKit, можливо створювати готові шаблонні проекти, що містять усі необхідні компоненти для побудови та запуску того чи іншого додатку [7]. Для прикладу, розглянемо шаблон типу “Application” (див. рис. 3). Шаблонні проекти за замовчуванням містять в собі мінімальний користувацький інтерфейс, а це в свою чергу дозволяє запустити створений проект та переглянути результат виконання на фізичному пристрої або в симуляторі.

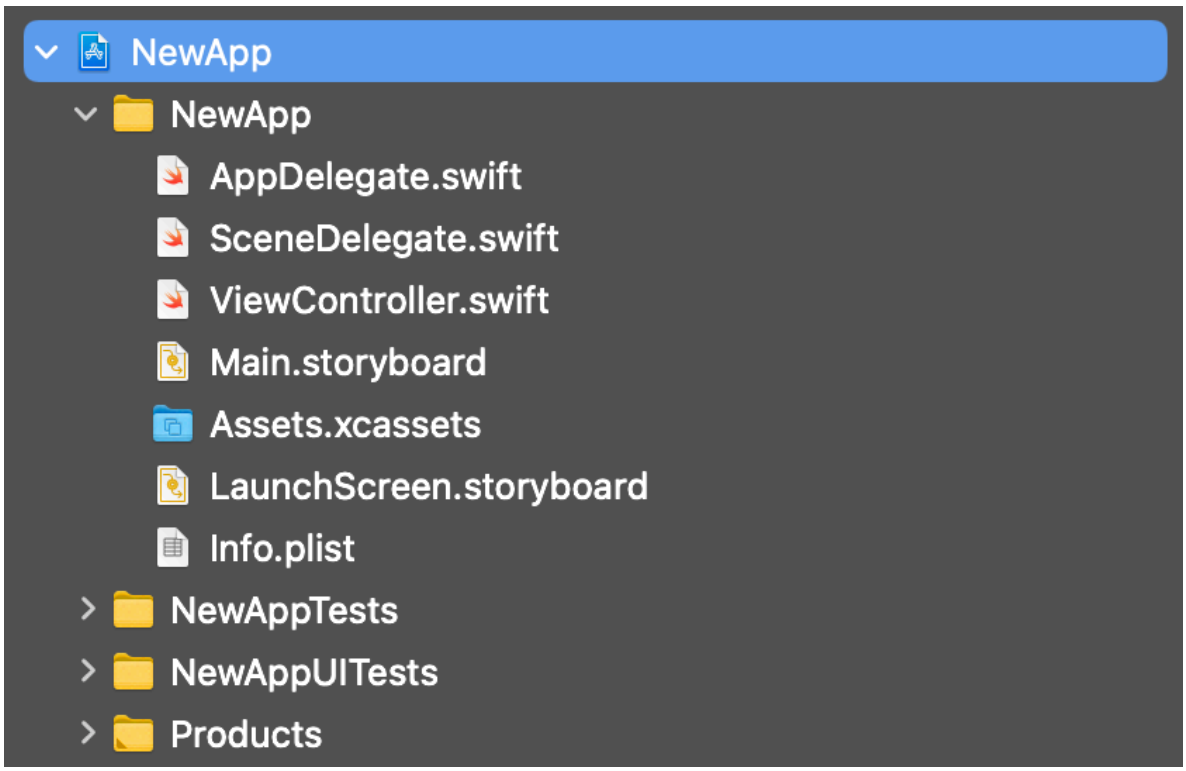

Рисунок 3 – приклад створеного шаблонного додатку типу “Application”

Інформація про основні властивості додатку вказана в файлі “Info.plist”. За умови створення шаблонного проекту, розробнику надається попередньо

налаштована версія цього файлу, проте даний файл може бути відредагований в будь-який момент. Для прикладу, програма може використовувати конкретне фізичне обладнання, як от камера, мікрофон чи датчик GPS. Всі функції пов’язані з цим обладнанням повинні бути узгоджені з кінцевим користувачем, тому інформація щодо запиту на використання цього обладнання має бути вказана в цьому файлі.

Структура додатків UIKit базується на патерні проектування Model-View-Controller (MVC) [8]. Основною ідеєю цього патерну є розділення об’єктів за призначенням. Об’єкти типу “Model” мають на меті представлення основних сутностей та керують даними програми. “View” об’єкти забезпечують візуальне представлення даних. Об'єкти типу “Controller” являються проміжними та виступають мостом між моделями та візуальними об'єктами, опрацювуючи дані між ними в різні проміжки часу.

Оглянемо типову структуру програми на основі UIKit (див. рис. 4). Розробник створює об’єкти, які представляють основні сутності чи структури даних конкретного додатку. UIKit надає елементи типу View, які за потреби можуть бути перевизначені задля коректного відображення даних. Обмін даними між моделями та елементами View - це контролери та об’єкти-делегати програми.

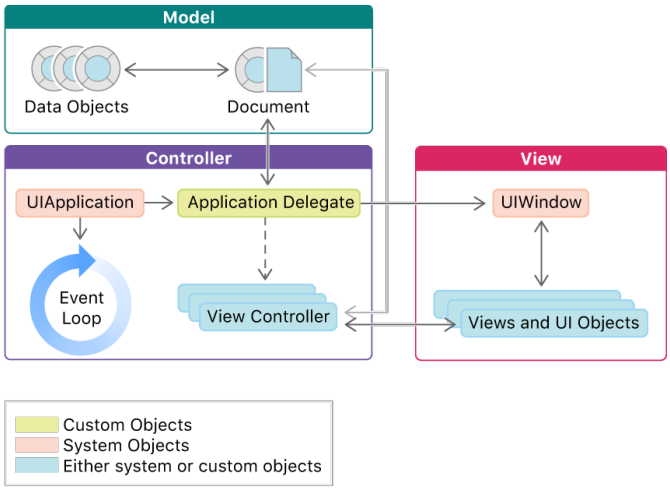

Рисунок 4 – основні компоненти додатку

Фреймворки UIKit та Foundation надають велику кількість основних типів, які використовуються для визначення об’єктів-моделей в різних застосунках. Foundation фреймворк визначає основні об’єкти, що представляють стрічки, масиви, числа та багато інших тиів даних. UIKit забезпечує розробника численними об’єктами, що подаються як елементи керування, або елементи представлення. Базовим елементом типу View є клас UIView, який займається відображенням вмісту на екрані. Об’єкт UIApplication є відправною точкою при запуску програми - він створює основний цикл подій та керує загальним життєвим циклом програми.

2.1.2. Interface Builder та Storyboards

Для створення графічного інтерфейсу користувача Xcode пропонує утиліту під назвою Interface Builder. Файли графічного інтерфейсу поділяються на два типи: xib та storyboard. Перший тип файлу зазвичай представляє собою один view controller, або ж елемент меню. Storyboard в свою чергу може містити в собі велику кількість контролерів та з’єднань між ними. Вміст цих файлів представлений в XML форматі, проте Xcode надає змогу редагувати їх в графічному вигляді.

Розглянемо приклад побудови графічного інтерфейсу користувача за допомогою storyboard. Кожен такий файл може бути розділений на сцени, переходи між ними та елементи управління цими переходами. Сцена представляє собою область екранного вмісту. Як правило, на пристроях з малими екранами використовується одна сцена, у випадку розробки додатків для iPad чи комп’ютерів Mac екран може містити декілька сцен. Елемент segue являє собою перехід від одної сцени до іншої. Кожен окремий storyboard також може бути зв’язаний з іншим за допомогою segue. Приклад storyboard, що містить в собі три сцени та 2 переходи між ними зображений нижче (див. рис. 5)

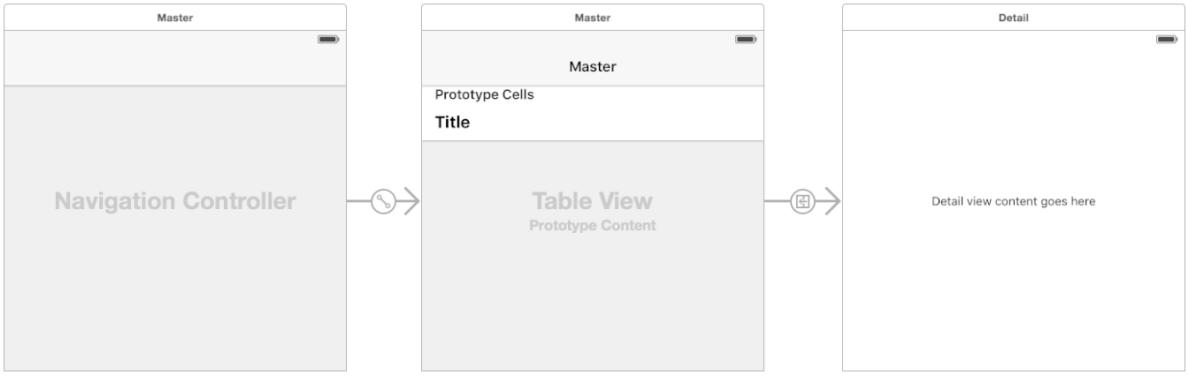

Рисунок 5 – storyboard з трьома сценами та переходами між ними

Перша сцена являє собою навігаційний контролер, який керує переміщенням користувача між двома іншими сценами. В результаті, кінцевий користувач бачитиме перед собою головне меню зі списком певних елементів, а також матиме можливість перейти до деталей окремого елементу. Навігаційний контролер також надає додаткові елементи керування, як от кнопка повернення назад.

2.1.3. Поєднання елементів графічного інтерфейсу з програмним кодом

Поведінка об’єктів графічного інтерфейсу зазвичай задається в програмному коді компонентів view controller. Код взаємодіє з об’єктами користувацького інтерфейсу через дії (action) та вихідні з’єднання (outlet).

З’єднання типу action використовується за потреби відправки повідомлення від елементу управління до програмного коду (див. рис. 6). Елемент управління в даному випадку - це об’єкт графічного інтерфейсу користувача, що спричиняє миттєві дії або певні видимі результати. Прикладом може стати натискання кнопки: повідомлення про дію відправляється до програмного коду з вказівками щодо оновлення даних. Повзунки, перемикачі, кнопки та текстові поля є прикладами часто використовуваних елементів керування в операційних системах Apple.

```
@IBAction func logoutButtonTapped(_ sender: Any) {
	//some logic
}

@IBAction func sliderMoved(_ sender: Any) {
	//some other logic
}
```

Рисунок 6 – приклад з’єднань типу “action”

Вихідне з’єднання формується у разі необхідності відправки повідомлення від програмного коду до графічного інтерфейсу. Об’єктом графічного інтерфейсу може бути вищезгаданий елемент управління, або ж любий інший об’єкт, що розміщений в конкретному storyboard файлі. Зазвичай, такі об’єкти як мітки, зображення, текстові поля та колекції елементів використовують з’єднання типу outlet (див. рис. 7).

```
@IBOutlet weak var descriptionLabel: UILabel!
@IBOutlet weak var loginTextField: UITextField!
if (loginTextField.text!.isEmpty) {
	descriptionLabel.text = “Provide login!”
}
```

Рисунок 7 – приклад з’єднань типу “outlet”

Створення екземплярів з’єднань відбувається після того як створиться view controller, проте це обов’язково здійсниться до того, як цей контролер відобразиться.

2.2. Java та Spring Framework в якості основних інструментів при створенні веб-сервера.

На сьогодні Java займає лідируючі позиції в сфері розробки серверних застосувань [9]. Наявність величезної кількості бібліотек, фреймворків, підтримка з боку представника і розробника Oracle робить дану мову гарним рішенням для створення серверного програмного забезпечення. Одним з найпопулярніших програмних каркасів для Java є Spring – це фреймворк з відкритим вихідним кодом, що дозволяє створювати та підтримувати архітектуру серверних застосувань різного рівня [10].

Spring являє собою набір інструментів для різного роду задач: робота з базами даних, розгортка веб-інфраструктури, керування безпекою застосунку тощо. Використання надбудови Spring Boot дозволяє з легкістю налаштовувати та запускати веб-застосування завдяки вбудованому веб серверу на зразок Tomcat, керувати залежностями та зовнішніми бібліотеками, а також автоматично обробляти конфігурацію застосунку.

2.2.1. Налаштування та побудова веб-сервера за допомогою Spring Boot

Spring має на меті виконувати роль контейнера з підтримкою інверсії управління. Sping Boot поєднує в собі велику кількість модулів, що дозволяють відмовитись від звичайної XML-конфігурації та в автоматичному режимі проводити діагностику і життєвий цикл веб-застосунку.

Ключовими точками Spring є компоненти, що використовуються під час роботи сервера [11]. Кожен з цих компонентів повинен виконувати свою роль та надавати зручний інтерфейс для доступу до тих чи інших функціональних можливостей окремого класу. Архітектурний підхід, що

дозволяє розділити програму на логічні частини використовує три типи компонентів: контролери, сервіси та репозиторії.

Контролери фактично представляють мережевий рівень веб-сервера та обробляють зовнішні HTTP-запити клієнтів. Spring надає низку анотацій, що дозволяють налаштовувати контролери, а також взаємодіяти з ними (див. рис. 8).

```
//imports

@Controller
@RequestMapping(“object”)
public class ObjectController {

   @GetMapping()
   public @ResponseBody Object getObjects () {
       //some logic
   }

   @PostMapping()
   public @ResponseBody Object createObject (
      @RequestBody final Object obj) {
       //some logic
   }
```

Рисунок 8 – типовий клас-контроллер

Клас, що використовується в якості контролера, позначається однойменною анотацією @Controller. Дана анотація наслідує базову @Component, та може бути замінена більш функціональною @RestController. В цьому випадку, розробнику не обов’язково вказувати в методах класу @ResponseBody для позначення типу об’єкта, що повертається клієнту. Кожен метод контролера може обробляти HTTP-запити різних типів і повинен бути позначений відповідною анотацією. @RequestMapping дозволяє вказати базовий URI до всіх методів даного класу. За бажанням, в параметрі анотації можна також вказати шлях до ресурсу в поєднанні з шляхом на рівні класу.

Компонент типу Service є проміжним між контролерами та репозиторіями. Зазвичай сервіси використовуються для реалізації бізнес-логіки додатку (див. рис. 9).

```
//imports

@Service
public class ObjectService {

   public List<Object> getObjects () {
       List<Object> objts = retrieveObjects();
       if (objts.isEmpty()){
           throw new ElementNotFoundException();
       }
   }

   public Object createObject (Object obj) {
       validateObjectUniqeness(obj);
       insert(obj);
   }
}
```

Рисунок 9 – приклад класу, що займається бізнес-логікою застосунку

Компоненти, що визначають поведінку різних об’єктів предметної області, як правило, позначають анотацією @Service.

Для взаємодії з СКБД, пошуку і зберігання даних використовують класи-репозиторії (див. рис. 10). Spring надає зручний інтерфейс JdbcTemplate, що полегшує роботу з SQL-подібними реляційними базами даних та JDBC. JdbcTemplate виконує всю роботу, що пов’язана зі з’єднанням до бази даних, керуванням виключень та обробкою помилок. Вбудовані анотації, що надає Spring, також містять в собі досить корисні функціональні можливості: для прикладу, розробник може винести тексти SQL-запитів в окремі файли, та доступатись до них через зручний інтерфейс @PropertySource.

```
//imports

@Repository
@PrpertySource(“classpath:sql/obj_queries.properties”)
public class ObjectDao {

   @Autowired
   private JdbcTempate jdbcTemplate;

   public List<Object> getAllObjects () {
       return this.jdbcTemplate.query(query);
   }

   public Object createObject (Object obj) {
       SQLParameterSource parameters = new MapSQLParameterSource()
              .addValue(“name”, obj.name())
              .addValue(“description”, obj.description());
       return this.template.update(query, parameters);

   }
}
```

Рисунок 10 – клас, що представляє собою репозиторій

2.2.2. Налаштування авторизації та автентифікації в Spring

Spring надає інструменти для побудови засобів автентифікації і контролю доступу, що поєднані в єдиному Spring Security фреймворці [12].

Фундаментальним об’єктом Spring Security є SecurityContextHolder. В цьому компоненті зберігається основна інформація про контекст безпеки самого застосунку. Контекст в свою чергу зберігає в собі інформацію про користувача, який в даний момент працює з додатком, та визначений в Spring Security як “Authentication”. Для валідної працездатності системи автентифікації використовуються також інтерфейс UserDetailsService - він використовується для створення екземпляру класу UserDetails, який в свою чергу надає інформацію для побудови об’єкту Authentication. Основнй інтерфейс, що відповідає за прийняття рішень в області контролю доступу має назву AccessDecisionManager. Даний інтерфейс містить в собі основний метод, який приймає об'єкт Authentication. Окрім інформації щодо поточного користувача, цей об’єкт містить в собі список атрибутів

метаданих безпеки, які застосовуються до цього об'єкта, наприклад, роль користувача в системі та його функціональні можливості. Конфігурація безпеки HTTP-запитів відбувається за допомогою WebSecurityConfigurerAdapter (див. рис. 11).

```
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(securedEnabled = true, proxyTargetClass = true)
public class SampleSecurity extends WebSecurityConfigurerAdapter {

  @Autowired
  private ImplementedUserDetailsService;

  @Bean
  public PasswordEncoder passwordEncoder() {
      return new BCryptPasswordEncoder(11);
  }

  @Autowired
  public void configureGlobal(AuthenticationManagerBuilder amb) {
      amb
         .passwordEncoder(passwordEncoder());
         .userDetailsService(customUserDetailsService);
  }

   @Override
   protected void configure (HttpSecurity httpSecur) {
       security
              .antMatchers(“/authorized/*”).hasRole(“STAFF”)
              .antMathcers(“/profile/*”).hasRole(“MEMBER”)
              .antMatchers(“/”, “/auth/**”).permitAll()
              .anyRequest().authenticated()
              .and()
              .httpBasic();
   }
```

Рисунок 11 – клас SampleSecurity, що реалізовує WebSecurityConfigurerAdapter

Конфігурація автентифікації та контролю доступу відбувається в наведеному вище рисунку: для налаштування засобів безпеки HTTP-сервісів необхідно наслідувати та перевизначити методи класу WebSecurityConfigurerAdapter. Метод configure() дозволяє встановити тип автентифікації та надати список уніфікованих ідентифікаторів ресурсів, що можуть бути надані лише певним типам користувачів. Інший метод configureGlobal() дозволяє визначити власну реалізацію інтерфейсу

UserDetailsService та налаштувати додаткові інструменти на зразок кодувальника паролів BCryptPasswordEncoder.

3. Опис реалізації програмного продукту

3.1. Обґрунтування вибору засобів розробки

Мобільний додаток, а саме його клієнтська частина розроблена на базі iOS за допомогою Swift та Xcode.

Розробка застосунків під операційні системи типу iOS, watchOS, iPadOS можлива за умови використання мови Objective-C, або ж більш сучасної Swift, тому вибір мови в даному випадку є досить очевидним. Функціональні можливості Swift в поєднанні з інтегрованим середовищем розробки Xcode дозволяють створювати потужні та конкурентні застосунки в порівнянні з іншими мобільними операційними системами та їх інструментами. Xcode надає низку можливостей для реалізації графічної складової додатку завдяки вбудованій утиліті Interface Builder. Поєднання графічних елементів та елементів керування з програмним кодом є досить зручним та функціональним. Наявність вичерпної документації також сприяє покращенню процесу розробки.

Задля комунікації с серверною частиною застосунку використано бібліотеку з відкритим вихідним кодом Alamofire [13]. Дане рішення дозволяє працювати з мережею на рівні HTTP-запитів. Бібліотека надає елегантний інтерфейс для роботи з мережею та значно спрощує використання мережевого стеку, що надається за замовчуванням в Foundation фреймворці. Інтерфейс надає низку функцій, що дозволяють працювати з запитами типу GET, POST, PUT, DELETE, працювати з авторизацією, обробляти відповіді від сервера та багато іншого.

Серверна частина проекту реалізована з використанням Java та Spring Framework. Надбудова Spring Boot дозволяє з легкістю конфігурувати веб-сервер та розширювати його у разі потреби без значних труднощів. IntelliJ IDEA, як інтегроване середовище розробки для Java досить тісно

інтегроване з Spring, та надає зручний інструментарій для роботи з самим фреймворком, базами даних та різними менеджерами автоматизації побудови і запуску проектів на зразок Maven [14].

Для роботи з даними, обрано СКБД PostgreSQL. Це потужна, об'єктно-реляційна система баз даних з відкритим кодом. Дана СКБД використовується в проектах різного масштабу завдяки своїй простоті, швидкодії, надійності та перевіреній архітектурі [15].

3.2. Опис розробки програми

Розробка клієнтського додатку проводилась за архітектурним патерном Model-View-Controller. Графічні компоненти та елементи інтерфейсу створені в файлі розмітки storyboard. Контролери використовують елементи view та надають функціональні можливості взаємодії з елементами керування на екрані. Класи типу "Model" являють собою основні сутності предметної області та використовуються для відображення на графічному інтерфейсі користувача. Екземпляри цих класів, а також об'єкти DTO беруть участь в комунікації між клієнтом та сервером.

Веб-сервер застосунку формує низка компонентів, що мають різне призначення. Контролери представляють мережевий рівень та займаються обробкою HTTP-запитів різних типів, сервіси являються інтерфейсами для реалізації своєрідної логіки додатку, а репозиторії беруть безпосередню участь в роботі з системою керування базами даних. Серверному застосуванню також притаманні класи, що реалізовують основні сутності системи, а також файли конфігурації і ресурсів.

3.3. Створення об'єктів та розробка головної програми

Графічний інтерфейс користувача складається з різних сцен, що містять в собі окремі елементи view. Основною сценою є Tab Bar Controller Scene. Контролер цієї сцени надає кінцевому користувачу можливість

переміщуватись між іншими елементами меню за допомогою переходів (segues). Елемент меню, що дозволяє шукати інших користувачів включає в себе навігаційний контролер. Це дає можливість перейти до конкретного профілю, обравши його в списку всіх користувачів. Profile Scene використовує перехід який є модальним – це означає, що користувач який не увійшов в систему, бачитиме перед собою view який є точкою призначення цього переходу, а саме - Login Page View. Останній view, в свою чергу містить перехід до вікна реєстрації, що здійснюється за умови натискання на кнопку “Register”.

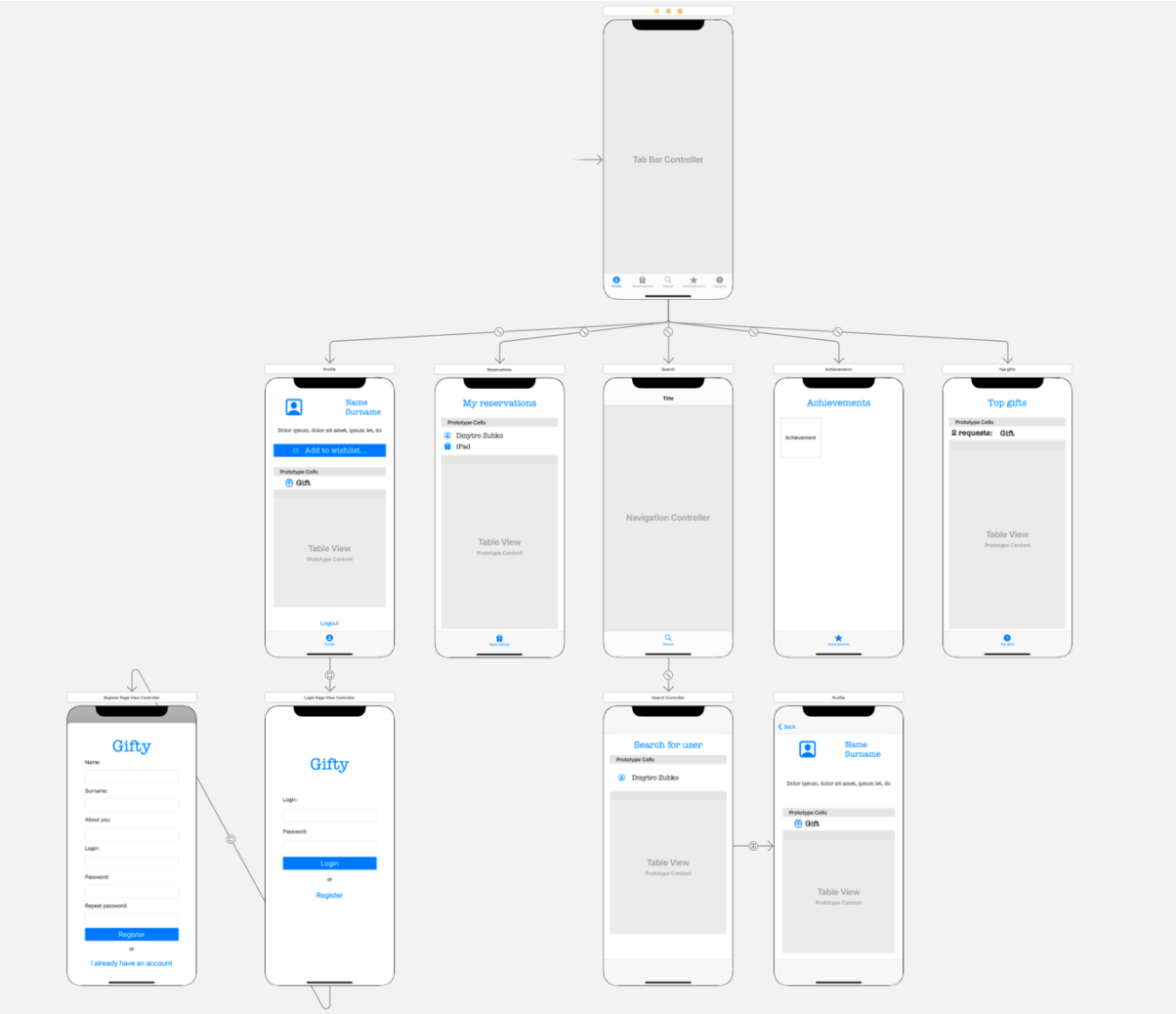

Рисунок 12 – основні складові графічного інтерфейсу користувача

Програмний код клієнтського застосунку поділений на три основні частини – presentation, model, viewcontroller. Presentation представляє класи, що розширюють стандартну поведінку графічних об’єктів чи елементів керування. Приклад такого класу - це RequestTableViewCell. Даний клас розширює можливості стандартної комірки списку з UIKit, та дозволяє використовувати додаткові параметри при створенні цього об’єкту (див. рис. 13).

```
import UIKit

class RequestTableViewCell: UITableViewCell{

    @IBOutlet weak var requestNameLabel: UILabel!
    @IBOutlet weak var isReservedImageView: UIImageView!

    override func awakeFromNib() {
        super.awakeFromNib()
    }

    override func setSelected(_ selected: Bool, animated: Bool) {
        super.setSelected(selected, animated: animated)
    }
}
```

Рисунок 13 – клас RequestTableViewCell з додатковими елементами UILabel та UIImageView

Класи-моделі представлені у вигляді основних сутностей. Прикладом такого класу є клас User, що містить всі необхідні параметри: ідентифікатор, прізвище, ім’я тощо.

Частина viewcontroller містить в собі класи, що безпосередньо взаємодіють з графічним інтерфейсом користувача через вихідні з’єднання, або дії, у випадку використання кнопок. ViewController класи також опрацьовують основну логіку мобільного додатку.

```
import UIKit
import Alamofire

class LoginPageViewController: UIViewController {

    @IBOutlet weak var loginTextField: UITextField!
    @IBOutlet weak var passwordTextField: UITextField!
    weak var userDelegate: UserDelegate?

    override func viewDidLoad() {
        super.viewDidLoad()
    }

    @IBAction func loginButtonTapped(_ sender: Any) {
        let userLogin = loginTextField.text
        let userPassword = passwordTextField.text
        //Create DTO and send request…
    }
}
```

Рисунок 14 – клас LoginPageViewController

Alamofire надає зручний інтерфейс для побудови та відправки HTTP-запитів на сервер (див. додаток «А»). Кожен запит, що надсилається за допомогою Alamofire працює в асинхронному режимі та не блокує основний потік програми, а використання closure-конструкцій дозволяє оновлювати дані в реальному часі.

Основними компонентами веб-серверу є контролери, сервіси, репозиторії та моделі. Всі ці компоненти логічно структуровані та розміщені в однойменних пакетах. Серверній частині застосунку також притаманні класи-утиліти, а також класи-мапери, що дозволяють створювати екземпляри об’єктів після отримання останніх з бази даних.

Контролери представлені у вигляді класів UserController, GiftController і AuthenticationController (див. додаток «Б»). Різновидом контролеру є також GlobalExceptionController, що перехоплює помилки, та надає змістовну відповідь клієнту (див. рис. 15).

```
//imports
@ControllerAdvice
public class GlobalExceptionController {

    @ExceptionHandler(LoginIsUsedException.class)
    public ResponseEntity<Object> loginIsUsedException() {
        return new ResponseEntity<>(
                "Login is already used!",
                HttpStatus.FORBIDDEN);
    }

    @ExceptionHandler(BadCredentialsException.class)
    public ResponseEntity<Object> invalidCredentialsException() {
        return new ResponseEntity<>(
                "Invalid login or password!",
                HttpStatus.FORBIDDEN);
    }


}
```

Рисунок 15 – клас GlobalExceptionController

Класи, що знаходяться в пакеті repository представляють рівень доступу до даних. Кожен такий клас розширює інтерфейс IGenericCRUD, та реалізовує методи базових операцій (створення, отримання та видалення об’єктів). DAO-класи також використовують NamedParameterJdbcTemplate для встановлення з’єднання з базою даних і обробки помилок. Тексти SQL-запитів знаходяться в ресурсах самого додатку (див. додаток «В»), та використовуються в DAO-класах за допомогою @PropertySource анотації.

3.4. Тестування програми і результати її виконання

Робота з мобільним додатком розпочинається зі входу в особистий профіль – користувач повинен надати дані для входу, або ж перейти безпосередньо до реєстрації. Якщо ж вхід на поточному пристрої вже був виконаний, користувач потрапляє на меню свого профілю після запуску додатку.

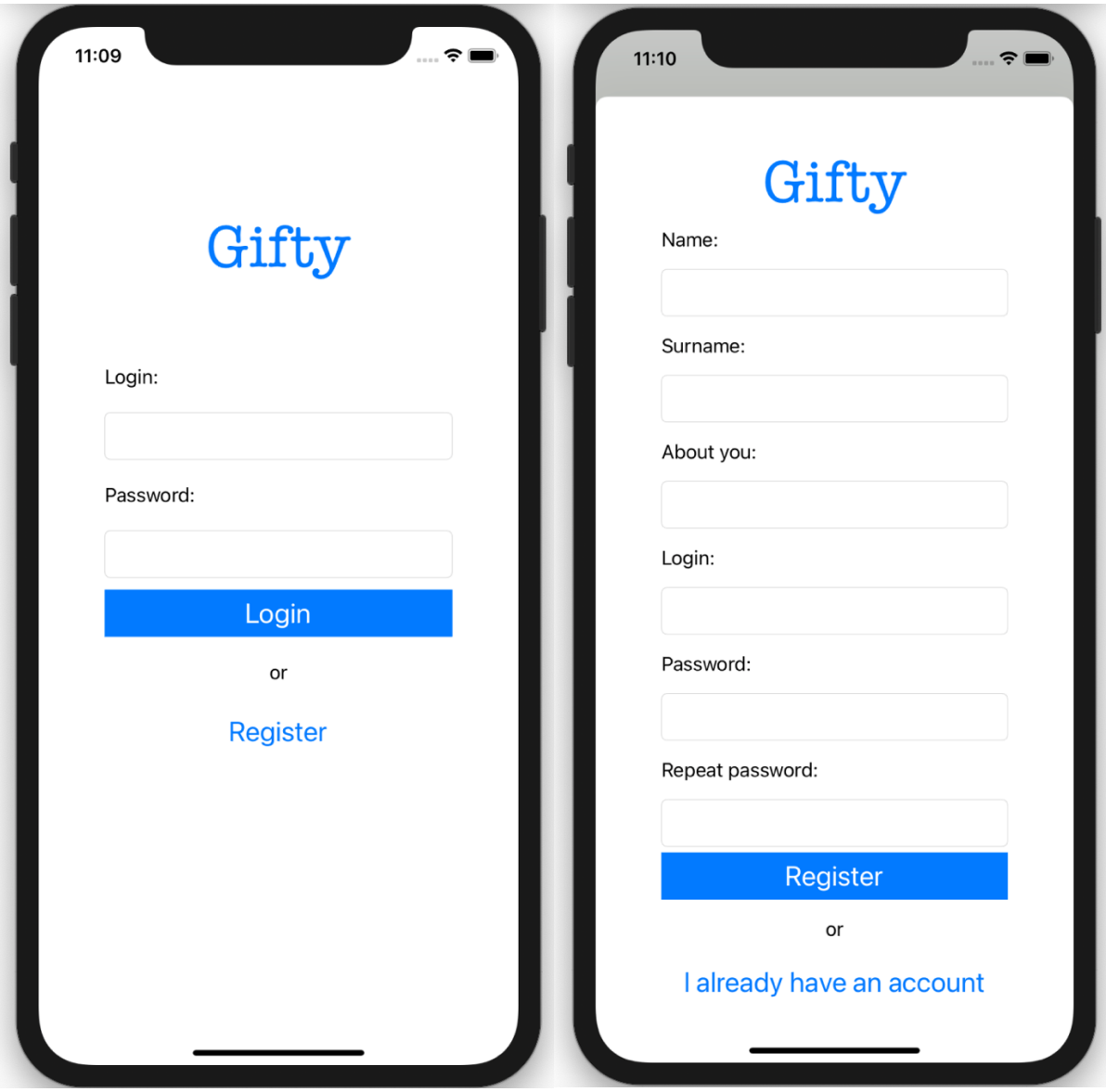

Рисунок 16 – вікно входу в систему, та вікно реєстрації

У разі виникнення певних помилок, що спричинені непередбачуваною поведінкою користувача, або проблемами сервера, буде відображено спливаюче вікно з текстом помилки. Прикладом такого вікна може бути повідомлення про невірні дані для входу, або повідомлення, що вказує на помилку при реєстрації спричинену вже існуючим логіном у базі даних (див. рис. 17).

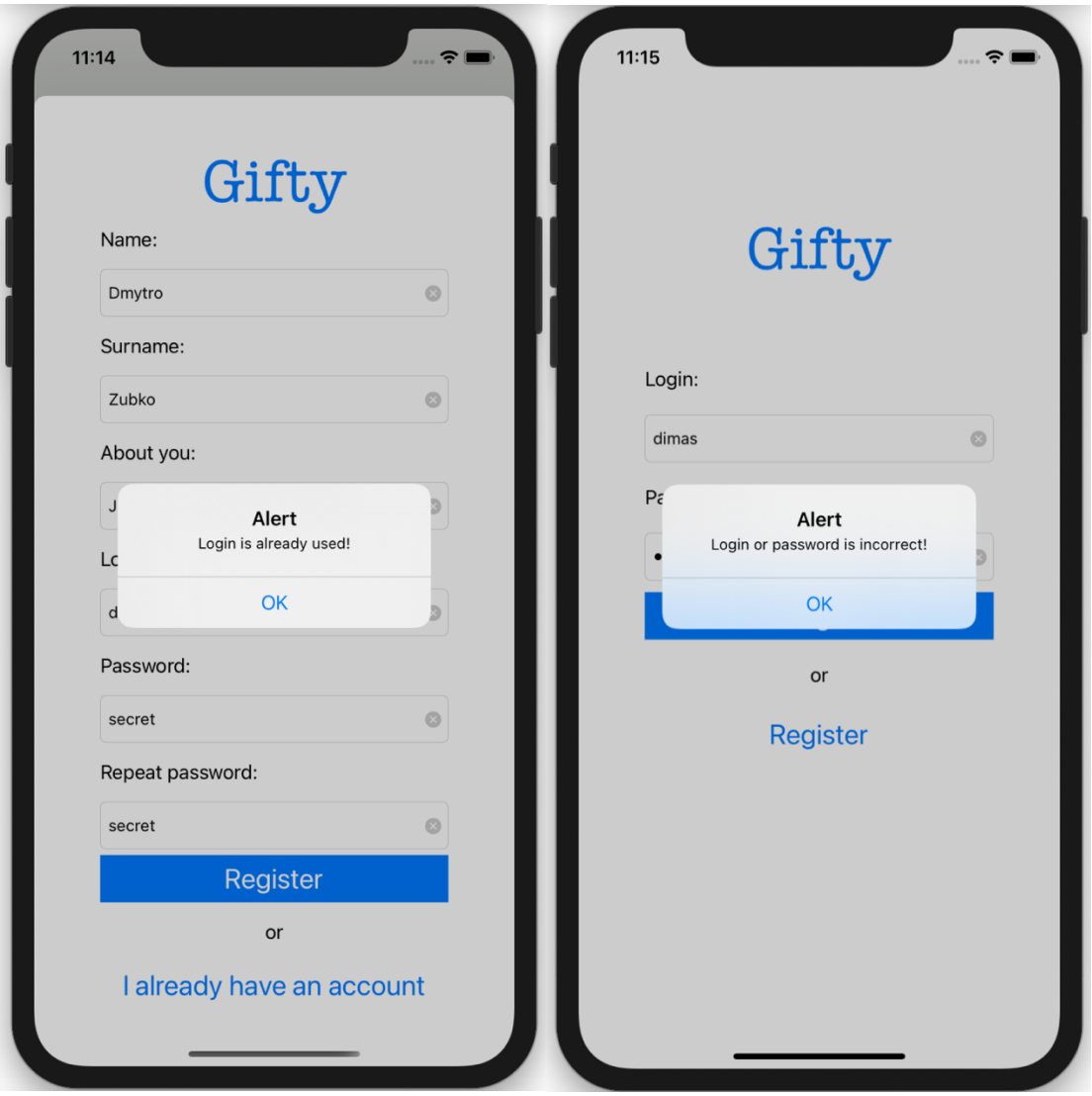

Рисунок 17 – приклади спливаючих вікон з текстом помилки

За умови вірно введених даних при вході в систему, користувач потрапляє до головного меню додатку. Елементи навігації розміщені в нижній частині екрану, за замовчуванням, показаний особистий профіль користувача. В даному елементі меню відображається інформація щодо самого користувача: персональні дані, список побажань, кнопка створення нового побажання і кнопка виходу з акаунту. Для створення нової позиції в списку побажань, необхідно натиснути на відповідну кнопку. Після заповнення назви подарунку, новий елемент буде додано на сторінку. Вже зарезервовані подарунки іншими користувачами системи позначені

зафарбованим зображенням біля самої назви. Для видалення чи підтвердження отримання певного подарунку, користувач може провести пальцем в ліво навпроти відповідного елементу. Підтвердивши дію в спливаючому вікні, позиція буде видалена зі списку побажань.

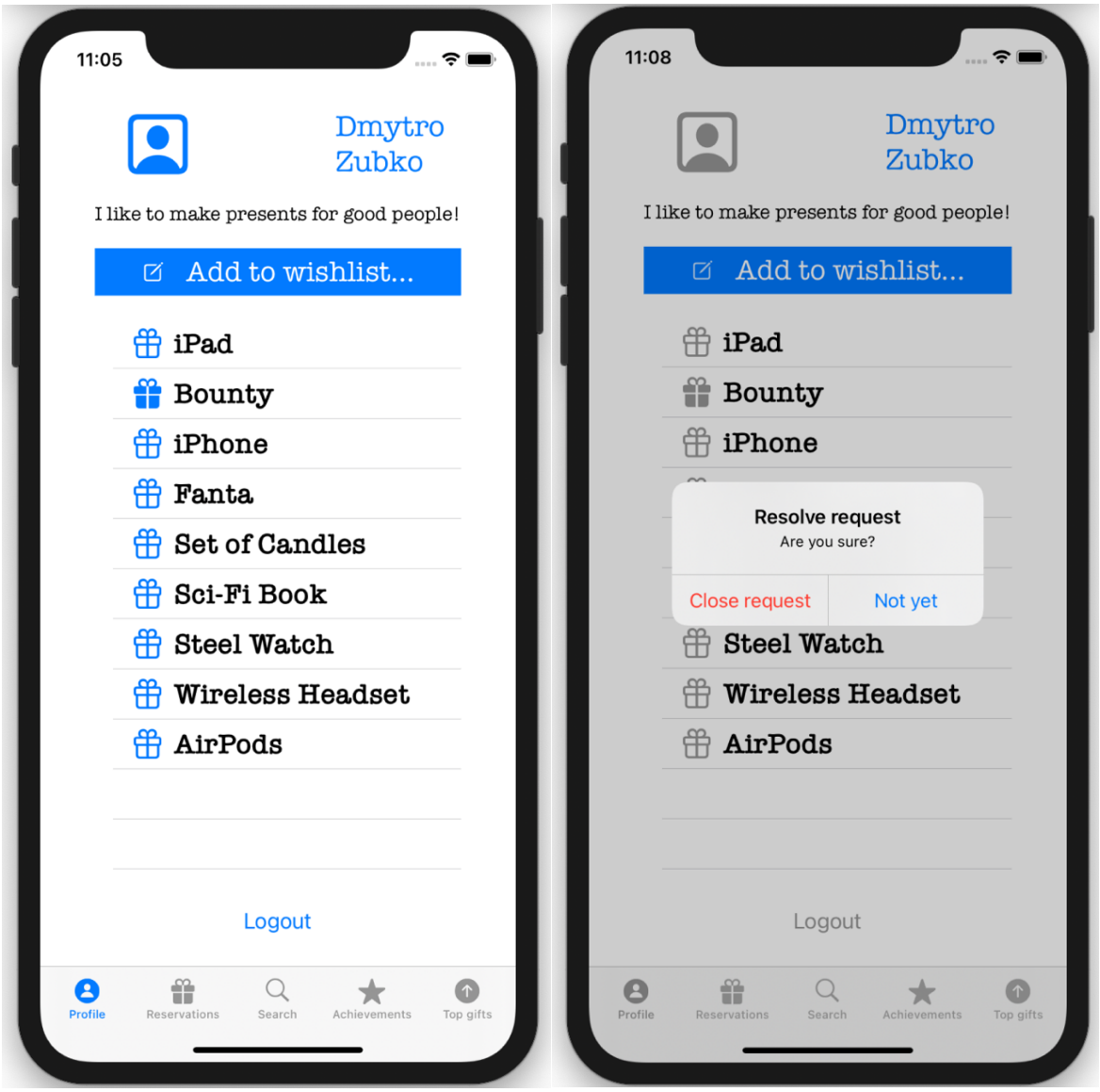

Рисунок 18 – особистий профіль користувача

Для пошуку користувачів та подальшої резервації подарунків можна скористатись елементом меню з назвою “Search”. Після пошуку конкретної особи, користувач може перейти до персонального профілю цієї людини, та за потреби зарезервувати подарунок зі списку побажань за допомогою

вищезгаданого жесту. Вже зарезервовані елементи іншими користувачами не можуть бути повторно заброньовані.

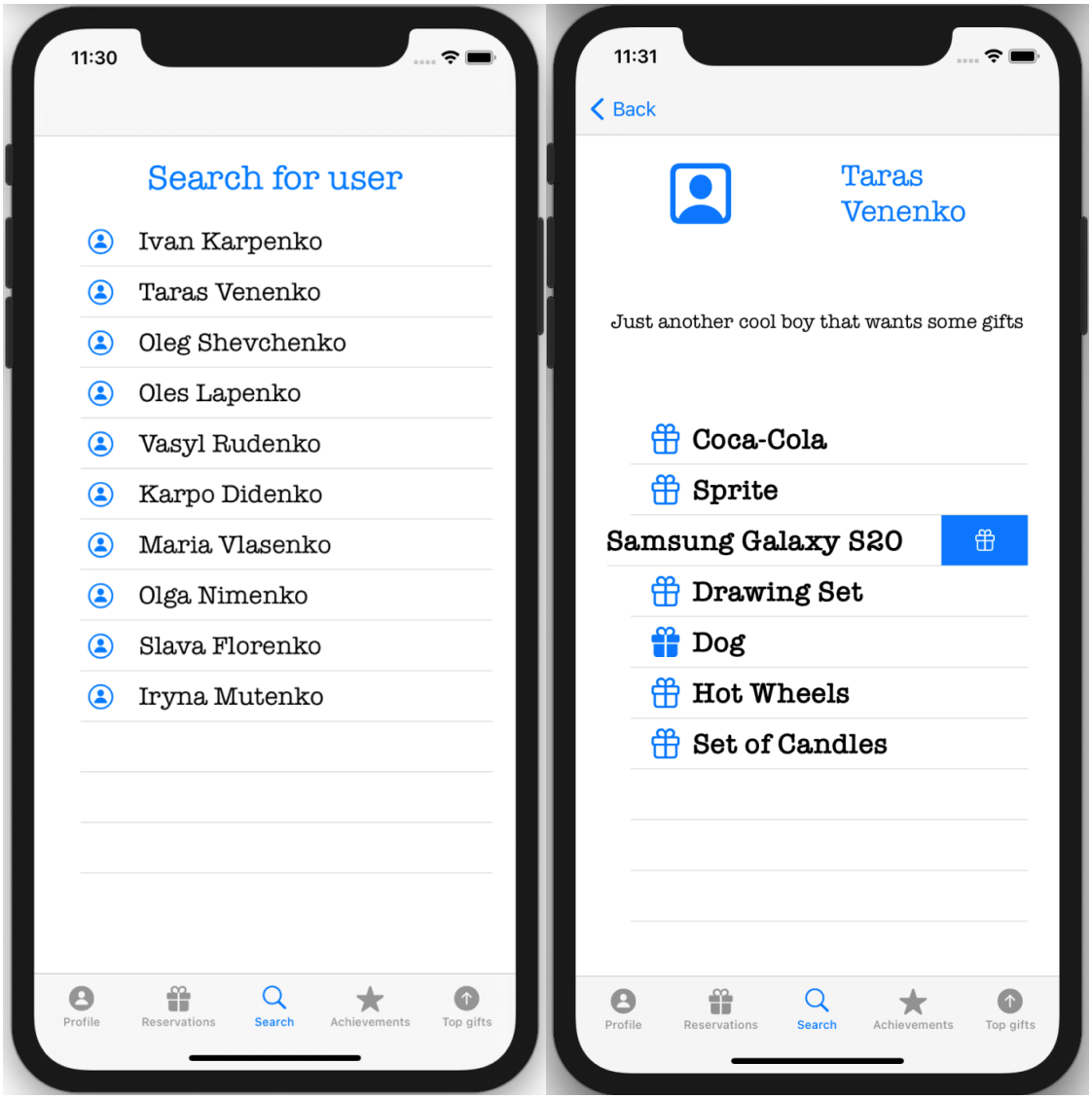

Рисунок 19 – пошук користувачів та резервація побажання

Список зарезервованих користувачем побажань можна віднайти в окремому елементі навігації під назвою “Reservations”. В даному списку надана інформація про назви подарунків та їх замовників. Резервація може бути скасована користувачем за допомогою змаху вліво. Після підтвердження скасування, позиція буде доступна для резервації іншими користувачами, а сам елемент списку буде видалено. Меню “Top Gifts”

надає змогу проглянути рейтинг найпопулярніших замовлень серед усіх користувачів. Рейтинг формується на основі кількості повторюваних побажань, та відсортований від найпопулярніших до найменш затребуваних.

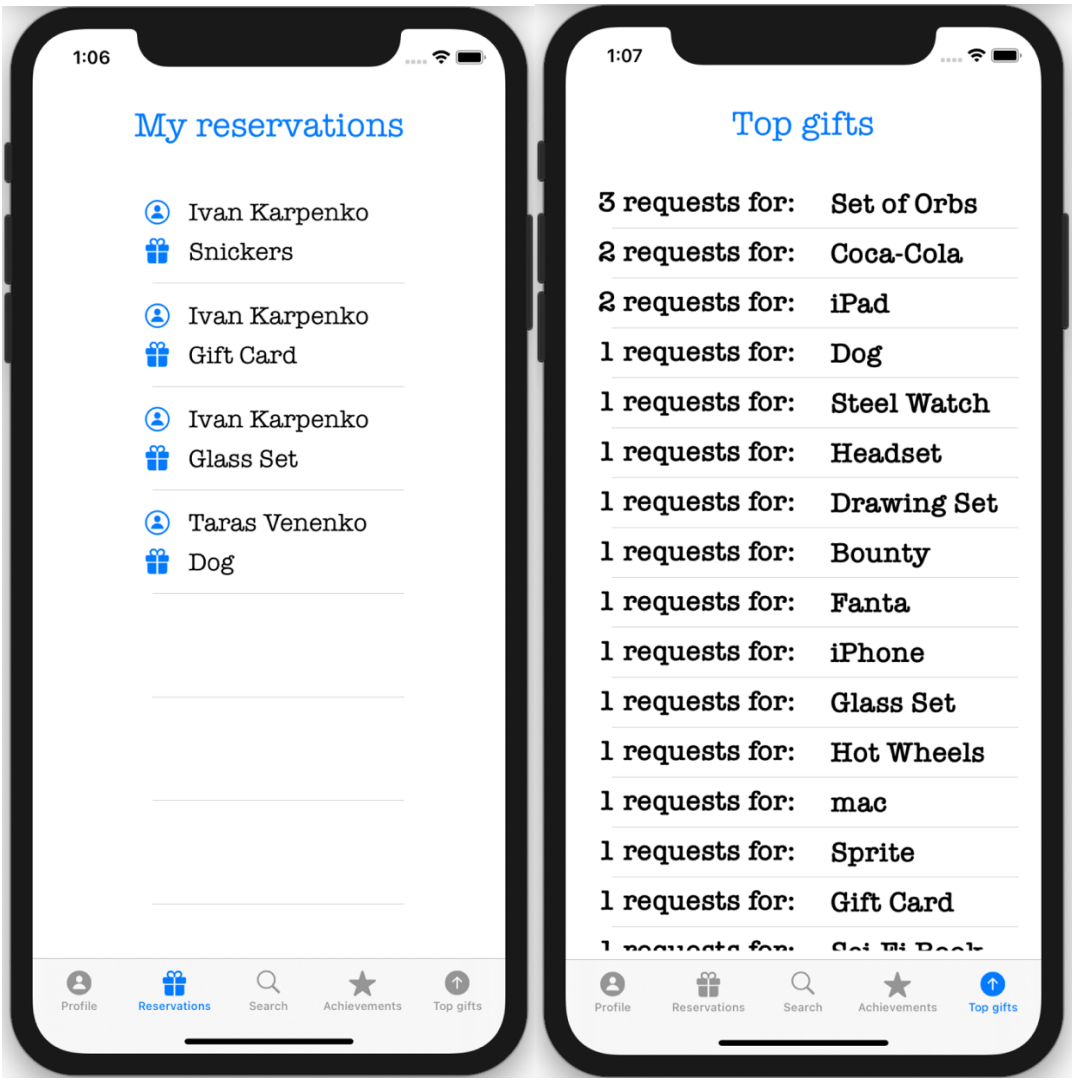

Рисунок 20 – резервації користувача та рейтинг подарунків

Висновок

В результаті виконання курсової роботи створено мобільний додаток на базі iOS, що реалізовує функціональні можливості, зазначені у постановці завдання. Висвітлено сучасний стан мобільних застосунків на ринку та досліджено основні принципи проектування програмного забезпечення даного типу. В рамках ведення процесу основної розробки розглянуто функціональні можливості сучасних бібліотек та інструментів, що використовуються на різних етапах проектування програмного забезпечення.

Застосунок, що є кінцевим результатом даної курсової роботи може бути з легкістю розширений та доповнений як з клієнтської частини, так і з боку сервера завдяки використаним патернам. Ідеєю для розширення може слугувати створення окремої версії додатку на базі операційної системи Android, з використанням вже існуючої серверної частини.

Список використаної літератури

1. Статистика онлайн-магазину Google Play [Електронний ресурс]: https://www.appannie.com/en/insights/market-data/google-play-all-time/
2. Статистика додатків в App Store [Електронний ресурс]: https://www.digger.ru/news/kolichestvo-prilozhenij-v-app-store-prevysilo-15-milliona
3. Додаток Christmas Gift List [Електронний ресурс]: https://play.google.com/store/apps/details?id=com.xmaslist
4. Додаток GiftPlanner [Електронний ресурс]: https://apps.apple.com/us/app/giftplanner/id479882306 - ?platform=iphone
5. Swift programming language [Електронний ресурс]: https://www.apple.com/swift/
6. SwiftUI [Електронний ресурс]: https://developer.apple.com/xcode/swiftui/
7. App development with UIKit [Електронний ресурс]: https://developer.apple.com/documentation/uikit/about_app_development_with_uikit
8. Model-View-Controller application design [Електронний ресурс]: https://techterms.com/definition/mvc
9. Usage statistics of server-side programming languages [Електронний ресурс]: https://w3techs.com/technologies/overview/programming_language
10. Spring Framework [Електронний ресурс]: https://spring.io/why-spring
11. Spring Annotation Type Component [Електронний ресурс]: https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/stereotype/Component.html
12. Spring Security access control framework [Електронний ресурс]: https://spring.io/projects/spring-security

13. Alamofire. Бібліотека для роботи з мережею у Swift [Електронний ресурс]: https://github.com/Alamofire/Alamofire - readme
14. Інтегроване середовище розробки IntelliJ IDEA [Електронний ресурс]: https://www.jetbrains.com/idea/
15. Система керування базами даних PostgreSQL [Електронний ресурс]: https://www.postgresql.org/about/

ДОДАТОК А. Клієнтський код – формування HTTP-запиту з використанням Alamofire

```
AF.request("http://localhost:9990/api/v1/auth/login",
                method: .post, parameters: loginDTO,
encoder:JSONParameterEncoder.default).responseDecodable(of: User.self) { response in

          if (response.response?.statusCode == 200){
             let user = response.value
             self.userDelegate?.userTransfered(user: user!, loginDto: loginDTO)
             UserDefaults.standard.setLoggedIn(value: true)
             self.dismiss(animated: true, completion: nil)
          } else if (resposne.response?.statusCode == 403){
             let alert = UIAlertController(title: "Error", message: "Login or password is
incorrect!", preferredStyle: UIAlertController.Style.alert)
             let okAction = UIAlertAction(title: "OK", style: UIAlertAction.Style.default) { (_)
in
             }
             alert.addAction(okAction)
             self.present(alert, animated: true, completion: nil)
          } else {
             let alert = UIAlertController(title: "Error", message: "Unknown error! Please try
again", preferredStyle: UIAlertController.Style.alert)

             let okAction = UIAlertAction(title: "OK", style: UIAlertAction.Style.default) { (_)
in
             }
             alert.addAction(okAction)
             self.present(alert, animated: true, completion: nil)
          }
       }
```

ДОДАТОК Б. Серверний код - GiftController

```
package com.megacorp.gifty.controller;

import com.megacorp.gifty.entity.GiftRating;
import com.megacorp.gifty.entity.GiftReservation;
import com.megacorp.gifty.entity.Gift;
import com.megacorp.gifty.entity.GiftRequest;
import com.megacorp.gifty.service.impl.GiftServiceImpl;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/v1")
public class GiftController {

    private GiftServiceImpl giftService;

    @Autowired
    public GiftController(final GiftServiceImpl giftService) {
        this.giftService = giftService;
    }

    @PostMapping(value = "user/{userId}/request")
    public ResponseEntity createUserGiftRequest(
            @PathVariable final int userId,
            @RequestBody final Gift gift) {
        giftService.insert(gift, userId);
        return new ResponseEntity(HttpStatus.CREATED);
    }

    @DeleteMapping(value = "user/{userId}/request/{giftId}")
    public ResponseEntity removeRequest(
            @PathVariable final int userId,
            @PathVariable final int giftId) {
        giftService.removeRequest(userId, giftId);
        return new ResponseEntity(HttpStatus.CREATED);
    }

    @GetMapping("/user/{userId}/request")
    public ResponseEntity<List<GiftRequest>> getUserGiftRequests(
            @PathVariable int userId) {
        return new ResponseEntity<>(giftService.getRequests(userId), HttpStatus.OK);
    }
```

```
    @GetMapping("/user/{userId}/reservation")
    public ResponseEntity<List<GiftReservation>> getUserReservations(
            @PathVariable int userId) {
        return new ResponseEntity<>(giftService.getUserReservations(userId), HttpStatus.OK);
    }

    @PatchMapping("/user/{requesterId}/reserve/{giftId}")
    public ResponseEntity reserveUserGiftRequest(
            @RequestParam int activeUserId,
            @PathVariable int requesterId,
            @PathVariable int giftId) {
        giftService.reserveRequest(activeUserId, requesterId, giftId);
        return new ResponseEntity(HttpStatus.OK);
    }

    @DeleteMapping("/user/{requesterId}/reserve/{giftId}")
    public ResponseEntity removeReservation(
            @RequestParam int activeUserId,
            @PathVariable int requesterId,
            @PathVariable int giftId) {
        giftService.removeReservation(activeUserId, requesterId, giftId);
        return new ResponseEntity(HttpStatus.OK);
    }

    @GetMapping("/gift/rating")
    public ResponseEntity<List<GiftRating>> getGiftRating() {
        return new ResponseEntity<>(giftService.getRating(), HttpStatus.OK);
    }

}
```

ДОДАТОК В. Ресурс з текстами SQL-запитів

```
find_gift_by_name = SELECT id, name \
                    FROM gifts \
                    WHERE name = :name

insert_gift       = INSERT INTO gifts \
                    (name) \
                    VALUES (:name)

insert_request    = INSERT INTO users_gifts \
                    (gift_id, requester_id) \
                    VALUES (:gift_id, :requester_id)

reserve_request   = UPDATE users_gifts \
                    SET reserver_id = :reserver_id \
                    WHERE gift_id = :gift_id AND requester_id = :requester_id

remove_reservation = UPDATE users_gifts \
                    SET reserver_id = NULL \
                    WHERE gift_id = :gift_id AND requester_id = :requester_id AND
reserver_id = :reserver_id

remove_request    = DELETE FROM users_gifts \
                    WHERE gift_id = :gift_id AND requester_id = :requester_id

find_user_requests = SELECT g.id, g.name, ug.requester_id, ug.reserver_id \
                    FROM gifts g \
                    INNER JOIN users_gifts ug on g.id = ug.gift_id \
                    WHERE ug.requester_id = :requester_id

find_user_reservations = SELECT g.id, g.name, u.name AS user_name, u.surname,
ug.requester_id, ug.reserver_id \
                    FROM gifts g \
                    INNER JOIN users_gifts ug on g.id = ug.gift_id \
                    INNER JOIN users u on ug.requester_id = u.id \
                    WHERE ug.reserver_id = :reserver_id

list_gift_rating  = SELECT ug.gift_id, g.name, COUNT(ug.gift_id) AS rating \
                    FROM users_gifts ug \
                    INNER JOIN gifts g on g.id = ug.gift_id \
                    GROUP BY g.name, ug.gift_id
```