МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ЕЛЕКТРОНІКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК
СЕКЦІЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ПРОЕКТУВАННЯ

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: «Мобільний додаток на базі операційної системи iOS для організації пасажирських перевезень компанією «Еліт Експресс»»

за спеціальністю 122 «Комп'ютерні науки»,
освітньо-професійна програма «Інформаційні технології проектування»

Виконавець роботи: студент групи ІТ-71 Акименко Владислав Валерійович

Кваліфікаційна робота бакалавра
захищена на засіданні ЕК
з оцінкою _____ «__» _____2021 р.

Науковий керівник _____ к.т.н., доц., Марченко А. В.
(підпис) (науковий ступінь, вчене звання, прізвище та ініціали)

Голова комісії _____ Шифрін Д. М.
(підпис) (науковий ступінь, вчене звання, прізвище та ініціали)

Засвідчую, що у цій дипломній роботі немає
запозичень з праць інших авторів
без відповідних посилань.

Студент _____
(підпис)

Суми-2021

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук
Секція інформаційних технологій проектування
Спеціальність 122 «Комп'ютерні науки»
Освітньо-професійна програма «Інформаційні технології проектування»

ЗАТВЕРДЖУЮ
Зав. секцією ІТП

В. В. Шендрик
«__» _____ 2021 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА СТУДЕНТУ

Акименко Владислав Валерійович

1 Тема роботи *Мобільний додаток на базі операційної системи iOS для організації пасажирських перевезень компанією «Еліт Експресс»*

керівник роботи *Марченко Анна Вікторівна, к.т.н., доцент*,

затверджені наказом по університету від « 14 » квітня 2021 р. №0181-VI

2 Строк подання студентом роботи «7» червня 2021 р.
3 Вхідні дані до роботи технічне завдання на розробку мобільного додатку

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) аналіз предметної області, проектування мобільного додатку, розробка мобільного додатку

5 Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) постановка задачі, дослідження продуктів-аналогів, вимоги до додатку, моделювання процесу бронювання місць, декомпозиція процесу бронювання місць, декомпозиція процесу вибору даних користувачем, діаграма бази даних, архітектура мобільного додатку, технології та інструменти розробки, екрани мобільного додатку, створення обмежень для елементів, приклад роботи додатку, сторінка додатку в AppStore, акт впровадження, висновки

6. Консультанти розділів роботи:

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7.Дата видачі завдання_____01.10.2020_____

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Визначення мети проекту	01.10.2020-30.10.2020	
2	Аналіз додатків-аналогів	01.11.2020-20.11.2020	
3	Узгодження технічного завдання	21.11.2020-31.12.2020	
4	Вибір засобів реалізації додатку	01.01.2021-15.01.2021	
5	Створення прототипу додатку	16.01.2021-10.02.2021	
6	Створення екранів додатку	11.02.2021-28.02.2021	
7	Розробка програмного коду	01.03.2021-01.04.2021	
8	Тестування мобільного додатку	02.04.2021-13.04.2021	
9	Розміщення в магазині додатків	14.04.2021-24.04.2021	
10	Оформлення документації	25.04.2021-07.06.2021	

Студент _____ Акименко В.В.
(підпис)

Керівник роботи _____ к.т.н., доц. Марченко А.В.
(підпис)

РЕФЕРАТ

Тема кваліфікаційної роботи бакалавра «Мобільний додаток на базі операційної системи iOS для організації пасажирських перевезень компанією «Еліт Експресс»».

Пояснювальна записка складається зі вступу, 3 розділів, висновків, списку використаних джерел із 10 найменувань, додатків. Загальний обсяг роботи − 79 сторінок, у тому числі 36 сторінок основного тексту, 1 сторінка списку використаних джерел, 42 сторінки додатків.

Кваліфікаційну роботу бакалавра присвячено розробці мобільного додатку організації пасажирських перевезень компанією «Еліт Експресс».

В роботі проведено дослідження області мобільних додатків, виконано аналіз додатків-аналогів, сформовано технічне завдання на розробку, побудовано модель процесу бронювання місць, виконано опис архітектури додатку, етапів розробки та продемонстровано роботу додатку.

Результатом проведеної роботи є розроблений мобільний додаток на базі ОС iOS для організації пасажирських перевезень.

Практичне значення роботи полягає у автоматизації важливих бізнес-процесів з організації пасажирських перевезень, зменшенню людських факторів, підвищенню якості діяльності компанії «Еліт Експресс».

Ключові слова: мобільний додаток, пасажирські перевезення, бронювання, iOS, Swift, Xcode

ЗМІСТ

ВСТУП

Сьогочасний швидкий темп життя та стрімкий розвиток науково-технічного прогресу дали міцний поштовх для розвитку технологій мобільного зв’язку та технологічних можливостей самих мобільних пристроїв. Вони є невід’ємною частиною нашого побуту. Ми користуємося смартфонами скрізь: вдома, на роботі, у дорозі, під час навчання та відпочинку. З плином часу все більше зростає їх функціонал.

Тепер неможливо заявити, що телефон потрібен лише для дзвінків, оскільки поява мобільних додатків збільшила потенціал смартфонів. Розваги, соціальні мережі, освіта, подорожі, спорт, послуги доставки – це лише частина із величезного списку сфер використання мобільних додатків.

Ще якихось 20 років тому людям доводилося їхати на автовокзал або телефонувати диспетчеру, щоб забронювати квитки. З появою веб-сайтів ця проблема відійшла на задній план, але повністю не зникла, оскільки користування мобільною версією сайту є досить дискомфортним.

Мобільні додатки дозволяють ліквідувати можливі незручності при користуванні веб-сайтами шляхом налагодження тісного зв’язку розробника з користувачем, що дає можливість створити зручний та інтуїтивно зрозумілий інтерфейс. Користувачам більше не потрібно заходити в браузер та шукати потрібний їм веб-сайт, що значною мірою пришвидшує взаємодію з послугою та заощаджує час.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Актуальність проблеми

Стрімкий розвиток технологій зумовив появу мобільних телефонів, що в свою чергу дозволило корпораціям звернути увагу на сегмент мобільних додатків для покращення своїх позицій на ринку. Статистика за 2020 рік показує, що серед користувачів, які мають смартфон, популярними операційними системами є Android та iOS. Близько 72% від усього мобільного ринку займають смартфони на базі ОС Android, в той час як на iOS 27,5%. Але незважаючи на таку перевагу по кількості користувачів, ситуація з доходами Apple App Store та Google Play протилежна: $72 мільярдів проти $55 мільярдів [1]. Це означає, що створення мобільних додатків на обидві системи все ще є актуальним.

Можливий функціонал мобільних додатків постійно розширюється, що дозволяє компаніям запускати власні додатки для приваблення клієнтів, зокрема в сфері пасажирських перевезень. Користувачеві достатньо 1 раз завантажити додаток, щоб отримати доступ до швидкого бронювання місць з можливістю збереження історії поїздок та зі своїм профілем без необхідності кожного разу заходити на веб-сайт.

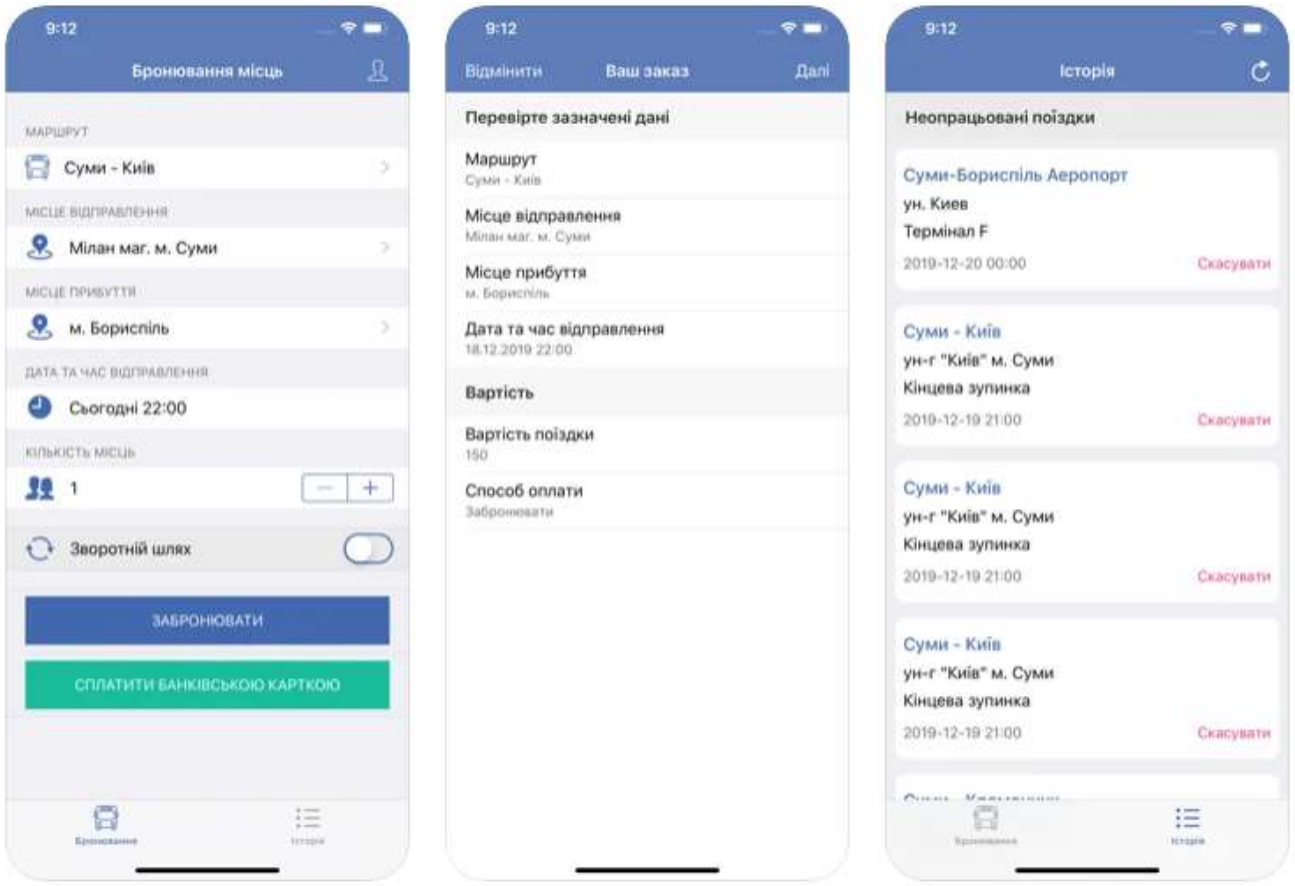
Під час розробки мобільного додатку варто враховувати його сферу використання для створення ліпшої оптимізації. Так, зокрема, мобільні додатки в свою чергу є кращою альтернативою мобільним версіям веб-сайтів. Серед їх переваг можна відзначити такі:

– швидкодія та продуктивність;
– зручний та інтуїтивний інтерфейс;
– відсутність необхідності постійного доступу до Інтернету;
– розширений функціонал;
– безпека персональних даних.

1.2 Аналіз мобільних додатків-аналогів

Під час роботи над проектом було виконано огляд існуючих мобільних додатків.

«Сервис Люкс м. Суми» – мобільний додаток, який дозволяє виконати бронювання подорожі із міста Суми до Києва, Харкова, Кременчука, Одеси, Кракова та у зворотному напрямку. Має зрозумілий та простий дизайн. Функціоналу додатку дозволяє виконати бронювання місць за обраним маршрутом, виконати оплату картою із додатку, переглянути історію поїздок (рис 1.1).

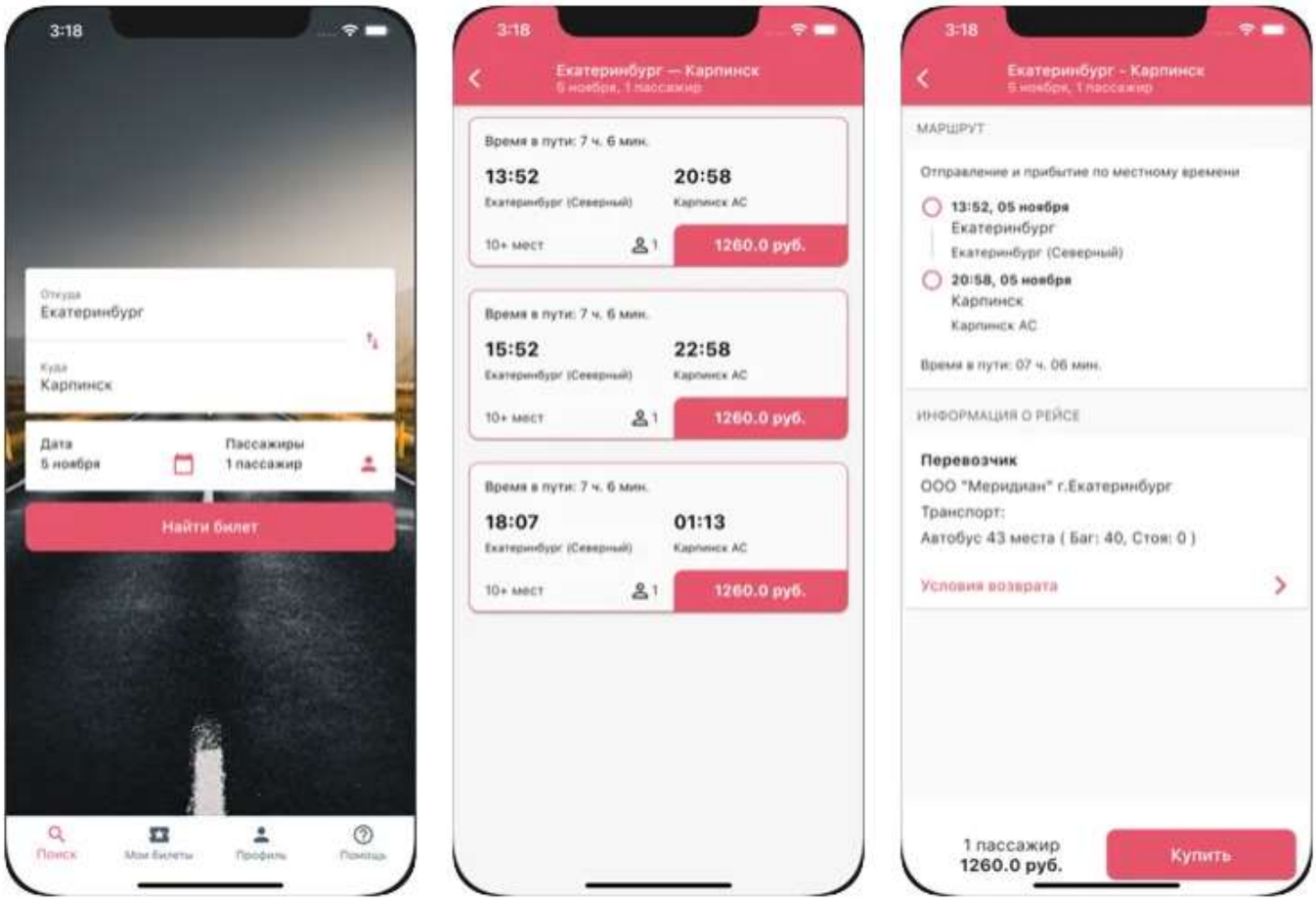


а б в

Рисунок 1.1 – Екрани додатку «Сервис Люкс м. Суми»:
головний екран бронювання (а), екран підтвердження бронювання (б),
історія поїздок (в)

До недоліків додатку можна віднести дублювання поїздки на екрані історії поїздок при бронюванні 2 і більше місць. У разі відміни бронювання необхідно скасувати кожне місце окремо, що може займати чимало часу.

«PROBUS – автобусные перевозки» – мобільний додаток організації пасажирських перевезень в Свердловській та Челябінській областях (рис 1.2). Додаток має сучасний та зручний дизайн. Користувач може виконати пошук рейсів за вказаним маршрутом.



а б в

Рисунок 1.2 – Екрани додатку«PROBUS»: екран пошуку рейсів (а), екран переліку рейсів за вказаними параметрами (б), екран бронювання (в)

Серед недоліків додатку можна відзначити відсутність локалізації, територіальне охоплення перевезень, оскільки додаток націлений на використання в 2 областях Росії та не є доцільним для використання жителям України.

За результатами аналізу додатків було складено порівняльну таблицю розглянутих аналогів та майбутнього додатку. Критерії для порівняння представлені в таблиці 1.1

Таблиця 1.1 – Порівняльна характеристика аналогів

Критерій	Сервис Люкс м. Суми	PROBUS – автобусные перевозки	Еліт Експресс
Інтерфейс	+	+	+
Розмір додатку	+	+-	+
Функціонал	+	+	+
Адаптивність	+	+	+
Швидкість реагування	+-	-	+
Актуальність для жителів України	+	-	+

1.3 Постановка задачі

Провівши аналіз предметної області та виконавши огляд мобільних додатків аналогів було встановлено мету проекту – розробка мобільного додатку на базі ОС iOS для організації перевезень для компанії «Еліт Експресс». Функціонал додатку має містити наступні функції:

– бронювання місць за маршрутом Суми-Київ
– оплата місць із додатку;
– перегляд історії поїздок;
– оплата або скасування поїздки;
– редагування профілю.

Для досягнення поставленої мети необхідно виконати наступні задачі:

– створення прототипу додатку;
– розробка макету додатку в IDE Xcode;
– розробка програмного коду;

– тестування роботи додатку;
– завантаження додатку в магазин App Store.

Розробка мобільного додатку буде виконуватися в середовищі розробки Xcode. Додаток буде написаний мовою програмування Swift, формат даних для отримання із серверу буде у вигляді JSON.

Повний список вимог до розроблюваного додатку наведений в технічному завданні в додатку А.

2 ПРОЕКТУВАННЯ МОБІЛЬНОГО ДОДАТКУ

Після проведення аналізу актуальності проблеми, встановлення мети проекту було визначено задачі та засоби реалізації мобільного додатку. Наступним етапом є процес проектування мобільного додатку, що слугує основою для коректної програмної реалізації продукту.

2.1 Діаграма нотації IDEF0 основного процесу мобільного додатку

Методологія IDEF0 знайшла широке використання завдяки простій та наглядній графічній нотації побудови моделі. Функції системи відображаються у вигляді прямокутників, в той час як основні зв’язки між зовнішніми та внутрішніми частинами системи у вигляді стрілок [2]. Діаграма дозволяє показати послідовний потік виконання головного процесу продукту.

На рисунку 2.1 представлена діаграма А-0 верхнього рівня. Вона складається з одного функціонального блоку, вхідних та вихідних даних, елементів управління та механізми для виконання функції.

У ході проведення аналізу предметної області були сформовані наступні дані:

– вхідні дані: прізвище користувача, номер телефону, база даних маршрутів, запит користувача, форма авторизації;

– вихідні дані: оновлена база даних, повідомлення про забронюване місце, оновлена історія поїздок, квитанція оплати бронювання;

– елементи управління: шаблон оплати, шаблон замовлення, шаблон квитка;

– механізми: мобільний додаток, платіжна система WayForPay, Диспетчер, СУБД MySQL, сервер.

Рисунок 2.1 – Контекстна діаграма А-0

Наступним етапом необхідно виконати декомпозицію діаграми А-0 шляхом декомпозиції головної функції на підпроцеси, що виконуються послідовно.

Таким чином було визначено наступні процеси:

- авторизація користувача;
- завантаження шаблону замовлення;
- вибір даних користувачем;
- підтвердження бронювання;
- оплата місць;
- перегляд історії поїздок.

Декомпозиція діаграми А-0 представлена на рисунку 2.2.

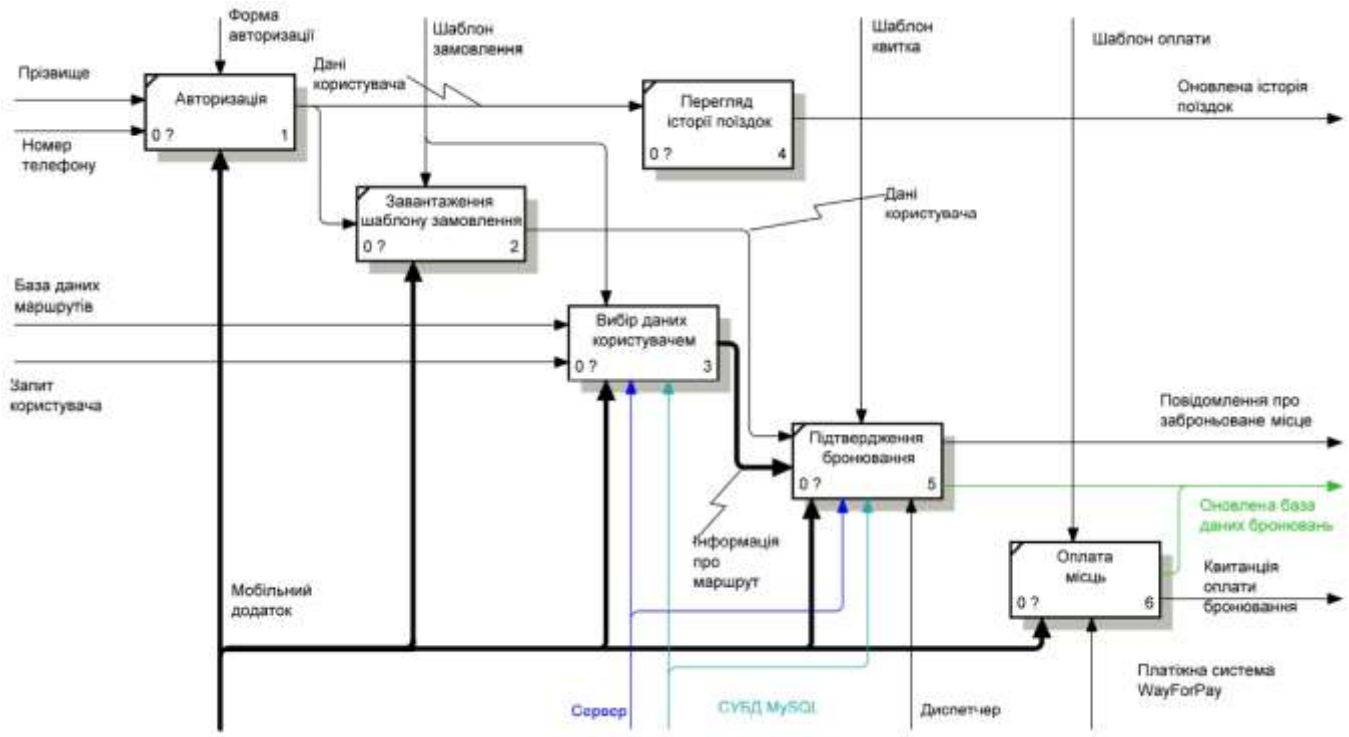


Рисунок 2.2 – Декомпозиція діаграми A-0

Для процесу «Вибір даних користувачем» було виконано декомпозицію. У результаті процес був розбитий на 6 процесів: «вибрати маршрут», «вибрати пункт відправлення», «вибрати пункт прибуття», «вибрати дати та часу», «вибрати кількість місць», «додати коментар». На рисунку 2.3 представлено декомпозицію процесу «Вибір даних користувачем».

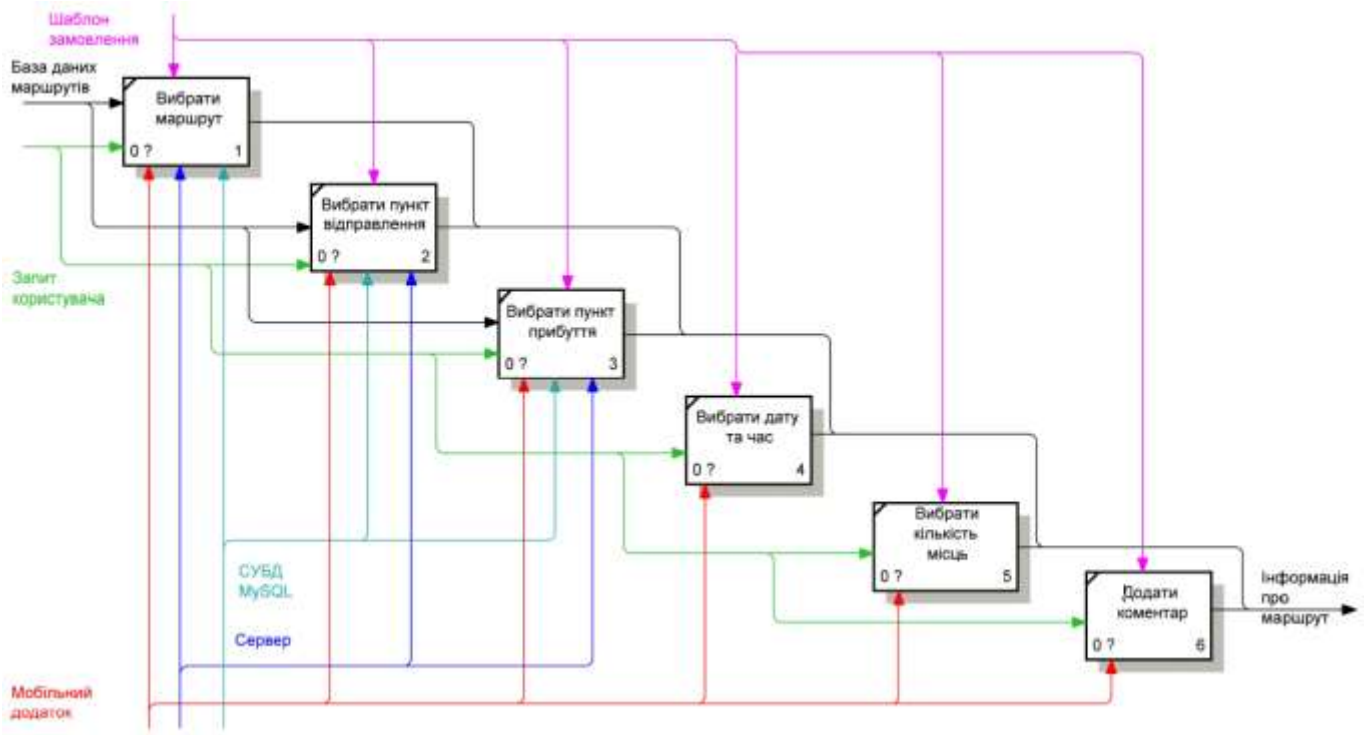


Рисунок 2.3 – Декомпозиція процесу «Вибір даних користувачем»

2.2 Діаграма варіантів використання мобільного додатку

UML - уніфікована мова моделювання (Unified Modeling Language) - це система позначень, яку можна застосовувати для об'єктно-орієнтованого аналізу і проектування.

Її можна використовувати для візуалізації, специфікації, конструювання та документування програмних систем.

Одним із елементів UML є діаграма використання, що дозволяє відобразити відносини між акторами та прецедентами та описати розроблювану систему на концептуальному рівні.

Прецедент – один із можливих сценаріїв взаємодії акторів з розробленим продуктом.

Окрім цього, діаграма описує функціональні вимоги до системи та допомагає узгодити ТЗ [3].

Оскільки для розроблюваного додатку немає розмежування до інформації то всі користувачі матимуть однаковий доступ. Основними операціями для актора «Користувач» є:

– редагування профілю;
– бронювання місць;
– оплата бронювання;
– перегляд історії поїздок;
– перегляд інформації про маршрути.

Діаграма варіантів використання в UML представлена на рисунку 2.4.

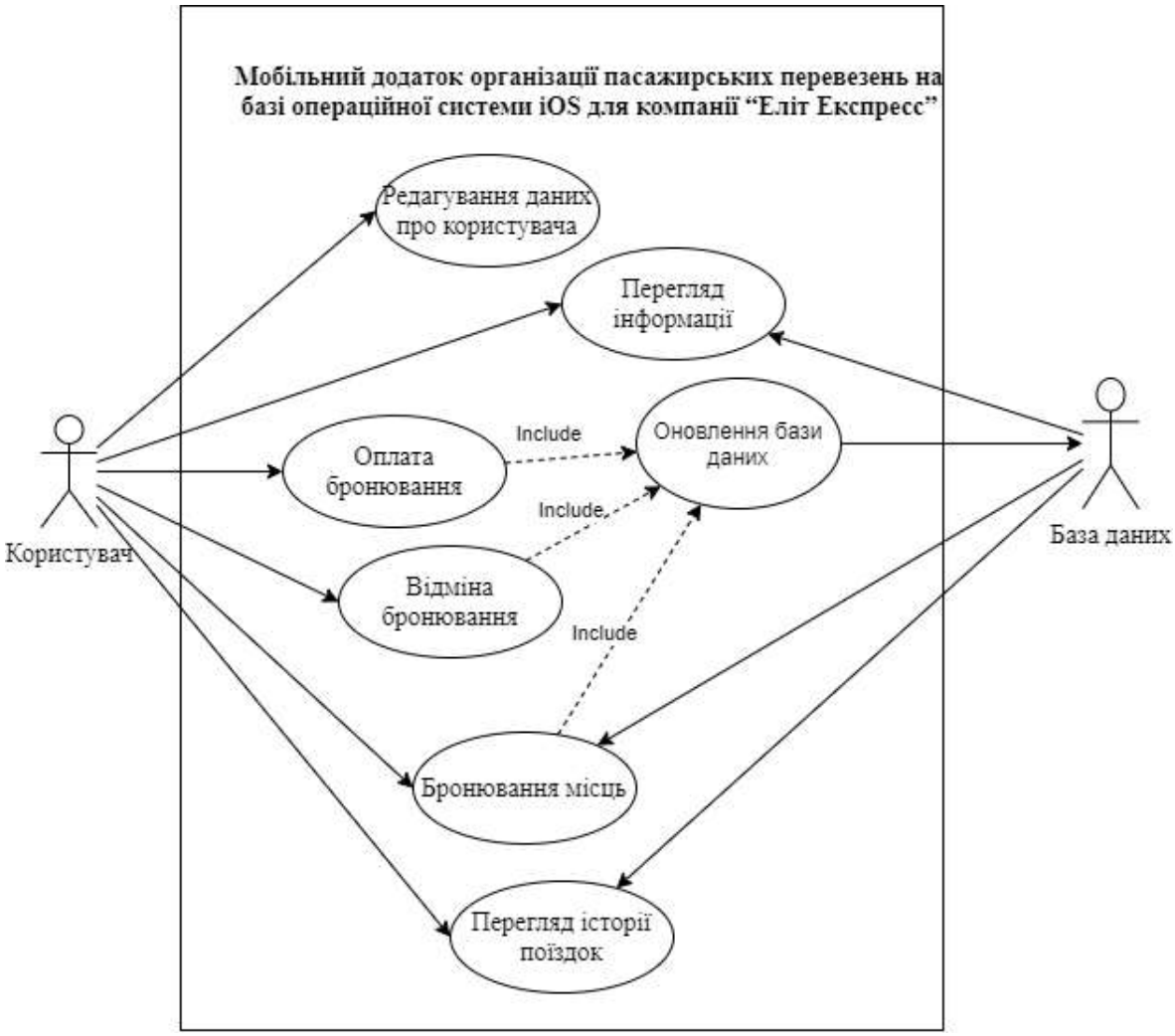


Рисунок 2.4 – Діаграма варіантів використання

2.3 Проектування моделі бази даних мобільного додатку

Розглянемо схему бази даних. На ній зображені таблиці «Zone», «Togo», «Reis» «Marshrut», що необхідні для відображення даних по рейсу. Таблиця «Orders» зберігає інформацію про бронювання користувачів, таблиця «Clients» містить інформацію про користувачів.

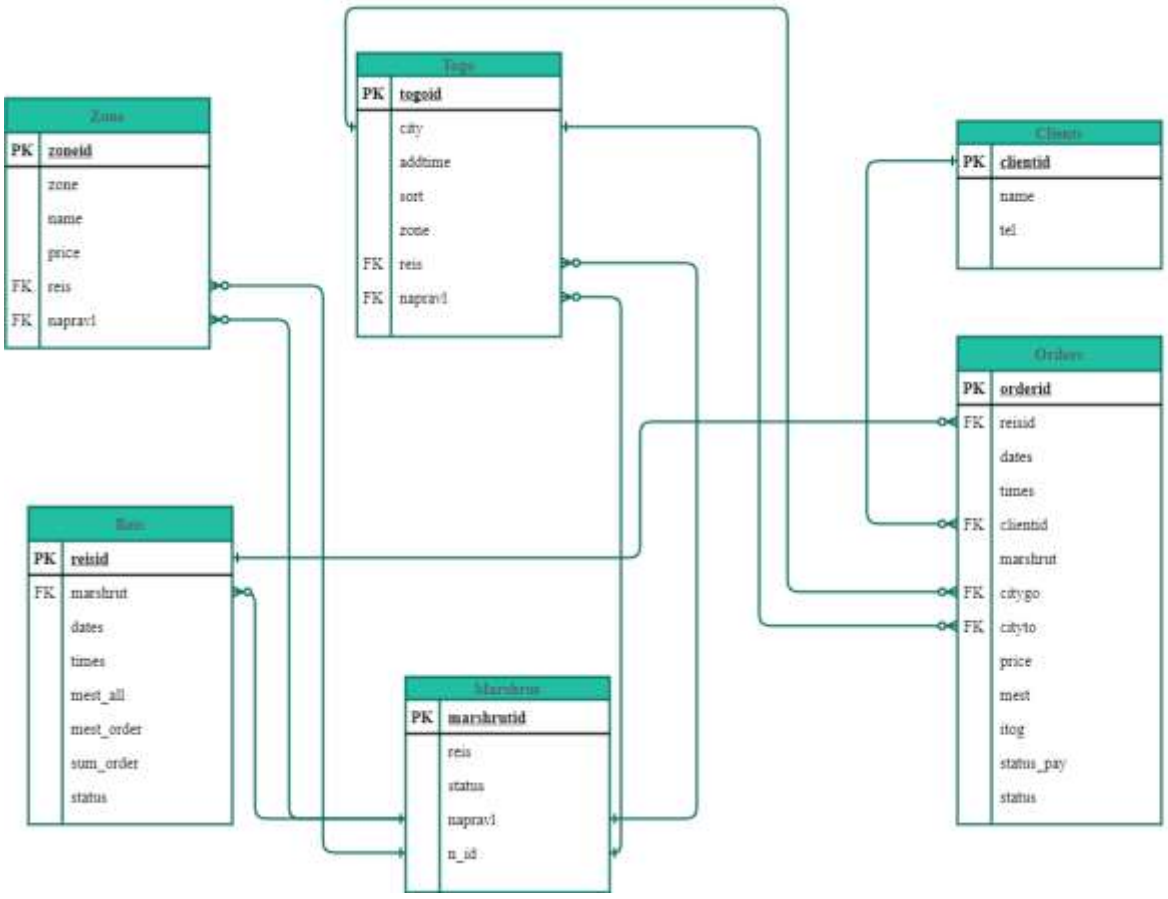


Рисунок 2.5 – ER-діаграма бази даних

Оскільки усі дані про маршрути та бронювання зберігаються на сервері, то було прийнято рішення отримувати необхідну інформацію у форматі JSON за допомогою POST-запитів із мобільного додатку. Структура зберігання даних в мобільному додатку представлена в таблиці 2.1.

Таблиця 2.1 – Структура зберігання даних в мобільному додатку

Назва	**Тип**	**Поле**	**Тип даних**	**Опис**
Route	Структура	code	String	Код маршруту
		route	String	Назва маршруту
PlaceElement	Структура	otprID	String	Ідентифікатор міста відправлення
		cityOtpr	String	Назва міста відправлення
		addtime	String	Час очікування після початку рейсу
		otprZone	String	Тарифна зона
		pribID	String	Ідентифікатор міста прибуття
		pribCity	String	Назва міста прибуття
		pribZone	String	Тарифна зона
TimeElement	Структура	Id	String	Ідентифікатор рейсу
		times	String	Час початку рейсу
		mesta	Integer	Кількість місць, що залишилися
ZoneElement	Структура	nameZone	String	Назва тарифної зони
		price	Double	Ціна
HistoryElement	Структура	id	String	Ідентифікатор поїздки
		reis	String	Назва рейсу
		dates	String	Дата поїздки
		times	String	Час поїздки
		citygo	String	Місце відправлення
		cityto	String	Місце прибуття
		marshrut	String	Код маршруту
		mest	String	Кількість заброньованих місць
		itog	Double	Загальна сума
		status	String	Статус бронювання
OrderInformation	Клас	orderRoute	String	Код маршруту
		orderPlaceAmount	Integer	Кількість місць
		orderDate	String	Дата поїздки
		orderTimeID	String	Ідентифікатор часу поїздки

Продовження таблиці 2.1

Назва	Тип	Поле	Тип даних	Опис
OrderInformation	Клас	orderDepartmentTime	String	Час відправлення
		orderIdFrom	String	Ідентифікатор місця відправлення
		orderIdTo	String	Ідентифікатор місця прибуття
		orderPrice	Double	Ціна за 1 квиток
		comments	String	Коментарі

Отримані дані із сервера зберігаються в структурах. Структура «Route» слугує для вибору типу маршруту. В структурі «PlaceElement» зберігаються дані про міста відправлення та прибуття, а в залежності від вибору відповідного пункту меню користувачем інформація про різні типи міст відображається окремо. Структура «TimeElement» необхідна для вибору рейсу користувачем. Структура «ZoneElement» дозволяє отримати ціну за вибраним маршрутом, містом прибуття та відправлення. У структурі «HistoryElement» зберігається інформація про поїздки користувачів. Клас «OrderInformation» зберігає інформацію про поточне бронювання.

3 РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ

3.1 Архітектура мобільного додатку

Під час розробки мобільного додатку важливо використовувати певний шаблон проектування з метою покращення роботи розроблюваної системи та якості документації. Досить часто при проектування мобільний додатків на базі ОС iOS використовується патерн проектування MVC (Model-View-Controller) згідно з яким модель даних, інтерфейс користувача та механізми взаємодії з користувачем поділені на 3 окремі частини, що взаємопов’язані між собою [4]. Модель відповідає за збереження даних додатку, вид відповідає за відображення візуальної складової додатку, а також за взаємодію користувача з додатком, контролер відповідає за зв’язок моделі та виду шляхом маніпуляції даних в залежності від дій користувача. Такий підхід до розробки є досить гнучким, що дозволяє виконувати зміни в окремих складових без істотного впливу на інші елементи. Більш того, даний шаблон проектування також дозволяє значною мірою зменшити складність великих додатків, спростити тестування та подальшу підтримку продукту.

Для даного мобільного додатку інтерфейс користувача буде створений за допомогою Interface Builder, що зберігатиме всі екрани додатку та переходи між ними. Моделлю даних буде виступати інформація, отримана з бази даних у вигляді структур, а також інформація про поточне замовлення. Взаємодія між інтерфейсом та моделлю даних буде оброблюватися за допомогою контролерів, які будуть відповідати певному екрану та міститимуть обробку подій згідно з діями користувача.

На рисунку 3.1 зображено архітектуру мобільного додатку.

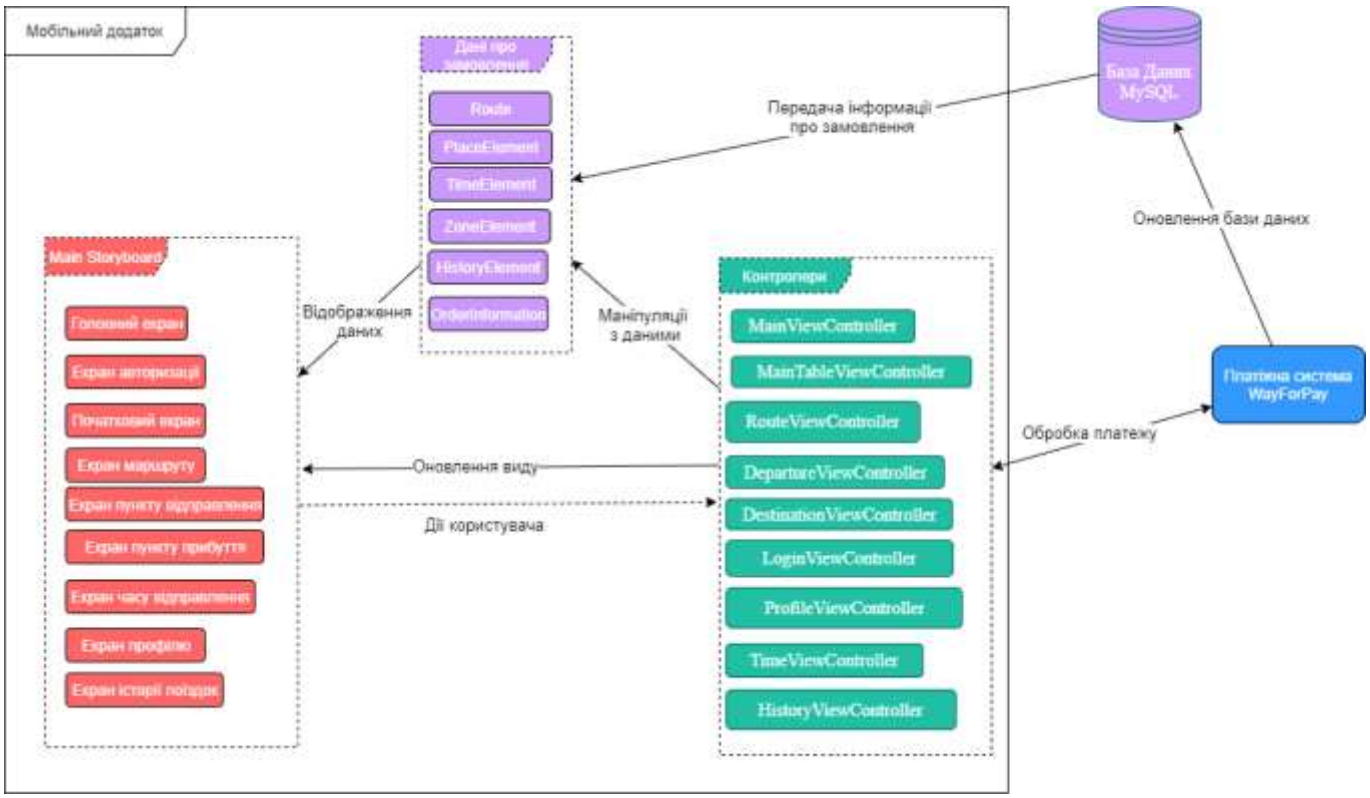


Рисунок 3.1 – Архітектура мобільного додатку

3.2 Програмна реалізація

Для розробки мобільного додатку на базі ОС iOS було обрано IDE XCode та мову програмування Swift. В першу чергу необхідно створити новий проект, вказавши його назву, команду розробників, а також ідентифікатор організації, спосіб створення інтерфейсу та мову програмування (рис. 3.2).

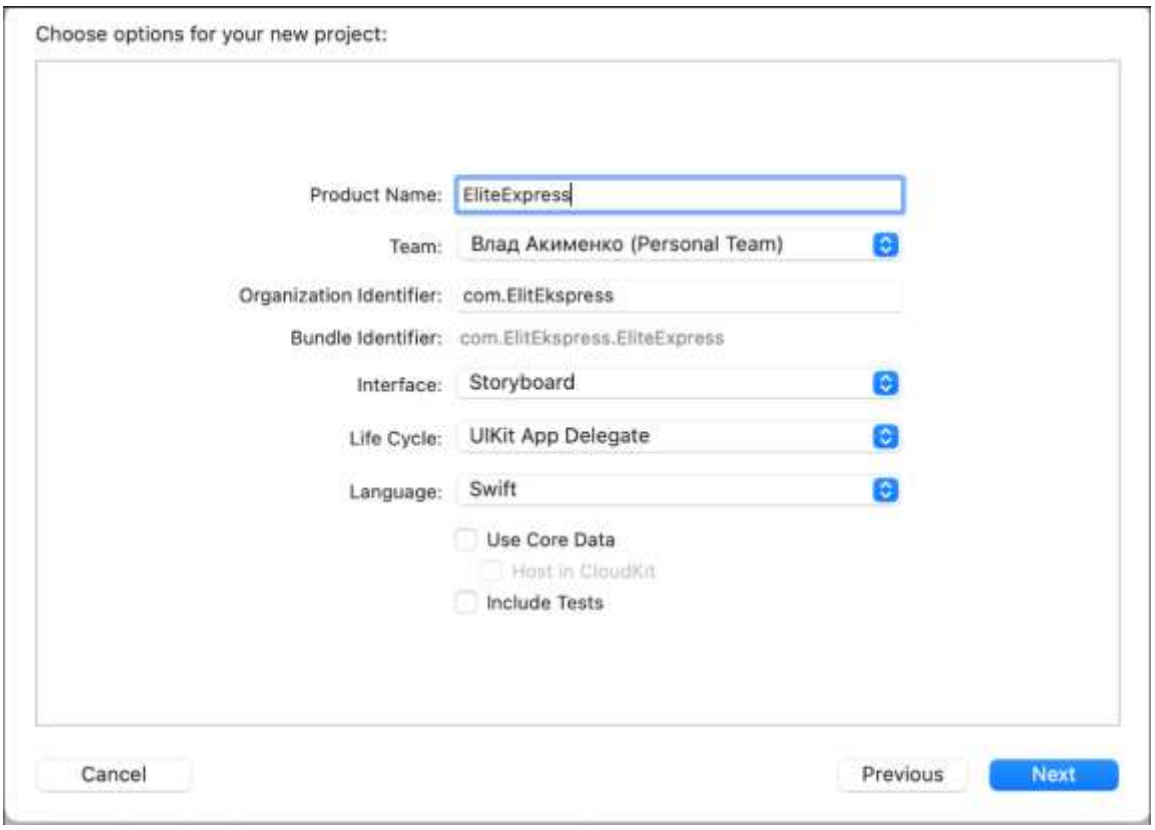


Рисунок 3.2 – Вікно створення нового проекту

Після створення проекту, XCode створює стандартний набір елементів нового проекту (рис. 3.3). У файлі EliteExpress.xcodeproj відображені властивості проекту (рис 3.4).

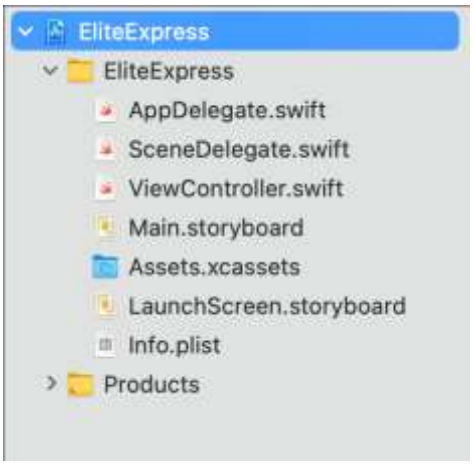


Рисунок 3.3 – Структура нового проекту

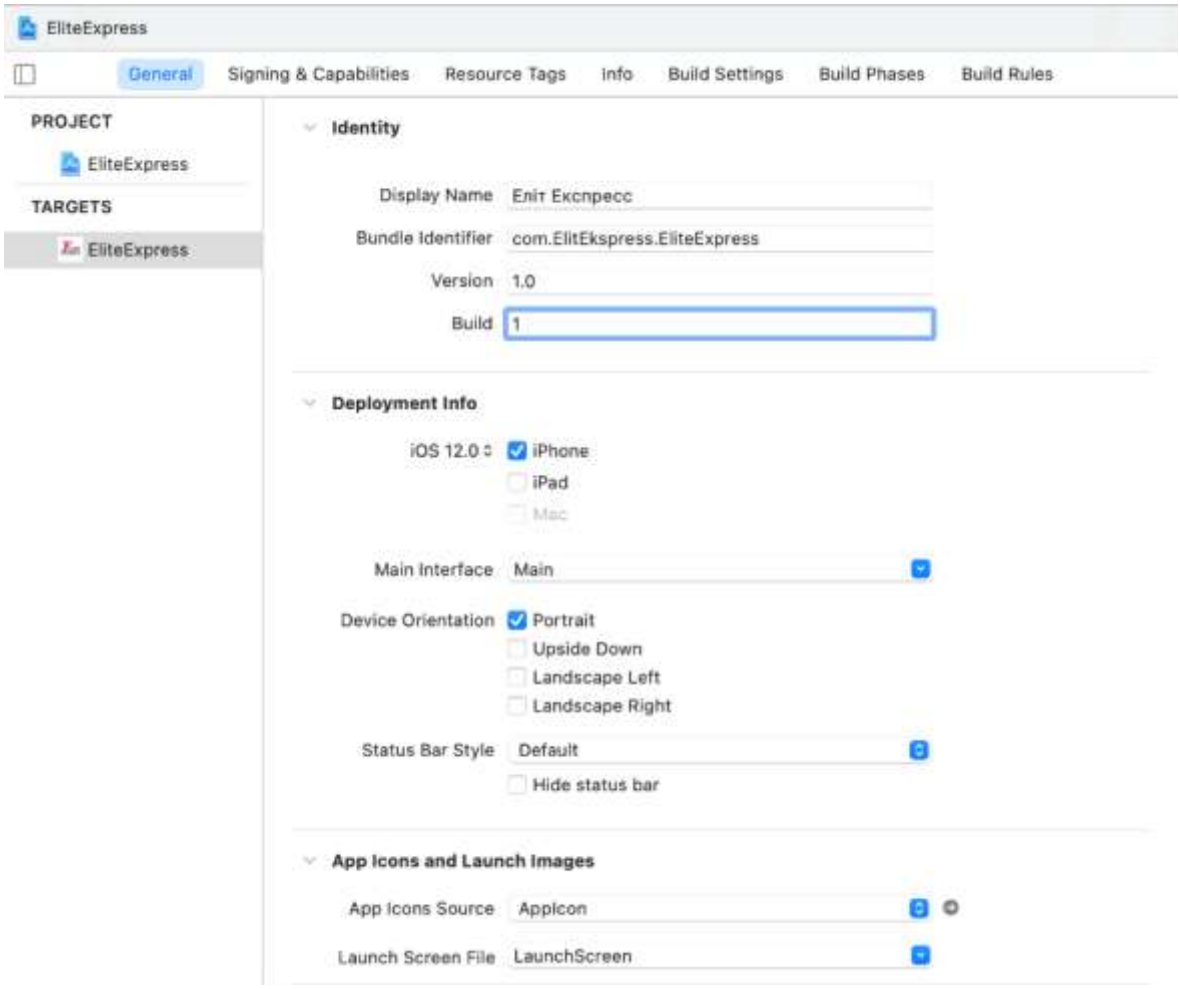


Рисунок 3.4 – Властивості проекту

В першу чергу в файлі Main.storyboard було створено екрани відповідно до розробленого прототипу, додано елементи інтерфейсу на кожен екран та задано переходи між екранами. Оскільки екран завантаження та авторизації не містять навігаційної панелі, то переходи між ними та на головний екран реалізовані за допомогою програмного коду. Усі екрани мобільного додатку зображені на рисунку 3.5.

Рисунок 3.5 – Екрани мобільного додатку

Для того, щоб уникнути проблем з відображенням інтерфейсу на різних пристроях необхідно кожному елементу додати обмеження (constraints) (рис. 3.6) та зафіксувати його положення відносно інших елементів або відносно виду [5].

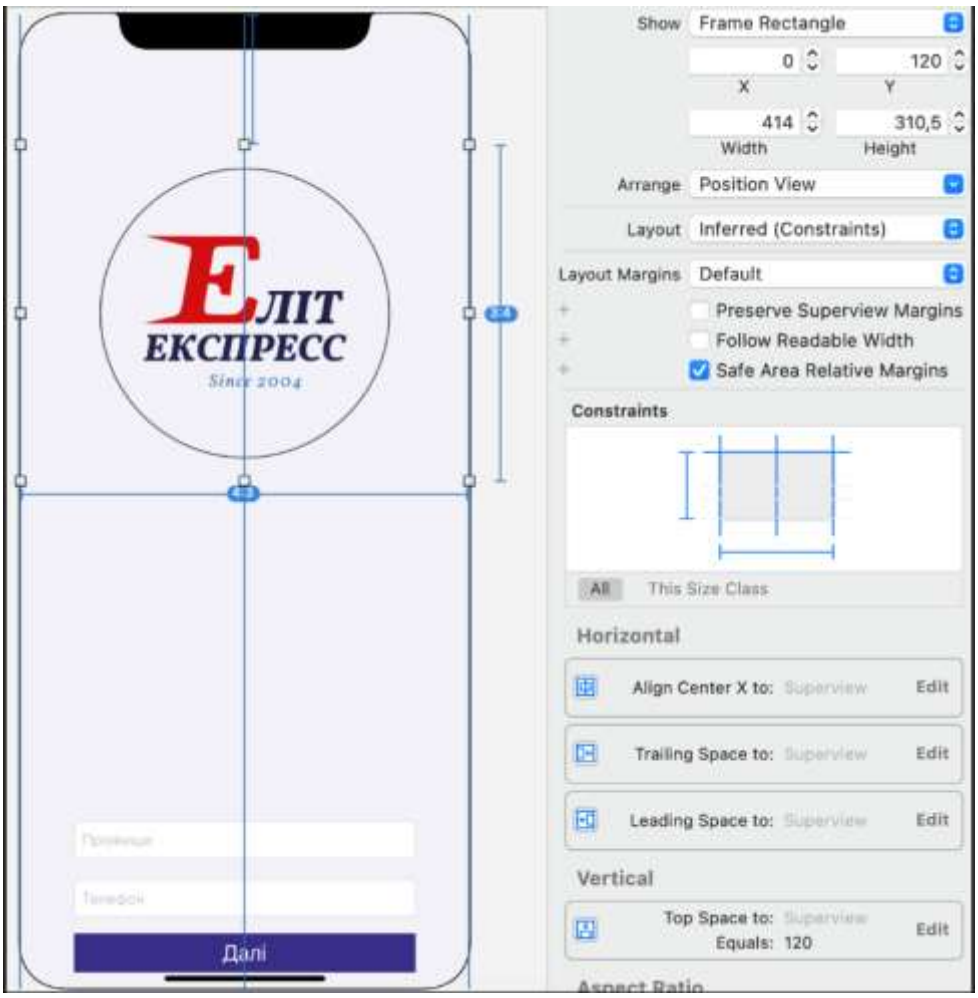


Рисунок 3.6 – Обмеження для зображення логотипу компанії

В таблиці 3.1 представлені класи контролери екранів та їх призначення.

Таблиця 3.1 – Класи мобільного додатку

Клас	**Призначення**
LoadViewController	Перевірка актуальності версії та доступу до мережі Інтернет
LoginViewController	Авторизація користувача
MainViewController	Перехід на профіль користувача або історію поїздок, бронювання місць за вибраними даними
MainTableViewController	Перехід на екрани вибору маршруту, пункту призначення та відправлення, часу відправлення
RouteTableViewController	Вибір маршруту поїздки
DepartureTableViewController	Вибір пункту відправлення
DestinationTableViewController	Вибір пункту прибуття

Продовження таблиці 3.1

Клас	**Призначення**
TimeTableViewController	Вибір часу поїздки
HistoryTableViewController	Відображення історії поїздок, відміна або оплата бронювання
ProfileTableViewController	Редагування профілю користувача

В таблиці 3.2 представлені класи з функціями, що використовуються декількома екранами та їх призначення.

Таблиця 3.2 – Класи з функціями для декількох екранів

Клас	**Призначення**
Reachability	Перевірка доступу до мережі Інтернет
Alerts	Повідомлення, що з’являються при відсутності мережі або при некоректно введених даних
KeyboardFunctions	Функції обробки дій з клавіатурою
LoadJSON	Завантаження даних з сервера та переведення їх в структури для зберігання в додатку
OrderInformation	Збереження інформації про поточне бронювання користувача
CommonFunctions	Функції переходу на інший екран, появи індикатору завантаження

Для реалізації оплати замовлення всередині додатку використовувався фреймворк Safari Services [6]. Даний фреймворк надає можливість вбудувати функціонал браузера Safari всередині додатку та використовувати його для оплати бронювання. Інформація про бронювання за допомогою POST запиту передається на сервер, після чого відкривається сторінка зі сформованими реквізитами бронювання та відбувається перехід на платіжну систему WayForPay. Передача даних відбувається за допомогою протоколу HTTPS [7].

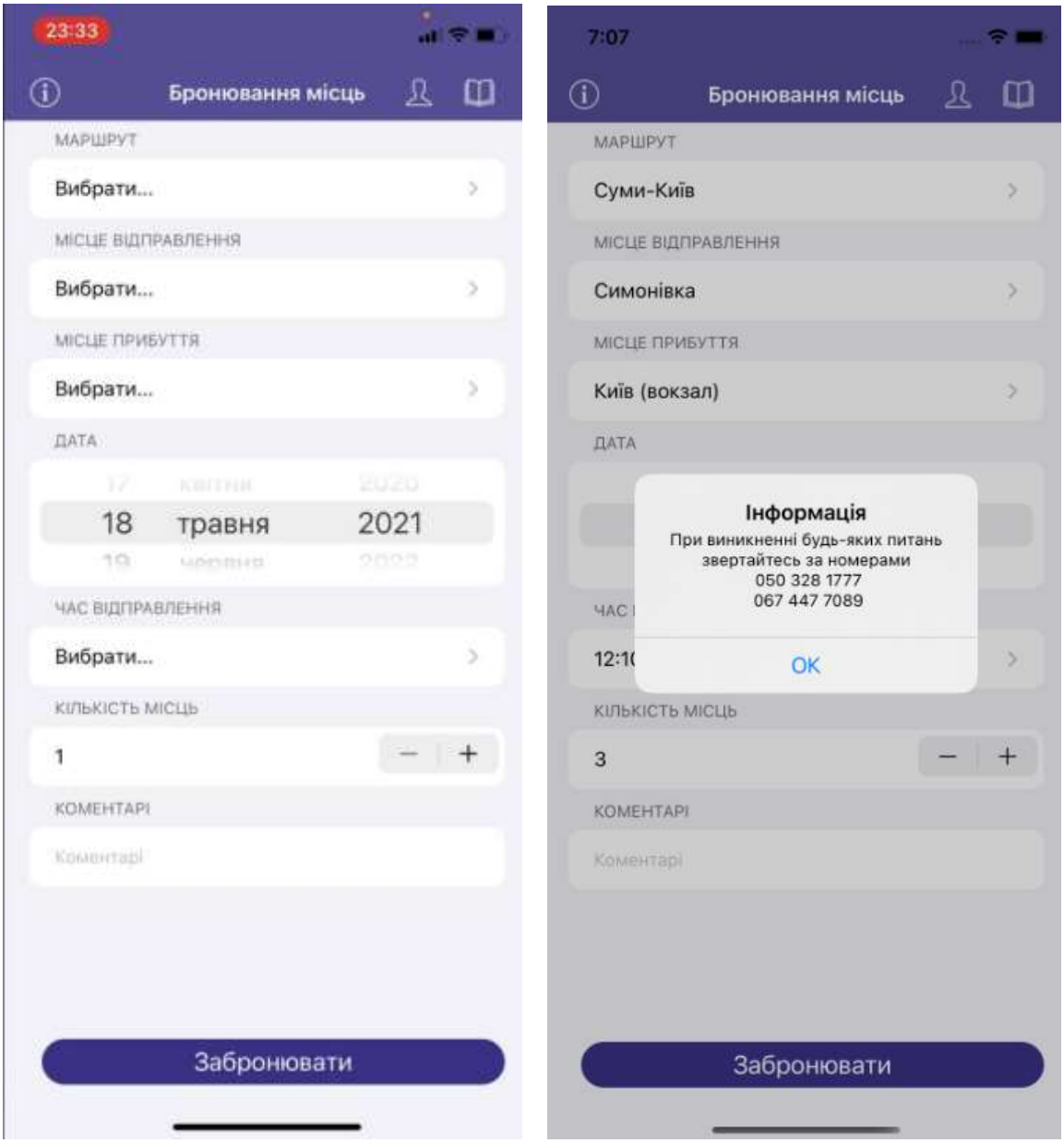
3.3 Використання мобільного додатку

Після запуску додатку з’являється екран авторизації (рис. 3.7). Для подальшої роботи з додатком необхідно ввести прізвище та номер телефону.

Рисунок 3.7 – Екран авторизації

Після успішної авторизації користувач потрапляє на головний екран додатку, де знаходяться поля для бронювання місць. Окрім цього, користувач має можливість

у полі навігації перейти на екран профілю, історії поїздок або відкрити повідомлення з щодо виникнення можливих питань (рис. 3.8).



а б

Рисунок 3.8 – Головний екран (а), інформаційна довідка (б)

На екрані профілю користувач має можливість відредагувати прізвище та змінити номер телефону. На рисунку 3.9 наведено профіль користувача

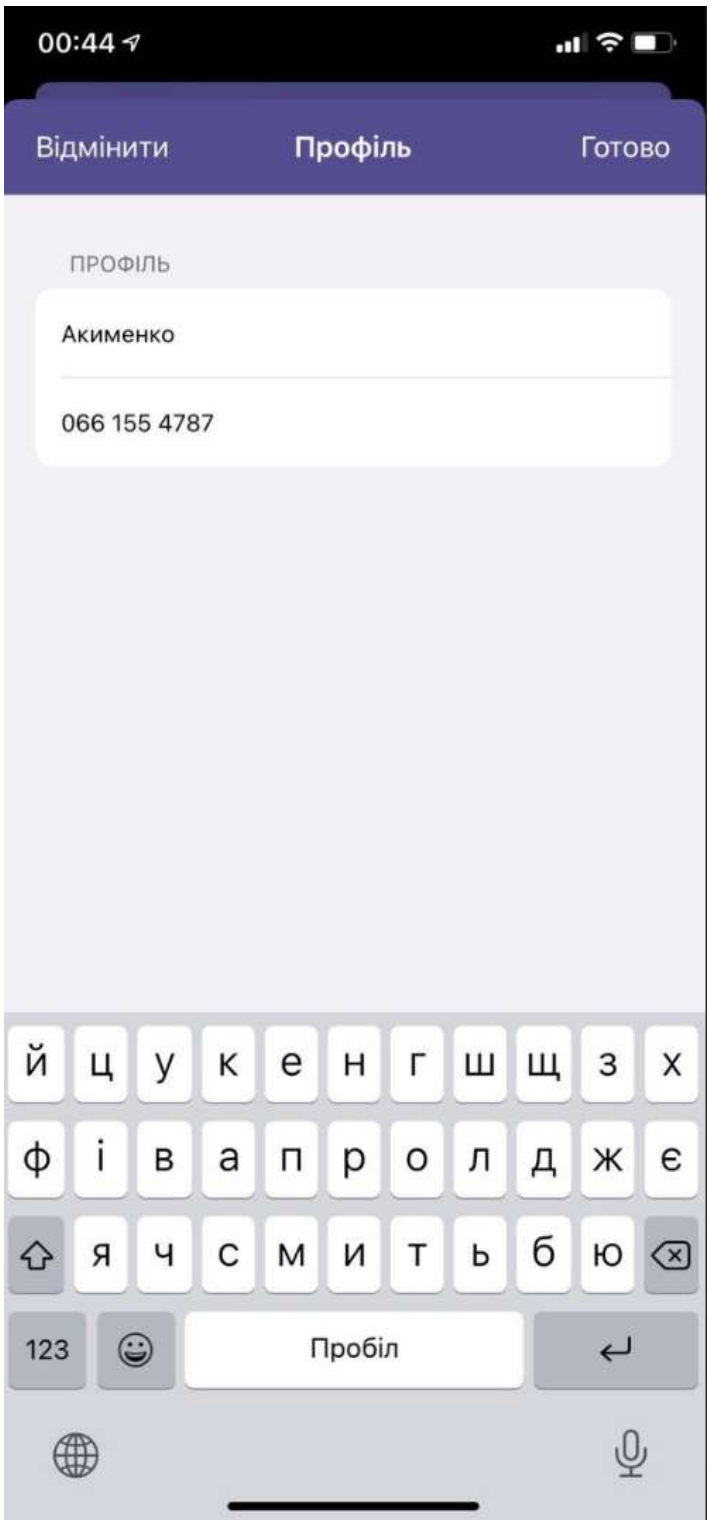


Рисунок 3.9 – Профіль користувача

На головному екрані при виборі полів маршруту, пункту відправлення, часу відправлення та пункту прибуття відбувається перехід на відповідні екрани (рис. 3.10-3.13).

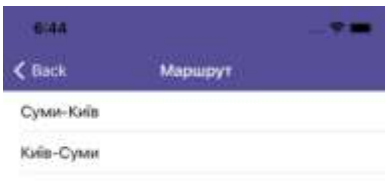


Рисунок 3.10 – Екран вибору маршруту

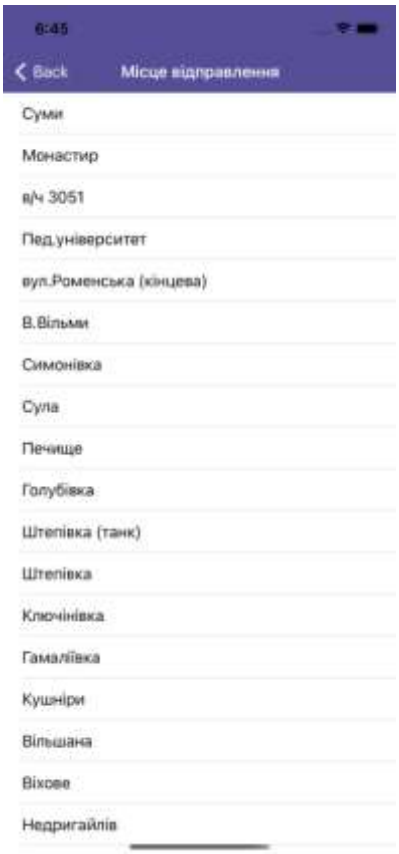


Рисунок 3.11 – Екран вибору місця відправлення

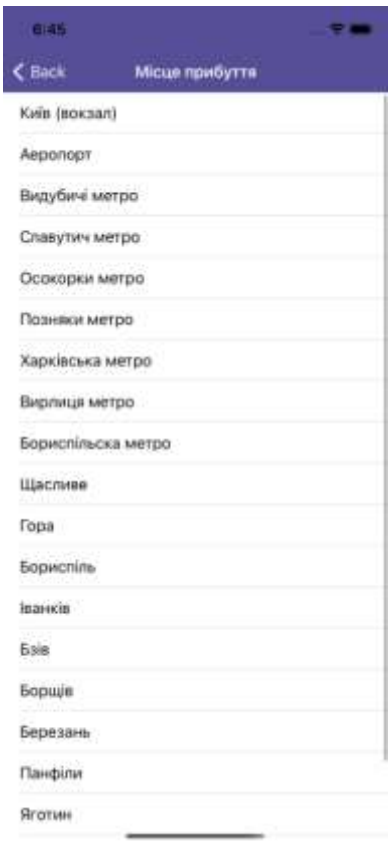
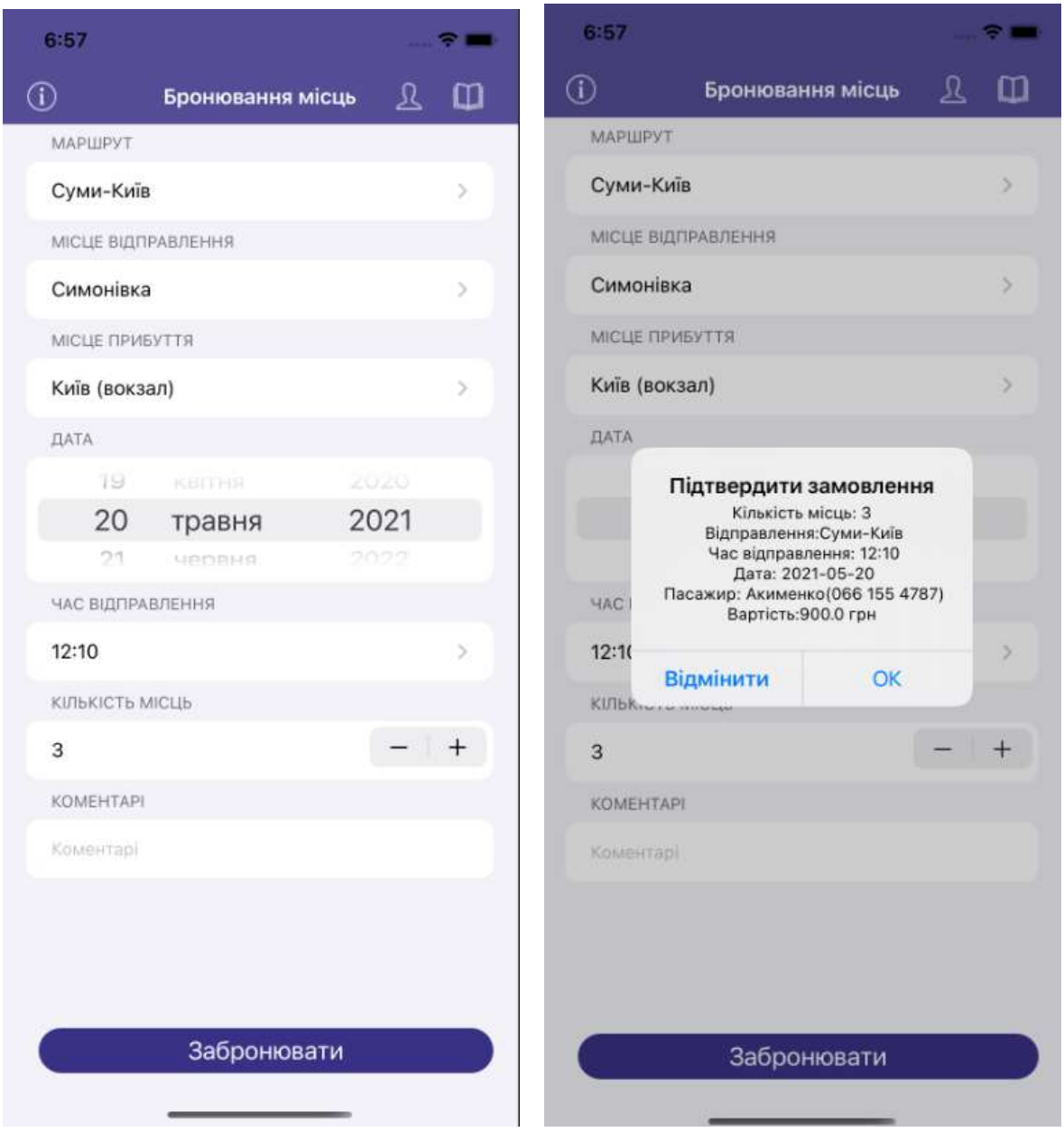


Рисунок 3.12 – Екран вибору пункту прибуття

Рисунок 3.13 – Екран вибору часу відправлення

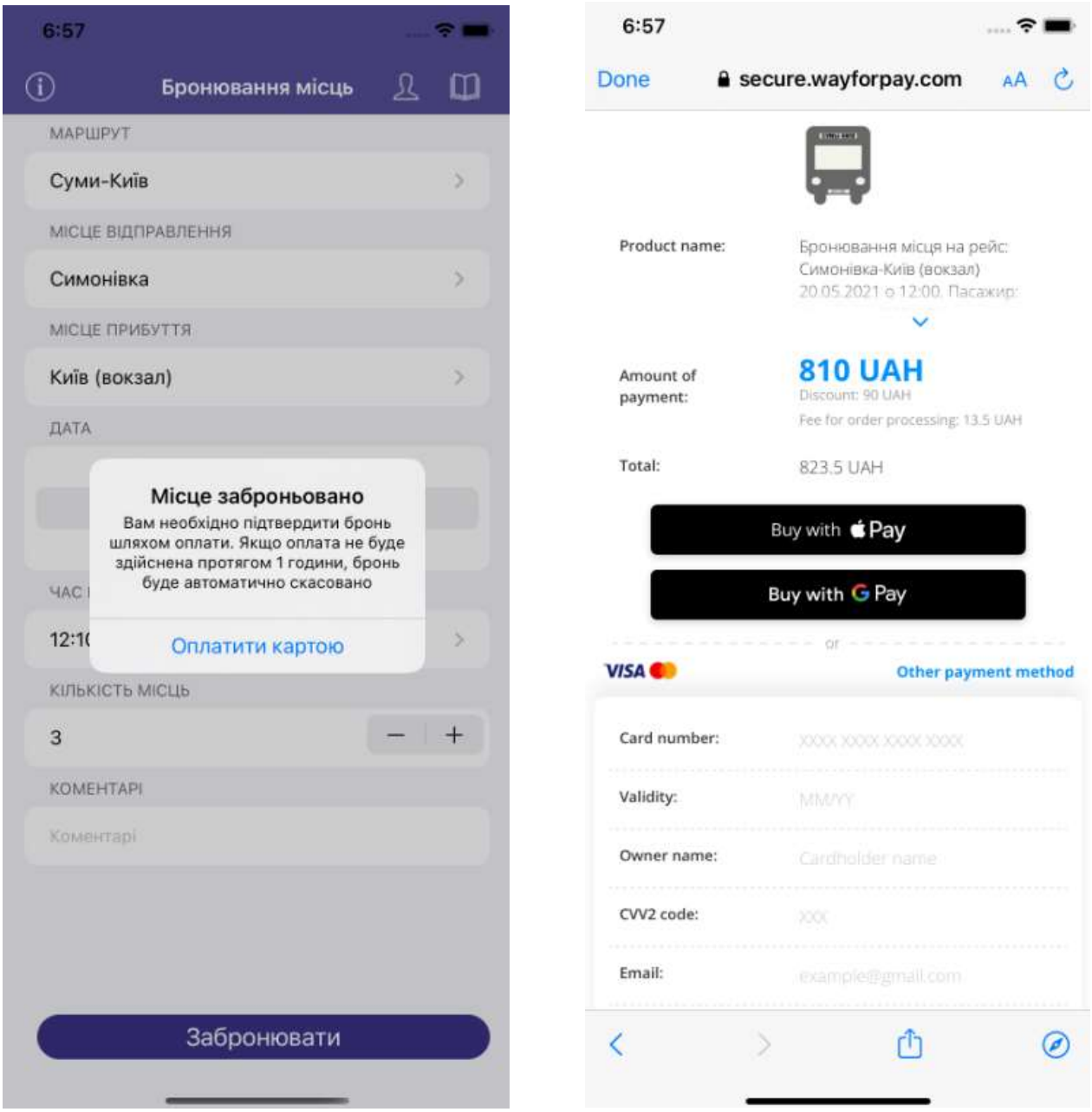
Коли усі дані заповнені необхідно натиснути кнопку «Забронювати». З'явиться повідомлення з підтвердженням замовлення (рис 3.14).



а б

Рисунок 3.14 – Головний екран із заповненими полями (а), підтвердження замовлення(б)

У випадку підтвердження замовлення з'явиться повідомлення про необхідність виконати оплату протягом 1 години (рис. 3.15 а). При натисканні кнопки «Оплатити картою» відбудеться перехід на екран з платіжною системою (рис. 3.15 б).

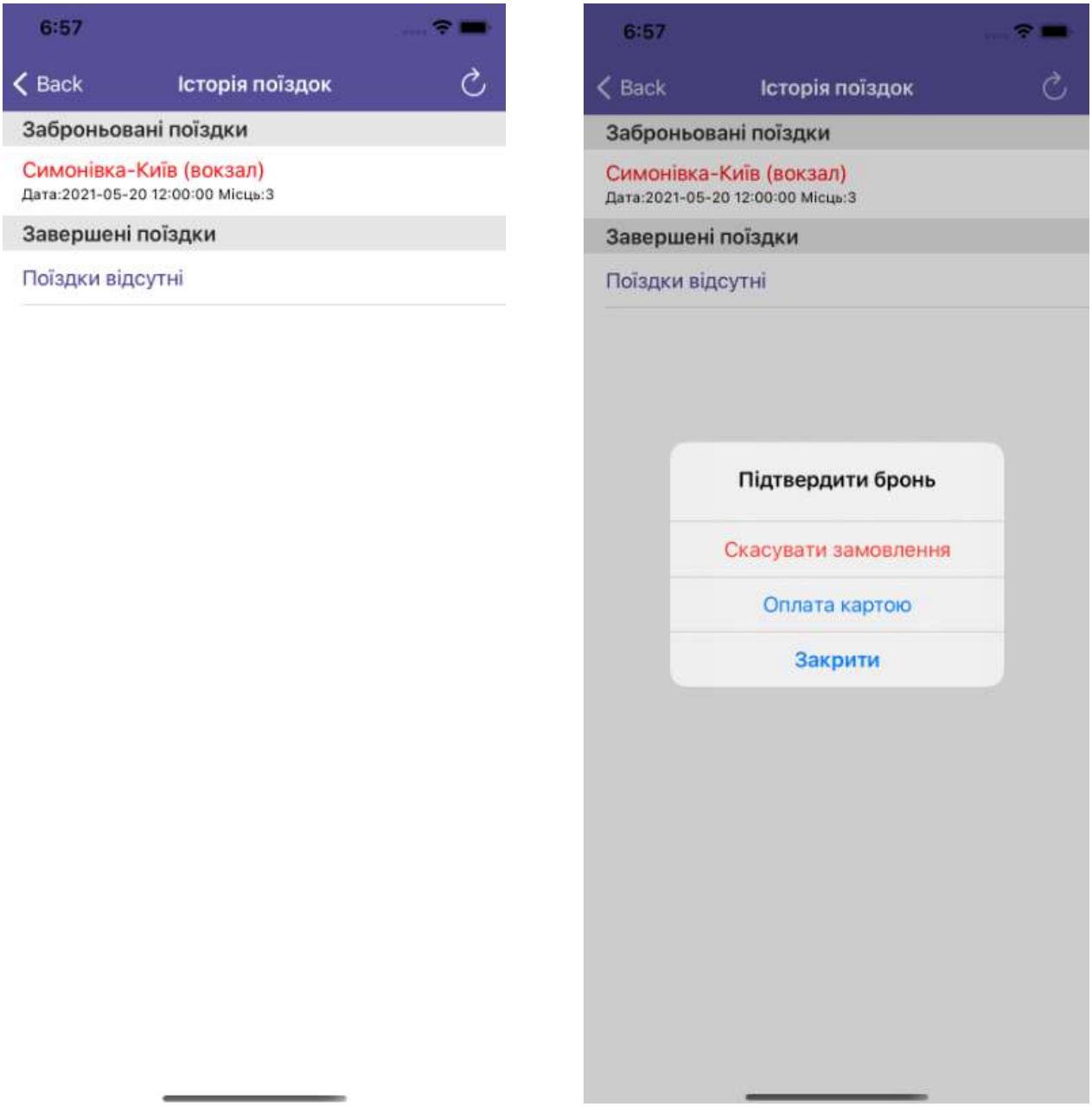


а б

Рисунок 3.15 – Повідомлення про успішно заброньоване місце (а), екран оплати замовлення(б)

На екрані історії поїздок будуть відображені завершені та заброньовані поїздки (рис. 3.16 а). Якщо з якоїсь причини користувач не виконав оплату броні, він може

оплатити місце шляхом натискання на відповідне поле бронювання та вибравши пункт «Оплата картою» (рис. 3.16 б). Окрім того, користувач має можливість скасувати бронь натиснувши «Скасувати замовлення».

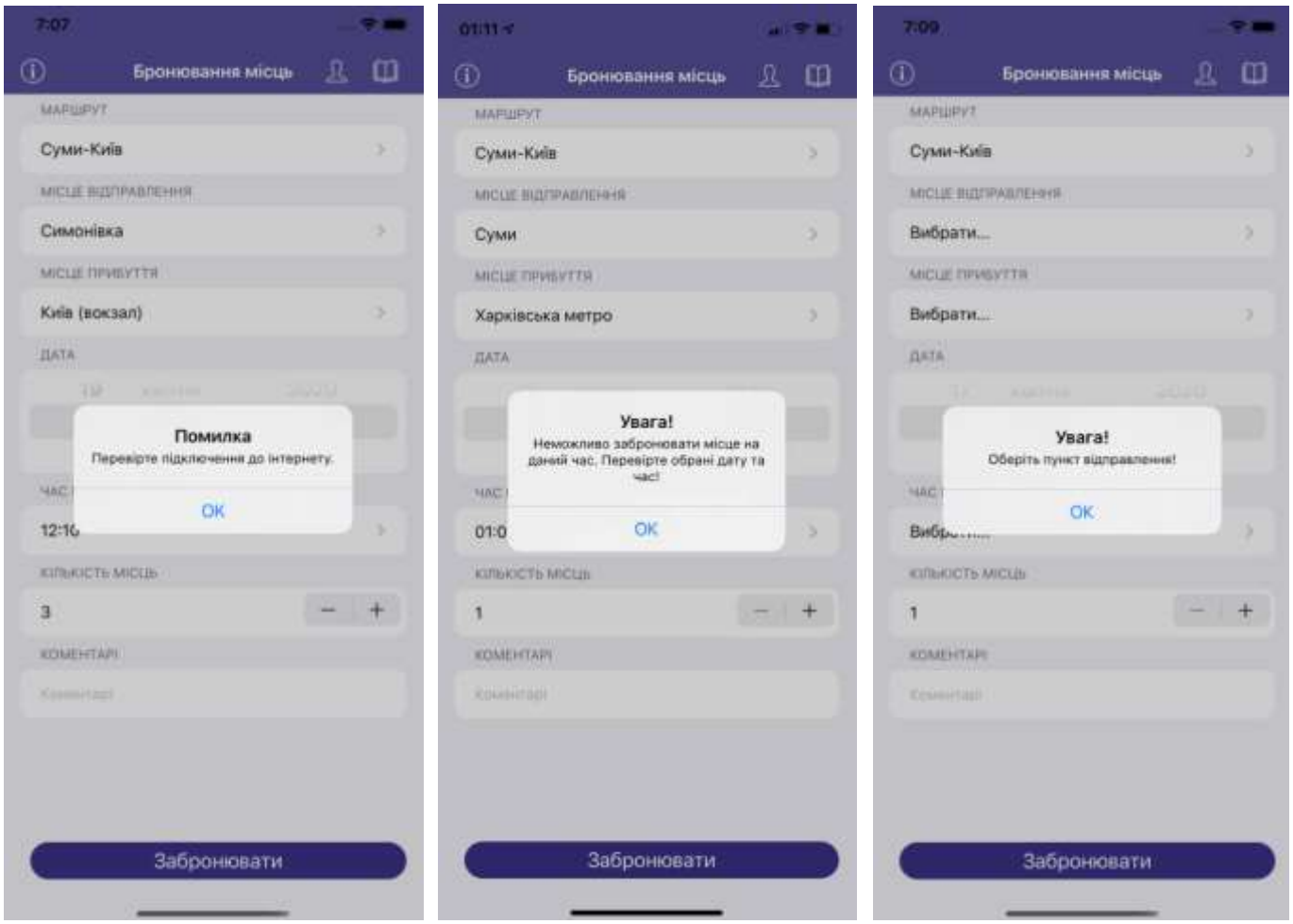


а б

Рисунок 3.16 – Екран історії поїздок (а), повідомлення з оплатою або скасуванням бронювання (б)

Мобільний додаток передбачає обробку виключних ситуацій: відсутність підключення до інтернету (рис. 3.17 а), неправильно вибрані дані про бронювання

(рис. 3.17 б) або ж відсутність певного пункту бронювання (рис. 3.17 в). У разі виникнення таких ситуацій користувач отримає повідомлення про помилку.



а б в

Рисунок 3.17 – Повідомлення про помилку: відсутнє підключення до інтернету (а), неправильно обрана дата або час (б), не вибраний пункт відправлення (в)

ВИСНОВКИ

У ході виконання дипломної роботи було виконано аналіз предметної області, визначено актуальність проблеми. Під час проведення аналізу мобільних додатків-аналогів було визначено переваги та недоліки існуючих продуктів та складено технічне завдання, що дозволить створити конкурентоспроможний мобільний додаток організації пасажирських перевезень.

Було сформовано мету роботи та визначені засоби для реалізації поставленої задачі. Мета роботи – розробка мобільного додатку організації пасажирських перевезень на базі ОС iOS. У якості засобів реалізації було обрано IDE Xcode, мову програмування Swift.

Під час проектування мобільного додатку було створено діаграму нотації IDEF0 процесу роботи додатку, а також було виконано декомпозицію основного процесу на підпроцеси, що дозволило краще змоделювати потік роботи програми. Розроблена діаграма варіантів використання дозволила визначити необхідну функціональність додатку.

Була розроблена схема бази даних, що дозволило реалізувати авторизацію користувача та історію поїздок, створити структури для зберігання даних під час роботи додатку та під час запитів на сервер. По завершенню проектування мобільного додатку було виконано програмну реалізацію проекту на основі створеного прототипу.

Результатом роботи є мобільний додаток, що є досить надійним та простим у використанні, має зручний інтерфейс, функціонал забезпечений в повній мірі. Мобільний додаток повністю задовольняє потреби замовника.

Мобільний додаток впроваджено в роботу компанії пасажирських перевезень, що підтверджується актом впровадження результатів дипломного проекту (додаток Г). Додаток завантажений у AppStore і доступний для використання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Android vs iOS Market Share 2020: Stats and Facts [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://www.mobileapps.com/blog/android-vs-ios-market-share.

2. Анісімов В. В. МЕТОДОЛОГИЯ IDEF0 [Електронний ресурс] / Віктор Володимирович Анісімов – Режим доступу до ресурсу: https://sites.google.com/site/anisimovkhv/learning/pris/lecture/tema6/tema6_2.

3. ГЛАВА 4 Диаграмма вариантов использования (use case diagram) [Електронний ресурс] – Режим доступу до ресурсу: http://khpi-iip.mipk.kharkiv.edu/library/case/leon/gl4/gl4.html#1.

4. Model–view–controller [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Model–view–controller.

5. Apple Inc. Understanding Auto Layout [Електронний ресурс] / Apple Inc. – 2018. – Режим доступу до ресурсу: https://developer.apple.com/library/archive/documentation/UserExperience/Conceptual/AutolayoutPG/index.html.

6. Safari Services framework [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/documentation/safariservices.

7. HTTP-метод POST [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/ru/docs/Web/HTTP/Methods/POST.

8. HyperText Transfer Protocol Secure [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/HTTPS.

9. The Swift programming language [Електронний ресурс]. – 2020. – Режим доступу до ресурсу: https://docs.swift.org/swift-book/.

10. UIKit framework [Електронний ресурс] – Режим доступу до ресурсу: https://developer.apple.com/documentation/uikit.

ДОДАТОК А

ТЕХНІЧНЕ ЗАВДАННЯ

на розробку інформаційної системи
«Мобільний додаток на базі операційної системи iOS для організації пасажирських перевезень компанією «Еліт Експресс»»

Суми 2021

1 ПРИЗНАЧЕННЯ Й МЕТА СТВОРЕННЯ МОБІЛЬНОГО ДОДАТКУ

Призначення мобільного додатку

Мобільний додаток призначений для замовлення місць клієнтами компанії «Еліт Експресс» за маршрутом Суми-Київ, оплату місць із додатку та взаємодії із історією поїздок.

Мета створення мобільного додатку

Метою створення мобільного додатку є покращення надання послуг компанією «Еліт Експресс», збільшення прибутку та розширення клієнтської бази.

Цільова аудиторія

До цільової аудиторії можна віднести постійних та потенційних клієнтів, жителів Сумської та Київської областей.

2 ВИМОГИ ДО МОБІЛЬНОГО ДОДАТКУ

2.1 Вимоги до мобільного додатку в цілому

2.1.1 Вимоги до структури й функціонування мобільного додатку

Мобільний додаток має бути розміщений в магазині додатків “AppStore”. Додаток створюється для використання на пристроях на базі ОС iOS.

2.1.2 Вимоги до персоналу

Від персоналу не вимагається особливих технічних навичок для експлуатації мобільного додатку, для підтримки мобільного додатку потрібні навички роботи з IDE Xcode, знання мов програмування Swift.

2.1.3 Вимоги до збереження інформації

Уся інформація буде зберігатися в базі даних та буде отримуватися за рахунок запитів на сервер. Номер телефону та прізвище користувача будуть зберігатися всередині додатку.

2.1.4 Вимоги до розмежування доступу

Розроблюваний мобільний додаток має бути загальнодоступним.

Усі користувачі мають однаковий доступ до додатку, розмежування за рівнем доступу до інформації відсутнє.

2.2 Структура мобільного додатку

2.2.1 Загальна інформація про структуру мобільного додатку

Структура мобільного додатку являє собою набір екранів з інформацією щодо бронювання та допоміжних діалогових вікон.

Такими екранами є:

Екран авторизації – перша сторінка при відкритті додатку, містить поля для заповнення даних про користувача.

Бронювання місць – головний екран для виконання бронювання місць.

Профіль – редагування даних про користувача.

Історія поїздок – перегляд історії поїздок користувача з можливістю оплати замовлення.

Маршрут – вибір маршруту поїздки.

Місце відправлення – вибір початкової точки маршруту.

Місце прибуття – вибір фінальної точки маршруту.

Час відправлення – вибір часу відправлення з початкової точки маршруту.

2.2.2 Навігація

Навігація у додатку відбувається шляхом переходу на наступний екран при виборі відповідного пункту меню на головній сторінці з можливістю повернутися на попередній екран за допомогою кнопки на навігаційній панелі.

2.2.3 Дизайн та структура додатку

Стиль мобільного додатку має бути сучасним, приємним для сприйняття, у якості основних кольорів було запропоновано використати фіолетові та сірі відтінки.

Прототип дизайну екранів додатку показано на рисунках 2.1-2.8.

Рисунок 2.1 – Екран авторизації

Рисунок 2.2 – Головний екран

Рисунок 2.3 – Профіль користувача

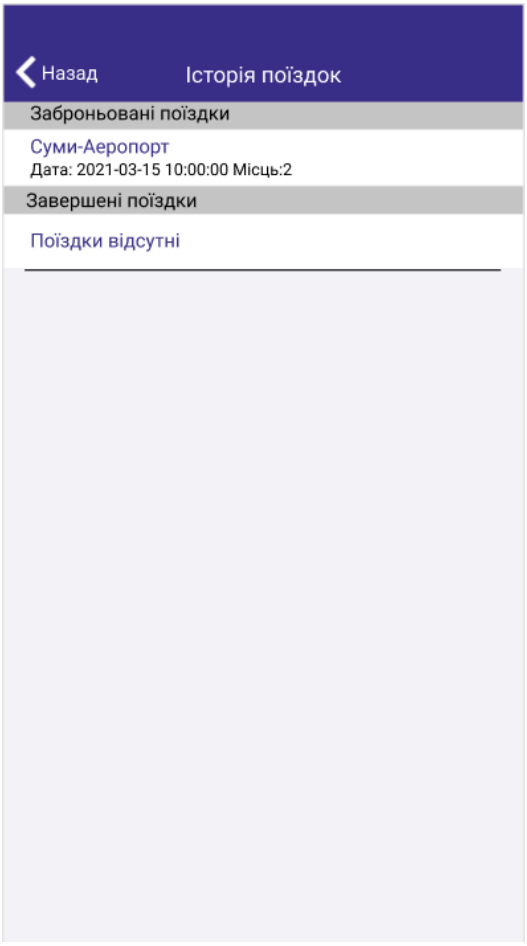


Рисунок 2.4 – Історія поїздок

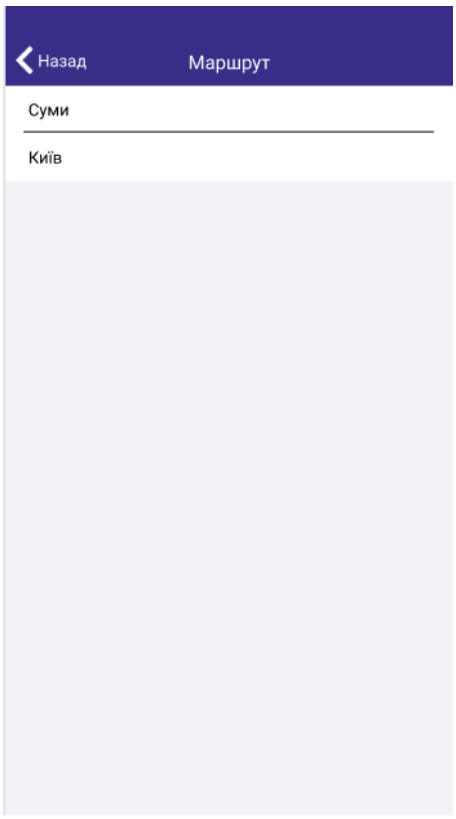


Рисунок 2.5 – Вибір маршруту

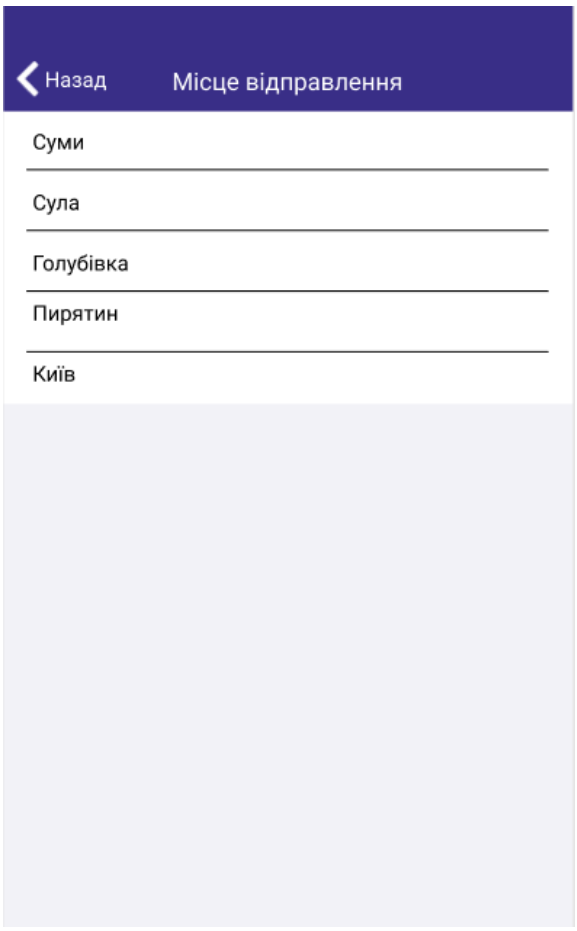


Рисунок 2.6 – Вибір пункту відправлення

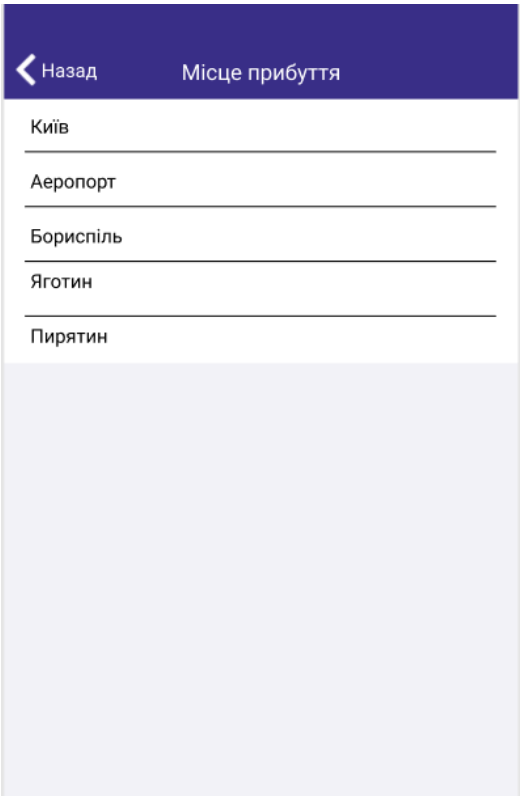


Рисунок 2.7 – Вибір пункту прибуття

Рисунок 2.8 – Вибір часу відправлення

2.3 Вимоги до функціонування системи

Відповідно до вимог замовника розроблений додаток має задовольняти таким функціональним вимогам:

- замовлення білетів за напрямком Суми-Київ;
- оплата білетів із додатку;
- можливість переглянути історію поїздок;
- скасування замовлення;
- відображення контактної інформації;

2.4 Вимоги до видів забезпечення

2.4.1 Вимоги до інформаційного забезпечення

Реалізація мобільного додатку відбувається з використанням:

- Xcode 12.0
- Swift 5.3
- JSON
- iOS Storyboards
- UIKit

2.4.2 Вимоги до лінгвістичного забезпечення

Мобільний додаток має бути виконаний українською мовою.

2.4.3 Вимоги до програмного забезпечення

Програмне забезпечення клієнтської частини повинне задовольняти наступним вимогам:

- Підтримка операційної системи iOS версії 12.0 і вище.
- Версія веб-браузера Safari версії 12.0 і вище

3 СКЛАД І ЗМІСТ РОБІТ ЗІ СТВОРЕННЯ МОБІЛЬНОГО ДОДАТКУ

Докладний опис етапів роботи зі створення мобільного додатку наведено в таблиці А.3.

Таблиця А.3 – Етапи створення мобільного додатку

№	Склад і зміст робіт	Строк розробки (у робочих днях)
1	Постановка задачі проекту	1 день
2	Складання технічного завдання	3 дні
3	Підготовка прототипу	5 днів
4	Створення макету дизайну мобільного додатку	2 дні
5	Розробка	20 днів
6	Робота над адаптивністю дизайну	2 дні
7	Тестування	1 день
8	Підготовка до випуску	1 день
9	Завантаження додатку у магазині “App Store”	1 день
10	Завершення роботи	1 день
	Загальна тривалість робіт	37 днів

4 ВИМОГИ ДО СКЛАДУ Й ЗМІСТУ РОБІТ ІЗ ВВЕДЕННЯ МОБІЛЬНОГО ДОДАТКУ В ЕКСПЛУАТАЦІЮ

Для введення мобільного додатку в експлуатацію на платформі “AppStore” необхідно створити аккаунт розробника Apple Developer. Для завантаження додатку необхідно мати підписаний та сертифікований файл розробника та створити збірку додатку для проходження валідації на платформі “App Store Connect”.

ДОДАТОК Б

ПЛАНУВАННЯ РОБІТ

Мета проекту: Розробити мобільний додаток на базі ОС iOS для організації пасажирських перевезень за маршрутом Суми-Київ. Проект буде виконаний вчасно, що підтверджено календарним планом проекту. Результати деталізації методом SMART розміщенні у таблиці Б.1.

Таблиця Б.1 – Деталізація мети методом SMART

Specific (конкретна)	Розробити мобільний додаток на базі ОС iOS для організації пасажирських перевезень.
Measurable (вимірювана)	Оскільки даний проект є комерційним, то результатом його роботи є розміщення проекту в магазині мобільних додатків AppStore.
Achievable (досяжна, узгоджена)	Ціль даного проекту вважається досяжною, оскільки розробник володіє необхідними навичками у створенні мобільних додатків мовою Swift та ознайомлений з необхідною кількістю методів, функцій технології UIKit. Мета була узгоджена з вимогами та потребами замовника.
Relevant (реалістична)	Для реалізації продукту проекту є всі необхідні технічні та програмні засоби(ноутбук зі встановленою ОС macOS, інтегроване середовище розробки (IDE) XCode), доступ до мережі Інтернет. Розробник досить кваліфікований для виконання поставлених задач.
Time-framed (обмежена в часі)	Мобільний застосунок розроблюється з обмеженням у часі на основі сформованого календарного плану та матриці відповідальності.

Планування змісту структури робіт. WBS є ієрархічною та інкрементною декомпозицією проекту у фази, кінцеві результати та пакети робіт. Цей інструмент спрямований на детальне планування, оцінку вартості, визначення та розподіл персональної відповідальності виконавців та на основні роботи і результати, що визначають зміст проекту.

На рисунку Б.1 приведена WBS-структура даного проекту.

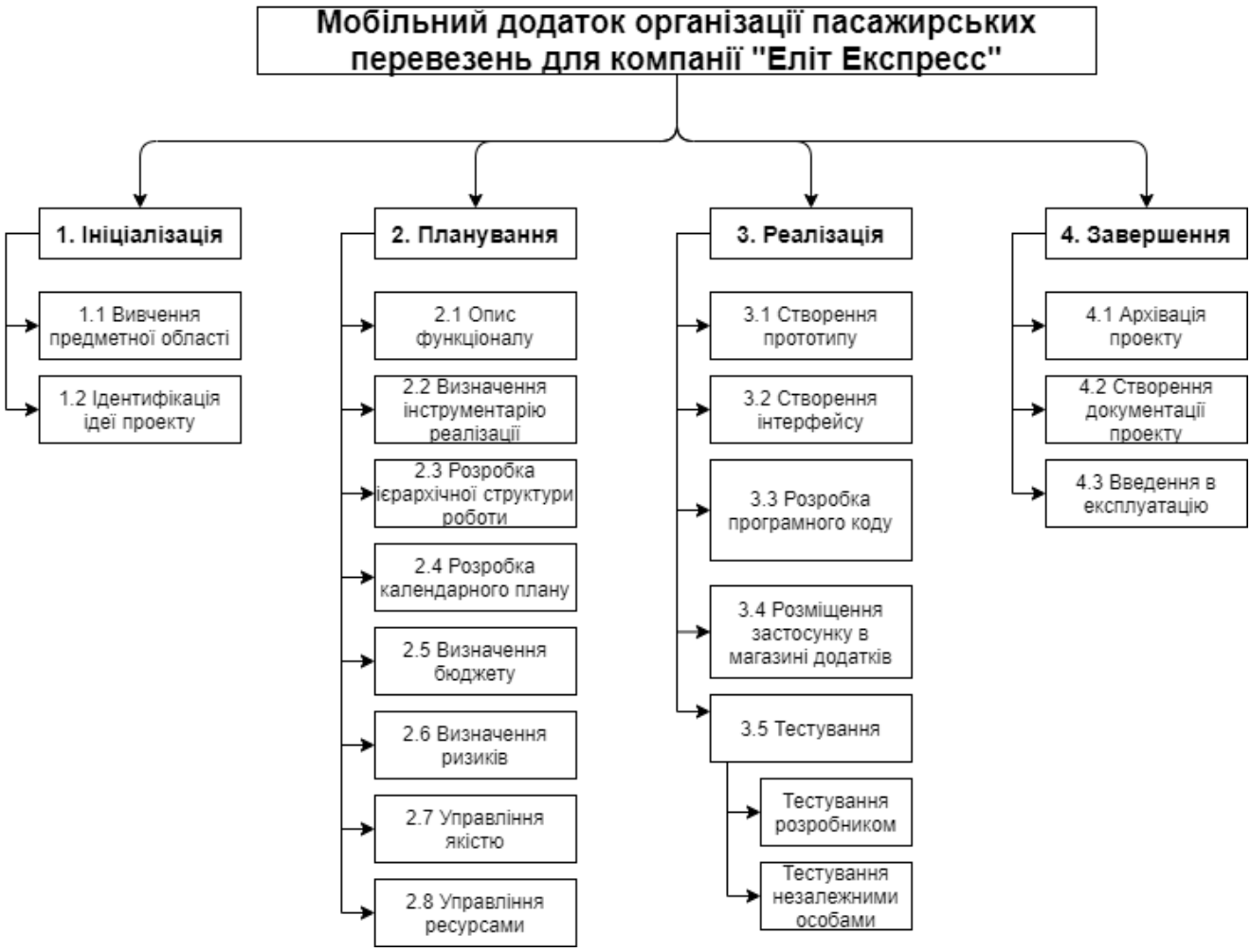


Рисунок Б.1 – WBS-структура проекту

Планування структури організації, для впровадження готового проекту (OBS).Після того, як була створена WBS структура проекту, наступним необхідним етапом є розробка OBS (Organization Breakdown Structure). Організаційна структура представляє собою графічне відображення учасників проекту та їх відповідальних осіб, які задіяні в реалізації проекту. Список виконавців, що функціонують в проекті

представлений в таблиці Б.2. Організаційна структура проекту зображена на рисунку Б.2.

Таблиця Б.2 – Виконавці проекту

Роль	Проектна роль
Розробник (Акименко В.В.)	Виконує розробку основного функціоналу проекту.
Менеджер проекту (Марченко А.В.)	Відповідає за виконання термінів, виконує збір та аналіз даних.
Тестувальник	Відповідає за тестування функціоналу проекту

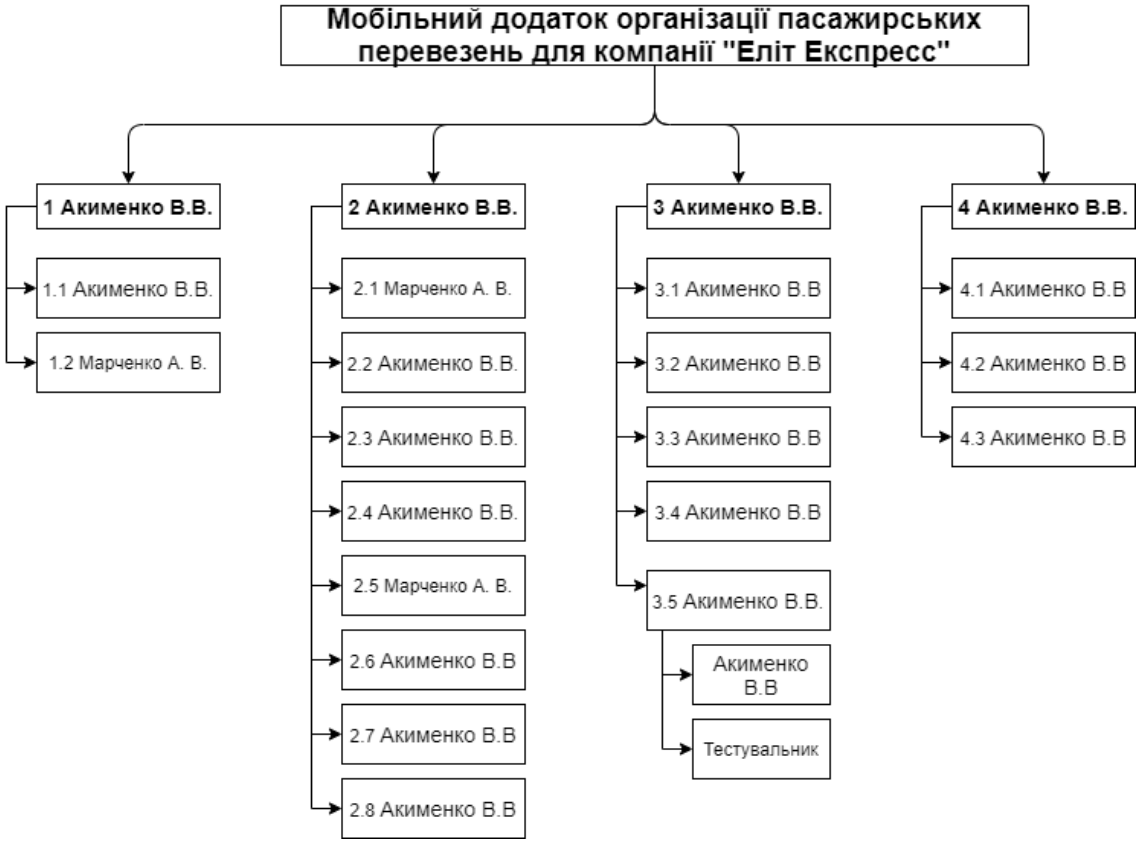


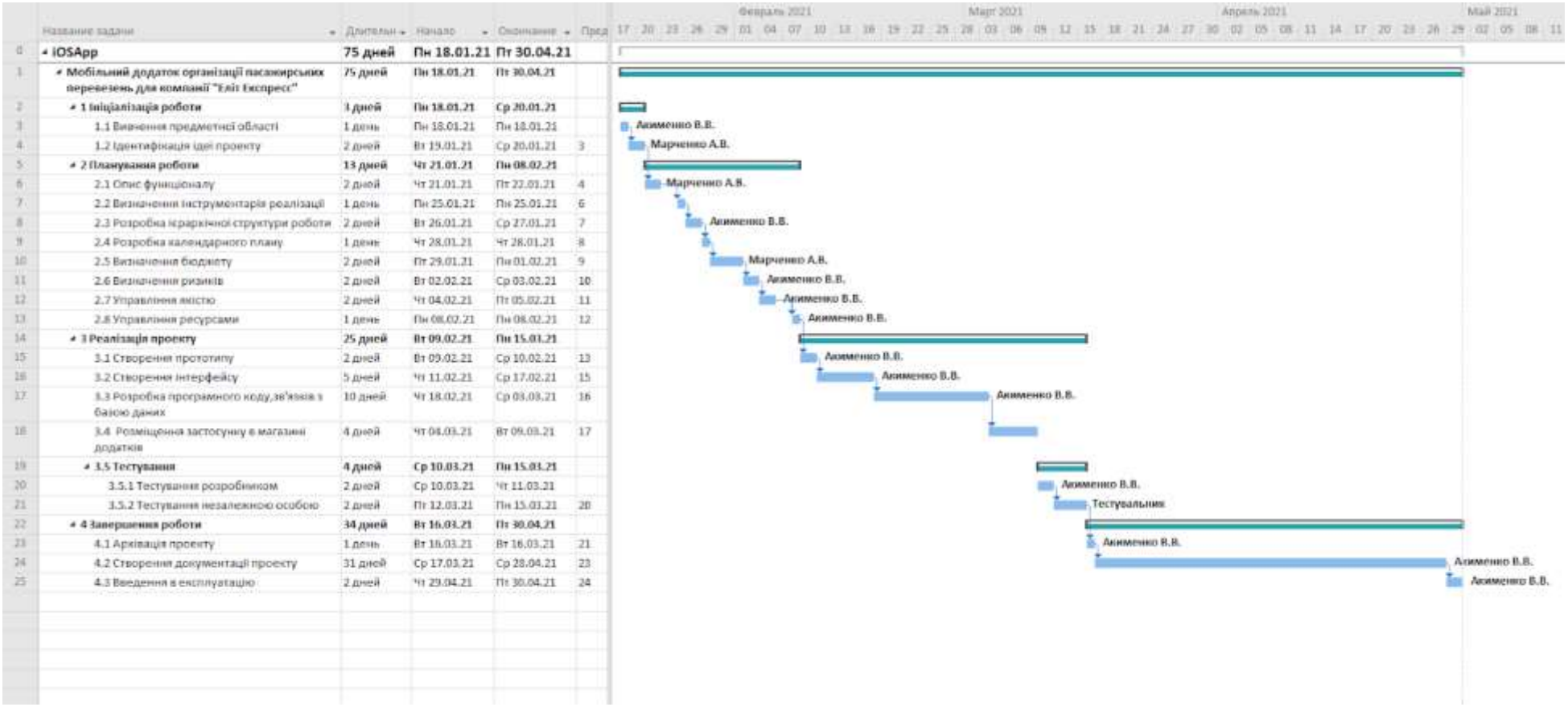
Рисунок Б.2 – OBS – структура проекту

Діаграма Ганта. Графіки та діаграми, спеціальні програми та процедури - це інструменти, які допомагають учасникам виконувати свої завдання та вдосконалювати їх. Як і PDM, діаграма Ганта вважається інструментом якості для

позначення цілей та завдань. Основною відмінністю цих інструментів є спосіб відображення інформації.

На відміну від PDM мережі, яка пропонує мережеву модель, управління проектами з діаграмами Ганта базується на форматі гістограми. Це допомагає відстежувати відсоток виконаної роботи над кожним завданням. Дуже важливо, щоб керівники проектів правильно розподіляли завдання та стежили за тим, щоб проект був виконаний вчасно. Діаграми Ганта фокусуються на відсотках кожного виконаного завдання. Крім того, діаграми Ганта найкраще підходять для проектів з невеликою кількістю взаємопов'язаних завдань.

Діаграма Ганта зображена на рисунку Б.3



	Название задачи	Длительн	Начало	Окончание	Пред
0	**iOSApp**	**75 дней**	**Пн 18.01.21**	**Пт 30.04.21**	
1	**Мобільний додаток організації пасажирських перевезень для компанії "Еліт Експресс"**	**75 дней**	**Пн 18.01.21**	**Пт 30.04.21**	
2	**1 Ініціалізація роботи**	**1 дней**	**Пн 18.01.21**	**Ср 20.01.21**	
3	1.1 Вивчення предметної області	1 день	Пн 18.01.21	Пн 18.01.21	
4	1.2 Ідентифікація ідеї проекту	2 дней	Вт 19.01.21	Ср 20.01.21	3
5	**2 Планування роботи**	**13 дней**	**Чт 21.01.21**	**Пн 08.02.21**	
6	2.1 Опис функціоналу	2 дней	Чт 21.01.21	Пт 22.01.21	4
7	2.2 Визначення інструментарія реалізації	1 день	Пн 25.01.21	Пн 25.01.21	6
8	2.3 Розробка ієрархічної структури роботи	2 дней	Вт 26.01.21	Ср 27.01.21	7
9	2.4 Розробка календарного плану	1 день	Чт 28.01.21	Чт 28.01.21	8
10	2.5 Визначення бюджету	2 дней	Пт 29.01.21	Пн 01.02.21	9
11	2.6 Визначення ризиків	2 дней	Вт 02.02.21	Ср 03.02.21	10
12	2.7 Управління якістю	2 дней	Чт 04.02.21	Пт 05.02.21	11
13	2.8 Управління ресурсами	1 день	Пн 08.02.21	Пн 08.02.21	12
14	**3 Реалізація проекту**	**25 дней**	**Вт 09.02.21**	**Пн 15.03.21**	
15	3.1 Створення прототипу	2 дней	Вт 09.02.21	Ср 10.02.21	13
16	3.2 Створення інтерфейсу	5 дней	Чт 11.02.21	Ср 17.02.21	15
17	3.3 Розробка програмного коду,зв'язків з базою даних	10 дней	Чт 18.02.21	Ср 03.03.21	16
18	3.4 Розміщення застосунку в магазині додатків	4 дней	Чт 04.03.21	Вт 09.03.21	17
19	**3.5 Тестування**	**4 дней**	**Ср 10.03.21**	**Пн 15.03.21**	
20	3.5.1 Тестування розробником	2 дней	Ср 10.03.21	Чт 11.03.21	
21	3.5.2 Тестування незалежною особою	2 дней	Пт 12.03.21	Пн 15.03.21	20
22	**4 Завершення роботи**	**34 дней**	**Вт 16.03.21**	**Пт 30.04.21**	
23	4.1 Архівація проекту	1 день	Вт 16.03.21	Вт 16.03.21	21
24	4.2 Створення документації проекту	31 дней	Ср 17.03.21	Ср 28.04.21	23
25	4.3 Введення в експлуатацію	2 дней	Чт 29.04.21	Пт 30.04.21	24

Рисунок Б.3 – Діаграма Ганта проекту

Аналіз ризиків. Ідентифікація ризиків визначає, які ризики можуть вплинути на проект, та документує характеристики цих ризиків. Визначення ризику не буде ефективним, якщо його не робити регулярно протягом усього проекту.

До ідентифікації ризиків має бути залучено якомога більше учасників: керівники проектів, замовники, користувачі, незалежні спеціалісти.

Класифікація ризиків:

1. За імовірністю виникнення:
   - слабо ймовірнісні;
   - мало ймовірнісні;
   - імовірні;
   - досить імовірні;
   - майже імовірні.
2. За величиною втрат:
   - мінімальна;
   - низька;
   - середня;
   - висока;
   - максимальна.

На основі цієї інформації була проведена класифікація ризиків для даного проекту, що показана в таблиці Б.3.

Таблиця Б.3 – Класифікація ризиків

Назва ризику	Ймовірність	Величина втрат
Не точно складене ТЗ	2	4
Недотримання графіку робіт	1	3
Збої в роботі програмного забезпечення	4	4
Збої в роботі апаратного забезпечення	4	4
Хвороба основного розробника	2	2
Некоректне тестування	2	1
Пошкодження файлів	4	5

На основі даної класифікації ризиків була побудована матриця ризиків проекту, що представлена на рисунку Б.4.

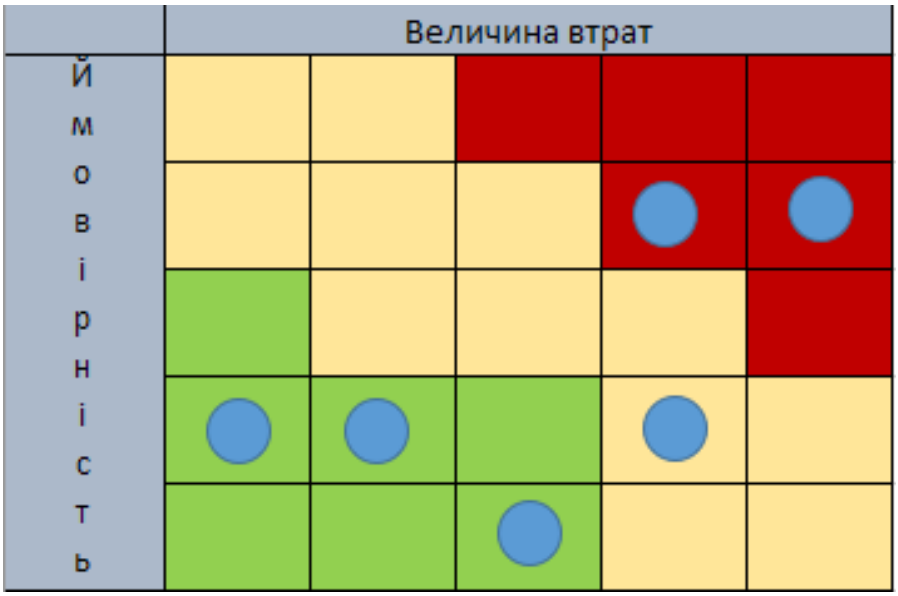


Рисунок Б.4 – Матриця ризиків

Після оцінки ризиків було складено план реагування на ризики (табл. Б.4)

Таблиця Б.4 – Реагування на ризики

Ризики проекту	Реакція на ризик
Не точно складене ТЗ	Зосередити увагу на моментах, що були виконані неправильно (після розмови із замовником) та внести відповідні правки до проекту.
Недотримання графіку робіт	1. Обговорення можливості внесення змін в умови реалізації з керівником та замовником. 2. Узгодити умови зміни вимог із замовником. Якщо це неможливо, реорганізувати роботу таким чином, щоб дотримати отримані терміни.

Продовження таблиці Б.4

Ризики проекту	Реакція на ризик
Збої в роботі програмного забезпечення	Перезапуск або переустановлення програми, яка дала збій, повернутися до попередньо-збереженої версії проекту, якщо останні дані остаточно втрачені.
Збої в роботі апаратного забезпечення	В найшвидший термін виконати ремонт апаратного забезпечення, знайти тимчасову заміну на час ремонту, якщо терміни виконання робіт не дозволяють затримки, завантажити останню резервну копію проекту та продовжувати роботу над проектом.
Хвороба основного розробника	Медичне страхування виконавця у випадку хвороби при роботі над проектом. Врахувати можливість відтермінування дат виконання завдань та залишити декілька резервних днів у разі хвороби.
Некоректне тестування	Відправити проект на додаткове тестування кваліфікованому спеціалісту.
Пошкодження файлів	Створювати резервні копії файлів після кожного етапу роботи, відновити роботу над завданнями з останньої версії проекту.

ДОДАТОК В

Файли коду реалізації

Файл LoadViewController.swift:

```
import UIKit

class LoadViewController: UIViewController {
    var version = AppVersion()
    let appVersion = Bundle.main.infoDictionary?["CFBundleShortVersionString"] as?
String
    //MARK:- VIEW CYCLE
    override func viewDidLoad() {
        super.viewDidLoad()
    }

    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(true)
        showSpinner()
        checkConnectionAndVersion()
    }

    // MARK: - CHECK CONNECTION
    func checkConnectionAndVersion(){
        if Reachability.isConnectedToNetwork() == true{
            getVersionJSON(postString: "")
            CommonFunctions.delay(2.0){
                self.removeSpinner()
                if self.version[0].appVersion.isEmpty == true
                {
                    print("cannot connect")
                    print(self.appVersion!)
                    CommonFunctions.changeViewController(controllerName:
"LoginViewController")
                }
                print(self.version[0].appVersion)
                print(self.appVersion!)
                if self.version[0].appVersion == self.appVersion! {
                    CommonFunctions.changeViewController(controllerName:
"LoginViewController")
                }
                else{
                    Alerts.versionAlert(controllerName: self)
                }
            }
        }
        else{
            let alert = UIAlertController(title: "РџPsPjPёP»PєP°", message:
"РџРµСЂРµIC-СЂС,Рµ РїС-РґРєР»СЂС‡РµPSPSCЦ РґPs С-PSC,РµСЂPSPµC,Сѓ.", preferredStyle:
.alert)
            alert.addAction(UIAlertAction(title: NSLocalizedString("OK", comment:
"Default action"), style: .default, handler: { _ in
                self.checkConnectionAndVersion()
            }))
            self.present(alert, animated: true, completion:  nil)
        }
    }
    func getVersionJSON(postString:String){
        LoadJSON.loadJson(page: "", fromURLString: postString) { (result) in
```

```
            switch result {
            case .success(let data):
                print(data)
                self.version = LoadJSON.parseJSON(jsonData: data, type:
AppVersion.self, controllerName: self) as! AppVersion
            case .failure(let error):
                print(error)
            }
        }
    }
}
```

Файл LoginViewController.swift:

```
import UIKit

class LoginViewController: UIViewController, UITextFieldDelegate{
    // MARK: - OUTLETS
    @IBOutlet weak var scrollView: UIScrollView!
    @IBOutlet weak var nameTextField: UITextField!
    @IBOutlet weak var phoneTextField: UITextField!
    @IBOutlet weak var continueButton: UIButton!

    var appDelegate = AppDelegate()
    let defaults = UserDefaults.standard

    //MARK:- VIEW CYCLE
    override func viewDidLoad() {
        super.viewDidLoad()
        showSpinner()
        continueButton.layer.cornerRadius = continueButton.frame.size.height/2
        if defaults.string(forKey: "name") != nil && defaults.string(forKey: "phone")
!= nil{
            self.view.isUserInteractionEnabled = false
            nameTextField.text = defaults.string(forKey: "name")
            phoneTextField.text = defaults.string(forKey: "phone")
            CommonFunctions.delay(0.5){
                self.removeSpinner()
                self.view.isUserInteractionEnabled = true
                if Reachability.isConnectedToNetwork() == true {
                    CommonFunctions.changeViewController(controllerName:
"MainViewController")
                }
                else {
                    Alerts.alert(controllerName: self)
                }
            }
        }
        else{
            removeSpinner()
        }
        registerForKeyboardNotifications()
        nameTextField.delegate = self
        phoneTextField.delegate = self
        self.hideKeyboardWhenTappedAround()
    }
    // MARK: - CONTINUE BUTTON
    @IBAction func ContinueButtonTapped(_ sender: Any) {
        checkDefaultData()
    }
    func checkDefaultData(){
        if KeyboardFunctions.numberOfChars(nameTextField: self.nameTextField,
phoneTextField: self.phoneTextField, controllerName: self) == true{
            KeyboardFunctions.checkInput(nameTextField: self.nameTextField,
phoneTextField: self.phoneTextField)
```

```
            if Reachability.isConnectedToNetwork() == true {
                CommonFunctions.changeViewController(controllerName:
"MainViewController")
            }
            else {
                Alerts.alert(controllerName: self)
            }
        }
    }
    // MARK: - KEYBOARD FUNCTIONS
    deinit {
        removeKeyboardNotifications()
    }
    func textFieldShouldBeginEditing(_ textField: UITextField) -> Bool {
        if textField == nameTextField{
            phoneTextField.resignFirstResponder()
            return true
        }
        else if textField == phoneTextField {
            nameTextField.resignFirstResponder()
            return true
        }
        return true
    }
    //Check input
    func textField(_ textField: UITextField, shouldChangeCharactersIn range: NSRange,
replacementString string: String) -> Bool {
        if textField == phoneTextField{
            guard let text = textField.text else { return false }
            let newString = (text as NSString).replacingCharacters(in: range, with:
string)
            textField.text = KeyboardFunctions.format(with: "XXX XXX XXXX",
phone:newString)
            return false
        }
        return true
    }
}
```

Файл MainViewController.swift:

```
import UIKit
import SafariServices

protocol SegueHandler: AnyObject {
    func segueToNext(identifier: String)
}
class MainViewController: UIViewController, SFSafariViewControllerDelegate,
SegueHandler {

    //MARK:- VARIABLES
    let order = OrderInformation.shared
    var zones = [ZoneElement]()
    var orderID = [OrderElement]()
    var defaults = UserDefaults.standard

    @IBOutlet weak var bookingButton: UIButton!

    override func viewDidLoad() {
        super.viewDidLoad()
        bookingButton.layer.cornerRadius = bookingButton.frame.size.height/2
        registerForKeyboardNotifications()

    }
```

```
    // MARK: - BOOK TRIP
    @IBAction func BookTrip(_ sender: Any) {
        bookingTripAlert()
    }
    @IBAction func AdditionalInfoAlert(_ sender: Any){
        Alerts.additionalInformation(controllerName: self)
    }
    func bookingTripAlert(){
        showSpinner()
        guard Reachability.isConnectedToNetwork() == true
        else {
            Alerts.alert(controllerName: self)
            return
        }
        if order.departureZone == "" || order.destinationZone == "" ||
order.orderDepartmentTime == "depTime" ||
CommonFunctions.checkCurrentDateWithChosen(deptime: order.orderDepartmentTime, date:
order.date) == false {
            removeSpinner()
            Alerts.missedDataAlert(controllerName: self)
        }
        else {
            guard order.availablePlaces >= Int(order.placeLabel)! else {
                removeSpinner()
                Alerts.missedAvailablePlaces(controllerName: self)
                return
            }
            let postString =
"app=ios&status=3&marshrut_reis=\(order.orderRoute)&zone=\(Int(order.departureZone)!+
Int(order.destinationZone)!)"
            print(postString)
            self.getZoneJSON(postString: postString)
            self.view.isUserInteractionEnabled = false
            DispatchQueue.main.asyncAfter(deadline: .now() + 1.0)
            {
                self.removeSpinner()
                self.view.isUserInteractionEnabled = true
                guard !self.zones.isEmpty
                else{
                    Alerts.parseErrorAlert(controllerName: self)
                    return
                }
                self.order.orderPrice = String(Double(self.zones[0].price!)!)
                self.orderMessageAlert()
            }
        }
    }
    // MARK: - RESULT MESSAGE ALERT
    func resultMessageAlert(){
        let alert = UIAlertController(title: "РћС–СЃС†Рµ Р·Р°Р±РѕРЅРµРІР°РЅРѕ",
message: "Р’Р°Рj РЅРµРѕР±Х…С–РґРЅРѕ РїС–РґС‚Рё,РІРµСЂРґРёС‚Ь СЄР»СЏС…РѕРj
РѕРїР»Р°С‚Рё. РїРєС‰Рѕ РѕРїР»Р°С‚Р° РЅСѓРґРµРЅРѕ Р·РґС–Р№СЃРЅРёРµРЅРѕ
РїСЂРѕС‚Рѕ 1 РіРѕРґРёРЅРё, Р±СЂРѕРЅСЊ Р±СѓРґРµ Р°РІС‚РѕРјР°С‚РёС‡РЅРѕ
СЃРєР°СЃРѕРІР°РЅРѕ" , preferredStyle: UIAlertController.Style.alert)
        alert.addAction(UIAlertAction(title: "РћРїР»Р°С‚РёС‚Рё РєР°СЂС‚Рѕ", style:
UIAlertAction.Style.default, handler:{action in
            let postString = "orderID=\(self.orderID[0].order)"
            if let url = URL(string: postString){
                let controller = SFSafariViewController(url: url)
                self.present(controller, animated: true, completion: nil)
                controller.delegate = self
            }
        }))
        self.present(alert, animated: true, completion:  nil)
```

```
    }
    func orderMessageAlert(){
        let amount = Double(self.zones[0].price!)! * Double(self.order.placeLabel)!
        let message = "РљС–Р»СЊРєС–СЃС‚СЊ: \(self.order.placeLabel) \n
Р'С–РґРїСЂР°РІР»РµРЅРЅСЏ:" + self.zones[0].nameZone! + "\n Р§Р°СЃ PIC-
РґРїСЂР°РІР»РµРЅРЅСЏ: \(self.order.timeLabel) \n Р”Р°С‚Р°: \(self.order.date) \n
РџР°СЃР°Р¶РёСЂ: \(self.defaults.string(forKey:
"name")!)(\(self.defaults.string(forKey: "phone")!)) \nР'Р°СЂС,С–СЃ,СЊ:\(amount)
РіСЂРЅ"
        let alert = UIAlertController(title: "РџС–РґС‚РІРµСЂРґРёС‚Рё
Р·Р°РјРѕРІР»РµРЅРќ", message: message , preferredStyle:
UIAlertController.Style.alert)
        alert.addAction(UIAlertAction(title: "Р'С–РґРјС–РЅРёС‚Рё", style:
UIAlertAction.Style.cancel, handler:{action in}))
        alert.addAction(UIAlertAction(title: "РћРљ", style:
UIAlertAction.Style.default, handler: {(alert:UIAlertAction!)->Void in
            let phone = UserDefaults.standard.string(forKey: "phone")!
            let separator = phone.components(separatedBy: " ")
            let phoneforStatus = separator[0]+separator[1]+separator[2]
            let postString =
"status=4&app=ios&marshrut_reis=\(self.order.orderRoute)&id_reis=\(self.order.orderTi
meID)"+"&date=\(self.order.orderDate)&"+"time=\(self.order.orderDepartmentTime)&fio=\
(String(describing: self.defaults.string(forKey:
"name")!))&"+"tel=\(phoneforStatus)"+"&mest=\(self.order.orderPlaceAmount)&"+"id_from
=\(self.order.orderIdFrom)&id_to=\(self.order.orderIdTo)&"+"price=\(self.order.orderP
rice)&info=\(self.order.comments)"
            print(postString)
            self.getOrderJSON(postString: postString)
            self.resultMessageAlert()
        }))
        self.present(alert, animated: true, completion:  nil)
    }
    // MARK: - JSON DATA
    func getZoneJSON(postString:String){
        LoadJSON.loadJson(page: "", fromURLString: postString) { (result) in
            switch result {
            case .success(let data):
                print(data)
                let str = String(decoding: data, as: UTF8.self)
                print(str)
                self.zones = LoadJSON.parseJSON(jsonData: data, type: Zone.self,
controllerName: self) as! Zone
                print(self.zones)
            case .failure(let error):
                print(error)
            }
        }
    }
    func getOrderJSON(postString:String){
        LoadJSON.loadJson(page: "", fromURLString: postString) { (result) in
            switch result {
            case .success(let data):
                print(data)
                self.orderID = LoadJSON.parseJSON(jsonData: data, type: Order.self,
controllerName: self) as! Order
            case .failure(let error):
                print(error)
            }
        }
    }
    // MARK: - SEGUE
    func segueToNext(identifier: String) {
        self.performSegue(withIdentifier:
                          identifier, sender: self)
```

```
    }
    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        if (segue.identifier == "tableviewEmbed") {
            let myTableViewController = segue.destination as! MainTableViewController
            myTableViewController.delegate = self
        }
    }
    // MARK: - KEYBOARD FUNCTIONS
    deinit {
        removeKeyboardNotifications()
    }

}
```

Файл MainTableViewController.swift:

```
import UIKit

class MainTableViewController: UITableViewController, UITextFieldDelegate
{

    //MARK:- VARIABLES
    weak var delegate: SegueHandler?
    var order = OrderInformation.shared
    //MARK:- OUTLETS
    @IBOutlet weak var RouteLabel: UILabel!
    @IBOutlet weak var DepartmentLabel: UILabel!
    @IBOutlet weak var DestinationLabel: UILabel!
    @IBOutlet weak var TimeLabel: UILabel!
    @IBOutlet weak var PlaceLabel: UILabel!
    @IBOutlet weak var PlaceStepper: UIStepper!
    @IBOutlet weak var CommentsTextField: UITextField!
    @IBOutlet weak var DatePicker: UIDatePicker!

    //MARK:- VIEW LIFE CYCLE
    override func viewDidLoad() {
        super.viewDidLoad()
        CommentsTextField.delegate = self
        self.hideKeyboardWhenTappedAround()
        datePickerSetup()
    }

    override func viewWillAppear(_ animated: Bool) {
        RouteLabel.text = order.routeLabel
        DepartmentLabel.text = order.departureLabel
        DestinationLabel.text = order.destinationLabel
        TimeLabel.text = order.timeLabel
        PlaceLabel.text = order.placeLabel
        order.printInfo()
    }


    @IBAction func PlaceChanged(_ sender: UIStepper) {
        PlaceLabel.text = String(Int(sender.value))
        order.placeLabel = String(Int(sender.value))
        order.orderPlaceAmount = Int(sender.value)
        order.printInfo()
    }
    //MARK:- KEYBORARD FUNCTIONS

    func textFieldDidEndEditing(_ textField: UITextField) {
        print(textField.text!)
        order.comments = textField.text ?? ""
    }
    //MARK: - DATEPICKER FUNCTIONS
```

```
    @objc func datePickerChange(datePicker: UIDatePicker)->String{
        if datePicker.isEqual(self.DatePicker){
            let dateFormatter = DateFormatter()
            dateFormatter.dateFormat = "YYYY-MM-dd"
            let date = dateFormatter.string(from: datePicker.date)
            order.date = date
            print(date)
            let previousDate = dateFormatter.string(from:
datePicker.date.addingTimeInterval(-60*60*24))
            order.previousDayDate = previousDate
            print(previousDate)
            order.orderDate = order.date
            order.orderDepartmentTime = "depTime"
            order.timeLabel = "P’PëP±CЪP°C,Pë..."
            TimeLabel.text = order.timeLabel
            return date
        }
        return ""
    }
    func datePickerSetup()->Void{
        var oneMonthInterval = TimeInterval()
        oneMonthInterval = 31 * 24 * 60 * 60
        let currentDate = Date()
        if #available(iOS 13.4, *) {
            DatePicker.preferredDatePickerStyle = UIDatePickerStyle.wheels
        }
        DatePicker.minimumDate = currentDate
        DatePicker.maximumDate = currentDate.addingTimeInterval(oneMonthInterval)
        DatePicker.addTarget(self, action: #selector(datePickerChange(datePicker:)),
for: .valueChanged)
        order.date = datePickerChange(datePicker: DatePicker)
    }
    // MARK: - Table view data source
    override  func tableView(_ tableView: UITableView, didSelectRowAt
                                indexPath: IndexPath){
        guard Reachability.isConnectedToNetwork() == true
        else {
            Alerts.alert(controllerName: self)
            return
        }
        switch indexPath.section {
        case 0:
            delegate?.segueToNext(identifier: "Route")
            break;
        case 1:
            if order.orderRoute == "route"{
                Alerts.routeAlert(controllerName: self)
            }
            else{
                delegate?.segueToNext(identifier: "Department")
            }
            break;
        case 2:
            if order.orderRoute == "route"{
                Alerts.routeAlert(controllerName: self)
            }
            else{
                delegate?.segueToNext(identifier: "Destination")
            }
            break;
        case 4:
            if order.orderIdFrom == "idFrom"{
                Alerts.dateAndTimeAlert(controllerName: self)
            }
```

```
            else{
                delegate?.segueToNext(identifier: "Time")
            }
            break;
        default:
            break;
        }

    }
    override func numberOfSections(in tableView: UITableView) -> Int {
        return 7
    }
    override func tableView(_ tableView: UITableView, numberOfRowsInSection section:
Int) -> Int {
        return 1
    }
    override func tableView(_ tableView: UITableView, heightForRowAt indexPath:
IndexPath) -> CGFloat {

        if indexPath.section == 3{
            return 100
        }
        else{

            let row = Int((tableView.frame.size.height-140)/12)
            return CGFloat(row)
        }

    }
    override func tableView(_ tableView: UITableView, heightForFooterInSection
section: Int) -> CGFloat {
        return 2
    }
    override func tableView(_ tableView: UITableView, heightForHeaderInSection
section: Int) -> CGFloat {
        return 30
    }
}
```

Файл HistoryTableViewController.swift:

```
import UIKit
import SafariServices

class HistoryTableViewController:
UITableViewController,SFSafariViewControllerDelegate {
    //MARK:- VARIABLES
    var history = [HistoryElement]()
    var finishedTripIndex = [Int]()
    var unfinishedTripIndex = [Int]()
    var phone = ""

    override func viewDidLoad() {
        super.viewDidLoad()
        showSpinner()
        getPhone()
        getHistory()
        tableView.tableFooterView = UIView()
    }
    @IBAction func refreshDataAction(_ sender: Any) {
        refreshData()
    }
    func refreshData()->Void{
        showSpinner()
        CommonFunctions.delay(1.0){
```

```
            self.history.removeAll()
            self.unfinishedTripIndex.removeAll()
            self.finishedTripIndex.removeAll()
            self.getHistory()
        }
    }
    func getPhone()->Void{
        let defaultphone = UserDefaults.standard.string(forKey: "phone")!
        let separator = defaultphone.components(separatedBy: " ")
        phone = separator[0]+separator[1]+separator[2]
    }
    func getHistory()->Void{
        let postString = "app=ios&status=5&phone=\(phone)"
        print(postString)
        getHistoryJSON(postString: postString)
    }
    //MARK:- JSON parsing
    func getHistoryJSON(postString:String){
        LoadJSON.loadJson(page: "", fromURLString: postString) { (result) in
            switch result {
            case .success(let data):
                print(data)
                self.history = LoadJSON.parseJSON(jsonData: data, type:History.self,
controllerName: self) as! History
                DispatchQueue.main.async {
                    self.getHistoryIndex()
                    self.tableView.reloadData()
                    self.removeSpinner()
                }
            case .failure(let error):
                print(error)
            }
        }
    }

    func getHistoryIndex(){
        if self.history.count > 0 {
            for i in 0...self.history.count-1 {
                switch Int(self.history[i].status!) {
                case 2,3:
                    self.unfinishedTripIndex.append(i)
                    break
                case 4:
                    self.finishedTripIndex.append(i)
                default:
                    break
                }
            }
            print(self.unfinishedTripIndex.count+self.finishedTripIndex.count)
        }
    }
    // MARK: - Table view data source

    override func numberOfSections(in tableView: UITableView) -> Int {
        return 2
    }

    override func tableView(_ tableView: UITableView, numberOfRowsInSection section:
Int) -> Int {

        if section == 0{
            if self.unfinishedTripIndex.count == 0{
                return 1
            }
```

```
            else{
                return self.unfinishedTripIndex.count
            }
        }
        else{
            if self.finishedTripIndex.count == 0{
                return 1
            }
            else{
                return self.finishedTripIndex.count
            }
        }
    }


    override func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {

        let cell = tableView.dequeueReusableCell(withIdentifier: "HistoryCell", for:
indexPath)
        cell.textLabel?.textColor =  .init(red:0.2385, green: 0.2050, blue: 0.5565,
alpha: 1.0)
        if indexPath.section == 0{
            if self.unfinishedTripIndex.count == 0{
                cell.textLabel?.text = "РџPsC—P·PrPєPë PIC-PrCЃCґC,PSC-"
                cell.detailTextLabel?.text = ""
            }
            else if self.unfinishedTripIndex.count > 0{
                let index = unfinishedTripIndex[indexPath.row]
                cell.textLabel?.text = "\(self.history[index].marshrut!)"
                cell.detailTextLabel?.text = "P”P°C,P°:\(self.history[index].dates!)
\(self.history[index].times!) РњC-CЃC†CЊ:\(self.history[index].mest!)"

                if Int(self.history[index].status!) == 2{
                    cell.textLabel?.textColor = .red
                }
            }
            return cell
        }
        else
        {
            if self.finishedTripIndex.count == 0{
                cell.textLabel?.text = "РџPsC—P·PrPєPë PIC-PrCЃCґC,PSC-"
                cell.detailTextLabel?.text = ""
            }
            else{
                let index = finishedTripIndex[indexPath.row]
                cell.textLabel?.text = "\(self.history[index].marshrut!)"
                cell.detailTextLabel?.text = "P”P°C,P°:\(self.history[index].dates!)
\(self.history[index].times!) РњC-CЃC†CЊ:\(self.history[index].mest!)"
            }
            return cell
        }
    }
    override func tableView(_ tableView: UITableView, titleForHeaderInSection
section: Int) -> String? {
        if section == 0 {
            return "P—P°P±CЂPsPSCЊPsPIP°PSC- PїPsC—P·PrPєPë"
        }
        if section == 1{
            return "P—P°PIPµCЂC€PµPSC- PїPsC—P·PrPєPë"
        }
        return ""
    }
```

```
    override func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
IndexPath) {
        if indexPath.section == 0 {
            guard unfinishedTripIndex.count != 0 else {return}
            guard Int(history[unfinishedTripIndex[indexPath.row]].status!) == 2 else
{return}

                if let url = URL(string: postString){
                    let controller = SFSafariViewController(url: url)
                    self.present(controller, animated: true, completion: nil)
                    controller.delegate = self
                }
            }))
            self.present(alert, animated: true, completion:  nil)
        }
    }
    // MARK: - Safari view controller
    func safariViewControllerDidFinish(_ controller: SFSafariViewController) {
        controller.dismiss(animated: true, completion: nil)
        refreshData()
    }
}
```

Файл ProfileTableViewController.swift:

```
import UIKit

class ProfileTableViewController: UITableViewController, UITextFieldDelegate {

    @IBOutlet weak var nameTextField: UITextField!
    @IBOutlet weak var phoneTextField: UITextField!

    let defaults = UserDefaults.standard

    override func viewDidLoad() {
        super.viewDidLoad()
        nameTextField.delegate = self
        phoneTextField.delegate = self
        if defaults.string(forKey: "name") != nil && defaults.string(forKey: "phone")
!= nil{
            nameTextField.text = defaults.string(forKey: "name")
            phoneTextField.text = defaults.string(forKey: "phone")
        }
        self.hideKeyboardWhenTappedAround()
        tableView.tableFooterView = UIView()
    }
    @IBAction func dismissProfileView(_ sender: Any){
        dismiss(animated: true, completion: nil)
    }
    @IBAction func saveUserDefaults(_ sender: Any) {
        if KeyboardFunctions.numberOfChars(nameTextField: self.nameTextField,
phoneTextField: self.phoneTextField, controllerName: self) == true{
            KeyboardFunctions.checkInput(nameTextField: self.nameTextField,
phoneTextField: self.phoneTextField)
            dismiss(animated: true, completion: nil)
        }
    }
    // MARK: - Table view data source

    override func numberOfSections(in tableView: UITableView) -> Int {return 1}
    override func tableView(_ tableView: UITableView, numberOfRowsInSection section:
Int) -> Int {return 2}
    override func tableView(_ tableView: UITableView, heightForRowAt indexPath:
IndexPath) -> CGFloat {return  tableView.frame.size.height/16}
```

```
    // MARK: - KEYBOARD FUNCTIONS
    deinit {
        removeKeyboardNotifications()
    }
    func textFieldShouldBeginEditing(_ textField: UITextField) -> Bool {
        if textField == nameTextField{
            phoneTextField.resignFirstResponder()
            return true
        }
        else if textField == phoneTextField {
            nameTextField.resignFirstResponder()
            return true
        }
        return true
    }
    //Check input
    func textField(_ textField: UITextField, shouldChangeCharactersIn range: NSRange,
replacementString string: String) -> Bool {
        if textField == phoneTextField{
            guard let text = textField.text else { return false }
            let newString = (text as NSString).replacingCharacters(in: range, with:
string)
            textField.text = KeyboardFunctions.format(with: "XXX XXX XXXX",
phone:newString)
            return false
        }
        return true
    }
}
```

Файл DepartureTableViewController.swift:

```
import UIKit

class DepartureTableViewController: UITableViewController {
    var places = [PlaceElement]()
    var arrayDepartment = [Int]()
    let order = OrderInformation.shared


    override func viewDidLoad() {
        super.viewDidLoad()
        let postString = "app=ios&status=2&marshrut_reis=\(order.orderRoute)"
        print(postString)
        showSpinner()
        getPlacesJSON(postString: postString)
        tableView.tableFooterView = UIView()
    }
    //MARK:- JSON
    func getPlacesJSON(postString:String){
        LoadJSON.loadJson(page: "", fromURLString: postString) { (result) in
            switch result {
            case .success(let data):
                print(data)
                self.places = LoadJSON.parseJSON(jsonData: data, type: Place.self,
controllerName: self) as! Place
                DispatchQueue.main.async {
                    self.addDigits()
                    self.tableView.reloadData()
                    self.removeSpinner()
                }
            case .failure(let error):
                print(error)
            }
        }
```

```
    }
    func addDigits(){
        if self.places.count > 1{
            for i in 0...self.places.count-1 {
                if self.places[i].otprID != nil {
                    self.arrayDepartment.append(i)
                }
            }
        }
    }

    // MARK: - Table view data source

    override func numberOfSections(in tableView: UITableView) -> Int {
        return 1
    }

    override func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
IndexPath) {
        order.departureLabel = self.places[arrayDepartment[indexPath.row]].cityOtpr!
        order.departureZone = self.places[arrayDepartment[indexPath.row]].optrZone!
        order.orderIdFrom = self.places[arrayDepartment[indexPath.row]].otprID!
        order.addTime = self.places[arrayDepartment[indexPath.row]].addtime!
        order.orderDepartmentTime = "depTime"
        order.timeLabel = "P’PëP±CЂP°C,Pë..."
        self.navigationController?.popViewController(animated: true)
    }
    override func tableView(_ tableView: UITableView, numberOfRowsInSection section:
Int) -> Int {
        return arrayDepartment.count
    }
    override func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier: "DepartmentCell",
for: indexPath)
        cell.textLabel?.text =
"\(self.places[arrayDepartment[indexPath.row]].cityOtpr!)"
        return cell
    }
}
```

Файл DestinationTableViewController.swift:

```
import UIKit

class DestinationTableViewController: UITableViewController {
    var places = [PlaceElement]()
    var arrayDestination = [Int]()
    let order = OrderInformation.shared

    override func viewDidLoad() {
        super.viewDidLoad()
        let postString =
"app=ios&timestamp=1&status=2&marshrut_reis=\(order.orderRoute)"
        print(postString)
        showSpinner()
        self.getPlacesJSON(postString: postString)
        tableView.tableFooterView = UIView()
    }
    //MARK:- JSON
    func getPlacesJSON(postString:String){
        LoadJSON.loadJson(page: "", fromURLString: postString) { (result) in
            switch result {
            case .success(let data):
                print(data)
```

```
                self.places = LoadJSON.parseJSON(jsonData: data, type: Place.self,
controllerName: self) as! Place
                DispatchQueue.main.async {
                    self.addDigits()
                    self.tableView.reloadData()
                    self.removeSpinner()
                }
            case .failure(let error):
                print(error)
            }
        }
    }
    func addDigits(){
        if self.places.count > 1{
            for i in 0...self.places.count-1 {
                if self.places[i].pribID != nil {
                    self.arrayDestination.append(i)
                }
            }
        }
    }

    // MARK: - Table view data source

    override func numberOfSections(in tableView: UITableView) -> Int {
        // #warning Incomplete implementation, return the number of sections
        return 1
    }
    override func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
IndexPath) {
        order.destinationLabel =
self.places[arrayDestination[indexPath.row]].pribCity!
        order.destinationZone =
self.places[arrayDestination[indexPath.row]].pribZone!
        order.orderIdTo = self.places[arrayDestination[indexPath.row]].pribID!
        self.navigationController?.popViewController(animated: true)
    }
    override func tableView(_ tableView: UITableView, numberOfRowsInSection section:
Int) -> Int {
        return arrayDestination.count
    }
    override func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier: "DestinationCell",
for: indexPath)
        cell.textLabel?.text =
"\(self.places[arrayDestination[indexPath.row]].pribCity!)"
        return cell
    }
}
```

Файл TimetableViewController.swift:

```
import UIKit

class TimeTableViewController: UITableViewController {

    //MARK:- VARIABLES
    let order = OrderInformation.shared
    var timeSample = [TimeSampleElement]()
    var timeHour = [String]()
    var currentDayTimes = [TimeElement]()
    var previousDayTimes = [TimeElement]()
    var addMinutes:String = ""
```

```
    override func viewDidLoad() {
        super.viewDidLoad()
        showSpinner()
        let postString2 =
"status=1&app=ios&marshrut_reis=\(order.orderRoute)&data_reis=\(order.date)"
        print(postString2)
        getCreatedTimesJSON(postString: postString2, caseElement: 1)
        let postString3 =
"status=1&app=ios&marshrut_reis=\(order.orderRoute)&data_reis=\(order.previousDayDate
)"
        print(postString3)
        getCreatedTimesJSON(postString: postString3, caseElement: 2)
        addMinutes = getAdditionalTime(addtime: order.addTime)
        let postString = "app=ios&status=7&marshrut_reis=\(order.orderRoute)"
        print(postString)
        getTimeJSON(postString:postString)
    }
    // MARK: - Table view data source

    override func numberOfSections(in tableView: UITableView) -> Int {
        return 1
    }

    override func tableView(_ tableView: UITableView, numberOfRowsInSection section:
Int) -> Int {
        return timeSample.count
    }
    override func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier: "TimeCell", for:
indexPath)
        cell.textLabel?.text = "\(timeHour[indexPath.row]):\(addMinutes)"
        return cell
    }
    override func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
IndexPath) {
        order.orderDepartmentTime = "\(timeHour[indexPath.row]):\(addMinutes)"
        order.timeLabel = order.orderDepartmentTime
        let time = timeDivision(chosenTime: order.orderDepartmentTime, additonalTime:
order.addTime)
        getTripId(tripTime: time.tripTime, timeCase: time.timeCase)
        self.navigationController?.popViewController(animated: true)
    }
    //MARK:- Order time calculation
    func timeDivision(chosenTime:String,additonalTime:String)-
>(tripTime:Int,timeCase:Int){
        var timeCase:Int
        let chosenTimeArray = chosenTime.components(separatedBy: ":")
        let chosenHour = Int(chosenTimeArray[0])!
        let chosenMinutes = Int(chosenTimeArray[1])!
        let additionalTimeArray = additonalTime.components(separatedBy: ":")
        let additionalHour = Int(additionalTimeArray[0])!
        let additionalMinutes = Int(additionalTimeArray[1])!
        var tripTime = 60*(chosenHour-additionalHour)+(chosenMinutes-
additionalMinutes)
        switch tripTime {
        case 0:
            timeCase = 1
            break
        case ..<0:
            timeCase = 2
            tripTime = 24*60+tripTime
            break
        case 1...:
```

```
            timeCase = 3
            break
        default:
            timeCase = 1
            break
        }
        return (tripTime,timeCase)

    }
    //MARK:- Changing order information
    func getTripId(tripTime:Int,timeCase:Int){
        switch timeCase {
        case 1:
            order.orderTimeID = "-999"
            order.orderDepartmentTime = "depTime"
            order.availablePlaces = 15
            break
        case 2:
            order.orderDate = order.previousDayDate
            guard self.previousDayTimes.count != 0  else {
                order.orderTimeID = "1"
                order.orderDepartmentTime = "\(tripTime/60):00"
                order.availablePlaces = 15
                return
            }
            for i in 0...self.previousDayTimes.count-1{
                let chosenTimeArray =
previousDayTimes[i].times.components(separatedBy: ":")
                let chosenHour = Int(chosenTimeArray[0])!
                if chosenHour*60 == tripTime{
                    order.orderTimeID = "\(previousDayTimes[i].id)"
                    order.orderDepartmentTime = "\(tripTime/60):00"
                    order.availablePlaces = previousDayTimes[i].mesta
                    break
                }
                else{
                    order.orderTimeID = "1"
                    order.orderDepartmentTime = "\(tripTime/60):00"
                    order.availablePlaces = 15
                }
            }
            break
        case 3:
            guard self.currentDayTimes.count != 0  else {
                order.orderTimeID = "1"
                order.orderDepartmentTime = "\(tripTime/60):00"
                order.availablePlaces = 15
                return
            }
            for i in 0...self.currentDayTimes.count-1{
                let chosenTimeArray =
currentDayTimes[i].times.components(separatedBy: ":")
                let chosenHour = Int(chosenTimeArray[0])!
                if chosenHour*60 == tripTime{
                    order.orderTimeID = "\(currentDayTimes[i].id)"
                    order.orderDepartmentTime = "\(tripTime/60):00"
                    order.availablePlaces = currentDayTimes[i].mesta
                    break
                }
                else{
                    order.orderTimeID = "1"
                    order.orderDepartmentTime = "\(tripTime/60):00"
                    order.availablePlaces = 15
                }
```

```
            }
            break
        default:
            order.orderTimeID = "-999"
            order.orderDepartmentTime = "depTime"
            order.availablePlaces = 15
            break
        }
    }
    //MARK:- JSON parsing
    func getTimeJSON(postString:String){
        LoadJSON.loadJson(page: "", fromURLString: postString) { (result) in
            switch result {
            case .success(let data):
                print(data)
                self.timeSample = LoadJSON.parseJSON(jsonData: data,
type:TimeSample.self, controllerName: self) as! TimeSample
                DispatchQueue.main.async {
                    self.getHour()
                    self.tableView.reloadData()
                    self.removeSpinner()
                }
            case .failure(let error):
                print(error)
            }
        }
    }
    func getCreatedTimesJSON(postString:String,caseElement: Int){
        LoadJSON.loadJson(page: "", fromURLString: postString) { (result) in
            switch result {
            case .success(let data):
                print(data)
                switch caseElement {
                case 1:
                    self.currentDayTimes = LoadJSON.parseJSON(jsonData: data,
type:Time.self, controllerName: self) as! Time
                    break
                case 2:
                    self.previousDayTimes = LoadJSON.parseJSON(jsonData: data,
type:Time.self, controllerName: self) as! Time
                    break
                default:
                    break
                }
            case .failure(let error):
                print(error)
            }
        }
    }
    //MARK:- Functions for table view
    func getAdditionalTime(addtime:String)->String{
        let timeStringArray = addtime.components(separatedBy: ":")
        let minutes = timeStringArray[1]
        return minutes
    }
    func getHour(){
        for i in 0...timeSample.count-1 {
            let timeStringArray = timeSample[i].time.components(separatedBy: ":")
            timeHour.append(timeStringArray[0])
        }
    }
}
```

Файл RouteTableViewController.swift:

```
import UIKit

class RouteTableViewController: UITableViewController {

    var routes = [Route]()
    var order = OrderInformation.shared

    override func viewDidLoad() {
        super.viewDidLoad()
        addRoutes()
        tableView.tableFooterView = UIView()
    }
    func addRoutes(){
        routes.append(Route(code:"1",route:"CCґPjPë-РљРёС—PI"))
        routes.append(Route(code: "2",route: "РљРёС—PI-РЎСґPjPë"))
    }
    // MARK: - Table view data source

    override func numberOfSections(in tableView: UITableView) -> Int {
        return 1
    }

    override func tableView(_ tableView: UITableView, numberOfRowsInSection section:
Int) -> Int {
        return routes.count
    }

    override func tableView(_ tableView: UITableView, cellForRowAt indexPath:
IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier: "RouteCell", for:
indexPath)
        cell.textLabel?.text = routes[indexPath.row].route
        return cell
    }

    override func tableView(_ tableView: UITableView, didSelectRowAt indexPath:
IndexPath) {
        order.routeLabel = self.routes[indexPath.row].route
        order.orderRoute = self.routes[indexPath.row].code
        order.departureLabel = "P’PëP±CЂP°C,Pë..."
        order.destinationLabel = "P’PëP±CЂP°C,Pë..."
        order.timeLabel = "P’PëP±CЂP°C,Pë..."
        order.orderIdFrom = "idFrom"
        self.navigationController?.popViewController(animated: true)
    }
}
```

Файл Reachability.swift:

```
import SystemConfiguration
public class Reachability {
    //MARK:- Internet connection
    class func isConnectedToNetwork() -> Bool {

        var zeroAddress = sockaddr_in(sin_len: 0, sin_family: 0, sin_port: 0,
sin_addr: in_addr(s_addr: 0), sin_zero: (0, 0, 0, 0, 0, 0, 0, 0))
        zeroAddress.sin_len = UInt8(MemoryLayout.size(ofValue: zeroAddress))
        zeroAddress.sin_family = sa_family_t(AF_INET)

        let defaultRouteReachability = withUnsafePointer(to: &zeroAddress) {
            $0.withMemoryRebound(to: sockaddr.self, capacity: 1) {zeroSockAddress in
                SCNetworkReachabilityCreateWithAddress(nil, zeroSockAddress)
```

```
            }
        }

        var flags: SCNetworkReachabilityFlags = SCNetworkReachabilityFlags(rawValue:
0)
        if SCNetworkReachabilityGetFlags(defaultRouteReachability!, &flags) == false
{
            return false
        }

        let isReachable = (flags.rawValue & UInt32(kSCNetworkFlagsReachable)) != 0
        let needsConnection = (flags.rawValue &
UInt32(kSCNetworkFlagsConnectionRequired)) != 0
        let ret = (isReachable && !needsConnection)

        return ret
    }

}
```

Файл LoadJSON.swift:

```
import Foundation
import UIKit

class LoadJSON{
    static func loadJson(page:String,fromURLString urlString: String,
                         completion: @escaping (Result<Data, Error>) -> Void) {
        let urlB = URL(string: page)!
        var request = URLRequest(url: urlB)
        request.setValue("application/x-www-form-urlencoded", forHTTPHeaderField:
"Content-Type")
        request.httpMethod = "POST"
        request.httpBody = urlString.data(using: .utf8)
        let urlSession = URLSession.shared.dataTask(with: request) { (data, response,
error) in
            if let error = error {
                completion(.failure(error))
            }

            if let data = data {
                completion(.success(data))
            }
        }
        urlSession.resume()
    }
    static func parseJSON<T:Decodable>(jsonData:
Data,type:T.Type,controllerName:UIViewController)->Decodable {
        do {
            let str = String(decoding: jsonData, as: UTF8.self)
            print(str)
            let decodedData = try JSONDecoder().decode(type,
                                                       from: jsonData)
            return decodedData
        } catch {
            print("Parse Error")
        }
        return 1
    }
}
```

Файл Commonfunctions.swift:

```
import Foundation
import UIKit

class CommonFunctions: UIViewController{
    //MARK: - DELAY FUNCTION
    static func delay(_ delay:Double, closure:@escaping ()->()) {
        let when = DispatchTime.now() + delay
        DispatchQueue.main.asyncAfter(deadline: when, execute: closure)
    }


    //MARK:- CHANGE CONTROLLER
    static func changeViewController(controllerName:String){
        let appDelegate = UIApplication.shared.delegate! as! AppDelegate
        let storyboard = UIStoryboard(name: "Main", bundle: nil)
        let initialViewController =
storyboard.instantiateViewController(withIdentifier: controllerName)
        UIView.transition(with: appDelegate.window!, duration: 0.5, options:
UIView.AnimationOptions.transitionFlipFromLeft, animations: {
            appDelegate.window?.rootViewController = initialViewController
        }, completion: nil)

    }
    // MARK: - CHECK DATE FUNCTION
    static func checkCurrentDateWithChosen(deptime:String, date:String)->Bool{
        let now = Date()
        let calendar = NSCalendar.current
        let currentHour = calendar.component(.hour , from: now)
        let currentMinute = calendar.component(.minute, from: now)
        let timeStringArray = deptime.components(separatedBy: ":")
        let orderHours = Int(timeStringArray[0])
        let orderMinutes = Int(timeStringArray[1])
        let dateFormatter = DateFormatter()
        dateFormatter.dateFormat = "YYYY-MM-dd"
        let str = dateFormatter.string(from: now)
        print(str)
        let current = dateFormatter.date(from: str)
        let dateLater = dateFormatter.date(from: date)
        let resultYear = calendar.compare(current!, to: dateLater!, toGranularity:
.year)
        let resultMonth = calendar.compare(current!, to: dateLater!, toGranularity:
.month)
        let resultDay = calendar.compare(current!, to: dateLater!, toGranularity:
.day)
        if resultYear == .orderedSame && resultMonth == .orderedSame && resultDay ==
.orderedDescending{
            return false
        }
        if resultYear == .orderedSame && resultMonth == .orderedSame && resultDay ==
.orderedSame{
            if orderHours! > currentHour {return true}
            else if orderHours! == currentHour {
                if orderMinutes! > currentMinute {return true}
                else{return false}
            }
            else{return false}
        }
        return true
    }


}
```

Файл KeyboardFunctions.swift:

```
import Foundation
import UIKit

class KeyboardFunctions: UIViewController{

    static func format(with mask: String, phone: String) -> String {
        let numbers = phone.replacingOccurrences(of: "[^0-9]", with: "", options:
.regularExpression)
        var result = ""
        var index = numbers.startIndex // numbers iterator
        // iterate over the mask characters until the iterator of numbers ends
        for ch in mask where index < numbers.endIndex {
            if ch == "X" {
                // mask requires a number in this place, so take the next one
                result.append(numbers[index])

                // move numbers iterator to the next index
                index = numbers.index(after: index)

            } else {
                result.append(ch) // just append a mask character
            }
        }
        return result
    }
    static func checkInput(nameTextField:UITextField, phoneTextField: UITextField){

        let name = nameTextField.text
        let phone = phoneTextField.text

        UserDefaults.standard.set(name, forKey: "name")
        UserDefaults.standard.set(phone, forKey: "phone")
    }
    //Check number of chars in fields
    static func numberOfChars(nameTextField: UITextField, phoneTextField:
UITextField, controllerName: UIViewController)->Bool{
        var digitInput = true
        if nameTextField.text?.count == 0{
            Alerts.secondNameAlert(controllerName: controllerName)
            digitInput = false
        }
        else if phoneTextField.text?.count != 12{
            Alerts.phoneAlert(controllerName: controllerName)
            digitInput = false
        }
        return digitInput
    }
}
extension UIViewController{

    @objc func registerForKeyboardNotifications(){
        NotificationCenter.default.addObserver(self, selector: #selector(kbWillShow),
name: UIResponder.keyboardWillShowNotification, object: nil)
        NotificationCenter.default.addObserver(self, selector: #selector(kbWillHide),
name:UIResponder.keyboardWillHideNotification, object: nil)
    }
    func removeKeyboardNotifications(){
        NotificationCenter.default.removeObserver(self, name:
UIResponder.keyboardWillShowNotification, object: nil)
        NotificationCenter.default.removeObserver(self, name:
UIResponder.keyboardWillHideNotification, object: nil)
    }
```

```
    @objc func kbWillShow(notification: NSNotification) {
        if self.view.frame.origin.y == 0 {
            self.view.frame.origin.y -= self.view.frame.height*30/100
        }
    }
    @objc func kbWillHide(notification: NSNotification) {
        if self.view.frame.origin.y != 0 {
            self.view.frame.origin.y = 0
        }
    }
    @objc func textFieldShouldReturn(_ textField: UITextField) -> Bool {
        textField.resignFirstResponder()
        return false
    }
    func hideKeyboardWhenTappedAround() {
        let tap = UITapGestureRecognizer(target: self, action:
#selector(UIViewController.dismissKeyboard))
        tap.cancelsTouchesInView = false
        view.addGestureRecognizer(tap)
    }

    @objc func dismissKeyboard() {
        view.endEditing(true)
    }
}
```

ДОДАТОК Г

ЗАТВЕРДЖУЮ
Заступник директора компанії
«Еліт Експресс»
_____ Головань М. О.
«28» березня 2021 р.

Акт

Впровадження результатів дипломного проекту

студента Сумського державного університету

Акименка Владислава Валерійовича

Даним актом підтверджується, що результати роботи студента Акименка В. В. на тему «Мобільний додаток на базі операційної системи iOS для організації пасажирських перевезень компанією «Еліт Експресс»» впроваджено в роботу компанії «Еліт Експресс».

Даний мобільний додаток автоматизує важливі бізнес-процеси компанії з організації пасажирських перевезень та забезпечує зручне управління поїздками користувачами.

Впровадження мобільного додатку в роботу організації дозволило підвищити продуктивність виконання основних типів робіт, зменшити людські фактори похибок, підвищити якість діяльності компанії «Еліт Експресс». Розроблений додаток забезпечує швидке та зручне бронювання місць за вказаним маршрутом, надійність при здійсненні оплати, доступ до історії поїздок.

Заступник директора компанії
«Еліт Експресс» Головань М. О.