**Міністерство Освіти І НАУКИ України**

**Національний університет "Львівська політехніка"**

**КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

**КУРСОВА РОБОТА**

**з дисципліни:** *“Об’єктно-орієнтоване програмування ”*

**На тему:** *“Автомобільна дорога”*

Студента групи ПЗ-24

спеціальності 6.121

“Інженерія програмного забезпечення”

Чорняка Л.О.

Керівник: доцент кафедри ПЗ

к.ф.-м.н., доцент Коротєєва Т. О.

Національна шкала\_\_\_\_\_\_\_\_\_\_\_\_\_

Кількіть балів\_\_\_Оцінка ECTS\_\_\_

Члени комісії \_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Львів 2023

**Завдання**

на курсову роботу з дисципліни “Об’єктно-орієнтоване програмування”

студента групи ПЗ-*24 Чорняк Л.О.*

***Тема: “Автомобільна дорога”***

**Завдання:** (варіант 28)

Створити Дорога

 Автомобільна дорога |  Тип (державна/регіональна/обласна/місцева) |  Протяжність | Кількість смуг |  Наявність пішохідної доріжки |  Наявність розділювача посередині дороги

 1)      Впорядкувати дороги за протяжністю методом підрахунку.

2)      Знайти найкоротшу дорогу, де найбільша кількість смуг.

3)      Знайти всі дороги, в яких наявні розділювачі посередині, кількість смуг >2 та згрупувати за типом.

4)      Визначити типи автомобільних доріг, з найбільшою протяжністю та наявністю пішохідних доріжок.

5)      Визначити автомобільні дороги з найбільшою кількістю смуг та наявними пішохідними доріжками які належать до регіональних.

Для класу створити: 1) Конструктор за замовчуванням; 2) Конструктор з параметрами; 3) конструктор копій; 4) перевизначити операції >>, << для зчитування та запису у файл. Для демонстрації роботи програми використати засоби візуального середовища програмування.

**Зміст завдання та календарний план його виконання**

|  |  |  |
| --- | --- | --- |
| **№ з/п** | **Зміст завдання** | **Дата** |
| 1 | Здійснити аналiтичний огляд лiтератури за заданою темою та обґрунтувати вибір інструментальних засобів реалізації. | 15.10 |
| 2 | Побудова UML діаграм | 15.10 |
| 3 | Розробка алгоритмів реалізації | 19.10 |
| 4 | Реалізація завдання (кодування) | 25.10 |
| 5 | Формування інструкції користувача | 27.10 |
| 6 | Оформлення звіту до курсової роботи згідно з вимогами Міжнародних стандартів, дотримуючись такої структури:   * зміст; * алгоритм розв‘язку задачі у покроковому представленні; * діаграми UML [клас](http://vns.lpnu.ua/mod/glossary/showentry.php?eid=54317&displayformat=dictionary)ів, прецедентів, послідовності виконання; * код розробленої програми з коментарями; * протокол роботи програми для кожного пункту завдання * інструкція користувача та системні вимоги; * опис виняткових ситуацій; * структура файлу вхідних даних; * висновки; * список використаних джерел. | 28.10 |

Завдання прийнято до виконання: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Чорняк Л.О.)**\_**

( пiдпис студента )

Керівник роботи: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_( Коротєєва Т. О.)

Дата видачі завдання\_\_\_\_\_\_\_\_\_\_6.09.2023\_\_

**Зміст**

[**1.** **Алгоритм розв’язку задачі у покроковому представленні** 4](#_Toc118244368)

[**2.** **Діаграми UML класів, прецедентів, послідовності виконання** 8](#_Toc118244369)

[**3.** **Код розробленої програми з коментарями** 12](#_Toc118244370)

[**4.** **Протокол роботи програми для кожного пункту завдання** 37](#_Toc118244371)

[**5.** **Інструкція користувача та системні вимоги** 55](#_Toc118244372)

[**6.** **Опис виняткових ситуацій** 64](#_Toc118244373)

[**7.** **Структура файлів вхідних даних** 67](#_Toc118244374)

[**Висновки** 68](#_Toc118244375)

[**Список використаних джерел** 69](#_Toc118244376)

# **Алгоритм розв’язку задачі у покроковому представленні**

Для реалізції поставлених перед мною задач, я створив структуру ‘Road’, яка представляє сутність дороги і має такі поля як тип, довжина, кількість смуг, наявність пішохідної доріжки, наявінсть розділювача посередині дороги. Та класс ‘RoadManager’, який зберігає дороги у вигляді масиву та реалізовує заданий функціонал.

**Впорядкування доріг за протяжністю методом підрахунку**, SBL **(sortByLength)**

SBL: Цей алгоритм використовує алгоритм сортування підрахунком для отримання відсортованого масиву довжин, а потім відображає цей масив на вихідний масив доріг, роблячи його відсортованим за довжиною.

1. Використовуючи алгоритм сортування підрахунком , отримуємо відсортований масив довжин **roadsLength**.
2. Вихідний масив **roads** копіюється у змінну **roadsCopy**.
3. Створюється порожній масив **sortedRoads**, який буде містити відсортовані дороги за довжиною.
4. Для кожної довжини з відсортованого масиву **roadsLength** виконуються наступні дії:

SBL4.1. Знаходиться індекс першого входження цієї довжини в **roadsCopy** за допомогою бібліотечної функції **firstIndex**.

SBL4.2. Знайдений елемент додається до **sortedRoads**.

SBL4.3. Видаляється вже доданий елемент із **roadsCopy**.

1. Функція повертає відсортований масив доріг за довжиною.

**Знаходження найкоротшої дороги, де найбільша кількість смуг, SR (shortestRoad)**

SR: Основна ідея цього алгоритму - знайти дорогу з найбільшою кількістю смуг, а серед цих доріг знайти ту, яка має найменшу довжину. Функція повертає цю найкоротшу дорогу або nil, якщо вхідний масив порожній.

1. Використовуючи бібліотечну функцію **max(by:)**, знаходиться дорога з максимальною кількістю смуг. Значення максимальної кількості смуг зберігається в змінній **maxAmountOfLanes**.
2. Створюється новий масив roads, в якому лише ті дороги, які мають кількість смуг рівну **maxAmountOfLanes**.
3. За допомогою бібліотечної функції **min(by:)** знаходится дорога з мінімальною довжиною серед відфільтрованих доріг.
4. Повертається знайдена найкоротша дорога.

**Знаходження всіх доріг, в яких наявні розділювачі посередині, кількість смуг >2 та групування їх за типом, GA (GroupingAlgorithm)**

GA: Щоб реалізувати цей алгоритм необдхідно використати клас **UITableView**, щоб зобразити дані у вигляжі таблиці. Для цього потрібно реалізувати методи інтерфейсу **UITableViewDataSource**.

1. Метод ‘**numberOfSections’**: Цей ме визначає кількість секцій у таблиці. Кількість секцій визначається кількістю всіх типів доріг.
2. Метод ‘**tableView(\_:numberOfRowsInSection:)**’: цей метод визначає кількість клітинок у конкретній секції таблиці. Кількість клітинок визначається кількістю доріг, які відповідають конкретному типу і умовам.
3. Метод ‘**tableView(\_:viewForHeaderInSection:)**’: Цей метод повертає відомості про заголовок (header) для кожної секції. Визначається заголовок для кожної секції на основі значення section, конфігурується та повертається.
4. Метод ‘**tableView(\_:cellForRowAt:)**‘: Цей метод повертає відомості для кожноого рядка таблиці. Визначається масив доріг для відповідної секції використовуючи метод з класу ‘**RoadManager’** ‘**roadsWithType(\_ type: Int)**’ . Отримується конкретний об'єкт **Road** поточного рядка таблиці, і конфігурується вміст клітинки на основі об'єкта **Road**. Повертається сконфігурована клітинка.

**Опис методу ‘roadsWithType’, RWT**

RWT: Цей алгоритм фільтрує дороги за вказаним типом та умовами

1. Визначається тип дороги на основі значення параметра **type**. Якщо значення **type** в межах визначених кейсів (0, 1, 2, 3), встановлюється відповідний тип, в іншому випадку залишається значення за замовчуванням ‘**regional’**.
2. Створюється порожній масив roads, в який будуть додаватися дороги, які відповідають визначеному типу та визначеним умовам.
3. Для кожної дороги перевіряється, чи вона задовольняє умови: наявність роздільної смуги, кількість смуг більше, тип дороги співпадає з визначеним типом.
4. Якщо дорога відповідає умовам, додається до масиву roads.
5. Повертається масив **roads**, який містить лише ті дороги, які відповідають визначеному типу та умовам.

**Визначення типів автомобільних доріг, з найбільшою протяжністю та наявністю пішохідних доріжок, LRT (longestRoadTypes)**

1. Використовуючи функцію **max(by:)**, знаходиться дорога з найбільшою довжиною серед усіх доріг. Отримана максимальна довжина зберігається у змінній **maxLength**.
2. Створюється порожній масив **longestRoadTypes**, який буде містити типи доріг з найбільшою довжиною та наявністю пішохідних доріжок.
3. Для кожної дороги перевіряється, чи вона має максимальну довжину та наявність пішохідної доріжок. Якщо ці умови виконуються, тип дороги додається до масиву **longestRoadTypes**.
4. Повертається масив, який містить типи доріг з найбільшою довжиною та наявністю пішохідних доріжок.

**Визначення автомобільних доріг з найбільшою кількістю смуг та наявними пішохіжними доріжками які належать до регіональних, RAL (roadsWithTheBiggestAmountOfLanes)**

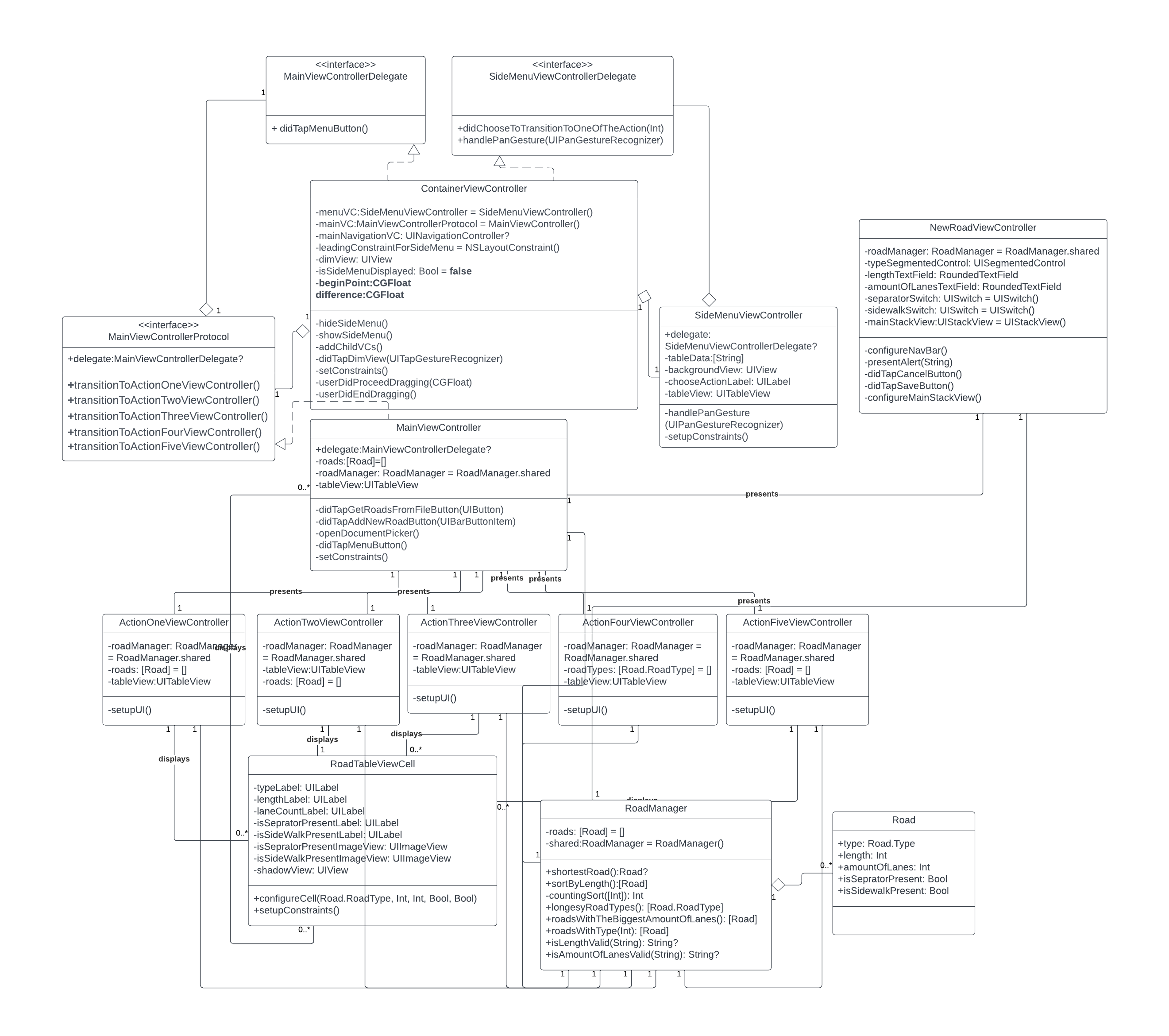
1. Використовуючи функцію **max(by:),** знаходиться дорога з найбільшою кількістю смуг. Значення максимальної кількості смуг руху зберігається у змінній **maxAmountOfLanes**.
2. Створюється новий масив **filteredRoads**.
3. Для кожної дороги перевіряється, чи її кількість пішохідних доріжок рівна **maxAmountOfLanes**, наявність пішохідної доріжки та її належність до регіональних. Якщо ці умови виконуються, дороги додається до масиву **filteredRoads**.
4. Повертається масив **filteredRoads**.

**Зчитування даних з файлу, FR (fileReading)**

1. Алгоритм для обробки вибору документів із файлової системи та записування інформації з цих документів у масив roads класу **RoadManager**, який використовується для відображення даних у таблиці.
2. Отримання URL-адреси вибраного файлу.
3. Читається вміст файлу. Якщо виникає помилка читання файлу, виводиться повідомлення про помилку.
4. Розділення вмісту файлу на окремі рядки і зберігання їх у масиві **lines**.
5. Кожен рядкок вмісту файлу розділяється на окремі компоненти, використовуючи пробіли як роздільники. Перевіряється, чи кількість компонентів у рядку дорівнює 5; якщо ні, викликається return, і обробка файлу завершується.
6. Отримуються значення для полів нового об'єкта Road з компонентів рядка. Створюється новий об'єкт Road з отриманими значеннями. Цей об'єкт додається до масиву **roads** класу **RoadManager**.
7. Оновлення таблиці, щоб відобразити нові дані.
8. Закриття екрана вибору файлу.

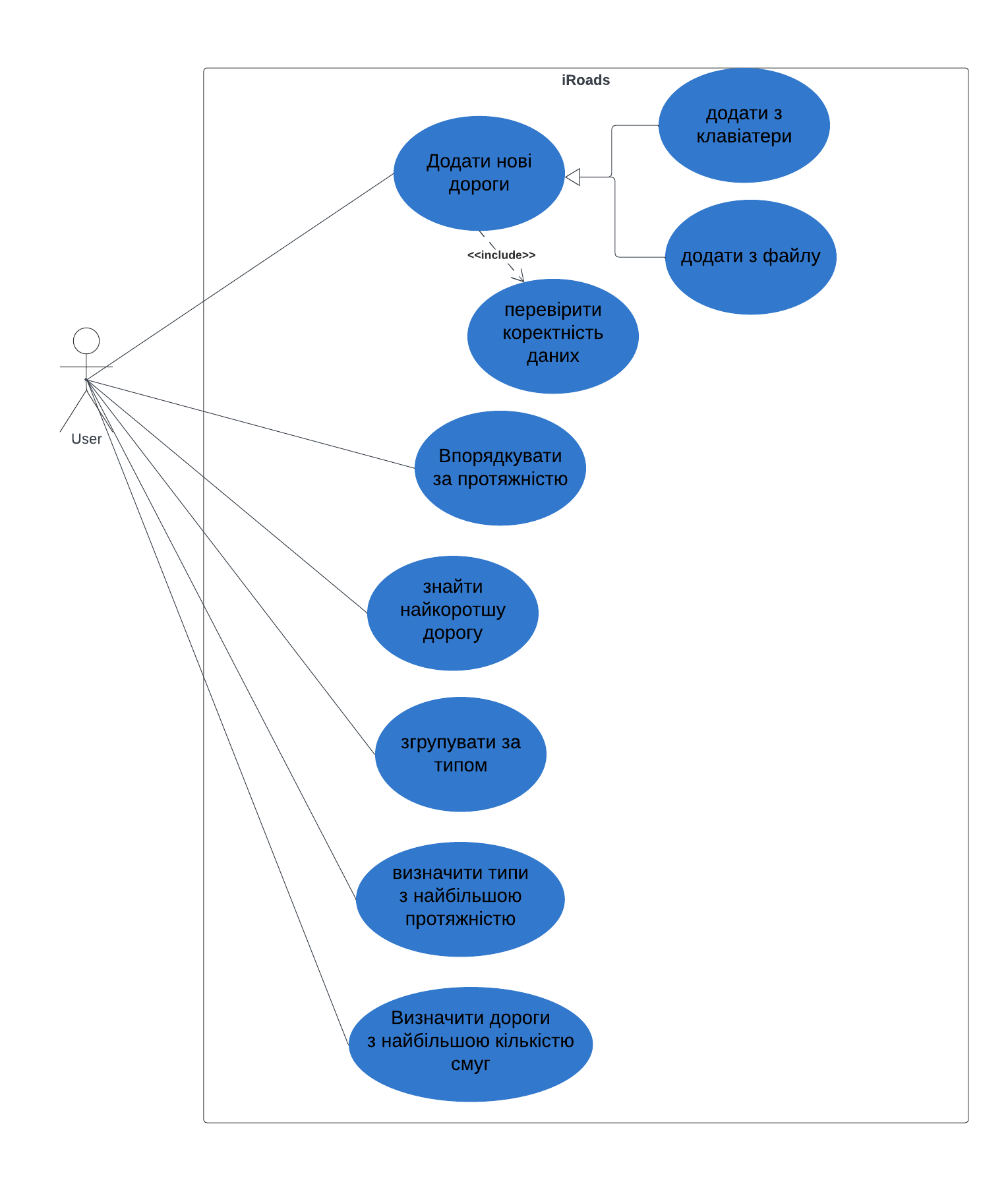
# **Діаграми UML класів, прецедентів, послідовності виконання**

1. Діаграма класів:



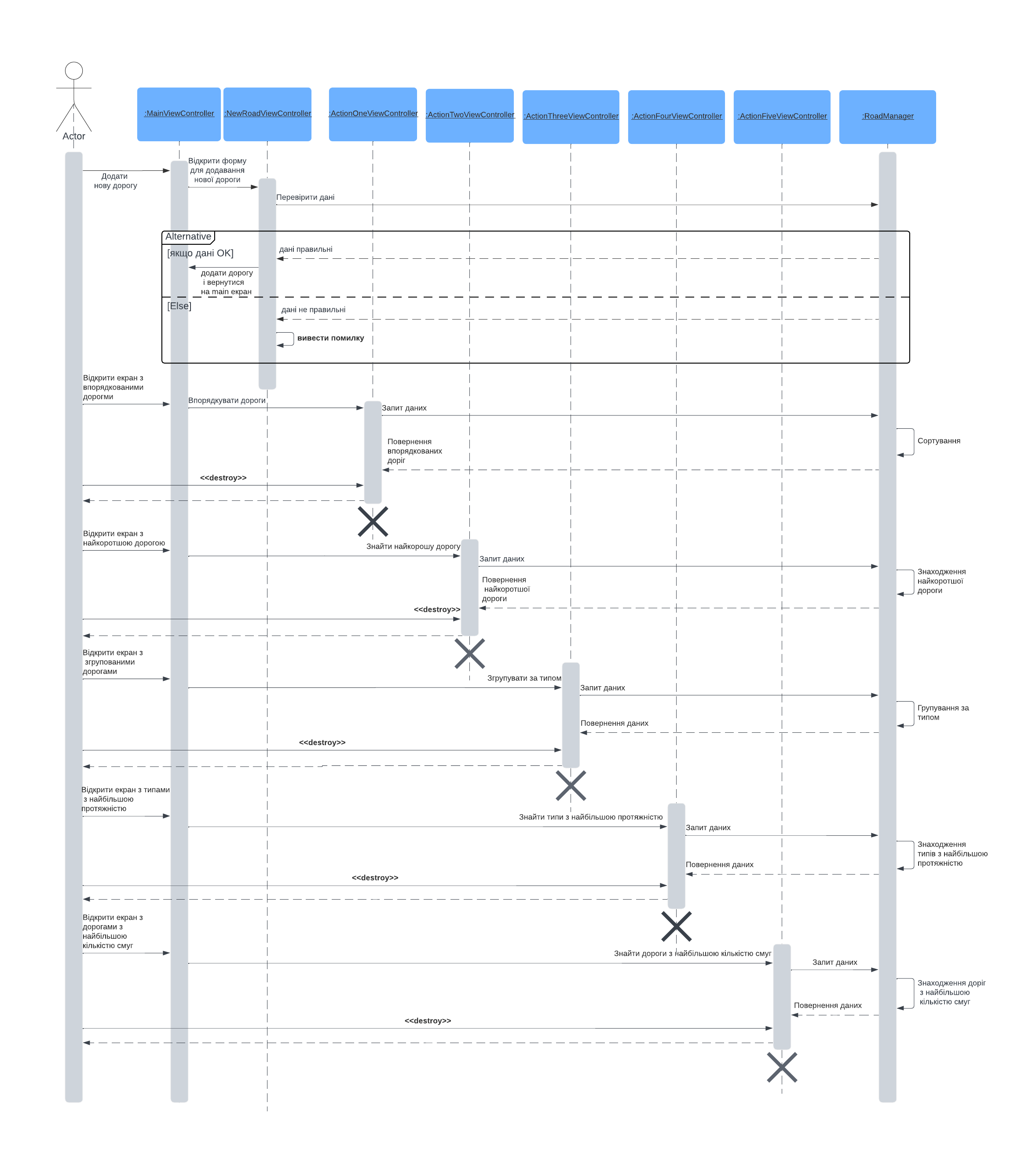
*Рис. 2.1. Діаграма класів*

1. Діаграма прецедентів:



*Рис. 2.2. Діаграма прецедентів*

1. Діаграми послідовності виконання:



*Рис. 2.3. Діаграма послідовності*

рис. 2.3. Діаграма UML послідовності виконання для звичайного користувача

# **Код розробленої програми з коментарями**

ContainerViewController.swift

**import** UIKit

**class** ContainerViewController: UIViewController {

**private** **let** menuVC = SideMenuViewController()

**private** **let** mainVC: MainViewControllerProtocol = MainViewController()

**private** **var** mainNavigationVC: UINavigationController?

**private** **var** leadingConstraintForSideMenu = NSLayoutConstraint()

**private** **enum** Constants {

**static** **let** sideMenuWidth: CGFloat = 225

}

**private** **let** dimView: UIView = {

**let** view = UIView()

view.layer.opacity = 0

view.backgroundColor = .black

view.translatesAutoresizingMaskIntoConstraints = **false**

view.isUserInteractionEnabled = **true**

**return** view

}()

**override** **func** viewDidLoad() {

**super**.viewDidLoad()

addChildVCs()

setConstraints()

}

**private** **var** isSideMenuDisplayed = **false**

**private** **func** hideSideMenu() {

UIView.animate(withDuration: 0.5, delay: 0, usingSpringWithDamping: 0.8, initialSpringVelocity: 0, options: .curveEaseInOut) {

**self**.leadingConstraintForSideMenu.constant = -Constants.sideMenuWidth

**self**.view.layoutIfNeeded()

**self**.dimView.layer.opacity = 0

} completion: { \_ **in**

**self**.isSideMenuDisplayed = **false**

}

}

**private** **func** showSideMenu() {

print("showSideMenu")

UIView.animate(withDuration: 0.5, delay: 0, usingSpringWithDamping: 0.8, initialSpringVelocity: 0, options: .curveEaseInOut) {

**self**.leadingConstraintForSideMenu.constant = 0

**self**.view.layoutIfNeeded()

**self**.dimView.layer.opacity = 0.2

} completion: { \_ **in**

**self**.isSideMenuDisplayed = **true**

}

}

**private** **func** addChildVCs() {

// Main

mainVC.delegate = **self**

**let** mainNavigationVC = UINavigationController(rootViewController: mainVC)

addChild(mainNavigationVC)

view.addSubview(mainNavigationVC.view)

mainNavigationVC.didMove(toParent: **self**)

**self**.mainNavigationVC = mainNavigationVC

view.addSubview(dimView)

**let** tapGestureRecognizer = UITapGestureRecognizer(target: **self**, action: **#selector**(didTapDimView))

tapGestureRecognizer.numberOfTapsRequired = 1

tapGestureRecognizer.numberOfTouchesRequired = 1

dimView.addGestureRecognizer(tapGestureRecognizer)

// Menu

addChild(menuVC)

menuVC.delegate = **self**

menuVC.view.backgroundColor = .white

view.addSubview(menuVC.view)

menuVC.didMove(toParent: **self**)

}

**@objc** **private** **func** didTapDimView(\_ gesture: UITapGestureRecognizer) {

print("dimViewTapped")

hideSideMenu()

}

**private** **func** setConstraints() {

**guard** **let** mainNavigationVC **else** { **return** }

mainNavigationVC.view.translatesAutoresizingMaskIntoConstraints = **false**

NSLayoutConstraint.activate([

mainNavigationVC.view.leadingAnchor.constraint(equalTo: view.leadingAnchor),

mainNavigationVC.view.trailingAnchor.constraint(equalTo: view.trailingAnchor),

mainNavigationVC.view.topAnchor.constraint(equalTo: view.topAnchor),

mainNavigationVC.view.bottomAnchor.constraint(equalTo: view.bottomAnchor)

])

NSLayoutConstraint.activate([

dimView.leadingAnchor.constraint(equalTo: view.leadingAnchor),

dimView.trailingAnchor.constraint(equalTo: view.trailingAnchor),

dimView.topAnchor.constraint(equalTo: view.topAnchor),

dimView.bottomAnchor.constraint(equalTo: view.bottomAnchor)

])

menuVC.view.translatesAutoresizingMaskIntoConstraints = **false**

NSLayoutConstraint.activate([

menuVC.view.topAnchor.constraint(equalTo: view.topAnchor),

menuVC.view.bottomAnchor.constraint(equalTo: view.bottomAnchor),

menuVC.view.leadingAnchor.constraint(equalTo: view.leadingAnchor),

menuVC.view.widthAnchor.constraint(equalToConstant: Constants.sideMenuWidth)

])

leadingConstraintForSideMenu = menuVC.view.leadingAnchor.constraint(equalTo: view.leadingAnchor, constant: -Constants.sideMenuWidth)

leadingConstraintForSideMenu.isActive = **true**

}

/// Початкова x координата початку drag жесту, щоб забрати side menu

**private** **var** beginPoint:CGFloat = 0.0

/// Відстань між початковою і поточною x координатами drag жесту

**private** **var** difference:CGFloat = 0.0

**override** **func** touchesBegan(\_ touches: Set<UITouch>, with event: UIEvent?) {

**if** **let** touch = touches.first {

**let** location = touch.location(in: view)

beginPoint = location.x

}

}

**override** **func** touchesMoved(\_ touches: Set<UITouch>, with event: UIEvent?) {

**if** **let** touch = touches.first {

**let** location = touch.location(in: view)

userDidProceedDragging(xLocation: location.x)

}

}

**override** **func** touchesEnded(\_ touches: Set<UITouch>, with event: UIEvent?) {

print("touchesEnded")

userDidEndDragging()

}

/// Функція викликається коли користувач продовжує тягнути side menu.

///

/// Спершу обчислюється відстань між початковою і поточною x координатами, і базуючись на цій відстані side mune суниться вліво

///

/// - **Parameters**:

/// - xLocation: x координата знаходження пальця користувача

**private** **func** userDidProceedDragging(xLocation: CGFloat) {

**guard** isSideMenuDisplayed **else** { **return** }

print("userDidProceedDragging")

**let** differenceFromBeginPoint = beginPoint - xLocation

**if** (differenceFromBeginPoint > 0 || differenceFromBeginPoint < 210) {

difference = differenceFromBeginPoint

**self**.leadingConstraintForSideMenu.constant = -differenceFromBeginPoint

**self**.dimView.alpha = 0.2 - (0.2 \* differenceFromBeginPoint / Constants.sideMenuWidth)

}

}

/// Ця функція викликається коли user завершує тягнути side menu. Якщо user потянув side menu достатньо далеко

/// воно ховається, інакше повертається назад у початкову позицію

**private** **func** userDidEndDragging() {

**if** (difference > 110) {

hideSideMenu()

} **else** {

**if** isSideMenuDisplayed {

showSideMenu()

}

}

}

}

**extension** ContainerViewController: SideMenuViewControllerDelegate {

**func** handlePanGesture(sender: UIPanGestureRecognizer) {

**switch** sender.state {

**case** .began:

**let** location = sender.location(in: view)

beginPoint = location.x

**case** .changed:

**let** location = sender.location(in: view)

userDidProceedDragging(xLocation: location.x)

**case** .ended:

userDidEndDragging()

**@unknown** **default**:

**break**

}

}

**func** didChooseToTransitionToOneOfTheAction(\_ action: Int) {

hideSideMenu()

**switch** action {

**case** 0: mainVC.transitionToActionOneViewController()

**case** 1: mainVC.transitionToActionTwoViewController()

**case** 2: mainVC.transitionToActionThreeViewController()

**case** 3: mainVC.transitionToActionFourViewController()

**case** 4: mainVC.transitionToActionFiveViewController()

**default**: **break**

}

}

}

**extension** ContainerViewController: MainViewControllerDelegate {

**func** didTapMenuButton() {

showSideMenu()

}

}

SideMenuViewControllerDelegate.swift

**import** UIKit

**protocol** SideMenuViewControllerDelegate: AnyObject {

**func** didChooseToTransitionToOneOfTheAction(\_ action: Int)

**func** handlePanGesture(sender: UIPanGestureRecognizer)

}

SideMenuViewController.swift

**import** UIKit

**class** SideMenuViewController: UIViewController {

**weak** **var** delegate: SideMenuViewControllerDelegate?

**private** **let** tableData = [

"Sort by length",

"Shortest road",

"Grouped by type",

"Longest road types",

"Roads With The Biggest Amount Of Lanes",

]

**private** **let** backgroundView: UIView = {

**let** view = UIView()

view.backgroundColor = colorLiteral(red: 0.2549019754, green: 0.2745098174, blue: 0.3019607961, alpha: 1)

view.translatesAutoresizingMaskIntoConstraints = **false**

view.layer.cornerRadius = 30

view.clipsToBounds = **true**

**return** view

}()

**private** **let** chooseActionLabel: UILabel = {

**let** label = UILabel()

label.numberOfLines = 2

label.text = "Choose Action"

label.font = .preferredFont(forTextStyle: .largeTitle)

label.textColor = .white

label.translatesAutoresizingMaskIntoConstraints = **false**

**return** label

}()

**private** **let** tableView: UITableView = {

**let** tableView = UITableView()

tableView.translatesAutoresizingMaskIntoConstraints = **false**

tableView.register(UITableViewCell.**self**, forCellReuseIdentifier: "cell")

tableView.isScrollEnabled = **false**

tableView.isUserInteractionEnabled = **true**

**return** tableView

}()

**override** **func** viewDidLoad() {

**super**.viewDidLoad()

tableView.dataSource = **self**

tableView.delegate = **self**

**let** panGesture = UIPanGestureRecognizer(target: **self**, action: **#selector**(handlePanGesture))

tableView.addGestureRecognizer(panGesture)

view.addSubview(backgroundView)

view.addSubview(chooseActionLabel)

view.addSubview(tableView)

setupConstraints()

}

**@objc** **private** **func** handlePanGesture(sender: UIPanGestureRecognizer) {

delegate?.handlePanGesture(sender: sender)

}

**private** **func** setupConstraints() {

NSLayoutConstraint.activate([

backgroundView.leadingAnchor.constraint(equalTo: view.leadingAnchor, constant: -30),

backgroundView.topAnchor.constraint(equalTo: view.topAnchor, constant: -30),

backgroundView.trailingAnchor.constraint(equalTo: view.trailingAnchor),

backgroundView.heightAnchor.constraint(equalToConstant: 200),

chooseActionLabel.leadingAnchor.constraint(equalTo: view.leadingAnchor, constant: 30),

chooseActionLabel.trailingAnchor.constraint(equalTo: view.trailingAnchor, constant: -30),

chooseActionLabel.topAnchor.constraint(equalTo: view.topAnchor, constant: 70),

tableView.topAnchor.constraint(equalTo: backgroundView.bottomAnchor, constant: 40),

tableView.leadingAnchor.constraint(equalTo: view.leadingAnchor),

tableView.trailingAnchor.constraint(equalTo: view.trailingAnchor),

tableView.heightAnchor.constraint(equalToConstant: 400)

])

}

}

**extension** SideMenuViewController: UITableViewDataSource, UITableViewDelegate {

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

tableData.count

}

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

**let** cell = tableView.dequeueReusableCell(withIdentifier: "cell", for: indexPath)

cell.textLabel?.text = tableData[indexPath.row]

cell.textLabel?.textColor = colorLiteral(red: 0.05882352963, green: 0.180392161, blue: 0.2470588237, alpha: 1)

cell.textLabel?.numberOfLines = 0

cell.textLabel?.font = .systemFont(ofSize: 25, weight: .bold)

**return** cell

}

**func** tableView(\_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {

print("selected")

tableView.deselectRow(at: indexPath, animated: **true**)

delegate?.didChooseToTransitionToOneOfTheAction(indexPath.row)

}

}

MainViewControllerProtocol.swift

**import** UIKit

**protocol** MainViewControllerProtocol: UIViewController {

**func** transitionToActionOneViewController()

**func** transitionToActionTwoViewController()

**func** transitionToActionThreeViewController()

**func** transitionToActionFourViewController()

**func** transitionToActionFiveViewController()

**var** delegate: MainViewControllerDelegate? { **get** **set** }

}

MainViewControllerDelegate.swift

**protocol** MainViewControllerDelegate: AnyObject {

**func** didTapMenuButton()

}

MainViewController.swift

**import** UIKit

**import** UniformTypeIdentifiers

**class** MainViewController: UIViewController {

**weak** **var** delegate: MainViewControllerDelegate?

**private** **var** roads: [Road] = []

**private** **let** roadManager = RoadManager.shared

**private** **let** tableView: UITableView = {

**let** tableView = UITableView()

tableView.translatesAutoresizingMaskIntoConstraints = **false**

tableView.register(RoadTableViewCell.**self**, forCellReuseIdentifier: "cell")

tableView.separatorStyle = .none

tableView.backgroundColor = .clear

**return** tableView

}()

**override** **func** viewDidLoad() {

**super**.viewDidLoad()

title = "Main"

view.backgroundColor = .systemBackground

navigationItem.leftBarButtonItem = UIBarButtonItem(image: UIImage(systemName: "list.dash"), style: .done, target: **self**, action: **#selector**(didTapMenuButton))

navigationController?.navigationBar.prefersLargeTitles = **true**

**let** addRoadButton = UIBarButtonItem(barButtonSystemItem: .add, target: **self**, action: **#selector**(didTapAddNewRoadButton))

**let** customButton = UIButton(configuration: .bordered())

customButton.addTarget(**self**, action: **#selector**(didTapGetRoadsFromFileButton), for: .touchUpInside)

customButton.setTitle("get roads from file", for: .normal)

**let** getRoadsFromFile = UIBarButtonItem(customView: customButton)

navigationItem.rightBarButtonItems = [addRoadButton, getRoadsFromFile]

tableView.dataSource = **self**

tableView.delegate = **self**

view.addSubview(tableView)

setConstraints()

}

**@objc** **private** **func** didTapGetRoadsFromFileButton(\_ sender: UIButton) {

openDocumentPicker()

}

**@objc** **private** **func** didTapAddNewRoadButton(\_ sender: UIBarButtonItem) {

**let** newRoadViewController = NewRoadViewController()

newRoadViewController.delegate = **self**

**let** destinationVC = UINavigationController(rootViewController: newRoadViewController)

present(destinationVC, animated: **true**)

}

**private** **func** openDocumentPicker() {

**let** supportedType: [UTType] = [.text]

**let** pickerViewController = UIDocumentPickerViewController(forOpeningContentTypes: supportedType, asCopy: **true**)

pickerViewController.allowsMultipleSelection = **false**

pickerViewController.shouldShowFileExtensions = **true**

pickerViewController.delegate = **self**

**self**.present(pickerViewController, animated: **true**, completion: **nil**)

}

**@objc** **private** **func** didTapMenuButton() {

print("didTapMenuButton")

delegate?.didTapMenuButton()

}

**private** **func** setConstraints() {

NSLayoutConstraint.activate([

tableView.leadingAnchor.constraint(equalTo: view.leadingAnchor),

tableView.trailingAnchor.constraint(equalTo: view.trailingAnchor),

tableView.bottomAnchor.constraint(equalTo: view.bottomAnchor),

tableView.topAnchor.constraint(equalTo: view.topAnchor)

])

}

}

**extension** MainViewController: UIDocumentPickerDelegate {

**func** documentPicker(\_ controller: UIDocumentPickerViewController, didPickDocumentsAt urls: [URL]) {

**guard** **let** fileURL = urls.first **else** {

print("No file selected")

**return**

}

**var** fileContent = String()

**do** {

fileContent = **try** String(contentsOf: fileURL)

} **catch** {

print("Error reading file: \(error.localizedDescription)")

}

**let** lines = fileContent.components(separatedBy: .newlines)

**for** line **in** lines {

**let** content = line.components(separatedBy: .whitespaces)

**guard** content.count == 5 **else** { **return** }

**guard** **let** type = Road.RoadType(rawValue: content[0]) **else** { **return** }

**let** length = Int(content[1]) ?? 0

**let** amountOfLanes = Int(content[2]) ?? 0

**guard** **let** isSeparatorPresent = Bool(content[3]) **else** { **return** }

**guard** **let** isSidewalkPresent = Bool(content[4]) **else** { **return** }

**let** road = Road(type: type, length: length, amountOfLanes: amountOfLanes, isSeparatorPresent: isSeparatorPresent, isSidewalkPresent: isSidewalkPresent)

roadManager.roads.append(road)

}

tableView.reloadData()

dismiss(animated: **true**)

}

}

**extension** MainViewController {

**private** **enum** Constants {

**static** **let** cellHeight: CGFloat = 290

}

}

**extension** MainViewController: UITableViewDelegate, UITableViewDataSource {

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

roadManager.roads.count

}

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

**let** road = roadManager.roads[indexPath.row]

**if** **let** cell = tableView.dequeueReusableCell(withIdentifier: "cell", for: indexPath) **as**? RoadTableViewCell {

cell.configureCell(

with: road.type,

length: road.length,

laneCount: road.amountOfLanes,

isSeparatorPresent: road.isSeparatorPresent,

isSideWalkPresent: road.isSidewalkPresent

)

**return** cell

}

**return** UITableViewCell()

}

**func** tableView(\_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {

tableView.deselectRow(at: indexPath, animated: **true**)

}

**func** tableView(\_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {

Constants.cellHeight

}

}

**extension** MainViewController: NewRoadViewControllerDelegate {

**func** didAddRoad() {

tableView.reloadData()

}

}

**extension** MainViewController: MainViewControllerProtocol {

**func** transitionToActionOneViewController() {

**let** destinationViewController = ActionOneViewController()

navigationController?.pushViewController(destinationViewController, animated: **true**)

}

**func** transitionToActionTwoViewController() {

**let** destinationViewController = ActionTwoViewController()

navigationController?.pushViewController(destinationViewController, animated: **true**)

}

**func** transitionToActionThreeViewController() {

**let** destinationViewController = ActionThreeViewController()

navigationController?.pushViewController(destinationViewController, animated: **true**)

}

**func** transitionToActionFourViewController() {

**let** destinationViewController = ActionFourViewController()

navigationController?.pushViewController(destinationViewController, animated: **true**)

}

**func** transitionToActionFiveViewController() {

**let** destinationViewController = ActionFiveViewController()

navigationController?.pushViewController(destinationViewController, animated: **true**)

}

}

ActionOneViewController.swift

**import** UIKit

**class** ActionOneViewController: UIViewController {

**private** **let** roadManager = RoadManager.shared

**private** **var** roads: [Road] = []

**private** **let** tableView: UITableView = {

**let** tableView = UITableView()

tableView.translatesAutoresizingMaskIntoConstraints = **false**

tableView.register(RoadTableViewCell.**self**, forCellReuseIdentifier: "cell")

tableView.separatorStyle = .none

tableView.backgroundColor = .clear

**return** tableView

}()

**override** **func** viewDidLoad() {

**super**.viewDidLoad()

title = "Sorted Roads"

view.backgroundColor = .systemBackground

roads = roadManager.sortByLength()

**self**.setupUI()

tableView.delegate = **self**

tableView.dataSource = **self**

}

**private** **func** setupUI() {

**self**.view.addSubview(tableView)

**self**.tableView.translatesAutoresizingMaskIntoConstraints = **false**

NSLayoutConstraint.activate([

**self**.tableView.topAnchor.constraint(equalTo: **self**.view.safeAreaLayoutGuide.topAnchor),

**self**.tableView.bottomAnchor.constraint(equalTo: **self**.view.safeAreaLayoutGuide.bottomAnchor),

**self**.tableView.leadingAnchor.constraint(equalTo: **self**.view.leadingAnchor),

**self**.tableView.trailingAnchor.constraint(equalTo: **self**.view.trailingAnchor),

])

}

}

**extension** ActionOneViewController: UITableViewDelegate, UITableViewDataSource {

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

roads.count

}

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

**let** road = roads[indexPath.row]

**if** **let** cell = tableView.dequeueReusableCell(withIdentifier: "cell", for: indexPath) **as**? RoadTableViewCell {

cell.configureCell(

with: road.type,

length: road.length,

laneCount: road.amountOfLanes,

isSeparatorPresent: road.isSeparatorPresent,

isSideWalkPresent: road.isSidewalkPresent

)

**return** cell

}

**return** UITableViewCell()

}

**func** tableView(\_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {

RoadTableViewCell.cellHeight

}

}

ActionTwoViewController.swift

**import** UIKit

**class** ActionTwoViewController: UIViewController {

**private** **let** roadManager = RoadManager.shared

**private** **var** roads: [Road] = []

**private** **let** tableView: UITableView = {

**let** tableView = UITableView()

tableView.translatesAutoresizingMaskIntoConstraints = **false**

tableView.register(RoadTableViewCell.**self**, forCellReuseIdentifier: "cell")

tableView.separatorStyle = .none

tableView.backgroundColor = .clear

**return** tableView

}()

**override** **func** viewDidLoad() {

**super**.viewDidLoad()

title = "Shortest Road"

view.backgroundColor = .systemBackground

roads = roadManager.sortByLength()

**self**.setupUI()

tableView.delegate = **self**

tableView.dataSource = **self**

}

**private** **func** setupUI() {

**self**.view.addSubview(tableView)

**self**.tableView.translatesAutoresizingMaskIntoConstraints = **false**

NSLayoutConstraint.activate([

**self**.tableView.topAnchor.constraint(equalTo: **self**.view.safeAreaLayoutGuide.topAnchor),

**self**.tableView.bottomAnchor.constraint(equalTo: **self**.view.safeAreaLayoutGuide.bottomAnchor),

**self**.tableView.leadingAnchor.constraint(equalTo: **self**.view.leadingAnchor),

**self**.tableView.trailingAnchor.constraint(equalTo: **self**.view.trailingAnchor),

])

}

}

**extension** ActionTwoViewController: UITableViewDelegate, UITableViewDataSource {

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

1

}

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

**if** **let** road = roadManager.shortestRoad() {

**if** **let** cell = tableView.dequeueReusableCell(withIdentifier: "cell", for: indexPath) **as**? RoadTableViewCell {

cell.configureCell(

with: road.type,

length: road.length,

laneCount: road.amountOfLanes,

isSeparatorPresent: road.isSeparatorPresent,

isSideWalkPresent: road.isSidewalkPresent

)

**return** cell

}

}

**return** UITableViewCell()

}

**func** tableView(\_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {

RoadTableViewCell.cellHeight

}

}

ActionThreeViewController.swift

**import** UIKit

**class** ActionThreeViewController: UIViewController {

**private** **let** roadManager = RoadManager.shared

**private** **let** tableView: UITableView = {

**let** tableView = UITableView()

tableView.backgroundColor = .systemBackground

tableView.separatorColor = .systemGray

tableView.register(CellHeader.**self**, forHeaderFooterViewReuseIdentifier: CellHeader.identifier)

tableView.register(RoadTableViewCell.**self**, forCellReuseIdentifier: "cell")

tableView.allowsSelection = **false**

**return** tableView

}()

**override** **func** viewDidLoad() {

**super**.viewDidLoad()

title = "Grouped"

view.backgroundColor = .systemBackground

**self**.setupUI()

//tableView.sectionHeaderTopPadding = 0

tableView.delegate = **self**

tableView.dataSource = **self**

}

**private** **func** setupUI() {

**self**.view.addSubview(tableView)

**self**.tableView.translatesAutoresizingMaskIntoConstraints = **false**

NSLayoutConstraint.activate([

**self**.tableView.topAnchor.constraint(equalTo: **self**.view.safeAreaLayoutGuide.topAnchor),

**self**.tableView.bottomAnchor.constraint(equalTo: **self**.view.safeAreaLayoutGuide.bottomAnchor),

**self**.tableView.leadingAnchor.constraint(equalTo: **self**.view.leadingAnchor),

**self**.tableView.trailingAnchor.constraint(equalTo: **self**.view.trailingAnchor),

])

}

}

**extension** ActionThreeViewController: UITableViewDelegate, UITableViewDataSource {

**func** numberOfSections(in tableView: UITableView) -> Int {

Road.RoadType.allCases.count

}

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

roadManager.roadsWithType(section).count

}

**func** tableView(\_ tableView: UITableView, viewForHeaderInSection section: Int) -> UIView? {

**guard** **let** header = tableView.dequeueReusableHeaderFooterView(withIdentifier: CellHeader.identifier) **as**? CellHeader **else** {

fatalError("Failed to create header cell.")

}

**var** headerTitle = ""

**switch** section {

**case** 0: headerTitle = "state"

**case** 1: headerTitle = "regional"

**case** 2: headerTitle = "oblasna"

**case** 3: headerTitle = "local"

**default**: **break**

}

header.configure(with: headerTitle)

**return** header

}

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

**let** roads = roadManager.roadsWithType(indexPath.section)

**let** road = roads[indexPath.row]

**if** **let** cell = tableView.dequeueReusableCell(withIdentifier: "cell", for: indexPath) **as**? RoadTableViewCell {

cell.configureCell(

with: road.type,

length: road.length,

laneCount: road.amountOfLanes,

isSeparatorPresent: road.isSeparatorPresent,

isSideWalkPresent: road.isSidewalkPresent

)

**return** cell

}

**return** UITableViewCell()

}

**func** tableView(\_ tableView: UITableView, heightForHeaderInSection section: Int) -> CGFloat {

// якщо немає жодної дороги з цим типом, тоді висота цього заголовку буде рівна 0, тобто

// заголовку не буде видно

**if** roadManager.roadsWithType(section).count == 0 { **return** 0}

**return** 44

}

**func** tableView(\_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {

RoadTableViewCell.cellHeight

}

}

CellHeader.swift

**import** UIKit

**class** CellHeader: UITableViewHeaderFooterView {

**static** **let** identifier = "CellHeader"

**private** **let** label: UILabel = {

**let** label = UILabel()

label.textColor = .label

label.textAlignment = .left

label.font = .systemFont(ofSize: 20, weight: .bold)

label.text = "Error"

**return** label

} ()

**public** **func** configure(with title: String) {

**self**.label.text = title

**self**.setupUI()

}

**private** **func** setupUI() {

**let** bView = UIView()

bView.backgroundColor = .systemGray

**self**.backgroundView = bView

**self**.addSubview(label)

label.translatesAutoresizingMaskIntoConstraints = **false**

NSLayoutConstraint.activate([

label.leadingAnchor.constraint(equalTo: **self**.layoutMarginsGuide.leadingAnchor),

label.trailingAnchor.constraint(equalTo: **self**.trailingAnchor),

label.centerYAnchor.constraint(equalTo: **self**.centerYAnchor)

])

}

}

ActionFourViewController.swift

**import** UIKit

**class** ActionFourViewController: UIViewController {

**private** **let** roadManager = RoadManager.shared

**private** **var** roadTypes: [Road.RoadType] = []

**private** **let** tableView: UITableView = {

**let** tableView = UITableView()

tableView.translatesAutoresizingMaskIntoConstraints = **false**

tableView.register(UITableViewCell.**self**, forCellReuseIdentifier: "cell")

tableView.separatorStyle = .none

tableView.backgroundColor = .clear

**return** tableView

}()

**override** **func** viewDidLoad() {

**super**.viewDidLoad()

title = "Longest Road Types"

view.backgroundColor = .systemBackground

roadTypes = roadManager.longestRoadTypes()

print("roadTypes: \(roadTypes)")

**self**.setupUI()

tableView.delegate = **self**

tableView.dataSource = **self**

}

**private** **func** setupUI() {

**self**.view.addSubview(tableView)

**self**.tableView.translatesAutoresizingMaskIntoConstraints = **false**

NSLayoutConstraint.activate([

**self**.tableView.topAnchor.constraint(equalTo: **self**.view.safeAreaLayoutGuide.topAnchor),

**self**.tableView.bottomAnchor.constraint(equalTo: **self**.view.safeAreaLayoutGuide.bottomAnchor),

**self**.tableView.leadingAnchor.constraint(equalTo: **self**.view.leadingAnchor),

**self**.tableView.trailingAnchor.constraint(equalTo: **self**.view.trailingAnchor),

])

}

}

**extension** ActionFourViewController: UITableViewDelegate, UITableViewDataSource {

**func** tableView(\_ tableView: UITableView, didDeselectRowAt indexPath: IndexPath) {

tableView.deselectRow(at: indexPath, animated: **true**)

}

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

roadTypes.count

}

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

**let** roadType = roadTypes[indexPath.row]

**let** cell = tableView.dequeueReusableCell(withIdentifier: "cell", for: indexPath)

**var** content = cell.defaultContentConfiguration()

content.text = roadType.rawValue

cell.contentConfiguration = content

cell.layoutIfNeeded()

**return** cell

}

**func** tableView(\_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {

44

}

}

ActionFiveViewController.swift

**import** UIKit

**class** ActionFiveViewController: UIViewController {

**private** **let** roadManager = RoadManager.shared

**private** **var** roads: [Road] = []

**private** **let** tableView: UITableView = {

**let** tableView = UITableView()

tableView.translatesAutoresizingMaskIntoConstraints = **false**

tableView.register(RoadTableViewCell.**self**, forCellReuseIdentifier: "cell")

tableView.separatorStyle = .none

tableView.backgroundColor = .clear

**return** tableView

}()

**override** **func** viewDidLoad() {

**super**.viewDidLoad()

title = "Biggest Amount Of Lanes"

view.backgroundColor = .systemBackground

roads = roadManager.roadsWithTheBiggestAmountOfLanes()

**self**.setupUI()

tableView.delegate = **self**

tableView.dataSource = **self**

}

**private** **func** setupUI() {

**self**.view.addSubview(tableView)

**self**.tableView.translatesAutoresizingMaskIntoConstraints = **false**

NSLayoutConstraint.activate([

**self**.tableView.topAnchor.constraint(equalTo: **self**.view.safeAreaLayoutGuide.topAnchor),

**self**.tableView.bottomAnchor.constraint(equalTo: **self**.view.safeAreaLayoutGuide.bottomAnchor),

**self**.tableView.leadingAnchor.constraint(equalTo: **self**.view.leadingAnchor),

**self**.tableView.trailingAnchor.constraint(equalTo: **self**.view.trailingAnchor),

])

}

}

**extension** ActionFiveViewController: UITableViewDelegate, UITableViewDataSource {

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

roads.count

}

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

**let** road = roads[indexPath.row]

**if** **let** cell = tableView.dequeueReusableCell(withIdentifier: "cell", for: indexPath) **as**? RoadTableViewCell {

cell.configureCell(

with: road.type,

length: road.length,

laneCount: road.amountOfLanes,

isSeparatorPresent: road.isSeparatorPresent,

isSideWalkPresent: road.isSidewalkPresent

)

**return** cell

}

**return** UITableViewCell()

}

**func** tableView(\_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {

RoadTableViewCell.cellHeight

}

}

NewRoadViewController.swift

**import** UIKit

**protocol** NewRoadViewControllerDelegate: AnyObject {

**func** didAddRoad()

}

**class** NewRoadViewController: UIViewController {

**weak** **var** delegate: NewRoadViewControllerDelegate?

**private** **let** roadManager = RoadManager.shared

**private** **let** typeSegmentedControl: UISegmentedControl = {

**let** items = ["state", "regional", "local", "oblasna"]

**let** segmentedControl = UISegmentedControl(items: items)

segmentedControl.selectedSegmentIndex = 0

**return** segmentedControl

}()

**private** **lazy** **var** lengthTextField: RoundedTextField = {

**let** textField = RoundedTextField()

textField.tag = 1

textField.delegate = **self**

textField.translatesAutoresizingMaskIntoConstraints = **false**

textField.heightAnchor.constraint(equalToConstant: 44).isActive = **true**

textField.placeholder = "length..."

**return** textField

}()

**private** **lazy** **var** amountOfLanesTextField: RoundedTextField = {

**let** textField = RoundedTextField()

textField.tag = 2

textField.delegate = **self**

textField.translatesAutoresizingMaskIntoConstraints = **false**

textField.heightAnchor.constraint(equalToConstant: 44).isActive = **true**

textField.placeholder = "amount of lanes..."

**return** textField

}()

**private** **let** separatorSwitch = UISwitch()

**private** **let** sidewalkSwitch = UISwitch()

**private** **let** mainStackView = UIStackView()

**override** **func** viewDidLoad() {

**super**.viewDidLoad()

configureMainStackView()

configureNavBar()

}

**private** **func** configureNavBar() {

title = "New Road"

**if** **let** appearance = navigationController?.navigationBar.standardAppearance {

**if** **let** customFont = UIFont(name: "Nunito-Bold", size: 40) {

appearance.titleTextAttributes = [.foregroundColor: UIColor(named: "NavigationBarTitle")!]

appearance.largeTitleTextAttributes = [.foregroundColor: UIColor(named: "NavigationBarTitle")!, .font: customFont]

}

**let** font = UIFont.systemFont(ofSize: 40)

appearance.titleTextAttributes = [.foregroundColor: colorLiteral(red: 0.9372549057, green: 0.3490196168, blue: 0.1921568662, alpha: 1)]

appearance.largeTitleTextAttributes = [.foregroundColor: colorLiteral(red: 0.9372549057, green: 0.3490196168, blue: 0.1921568662, alpha: 1), .font: font]

navigationController?.navigationBar.standardAppearance = appearance

navigationController?.navigationBar.compactAppearance = appearance

navigationController?.navigationBar.scrollEdgeAppearance = appearance

}

navigationController?.navigationBar.prefersLargeTitles = **true**

navigationController?.navigationBar.tintColor = colorLiteral(red: 0.9372549057, green: 0.3490196168, blue: 0.1921568662, alpha: 1)

navigationItem.leftBarButtonItem = UIBarButtonItem(barButtonSystemItem: .cancel, target: **self**, action: **#selector**(didTapCancelButton))

navigationItem.rightBarButtonItem = UIBarButtonItem(barButtonSystemItem: .save, target: **self**, action: **#selector**(didTapSaveButton))

}

**private** **func** presentAlert(text: String) {

**let** alert = UIAlertController(title: "Oops", message: text, preferredStyle: .alert)

**let** okAction = UIAlertAction(title: "OK", style: .default)

alert.addAction(okAction)

present(alert, animated: **true**)

}

**@objc** **private** **func** didTapCancelButton() {

dismiss(animated: **true**)

}

/// Функція викликається коли користувач натиснув кнопку save

///

/// Функція спочатку збирає дані з текстових полів. Потім викликає функції RoadManager щоб перевірити їхню коректність.

/// Якщо дані не коректні виводиться повідомлення про помилку і функція завершується. В іншому випадку збираються решта даних,

/// формується новий об'єкт Road, додається до RoadManager, викликається метод делегата про те, що дорога додана і екран забирається

**@objc** **private** **func** didTapSaveButton() {

**guard** **let** length = lengthTextField.text **else** { **return** }

**guard** **let** amountOfLanes = amountOfLanesTextField.text **else** { **return** }

**do** {

**try** roadManager.checkLength(length: length)

} **catch** {

presentAlert(text: error.localizedDescription)

**return**

}

**if** **let** errorMessage = roadManager.isAmountOfLanesValid(amountOfLanes: amountOfLanes) {

presentAlert(text: errorMessage)

} **else** {

**var** type: Road.RoadType = .regional

**switch** typeSegmentedControl.selectedSegmentIndex {

**case** 0: type = .state

**case** 1: type = .regional

**case** 2: type = .local

**case** 3: type = .oblasna

**default**: **break**

}

**let** road = Road(

type: type,

length: Int(length) ?? 0,

amountOfLanes: Int(amountOfLanes) ?? 0,

isSeparatorPresent: separatorSwitch.isOn,

isSidewalkPresent: sidewalkSwitch.isOn

)

roadManager.addRoad(road: road)

delegate?.didAddRoad()

dismiss(animated: **true**)

}

}

**private** **func** configureMainStackView() {

view.backgroundColor = .systemBackground

mainStackView.axis = .vertical

mainStackView.distribution = .equalSpacing

mainStackView.translatesAutoresizingMaskIntoConstraints = **false**

**let** typeLabel = UILabel()

typeLabel.text = "Type"

typeLabel.textColor = .black

**let** typeStackView = UIStackView()

typeStackView.spacing = 10

typeStackView.axis = .vertical

typeStackView.addArrangedSubview(typeLabel)

typeStackView.addArrangedSubview(typeSegmentedControl)

**let** lengthLabel = UILabel()

lengthLabel.text = "Length"

**let** lengthStackView = UIStackView()

lengthStackView.spacing = 10

lengthStackView.axis = .vertical

lengthStackView.addArrangedSubview(lengthLabel)

lengthStackView.addArrangedSubview(lengthTextField)

**let** amountOfLanesLabel = UILabel()

amountOfLanesLabel.text = "Amount of lanes"

**let** amountOfLanesStackView = UIStackView()

amountOfLanesStackView.spacing = 10

amountOfLanesStackView.axis = .vertical

amountOfLanesStackView.addArrangedSubview(amountOfLanesLabel)

amountOfLanesStackView.addArrangedSubview(amountOfLanesTextField)

**let** separatorLabel = UILabel()

separatorLabel.text = "Separator: "

**let** separatorStackView = UIStackView()

separatorStackView.axis = .horizontal

separatorStackView.addArrangedSubview(separatorLabel)

separatorStackView.addArrangedSubview(separatorSwitch)

**let** sidewalkLabel = UILabel()

sidewalkLabel.text = "Sidewalk: "

**let** sidewalkStackView = UIStackView()

sidewalkStackView.axis = .horizontal

sidewalkStackView.addArrangedSubview(sidewalkLabel)

sidewalkStackView.addArrangedSubview(sidewalkSwitch)

[typeStackView, lengthStackView, amountOfLanesStackView, separatorStackView, sidewalkStackView].forEach {mainStackView.addArrangedSubview($0)}

view.addSubview(mainStackView)

mainStackView.spacing = 30

NSLayoutConstraint.activate([

mainStackView.topAnchor.constraint(equalTo: view.safeAreaLayoutGuide.topAnchor, constant: 20),

mainStackView.leadingAnchor.constraint(equalTo: view.layoutMarginsGuide.leadingAnchor),

mainStackView.trailingAnchor.constraint(equalTo: view.layoutMarginsGuide.trailingAnchor),

mainStackView.heightAnchor.constraint(equalToConstant: 400)

])

}

}

**extension** NewRoadViewController: UITextFieldDelegate {

**func** textFieldShouldReturn(\_ textField: UITextField) -> Bool {

**if** **let** nextTextField = view.viewWithTag(textField.tag + 1) {

textField.resignFirstResponder()

nextTextField.becomeFirstResponder()

}

**return** **true**

}

}

RoadTableViewCell.swift

**import** UIKit

**class** RoadTableViewCell: UITableViewCell {

**private** **let** typeLabel: UILabel = {

**let** label = UILabel()

label.translatesAutoresizingMaskIntoConstraints = **false**

**return** label

}()

**private** **let** lengthLabel: UILabel = {

**let** label = UILabel()

label.textColor = .secondaryLabel

label.font = .systemFont(ofSize: 30, weight: .medium)

label.translatesAutoresizingMaskIntoConstraints = **false**

**return** label

}()

**private** **let** laneCountLabel: UILabel = {

**let** label = UILabel()

label.textColor = .secondaryLabel

label.font = .systemFont(ofSize: 30, weight: .medium)

label.translatesAutoresizingMaskIntoConstraints = **false**

**return** label

}()

**private** **let** isSeparatorPresentLabel: UILabel = {

**let** label = UILabel()

label.text = "Is separator present:"

label.textColor = .secondaryLabel

label.font = .systemFont(ofSize: 30, weight: .medium)

label.translatesAutoresizingMaskIntoConstraints = **false**

**return** label

}()

**private** **let** isSideWalkPresentLabel: UILabel = {

**let** label = UILabel()

label.text = "Is sidewalk present:"

label.textColor = .secondaryLabel

label.font = .systemFont(ofSize: 30, weight: .medium)

label.translatesAutoresizingMaskIntoConstraints = **false**

**return** label

}()

**private** **let** isSeparatorPresentImageView: UIImageView = {

**let** imageView = UIImageView()

**let** config = UIImage.SymbolConfiguration(font: .boldSystemFont(ofSize: 20))

imageView.preferredSymbolConfiguration = config

imageView.translatesAutoresizingMaskIntoConstraints = **false**

**return** imageView

}()

**private** **let** isSideWalkPresentImageView: UIImageView = {

**let** imageView = UIImageView()

**let** config = UIImage.SymbolConfiguration(font: .boldSystemFont(ofSize: 20))

imageView.preferredSymbolConfiguration = config

imageView.translatesAutoresizingMaskIntoConstraints = **false**

**return** imageView

}()

**private** **enum** UIConstants {

**static** **let** symbolSize: CGFloat = 30

**static** **let** labelHeight: CGFloat = 32

}

**static** **let** cellHeight: CGFloat = 290

**override** **init**(style: UITableViewCell.CellStyle, reuseIdentifier: String?) {

**super**.init(style: style, reuseIdentifier: reuseIdentifier)

**self**.selectionStyle = .none

contentView.addSubview(shadowView)

contentView.addSubview(typeLabel)

contentView.addSubview(lengthLabel)

contentView.addSubview(laneCountLabel)

contentView.addSubview(isSeparatorPresentLabel)

contentView.addSubview(isSideWalkPresentLabel)

contentView.addSubview(isSeparatorPresentImageView)

contentView.addSubview(isSideWalkPresentImageView)

setupConstraints()

}

**private** **let** shadowView: UIView = {

**let** view = UIView()

view.translatesAutoresizingMaskIntoConstraints = **false**

view.layer.shadowRadius = 5

view.layer.shadowColor = UIColor.black.cgColor

view.layer.shadowOffset = CGSize(width: 5, height: 5)

view.layer.shadowOpacity = 0.5

view.layer.masksToBounds = **false**

view.layer.cornerRadius = 20

view.backgroundColor = UIColor.gray.withAlphaComponent(0.25)

**return** view

}()

**override** **func** layoutSubviews() {

**super**.layoutSubviews()

**self**.backgroundColor = .clear

contentView.frame = contentView.frame.inset(by: UIEdgeInsets(top: 10, left: 17, bottom: 10, right: 17))

contentView.layer.cornerRadius = 20

contentView.layer.masksToBounds = **false**

}

**func** configureCell(with type: Road.RoadType, length: Int, laneCount: Int, isSeparatorPresent: Bool, isSideWalkPresent: Bool) {

**let** font: [NSAttributedString.Key: **Any**] = [.font: UIFont.systemFont(ofSize: 30, weight: .medium)]

**let** color1: [NSAttributedString.Key: **Any**] = [.foregroundColor : UIColor.secondaryLabel]

**let** color2: [NSAttributedString.Key: **Any**] = [.foregroundColor: UIColor.label]

**let** typeString = "Type: \(type.rawValue)"

**let** typeAttributedString = NSMutableAttributedString(string: typeString)

typeAttributedString.addAttributes(font, range: NSRange(location: 0, length: typeString.count ))

typeAttributedString.addAttributes(color1, range: NSRange(location: 0, length: "Type: ".count))

typeAttributedString.addAttributes(color2, range: NSRange(location: "Type: ".count, length: type.rawValue.count))

typeLabel.attributedText = typeAttributedString

**let** lengthString = "Length: \(length)"

**let** lengthAttributedString = NSMutableAttributedString(string: lengthString)

lengthAttributedString.addAttributes(font, range: NSRange(location: 0, length: lengthString.count))

lengthAttributedString.addAttributes(color1, range: NSRange(location: 0, length: "Type: ".count))

lengthAttributedString.addAttributes(color2, range: NSRange(location: "Length: ".count, length: "\(length)".count))

lengthLabel.attributedText = lengthAttributedString

**let** laneCountString = "Amount of lanes: \(laneCount)"

**let** laneCountAttributedString = NSMutableAttributedString(string: laneCountString)

laneCountAttributedString.addAttributes(font, range: NSRange(location: 0, length: laneCountString.count))

laneCountAttributedString.addAttributes(color1, range: NSRange(location: 0, length: "Amount of lanes: ".count))

laneCountAttributedString.addAttributes(color2, range: NSRange(location: "Amount of lanes: ".count, length: "\(laneCount)".count))

laneCountLabel.attributedText = laneCountAttributedString

**let** checkmarkImage = UIImage(systemName: "checkmark")

checkmarkImage?.applyingSymbolConfiguration(.init(font: .boldSystemFont(ofSize: 40)))

**let** xmarkImage = UIImage(systemName: "xmark")

xmarkImage?.applyingSymbolConfiguration(.init(font: .boldSystemFont(ofSize: 40)))

isSeparatorPresentImageView.image = isSeparatorPresent ? checkmarkImage : xmarkImage

isSeparatorPresentImageView.tintColor = isSeparatorPresent ? .yellow : .red

isSideWalkPresentImageView.image = isSideWalkPresent ? checkmarkImage : xmarkImage

isSideWalkPresentImageView.tintColor = isSideWalkPresent ? .yellow : .red

}

**private** **func** setupConstraints() {

NSLayoutConstraint.activate([

shadowView.leadingAnchor.constraint(equalTo: contentView.leadingAnchor),

shadowView.trailingAnchor.constraint(equalTo: contentView.trailingAnchor),

shadowView.topAnchor.constraint(equalTo: contentView.topAnchor),

shadowView.bottomAnchor.constraint(equalTo: contentView.bottomAnchor),

typeLabel.topAnchor.constraint(equalTo: contentView.topAnchor, constant: 20),

typeLabel.leadingAnchor.constraint(equalTo: contentView.leadingAnchor, constant: 20),

typeLabel.heightAnchor.constraint(equalToConstant: UIConstants.labelHeight),

lengthLabel.topAnchor.constraint(equalTo: typeLabel.bottomAnchor, constant: 20),

lengthLabel.leadingAnchor.constraint(equalTo: contentView.leadingAnchor, constant: 20),

lengthLabel.heightAnchor.constraint(equalToConstant: UIConstants.labelHeight),

laneCountLabel.topAnchor.constraint(equalTo: lengthLabel.bottomAnchor, constant: 20),

laneCountLabel.leadingAnchor.constraint(equalTo: contentView.leadingAnchor, constant: 20),

laneCountLabel.heightAnchor.constraint(equalToConstant: UIConstants.labelHeight),

isSeparatorPresentLabel.topAnchor.constraint(equalTo: laneCountLabel.bottomAnchor, constant: 20),

isSeparatorPresentLabel.leadingAnchor.constraint(equalTo: contentView.leadingAnchor, constant: 20),

isSeparatorPresentLabel.heightAnchor.constraint(equalToConstant: UIConstants.labelHeight),

isSideWalkPresentLabel.topAnchor.constraint(equalTo: isSeparatorPresentLabel.bottomAnchor, constant: 20),

isSideWalkPresentLabel.leadingAnchor.constraint(equalTo: contentView.leadingAnchor, constant: 20),

isSideWalkPresentLabel.bottomAnchor.constraint(equalTo: contentView.bottomAnchor, constant: -20),

isSideWalkPresentLabel.heightAnchor.constraint(equalToConstant: UIConstants.labelHeight),

isSeparatorPresentImageView.centerYAnchor.constraint(equalTo: isSeparatorPresentLabel.centerYAnchor),

isSeparatorPresentImageView.leadingAnchor.constraint(equalTo: isSeparatorPresentLabel.trailingAnchor, constant: 10),

isSeparatorPresentImageView.heightAnchor.constraint(equalToConstant: UIConstants.symbolSize),

isSeparatorPresentImageView.widthAnchor.constraint(equalToConstant: UIConstants.symbolSize),

isSideWalkPresentImageView.centerYAnchor.constraint(equalTo: isSideWalkPresentLabel.centerYAnchor),

isSideWalkPresentImageView.leadingAnchor.constraint(equalTo: isSeparatorPresentImageView.leadingAnchor),

isSideWalkPresentImageView.heightAnchor.constraint(equalToConstant: UIConstants.symbolSize),

isSideWalkPresentImageView.widthAnchor.constraint(equalToConstant: UIConstants.symbolSize)

])

}

**required** **init**?(coder: NSCoder) {

fatalError("init(coder:) has not been implemented")

}

}

RoadManager.swift

**import** Foundation

**class** RoadManager {

**public** **var** roads: [Road] = []

**static** **let** shared = RoadManager()

**private** **init**() {}

/// Знаходить найкоротшу дорогу, де найбільша кількість смуг

/// - **Returns**: дорогу яка віповідає критеріям або nil, якщо масив доріг пустий

**public** **func** shortestRoad() -> Road? {

**guard** **let** maxAmountOfLanes = roads.max(by: { $0.amountOfLanes < $1.amountOfLanes })?.amountOfLanes **else** { **return** **nil** }

**let** roads = roads.filter { $0.amountOfLanes == maxAmountOfLanes }

**let** road = roads.min { $0.length < $1.length }

**return** road

}

/// Впорядковує дороги за протяжністю методом підрахунку.

/// - **Returns**: відсортовані дороги

**public** **func** sortByLength() -> [Road] {

**let** roadsLength = countingSort( roads.map {$0.length} )

**var** roadsCopy = roads

**var** sortedRoads: [Road] = []

**for** length **in** roadsLength {

**guard** **let** index = roadsCopy.firstIndex(where: {$0.length == length}) **else** { **return** []}

sortedRoads.append(roadsCopy[index])

roadsCopy.remove(at: index)

}

**return** sortedRoads

}

**private** **func** countingSort(\_ array: [Int])-> [Int] {

**guard** array.count > 0 **else** {**return** []}

**let** maxElement = array.max() ?? 0

**var** countArray = [Int](repeating: 0, count: Int(maxElement + 1))

**for** element **in** array {

countArray[element] += 1

}

**for** index **in** 1 ..< countArray.count {

**let** sum = countArray[index] + countArray[index - 1]

countArray[index] = sum

}

**var** sortedArray = [Int](repeating: 0, count: array.count)

**for** index **in** stride(from: array.count - 1, through: 0, by: -1) {

**let** element = array[index]

countArray[element] -= 1

sortedArray[countArray[element]] = element

}

**return** sortedArray

}

/// Визначає типи автомобільних доріг, з найбільшою протяжністю та наявністю пішохідних доріжок.

/// - **Returns**: типи автомобільних доріг, які відповідають критеріям

**public** **func** longestRoadTypes() -> [Road.RoadType] {

**guard** **let** maxLength = roads.max(by: { $0.length < $1.length })?.length **else** { **return** []}

**var** longestRoadTypes = Set<Road.RoadType>()

roads.forEach { road **in**

**if** road.length == maxLength && road.isSidewalkPresent == **true** {

longestRoadTypes.insert(road.type)

}

}

**return** longestRoadTypes.map {$0}

}

/// Визначає автомобільні дороги з найбільшою кількістю смуг та наявними пішохідними доріжками які належать до регіональних.

/// - **Returns**: дороги, які відповідають критеріям

**public** **func** roadsWithTheBiggestAmountOfLanes() -> [Road] {

**guard** **let** maxAmountOfLanes = roads.max(by: { $0.amountOfLanes < $1.amountOfLanes })?.amountOfLanes **else** { **return** [] }

**let** filteredRoads = roads.filter { road **in**

**if** road.amountOfLanes == maxAmountOfLanes && road.isSidewalkPresent == **true** && road.type == .regional {

**return** **true**

} **else** {

**return** **false**

}

}

**return** filteredRoads

}

/\*\*

Знаходить всі дороги, які належать до заданого типу, наявні розділювачі посередині, кількість смуг >2.

- **Parameters**:

- type: Ціле число, яке представляє тип дороги.

- **Returns**: дороги, які відповідають критеріям

\*/

**public** **func** roadsWithType(\_ type: Int) -> [Road] {

**var** roadType: Road.RoadType = .regional

**switch** type {

**case** 0: roadType = .state

**case** 1: roadType = .regional

**case** 2: roadType = .oblasna

**case** 3: roadType = .local

**default**: **break**

}

**var** roads: [Road] = []

**for** road **in** **self**.roads {

**if** road.isSeparatorPresent == **true** && road.amountOfLanes > 2 {

**if** road.type == roadType {

roads.append(road)

}

}

}

**return** roads

}

**public** **func** addRoad(road: Road) {

roads.append(road)

}

/\*\*

Перевіряє коректність довжини

- **Parameters**:

- length: довжина

- **Returns**: опис помилки, або nil якщо довжина коректна

\*/

**public** **func** checkLength(length: String) **throws** {

**if** length.isEmpty {

**throw** RoadError.emptyLength(description: "you didn't enter length")

}

**guard** **let** length = Int(length) **else** {

**throw** RoadError.incorrectLength(description: "you entered incorrect length")

}

**if** length < 0 {

**throw** RoadError.negativeLength(description: "you entered negative length")

}

}

/\*\*

Перевіряє коректність кількості смуг

- **Parameters**:

- amountOfLanes: кількості смуг

- **Returns**: опис помилки, або nil якщо кількості смуг коректна

\*/

**public** **func** isAmountOfLanesValid(amountOfLanes: String) -> String? {

**var** errorMessage: String? = **nil**

**if** amountOfLanes.isEmpty {

errorMessage = "you didn't enter amount of lanes"

} **else** **if** Int(amountOfLanes) == **nil** {

errorMessage = "you entered incorrect amount of lanes"

}

**return** errorMessage

}

}

RoadModel.swift

**import** Foundation

**struct** Road {

**enum** RoadType: String, CaseIterable {

**case** state

**case** regional

**case** oblasna

**case** local

}

**init**(type: RoadType, length: Int, amountOfLanes: Int, isSeparatorPresent: Bool, isSidewalkPresent: Bool) {

**self**.type = type

**self**.length = length

**self**.amountOfLanes = amountOfLanes

**self**.isSeparatorPresent = isSeparatorPresent

**self**.isSidewalkPresent = isSidewalkPresent

}

**init**() {

type = .regional

length = 0

amountOfLanes = 0

isSeparatorPresent = **false**

isSidewalkPresent = **false**

}

**public** **var** type: RoadType

**public** **var** length: Int

**public** **var** amountOfLanes: Int

**public** **var** isSeparatorPresent: Bool

**public** **var** isSidewalkPresent: Bool

}

RoadError.swift

**import** Foundation

**enum** RoadError: LocalizedError {

**case** emptyLength(description: String)

**case** negativeLength(description: String)

**case** incorrectLength(description: String)

**var** errorDescription: String? {

**switch** **self** {

**case** .emptyLength(**let** description): **return** description

**case** .negativeLength(**let** description): **return** description

**case** .incorrectLength(**let** description): **return** description

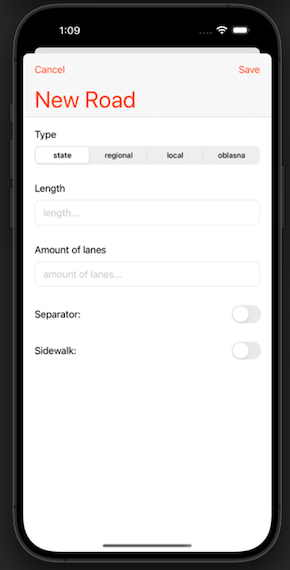
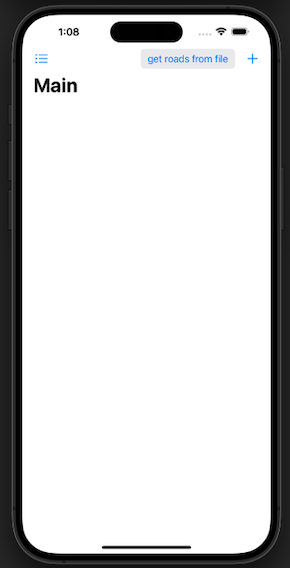
}

}

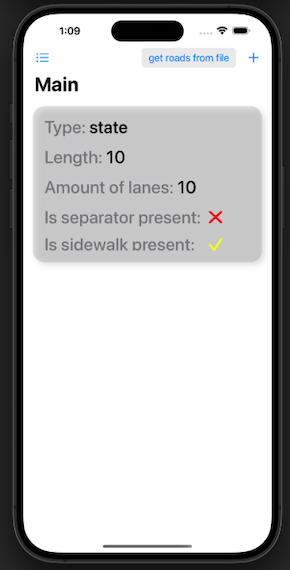
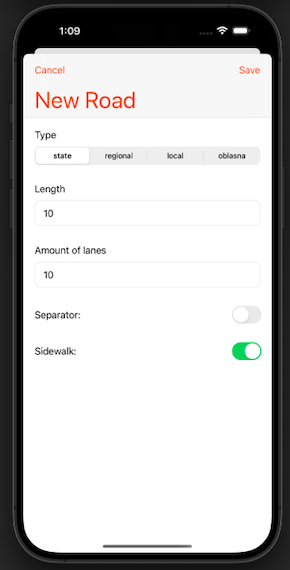
}

**4. Протокол роботи програми для кожного пункту**

1. Ввід даних вручну

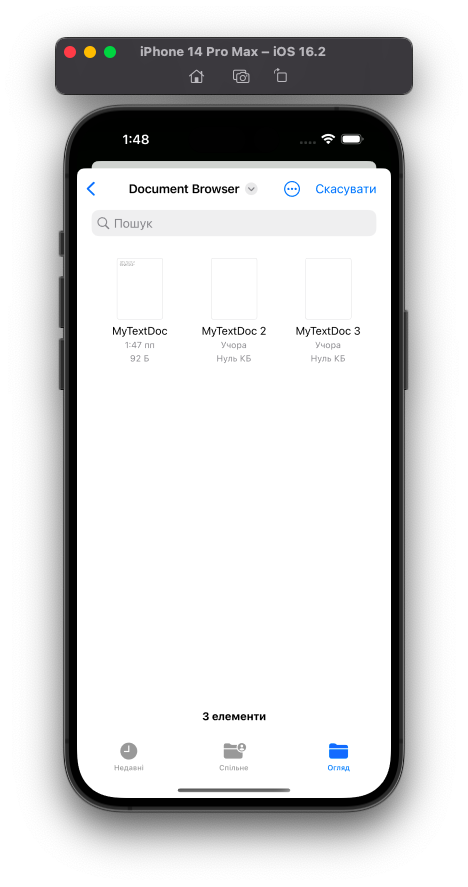


*Рис. 4.1. Початковий екран Рис. 4.2 Форма для введення даних*

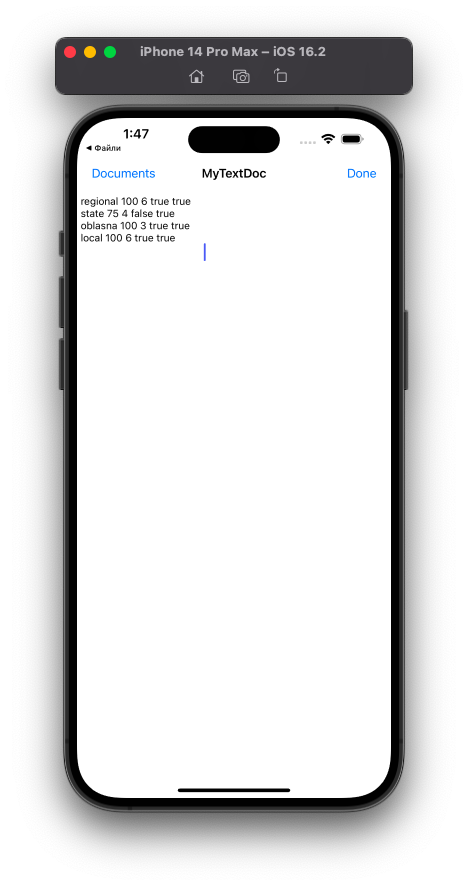


*Рис. 4.3 Введення даних Рис. 4.4 Відображення введених даних*

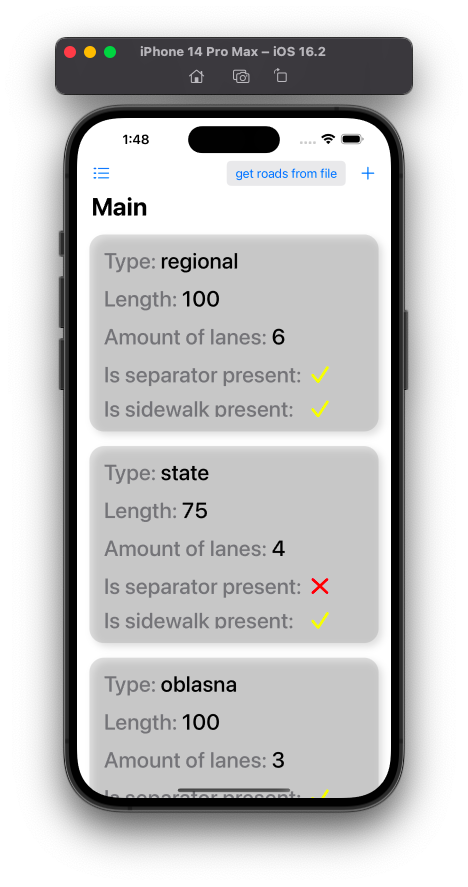
2. Ввід даних з файлу



*Рис. 4.5 Форма для вибору файлу*

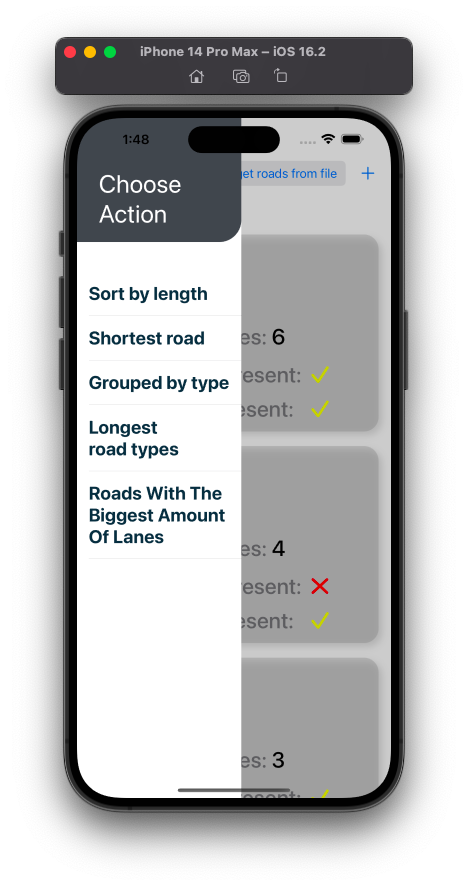


*Рис. 4.6 вибраний файл*

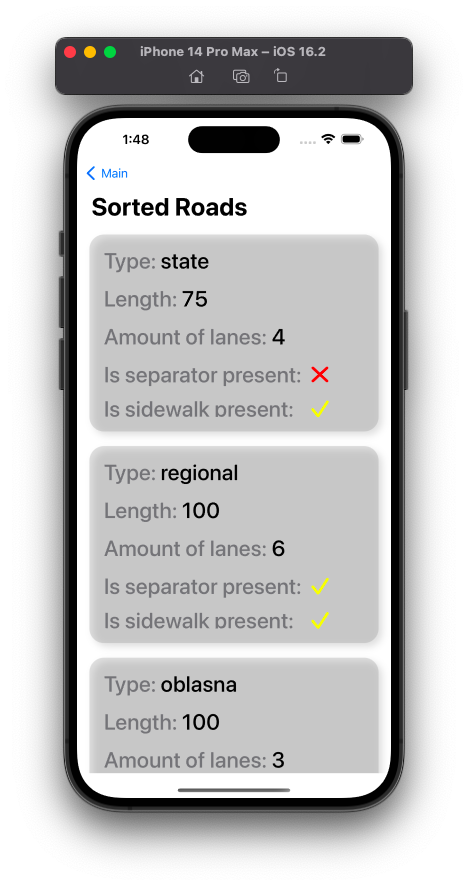


*Рис. 4.7 відображення вмісту файла*

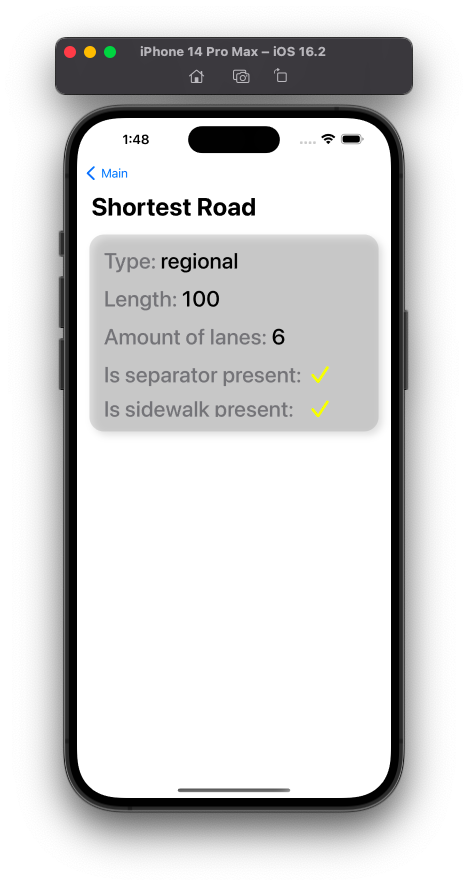
3. Виконання функціоналу поставленого у завданні



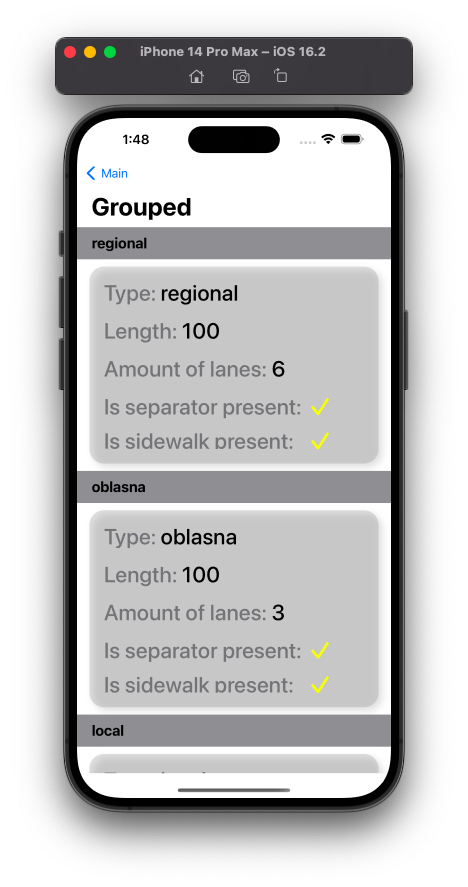
*Рис. 4.7. Відкриття side menu та вибір конкретної дії*

****

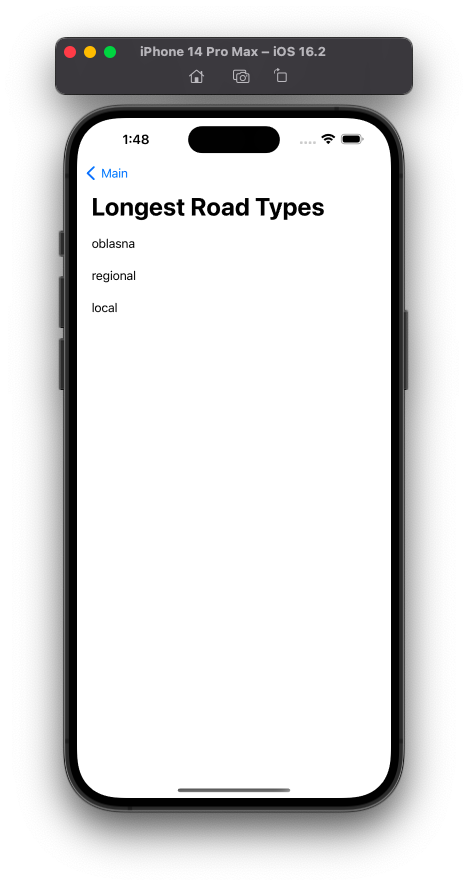
*Рис. 4.8. Впорядкування доріг за протяжністю методом підрахунку*



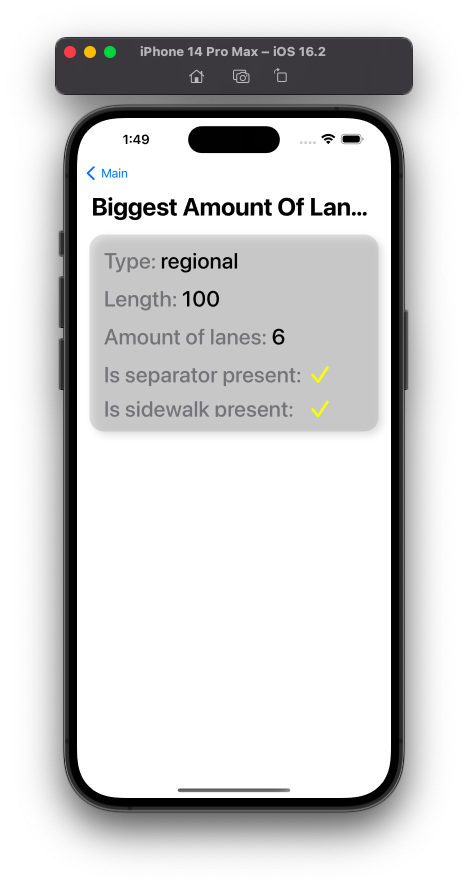
*Рис 4.9. Знаходження найкоротшої дороги, де найбільша кількість смуг*



*Рис 4.10 Знаходження всіх доріг, в яких наявні розділювачі посередині, кількість смуг >2 та згрупувати за типом.*



*Рис 4.11. Визначення типів автомобільних доріг, з найбільшою протяжністю та наявністю пішохідних доріжок.*



*Рис 4.12. Визначення автомобільних доріг з найбільшою кількістю смуг та наявними пішохідними доріжками які належать до регіональних.*

**5. Інструкція користувача та системні вимоги**

**1. Компоненти ПЗ**

Додаток розроблений мовою програмування swift використовуючи фреймоворк для стоврення інтерфейсів UIKit, у середовищі розробки Xcode. Додаток був розроблений під пристрої iPhone і iPad.

**2. Встановлення ПЗ**

Щоб встановити додаток, потрібно знайти його у додатку App Store під назвою ‘iRoads’, встановити. Для запуску потрібно натиснути на іконку програми. Причому треба звернути увагу чи не застаріла у вас версія операційної системи, та чи достатньо пам’яті на пристрої.

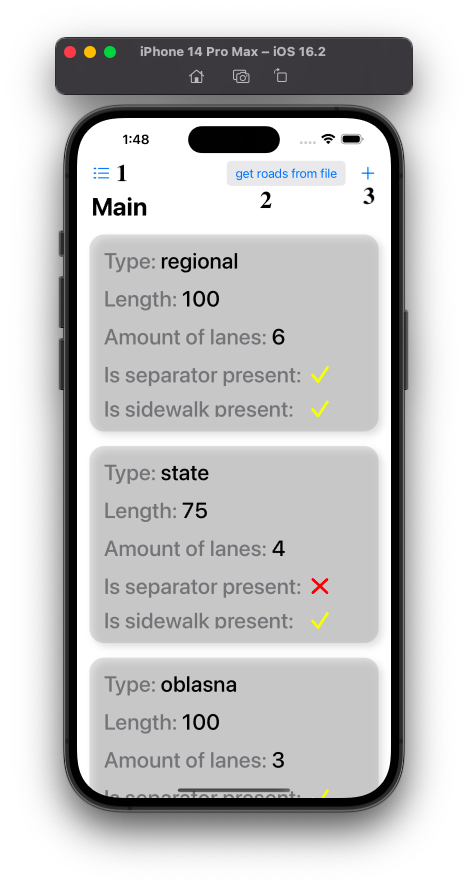
**3. Налаштування ПЗ**

Додаток позволяє змінювати чорну та білі теми. Жодних інших налаштувань додаток не підтримує і не потребує.

**4. Базові функції ПЗ**

Функції головного екрану (рис. 5.1.) :

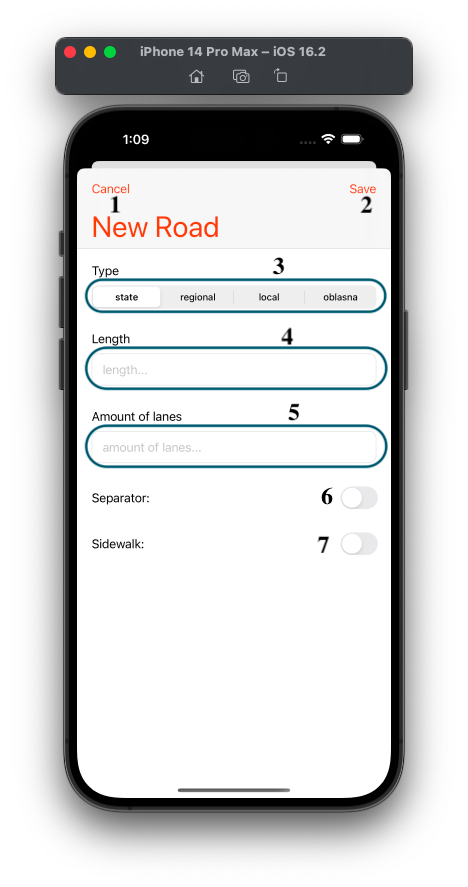
1. Відкриває side menu
2. Відкриває екран вибору файла
3. Відкриває екран введення даних вручну



*Рис. 5.1. Головний екран*

Функції екрану введення даних вручну (Рис. 5.2.):

1. Закриває поточний екран без збереження даних
2. Закриває поточний екран з збереженням даних
3. Група кнопок для вибору типу дороги
4. Текстове поле для вводу довжини дороги
5. Текстове поле для вводу кількості смуг дороги
6. Перемикач для визначення чи має дорога розділювач
7. Перемикач для визначення чи має дорога пішохідну доріжку

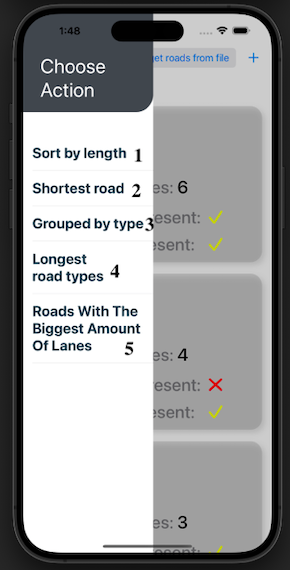


*Рис. 5.2 Екран для вводу даних*

Функції екрану side menu (рис. 5.3):

1. Перехід до екрана впорядкування доріг за протяжністю методом підрахунку
2. Перехід до екрана знаходження найкоротшої дороги, де найбільша кількість смуг
3. Перехід до екрана знаходження всіх доріг, в яких наявні розділювачі посередині, кількість смуг >2 та групування їх за типом.
4. Перехід до екрана визначення типів автомобільних доріг, з найбільшою протяжністю та наявністю пішохідних доріжок.

5) Перехід до екрана визначення автомобільних доріг з найбільшою кількістю смуг та наявними пішохідними доріжками які належать до регіональних.



*Рис. 5.3. Екран side menu*

Функції екрану впорядкування доріг за протяжністю методом підрахунку (рис. 5.4.)

1) Повернення до головного екрану

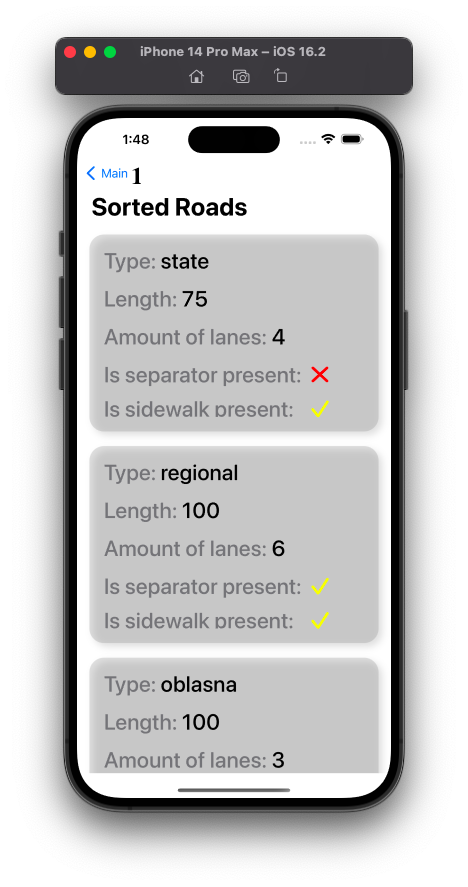


Рис 5.4. Екран впорядкування доріг за протяжністю методом підрахунку

**5. Аналіз помилок**

1) Якщо помилка виникла при зчитуванні даних з файлу перевірте чи його заповнено так як вказано в пункті ‘Структура файлу вхідних даних’.

2) При виникненні помилки при введенні даних вручну прочитайте повідомлення, натисніть OK та попробуйте вирішити проблему на основі повідомлення.

**6. Системні вимоги**

1) операційна система iOS версією не нижче 15.0.

# **7. Опис виняткових ситуацій**

* Залишення пустим поля для вводу довжини

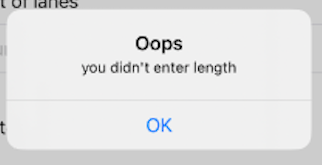


Рис. 6.1. Помилка при залишенні пустим поля для вводу довжини

* Залишення пустим поля для вводу кількості смуг

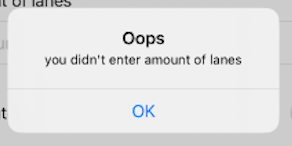


Рис. 6.2. Помилка при залишенні пустим поля для вводу кількості смуг

* Введення некоректних символів у поле для вводу довжини

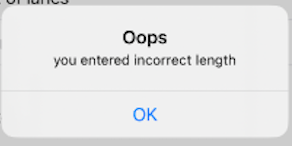


Рис. 6.3. Помилка при введенні некоректних символів у поле для вводу довжини

* Введення некоректних символів у поле для вводу кількості смуг

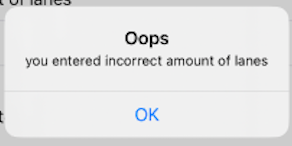


Рис. 6.4. Помилка при введенні некоректних символів у поле для вводу кількості смуг

* Некоректний вміст файлу

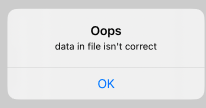


Рис. 6.4. Помилка при некоректному вмісту файлу

# **Структура файлів вхідних даних**

Користувач може ввести дані з текстового файлу. Кожен рядок представляє інформацію про одну дорогу. Перший елемент рядка це тип, він може мати значення “regional”, “oblasna”, “state” або “local”. Другий елемент довжина – позитивне число. Третій елемент кількість смуг – позитивне число. Останні два елемента наявність розділювача та пішохідної доріжки, вони можуть мати значення “true” або “false”.

Приклад коректного заповнення файлу:

State 100 4 true false

**Висновок**

Під час виконання курсової роботи я значно поглибив та закріпив свої навички та теоретичні знання ООП, яке я вивчав минулого року. Під час розробки своєї програми я застосовував такі принципи ООП як наслідування, інкапсуляція, поліморфізм, що мені дуже допомогло зрозуміти саме їхню практичну цінність.

Також я закріпив багато навичок з дисципліни “Вступ до інженерії програмного забезпечення”. Це включає моделювання UML діаграм різних рівнів абстракції. Спершу я побудував діаграму прецедентів, яка описує бізнес логіку додатку, а потім діаграму класів та послідовності. Для побудови якісного та зручного інтерфейса користувача мені дуже пригодилися знання десяти евристик. Також знання з ВІПЗу мені значно допомогли при визначенні системних вимог, написанні інструкції користувача та коментарів у коді.

Для написання додатку я використовував мову програмування swiftта фреймворк для побудови інтерфейсів UIKit. Виконуючи курсову роботу я здобув багато досвіду у програмуванні цією мовою. Я створював користувацькі типи (class, struct, enum), публічні та приватні методи і поля, ініціалізатори. Використовував опційні типи. Також навчився використовувати патерни проєктування singleton при програмуванні класу RoadManagerта патерн delegate при передачі даних між екранами. Я опанував елементи фреймворку UIKit: клас UILabel для відображення тексту, UITextField для відображення поля для вводу інформації, UIAlertController для відображення повідомлення, UITableView для відображення даних у вигляді списку, UIButton для відображення кнопки, UIGestureRecognizer для роботи з жестами. Реалізував програму з кількома екранами та навчився роботи навігацію між ними.

# **Список використаних джерел**

1. **Об'єктно-орієнтоване програмування**: методичні вказівки до виконання курсової роботи для студентів спеціальності 6.121 «Інженерія програмного забезпечення» / Укл. Коротєєва Т.О., Дяконюк Л.М.– Львів: Національний університет “Львівська політехніка ” кафедра програмного забезпечення, 2023. – 27с.
2. Левус Є., Мельник Н. Вступ до інженерії програмного забезпечення : навч. посіб. Львів : Вид-во Львів. політехніки, 2018. 246 с.
3. Левус Є.В., Марусенкова Т.А., Нитребич О.О.. Життєвий цикл програмного забезпечення: навчальний посібник. Львів : Видавництво Львівської політехніки, 2017. 208 с.
4. Apple developer documentation [Електронний ресусрс] : [Веб-сайт]. -Режим доступу: <https://developer.apple.com/documentation/>
5. Hacking with swift [Електронний ресусрс] : [Веб-сайт]. -Режим доступу: [https://www.hackingwithswift.com](https://www.hackingwithswift.com/)
6. Swift documentation [Електронний ресусрс] : [Веб-сайт]. -Режим доступу: <https://www.swift.org/documentation/>