Zintegrowany Program Rozwoju Politechniki Lubelskiej - część druga

POLITECHNIKA LUBELSKA
WYDZIAŁ ELEKTROTECHNIKI I INFORMATYKI

KIERUNEK STUDIÓW INFORMATYKA

MATERIAŁY DO ZAJĘĆ LABORATORYJNYCH

Programowanie w języku SWIFT

Autor:
dr inż. Maria Skublewska-Paszkowska

Lublin 2021

Fundusze Europejskie Wiedza Edukacja Rozwój
Rzeczpospolita Polska
Unia Europejska Europejski Fundusz Społeczny

LABORATORIUM 1. ZAPOZNANIE ZE ŚRODOWISKIEM PROGRAMISTYCZNYM XCODE.

Cel laboratorium:

Poznanie środowiska deweloperskiego Xcode.

Zakres tematyczny zajęć:

- tworzenie projektów w środowisku Xcode,
- podstawy pracy w środowisku Xcode,
- tworzenie aplikacji konsolowych,
- tworzenie aplikacji na system macOS z inferfejsem Storyboard,
- tworzenie aplikacji na system macOS z inferfejsem SwiftUI,
- debugowanie programu.

Pytania kontrolne:

1. Do czego służy środowisko Xcode?
2. W jakich językach można wytwarzać oprogramowanie?
3. Jakie narzędzia ma wbudowane środowisko Xcode?
4. Jakie typy aplikacji można tworzyć?

Xcode to środowisko deweloperskie dedykowane na komputery Mac, dzięki któremu można tworzyć aplikacje dla systemów takich jak: iOS, macOS, tvOS i watchOS. Środowisko ma wbudowane takie języki jak C, C++, Objective-C oraz Swift, a także narzędzia do tworzenia aplikacji m.in. na urządzenia iPhone, iPad, Mac, Apple TV. Xcode to zintegrowane środowisko programistyczne (ang. Integrated Development Environment – IDE) firmy Apple. Oznacza to, że programiści mają możliwość korzystania z wielu wbudowanych narzędzi takich jak:

- edytora **Interface Builder** – do definiowania elementów graficznych widoku poprzez ich wybranie i przeciągnięcie na kanwę;
- narzędzia **Storyboard** – do tworzenia graficznego interfejsu użytkownika (GUI), widoków i przejść między nimi;
- **debuggera** – do uruchamiania i testowania programu (m.in. zatrzymanie programu w określonych miejscach programu, wyświetlenie zawartości pamięci, procesora);
- **symulatora** – dla aplikacji dedykowanych na system iOS;
- **dokumentacji**.

Xcode w wersji 12 został wypuszczony na rynek w czerwcu 2021 roku. Wspiera on język programowania Swift w wersji 5.3 oraz pakiety SDK (ang. Software Development Kits) dla iOS 14, iPadOS 14, tvOS 14, watchOS 7 i macOS Big Sur. Uniwersalne aplikacje (macOS Universal) dedykowane są zarówno dla Apple Silicon oraz komputerach Mac z procesorami Intel.

Właściwości Xcode 12:

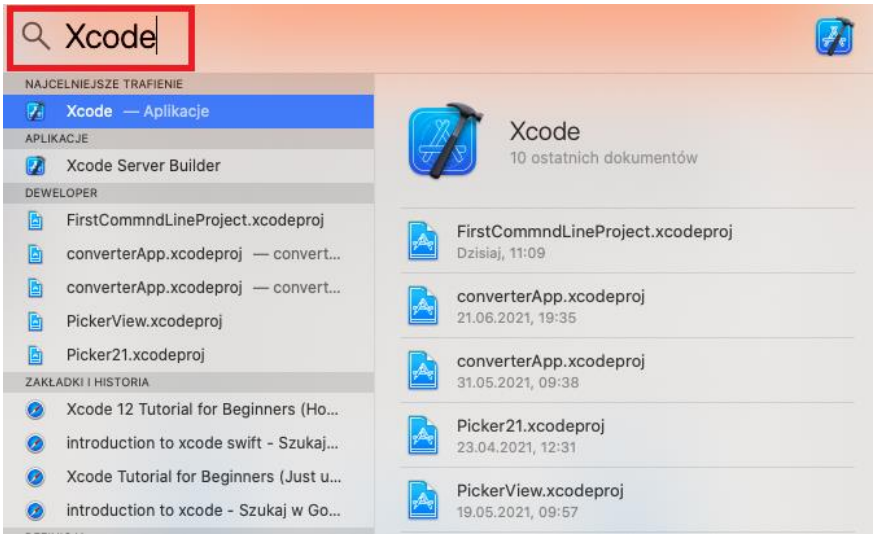
- dokumenty można otwierać we własnej karcie, co ułatwia szybkie przełączanie się między plikami przy zachowaniu konfiguracji Xcode;
- dodana obsługa podglądu widżetów, klipów aplikacji i zawartości w pakietach Swift (w celu płynnego podgląd na żywo na urządzeniu, Xcode instaluje nową aplikację Xcode Previews na iOS 14 i iPadOS);
- nowy protokół *LibraryContentProvider*, który umożliwia wyświetlanie widoków i modyfikatorów w bibliotece Xcode;
- uzupełnienia kodu mają nowy interfejs użytkownika, ułatwiający znalezienie uzupełnienia (uzupełnienia są również dokładniejsze i do 12 razy szybsze);
- obsługa zasobów graficznych Scalable Vector Graphic (SVG), które zachowują swoją reprezentację wektorową na systemach macOS 10.15 lub nowszych, iOS 13 lub nowszych oraz iPadOS 13 lub nowszych;
- debugowanie – w przypadku, gdy wystąpi awaria w debugerze, Xcode drukuje komunikaty o awariach w konsoli, podobnych do tych wyświetlanych w CrashReporter;
- obsługa nowego *safeAreaLayoutGuide* w NSView wprowadzonego w systemie macOS 11;
- nowa minimapa dla płótna Interface Builder;
- opcja *Current Date* dla NSDatePicker w Interface Builder;
- symulator może wyświetlać symulowane urządzenie w trybie pełnoekranowym lub umieszczać jego okno obok Xcode;
- symulator obsługuje procesy 64-bitowe i 32-bitowe dla watchOS 7.

Minimalne wymagania dla środowiska Xcode 12 to:

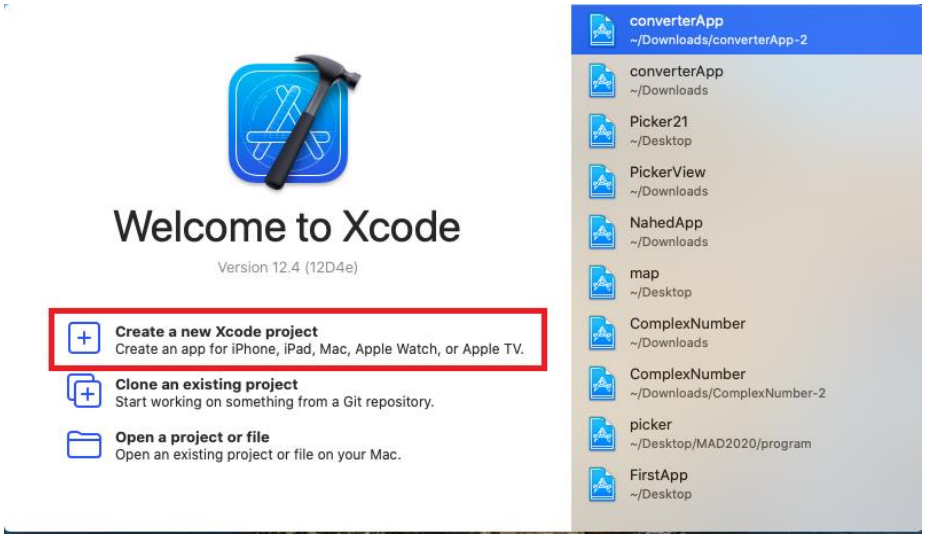
- komputer Mac z systemem macOS 11 (Big Sur);
- 4GB pamięci RAM;
- 8 GB wolnego miejsca na dysku;
- komputer 2013-2015 lub nowszy: Mac, MacBook, iMac lub Mac mini.

Najprościej jest pobrać i zainstalować środowisko Xcode ze sklepu App Store. Jest to narzędzie bezpłatne. Należy posiadać Apple ID i być zalogowanym. Po instalacji Xcode można uruchomić. W prawym górnym rogu, na pasku można wybrać wyszukiwanie i wpisać Xcode (Rys. 1.1). Zostanie wyszukane środowisko deweloperskie oraz inne programy pisane w tym środowisku.

Po wybraniu środowiska, zostanie ono uruchomione (Rys. 1.2). Po prawej stronie znajdują się wcześniej utworzone programy. Aby utworzyć nowy projekt, należy wybrać pierwszą opcję *Create a new Xcode project*.



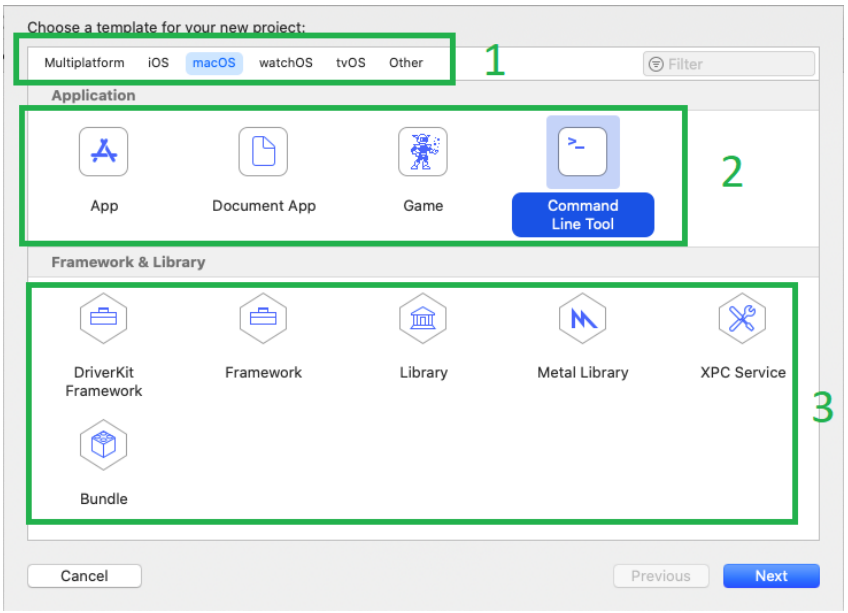
Rys. 1.1. Wyszukanie środowiska Xcode



Rys. 1.2. Uruchomienie środowiska Xcode

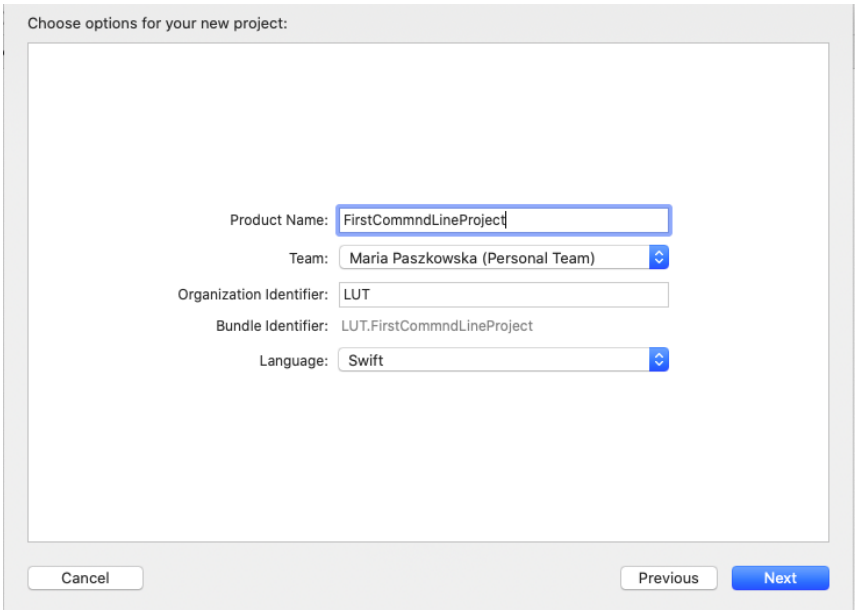
Następnie należy wybrać szablon projektu, rodzaj tworzonej aplikacji lub szkielet programistyczny (ang. framework), albo bibliotekę. Przedstawiono to na Rys. 1.3:

1. W Xcode 12 do wyboru jest 6 szablonów do tworzenia programów:
   a. wieloplatformowych;
   b. na system mobilny iOS;
   c. na system watchOS dedykowany dla *Apple Watch*;
   d. na system tvOS dedykowany na Apple TV;
   e. inne (nie wymienione powyżej).
2. W każdym szablonie znajdują się typy aplikacji, które można wybrać.
3. Jeśli istnieje taka potrzeba, można wybrać tworzenie frameworku lub biblioteki, dla wybranego szablonu.



Rys. 1.3. Wybranie szablonu i rodzaju aplikacji

Kolejnym krokiem jest zdefiniowanie oraz wybranie opcji dla tworzonego projektu, co zostało przedstawione na Rys. 1.4. Należy wpisać nazwę projektu (*Product Name*), podać zespół (*Team*) i identyfikator organizacji (*Organization Identifier*), jeśli istnieje. Nazwa aplikacji pojawi się później w *App Store* i na urządzeniu po jej zainstalowaniu. Nazwa produktu musi mieć co najmniej 2 znaki i nie więcej niż 255 bajtów. Identyfikator organizacji to ciąg odwrotnego DNS, który jednoznacznie identyfikuje firmę. Na podstawie tych informacji tworzony jest unikalny identyfikator aplikacji (*Bundle Identifier*). Ostatnią opcją do wyboru jest język programowania. Można wybrać jeden z: Swift, Objective-C, C++ oraz C.

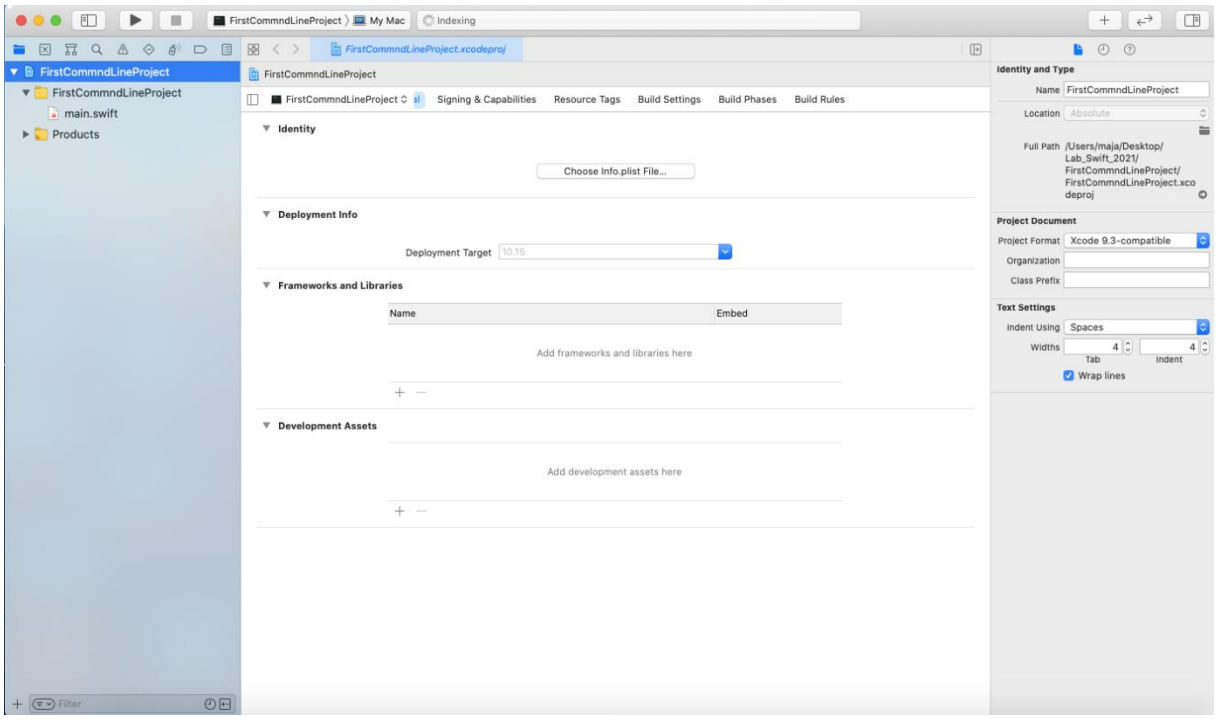


Rys. 1.4. Wybranie opcji dla projektu

Następnie należy zdefiniować ścieżkę do projektu.

Po utworzeniu projektu, zostanie on otworzony, co przedstawiono na Rys. 1.5. Pierwsza strona, która się domyślnie otwiera zawiera właściwości wprowadzone podczas tworzenia projektu. Można tutaj nadać numer projektu, a także dodać frameworki, czy biblioteki, jeśli wymaga tego projekt. W kolejnych zakładkach znajdują się właściwości dotyczące:

- *Signing and capabilities* – zapewnia aplikacji dostęp do usługi aplikacji dostarczanej przez firmę Apple, takiej jak CloudKit, Game Center lub Zakup w aplikacji. Aby korzystać z niektórych usług aplikacji, należy udostępnić aplikację, dodając funkcję w edytorze projektów Xcode, która poprawnie skonfiguruje usługę aplikacji. Xcode edytuje pliki uprawnień i listy właściwości informacji, dodaje powiązane struktury i konfiguruje zasoby podpisywania.
- *Resource Tags* – to narzędzia do tworzenia i edytowania tagów, dodawania i usuwania zasobów będących częścią tagu oraz określania, kiedy zasoby skojarzone z tagiem są pobierane przez system operacyjny.
- *Build Settings* – to ustawienia dotyczące rozlokowania (ang. *Deployment*), pakietów (ang. *Packaging*), podpisów (ang. *Signing*), kompilatora języka Swift (ang. *Swift Compiler - Language*) oraz właściwości zdefiniowane przez użytkownika (ang. *User-Defined*).
- *Build Phases* – to ustawienia kompilacji.
- *Build Rules* – to ustawienia reguł dla kompilacji.



Rys. 1.5. Informacje o projekcie dla aplikacji konsolowej

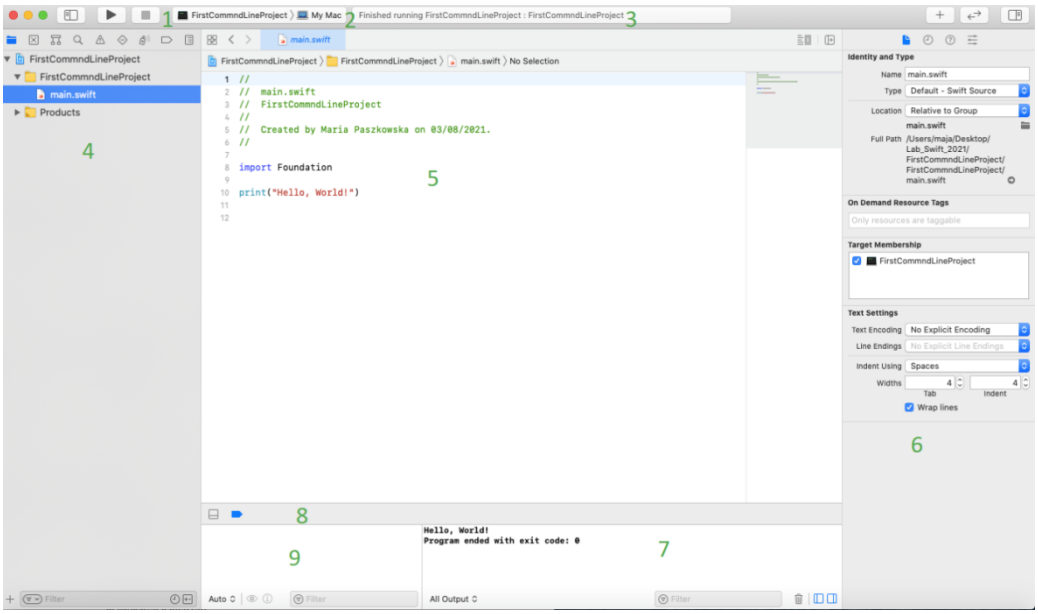
Elementy Xcode zostały przedstawione na Rys. 1.6:

1. Uruchamianie i zatrzymywanie aplikacji (ang. *Build and then run, Stop*).

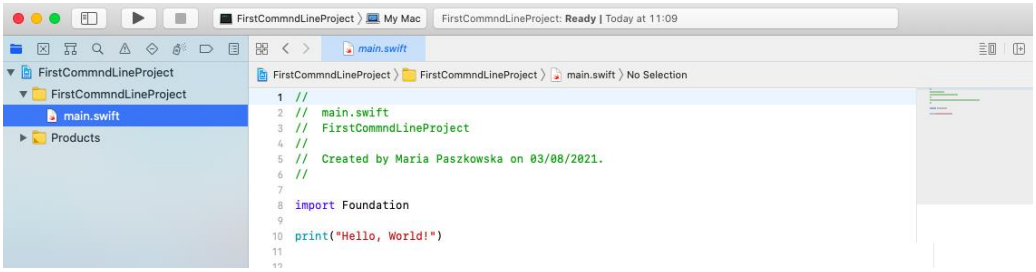
2. Uruchomienie aplikacji dotyczy określonego celu (ang. *Target*).
3. U góry, pośrodku, znajduje się pasek stanu. To on informuje, co się aktualnie dzieje podczas kompilacji i uruchamiania. Tutaj zostanie wyświetlona informacja o błędach.
4. Nawigacja (ang. *Navigation*) pomaga zlokalizować zasoby w projekcie Xcode, takie jak pliki Swift, problemy, czy punkty przerwania.
5. Edytor (ang. *Editor*) służy do programowania. To tutaj pisany jest kod w języku Swift. Jeśli projekt zawiera *Interface Builder*, w tym miejscu tworzony jest interfejs użytkownika.
6. Inspektor (ang. *Inspector*) pomaga w sprawdzaniu i dostosowywaniu atrybutów plików, elementów interfejsu użytkownika itp.
7. Konsola (ang. *Console*) służy do wyświetlania danych wyjściowych podczas działania aplikacji. Umożliwia także wprowadzanie danych.
8. Obszar debugowania (ang. *Debugging area*) używany jest do debugowania aplikacji, aby zobaczyć wyniki debugowania i co się dzieje, gdy aplikacja ulegnie awarii lub wystąpił błąd.
9. Widok zmiennych (ang. *Variable view*) pozwala na sprawdzenie wartości zmiennych w czasie wykonywania programu.

Przydatne skróty podczas obsługi Xcode to:

- **Build**: Command + B;
- **Run**: Command + R;
- **Test**: Command + U;
- **Stop**: Command +.



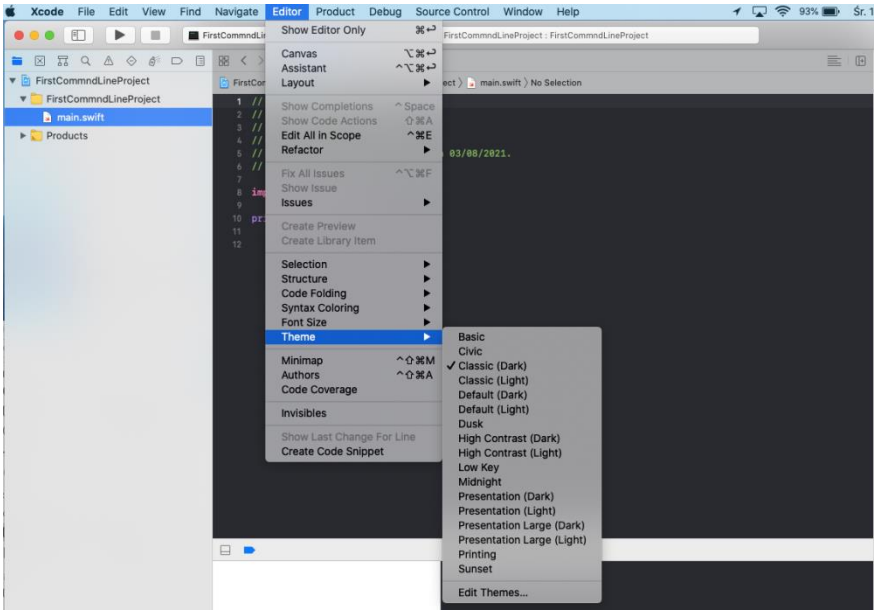
Rys. 1.6. Struktura Xcode



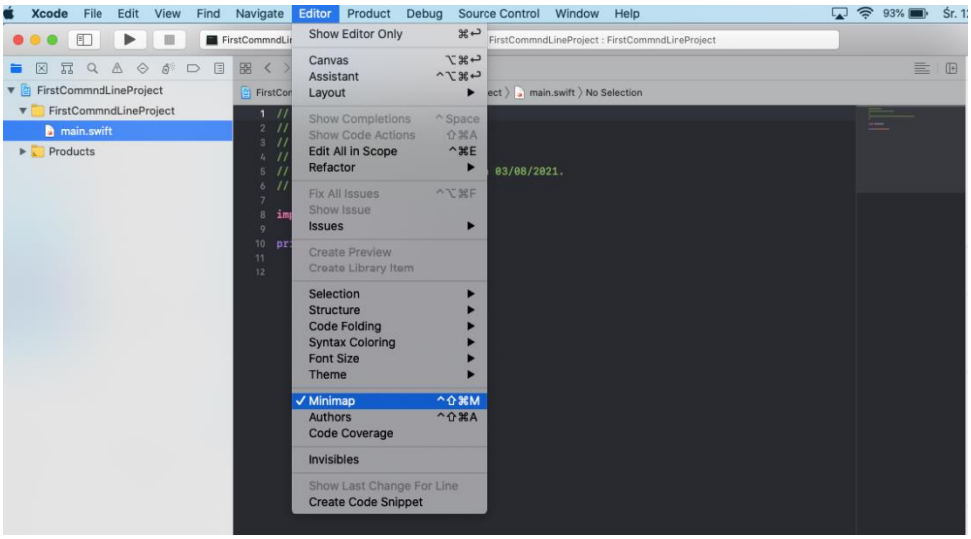
Rys. 1.7. Plik main.swift dla aplikacji konsolowej

Dla aplikacji konsolowych, plik *main.swift* jest głównym plikiem projektu (Rys. 1.7). W projekcie jest zaimportowany framework *Foundation*. Zapewnia on podstawową funkcjonalność dla aplikacji i struktur, w tym przechowywanie i trwałość danych, przetwarzanie tekstu, obliczanie daty i godziny, sortowanie i filtrowanie oraz obsługę sieci. Klasy, protokoły i typy danych zdefiniowane przez Foundation są używane w zestawach SDK dla systemów macOS, iOS, watchOS i tvOS.

Ustawienia środowiska Xcode można zmienić za pomocą menu znajdującego się na górze pulpitu, pod warunkiem, że okno projektu jest aktywne. Przykładowe ustawienia dotyczące zmiany tematu oraz widoku minimapy zostały przedstawione na Rys. 1.8 i 1.9.



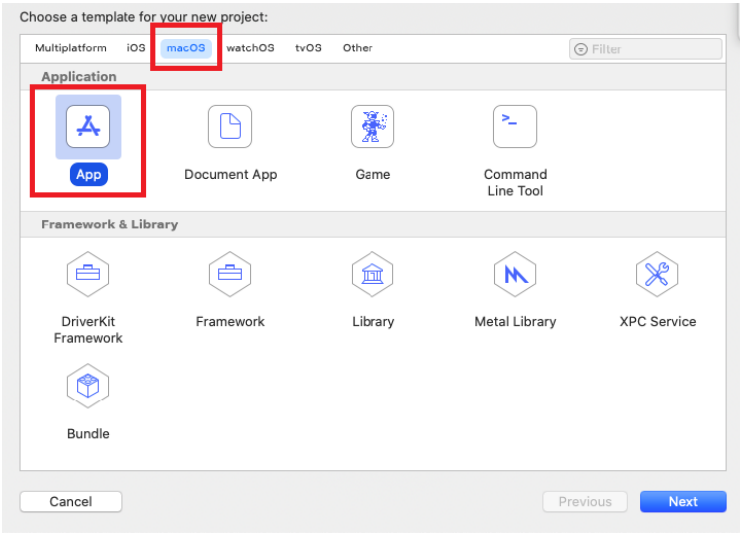
Rys. 1.8. Zmiana tematu środowiska Xcode



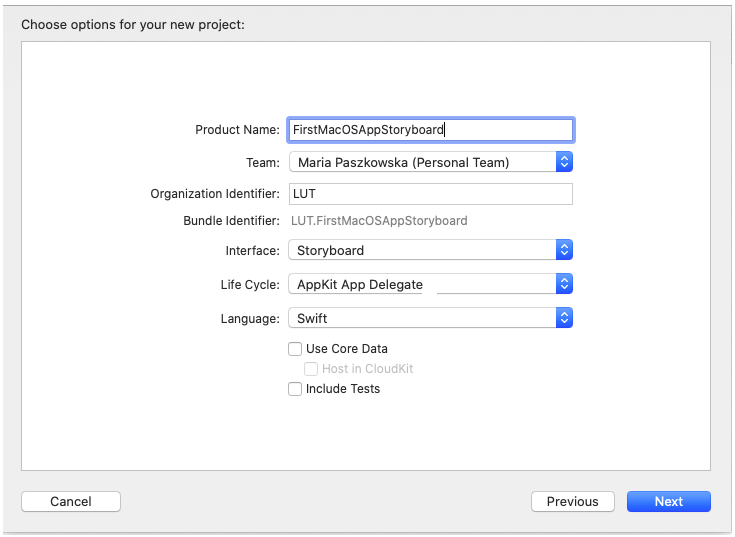
Rys. 1.9. Ustawienie widoku minimapy środowiska Xcode

Utworzenie nowego projektu dla aplikacji macOS

Bardziej zaawansowanym projektem jest aplikacja z interfejsem użytkownika (Rys. 1.10). Projekt dla aplikacji dla systemu macOS może zostać wykonany z zastosowaniem interfejsu *Storyboard* (Rys. 1.11) lub *SwiftUI* (Rys. 1.16).



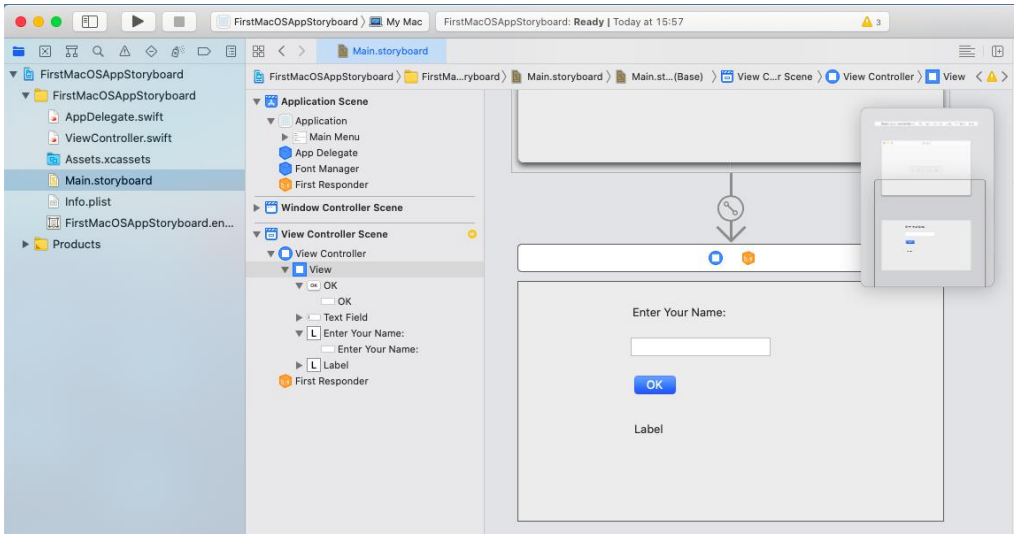
Rys. 1.10. Utworzenie projektu dla systemu macOS



Rys. 1.11. Utworzenie projektu dla systemu macOS z interfejsem Storyboard

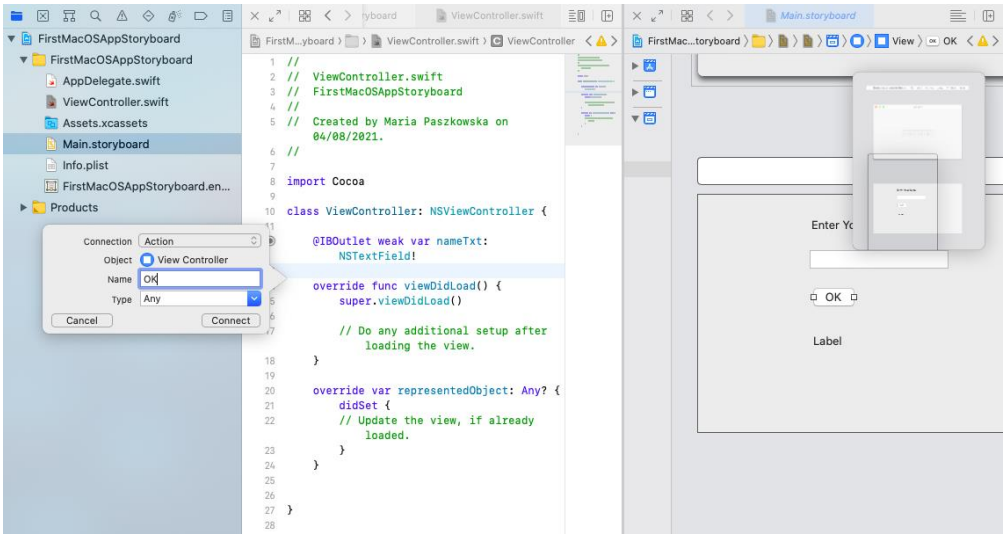
W przypadku pierwszego rozwiązania struktura plików jest bardziej złożona niż dla aplikacji konsolowej (Rys. 1.12). Są to m.in:

- *AppDelegate.swift* – klasa ma za zadanie obsługiwać zdarzenia cyklu życia aplikacji, czyli odpowiadać na uruchamianą aplikację, działać w tle, na pierwszym planie, czy otrzymywać dane.
- *ViewController.swift* – klasa definiuje wspólne zachowanie wszystkich kontrolerów widoku. W klasie tej tworzone są zmienne, stałe i metody. Przeprowadzana jest aktualizacja widoków w odpowiedzi na zmiany danych źródłowych, definiowane są interakcje użytkowników z widokami, następuje zarządzanie układem interfejsu oraz koordynowanie obiektami.
- *Main.Storyboard* – plik, który zawiera zbiór widoków i informacje o tym, jak te widoki są powiązane. Przy użyciu narzędzia Interface Builder tworzony jest interfejs użytkownika.

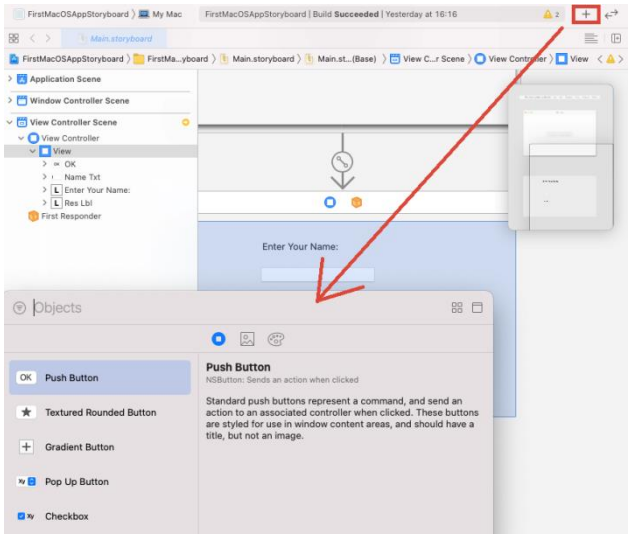


Rys. 1.12. Projekt aplikacji macOS z interfejsem Storyboard

Na Rys. 1.12 przedstawiono prosty interfejs użytkownika Storyboard dla aplikacji macOS w pliku składający się z dwóch etykiet, jednego pola tekstowego i przycisku. W celu dodania elementów interfejsu graficznego, należy je wyszukać i przeciągnąć na kanwę (Rys. 1.13). Utworzony widok nie ma żadnego powiązania z klasą kontrolera widoku. Należy więc powiązać poszczególne elementy widoku z kontrolerem widoku. W tym celu przy wciśniętym przycisku ctrl należy zaznaczyć i przeciągnąć element do klasy kontrolera. Powinna ukazać się niebieska linia. Po zwolnieniu przycisku, wyświetli się okno, w którym należy zdefiniować nazwę zmiennej oraz jej typ (Outlet dla zmiennych lub Action dla metod). Na Rys. 1.14 przedstawiono powiązanie z przyciskiem OK.

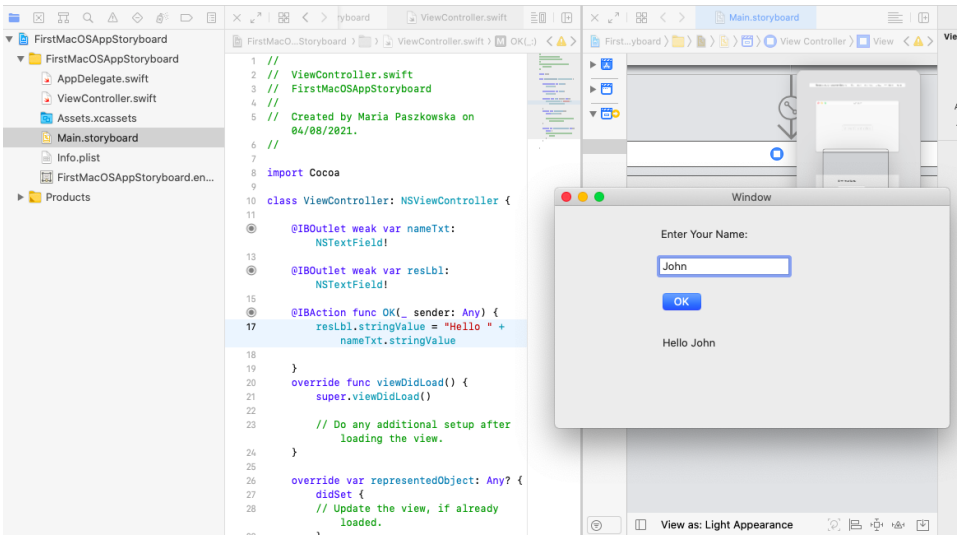


Rys. 1.13. Utworzenie powiązania pomiędzy interfejsem oraz kontrolerem widoku



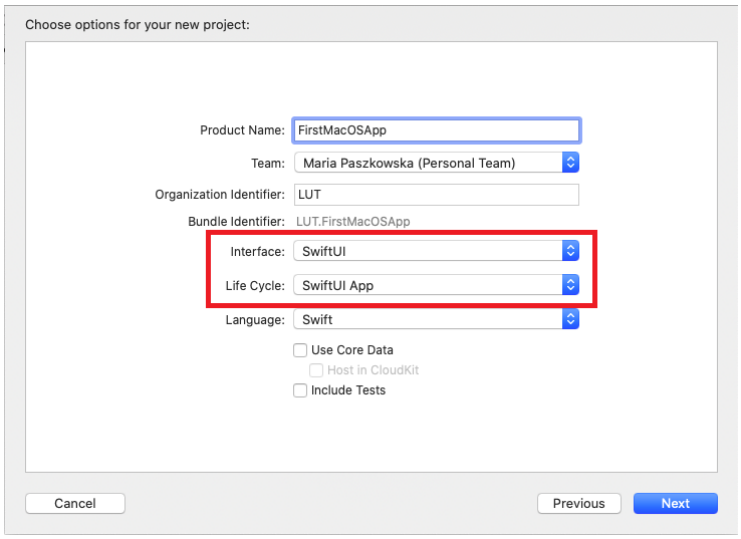
Rys. 1.14. Dodanie elementów graficznych

Po implementacji metody przycisku *OK*, działanie aplikacji zostało przedstawione na Rys. 1.15.

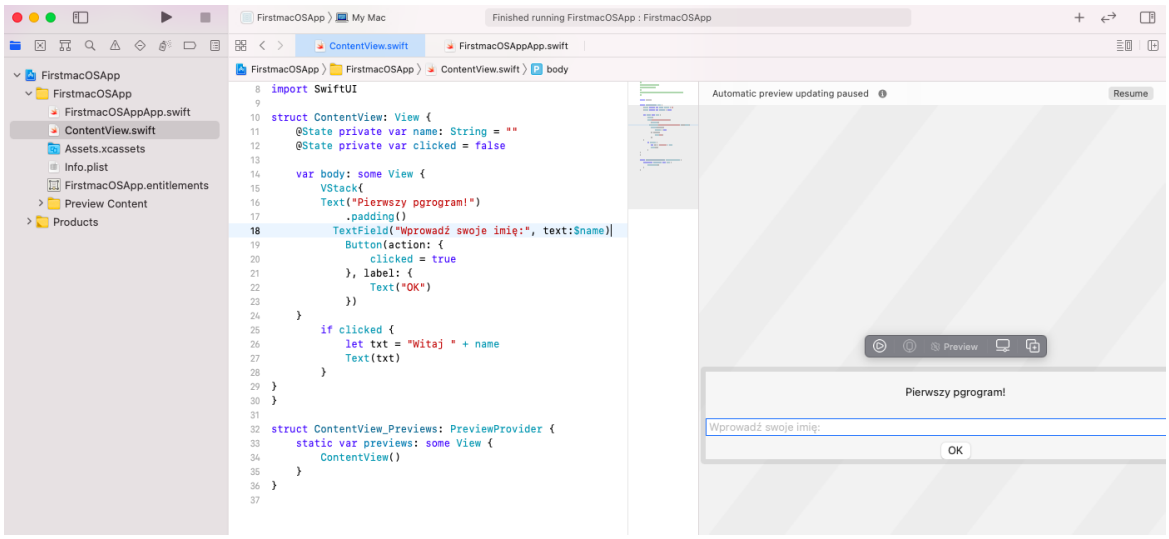


Rys. 1.15. Aplikacja na system macOS z interfejsem Storyboard

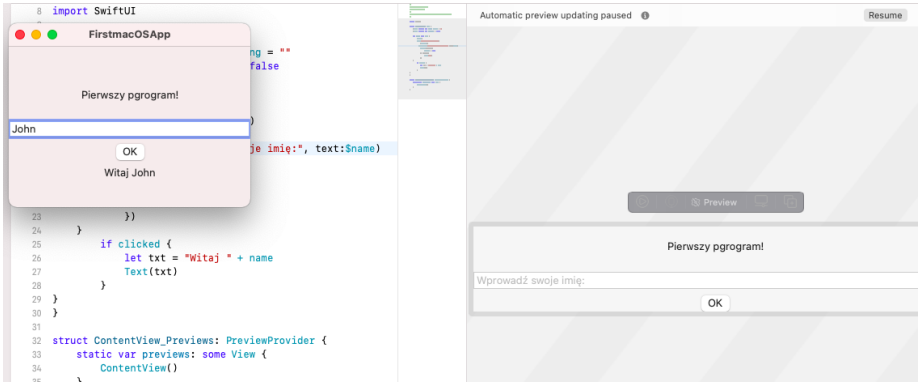
Tworzenie **aplikacji macOS z interfejsem SwiftUI** jest przedstawione na Rys. 1.16. Należy wybrać interfejs *SwiftUI* oraz dla cyklu życia aplikacji – *SwiftUI App*. Aplikacja jest implementowana bez Storyboard. W klasie *ContentView.swift* generowany jest cały interfejs (Rys. 1.17). Przedstawiona aplikacja składa się z etykiety, pola tekstowego oraz przycisku. Zdefiniowany interfejs jest automatycznie wygenerowany po prawej stronie kodu. Funkcja ta jest możliwa tylko od systemu *Big Sure* oraz *Xcode 12.5*. Po wpisaniu imienia i wciśnięciu przycisku, generowany jest tekst powitalny i wyświetlany poniżej. Dodatkowo ustawiono rozmiar aplikacji na 300x200 w pliku *FirstmacOSAppApp.swift* przy zastosowaniu funkcji *frame()*. Działanie programu zostało przedstawione na Rys. 1.18.



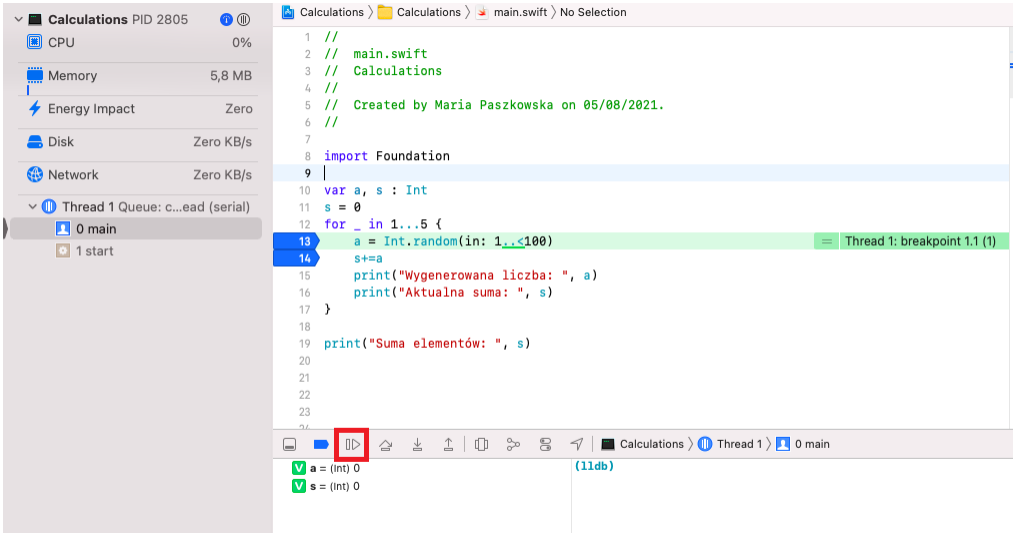
Rys. 1.16. Tworzenie aplikacji na system macOS z interfejsem SwiftUI



Rys. 1.17. Aplikacja na system macOS z interfejsem SwiftUI

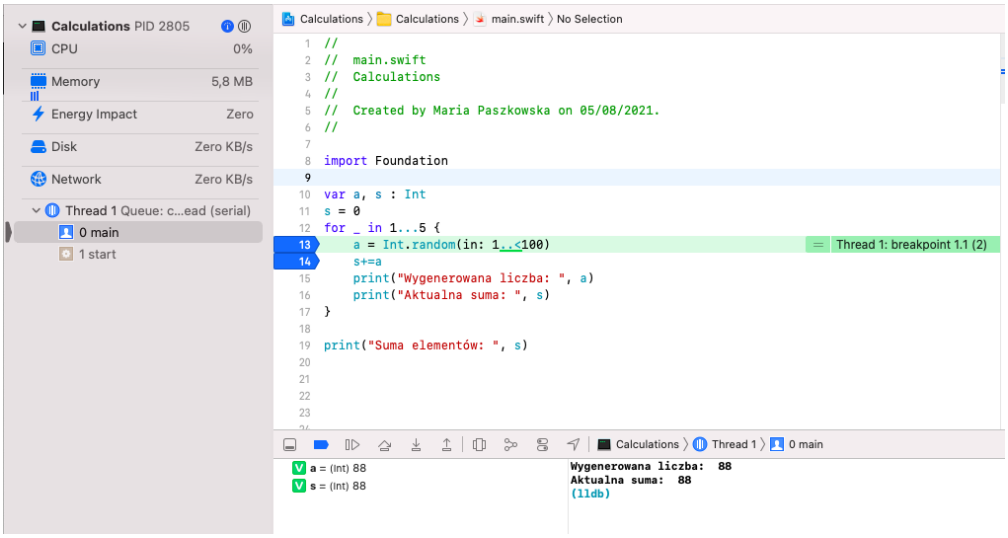


Rys. 1.18. Wynik aplikacji na system macOS z interfejsem SwiftUI



Rys. 1.19. Debugowanie aplikacji

Debugowanie (ang. *Debugging*) to proces, który pomaga zlokalizować przyczynę błędu. Jedną z możliwości debugowania jest użycie punktów przerwania (ang. *Breakpoints*), które pozwalają zatrzymać działanie programu w momencie zdefiniowanym przez programistę. Od tego momentu można krok po kroku śledzić działanie aplikacji. Proces ten został przedstawiony na Rys. 1.19 oraz 1.20.



Rys. 1.19. Debugowanie aplikacji- wartości zmiennych

Zadanie 1.1. Utwórz aplikację konsolową

Polecenie 1. Utwórz prostą aplikację konsolową, która wylosuje 3 liczby całkowite, wyświetli je, a następnie obliczy ich sumę i średnią.
Do losowania liczb można zastosować funkcję `random(in:)`.

Polecenie 2. Wykonaj debugowanie aplikacji.

Zadanie 1.2. Utwórz aplikację na system macOS z interfejsem SwiftUI

Polecenie 1. Utwórz prostą aplikację, która wczyta 3 liczby całkowite od użytkownika, obliczy sumę oraz średnią arytmetyczną. Należy założyć poprawność danych.

Polecenie 2. Wykonaj debugowanie aplikacji.

Zadanie 1.3. Utwórz aplikację na system macOS z interfejsem Storyboard

Polecenie 1. Utwórz prostą aplikację, która wczyta 3 liczby całkowite od użytkownika, obliczy sumę oraz średnią arytmetyczną. Należy założyć poprawność danych.

Polecenie 2. Wykonaj debugowanie aplikacji.

Zintegrowany Program Rozwoju Politechniki Lubelskiej - część druga

Materiały zostały opracowane w ramach projektu
„Zintegrowany Program Rozwoju Politechniki Lubelskiej – część druga”,
umowa nr **POWR.03.05.00-00-Z060/18-00**
w ramach Programu Operacyjnego Wiedza Edukacja Rozwój 2014-2020
współfinansowanego ze środków Europejskiego Funduszu Społecznego