День 1.

2.2.2 Выполнение работ по Модулю Б. Верстка приложения – 2 часа, практическая работа

Разработка пользовательского интерфейса (UI) и пользовательского опыта (UX) мобильного приложения.

1. *Задание чемпионата. Критерии оценивания по данному модулю. Принципы проверки и оценки.*
2. *Создание проекта в XCode. Основы языка Swift..*
3. *SwiftUI. Знакомство с базовыми компонентами – кнопки, поля ввода, изображения.*
4. *Разработка одного-двух экранов из задания регионального чемпионата*

Задание чемпионата и критерии оценивания

На примере сессии 2 регионального чемпионата

Общее

Необходимо разработать мобильное приложение для смартфона, удовлетворяющее следующим требованиям:

Минимальная версия ОС, поддерживаемая приложением, должна быть: Android - 11.0, iOS - 14.0.

В качестве бэкенда будет использован Supabase. Для авторизации в supabase используйте учетную запись, **выданную главным экспертом.**

В работе необходимо использовать систему контроля версий Git, который предоставляет организатор.

Необходимо строго следовать предложенному дизайну.

Необходимо осуществлять комментирование кода в созданных классах. Обязательны следующие комментарии:

- Описание назначения класса
- Дата создания
- Автор создания
- Описание назначения вложенных элементов программного кода

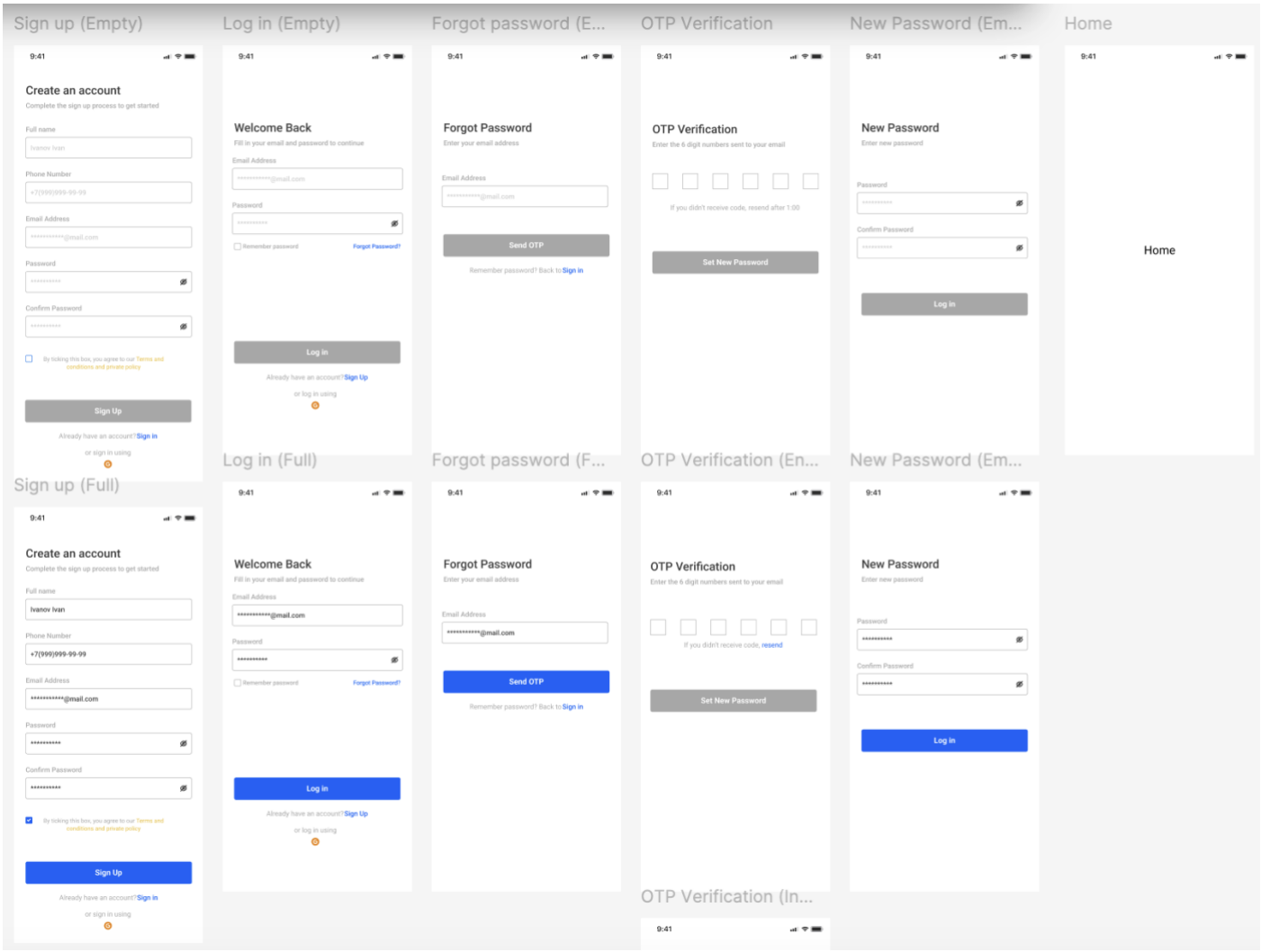
При разработке проекта приложения вам необходимо использовать архитектуру (см.файл с описанием архитектуры), в которой будут разделены слои бизнес-логики, представлений и домена. Изменение бизнес-логики и/или представления одного из экранов не должно повлечь за собой изменение других экранов и нарушение работоспособности приложения, за исключением переходов. Допускается использование SupaBase.

Файлы проекта распределены по папкам в соответствии с архитектурой. Допустимо использование папки Common для общих файлов.

Вся верстка должна быть адаптивной (следует учитывать разные размеры экранов). Необходимо:

- **Избегать появления большого пустого пространства;**
- **Следить за отсутствием искажения элементов;**
- **Все элементы должны полностью находится в границах и на месте, указанном в макете;**
- **Учитывать расстояние между элементами;**
- **Используйте шрифты согласно макету.**
- **Дизайн предложен в Figma:**

https://www.figma.com/file/guyXW190UglGSwE2CCyafu/OECH-APP-Final?type=design&node-id=0-1&mode=design&t=lrg58KLyrOO3iQa5-0



Необходимо корректно обрабатывать запросы к серверу. В случае получения ошибки от сервера или отсутствия соединения с сетью Интернет необходимо отобразить соответствующий текст ошибки в диалоговом окне, которое должно закрываться только пользователем.

В процессе обмена данными с сервером должна осуществляться стандартная индикация.

Необходимо во время сессии работать в ветке “Path-X”, где X – это номер сессии. По завершению сессии необходимо сделать средствами Giltab Merge, с основной веткой, которая должна называться “main”, при этом ветка удалятся не должна.

1. Создайте экран «Sign Up», как на макете:

– Реализуйте проверку email на корректность (соответствие паттерну «name@domenname.ru», где имя и доменное имя может состоять только из маленьких букв и цифр). При некорректном заполнении необходимо отобразить ошибку любым способом;

– Реализуйте возможность отображения пароля;

– Реализуйте повторный ввод пароля для подтверждения;

– Реализовать просмотр политики конфиденциальности, которая хранится локально в проекте в виде PDF файле и открывается в свободной форме;

– При нажатии на кнопку «Sign Up» осуществляется переход на экран «Log In»

– Регистрация и переход на экран «Log In» осуществляется только при согласии с Условиями и политикой конфиденциальности

– Реализуйте возможность перехода на экран «Log In» при нажатии на «Sign in»

– Реализуйте отправку запроса на сервер для регистрации

– Реализуйте возможность регистрации посредством использования активного аккаунта Google

2. Создайте экран «Log In», как на макете:

- Реализуйте возможность отображения пароля
- При нажатии на «Forgot Password» осуществляется переход на экран «Forgot Password»
- При успешной авторизации осуществляется переход на экран «Home»
- Реализуйте возможность перехода на экран «Sign Up» при нажатии на «Sign Up»
- Реализуйте отправку запроса на сервер для авторизации с помощью почты и пароля
- Реализуйте возможность авторизации посредством использования активного аккаунта Google
- Реализуйте возможность сохранение пароля
- Обеспечить безопасное хранение пароля используя SHA-512

3. Создайте экран «Forgot Password», как на макете:

- При нажатии на кнопку «Send OTP», при наличии в поле ввода корректного e-mail, осуществляется переход на экран «OTP Verification»
- Реализуйте возможность перехода на экран «Log In» при нажатии на «Sign in»
- Реализуйте отправку запроса на сервер для получения кода

4. Создайте экран «OTP Verification», как на макете:

- Реализуйте возможность повторного запроса кода по истечению таймера 01:00 минута;
- Пока код не введён, кнопка «Set New Password» не активна;
- После ввода символа, соответствующая рамка квадрата окрашивается в синий цвет;
- Если код-пароль введён не верно, то все рамки квадратов становятся красными;
- При корректном коде при нажатии на кнопку «Set New Password» осуществляется переход на экран «New Password»
- Реализуйте отправку кода на сервер для верификации
- Реализуйте повторную отправку кода на сервер для верификации

5. Создайте экран «New Password», как на макете:

- Реализуйте проверку совпадения паролей;
- При нажатии на «Log In» осуществляется изменение пароля и переход на экран «Home»;
- Реализуйте отправку запроса на сервер для изменения пароля

Критерии оценивания по модулю

Тип аспекта	Аспект	Методика проверки аспекта	Требование или номинальный размер	Проф. задача	
И	Экран «Sign Up» соответствует макету	минус 0,1 за каждый элемент	15	3	1,50
И	Экран «Sign Up». Реализована			3	0,10
И	Экран «Sign Up». Реализована			3	0,05
И	Экран «Sign Up». Реализован			3	0,05
И	Экран «Sign Up». Реализован просмотр			3	0,4
И	Экран «Sign Up». При нажатии на			3	0,1
И	Экран «Sign Up». Регистрация и			3	0,2
И	Экран «Sign Up». Реализована			3	0,10
И	Экран «Log In» соответствует макету	минус 0,1 за каждый элемент	12	3	1,2
И	Экран «Log In». Реализована			3	0,05
И	Экран «Log In». При нажатии на			3	0,1
И	Экран «Log In». При успешной			3	0,1
И	Экран «Log In». Реализвована			3	0,1
И	Экран «Forgot Password»	минус 0,05 за каждый элемент	6	3	0,30
И	Экран «Forgot Password». При нажатии			3	0,2
И	Экран «Forgot Password». Реализуйте			3	0,10
И	Экран «OTP Verification» соответствует	минус 0,05 за каждый элемент	5	3	0,25
И	Экран «OTP Verification». Реализована			3	0,40
И	Экран «OTP Verification». Пока код не			3	0,10
И	Экран «OTP Verification». После ввода			3	0,10
И	Экран «OTP Verification». Если код-			3	0,10
И	Экран «OTP Verification». При			3	0,2
И	Экран «New Password» соответствует	минус 0,05 за каждый элемент	6	3	0,3
И	Экран «New Password». Реализуйте			3	0,05
И	Экран «New Password». При нажатии			3	0,1

Среда XCode

Среда программирования **Xcode** - инструмент, специально разработанный компанией Apple для написания различных программ на языке программирования Swift.

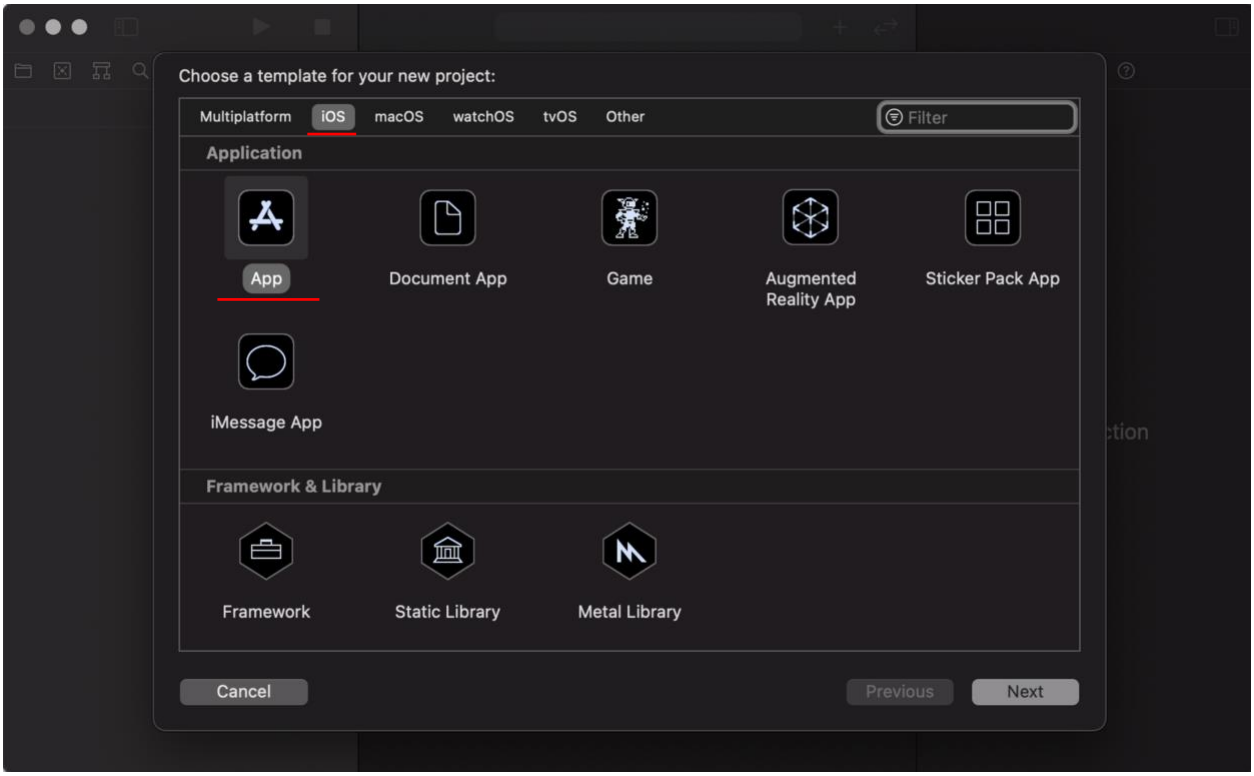
В верхней части экрана расположено меню (рис.ниже). С его помощью можно управлять программой: создавать новые проекты, изменять некоторые настройки и совершать другие подобные действия.

Создание проекта

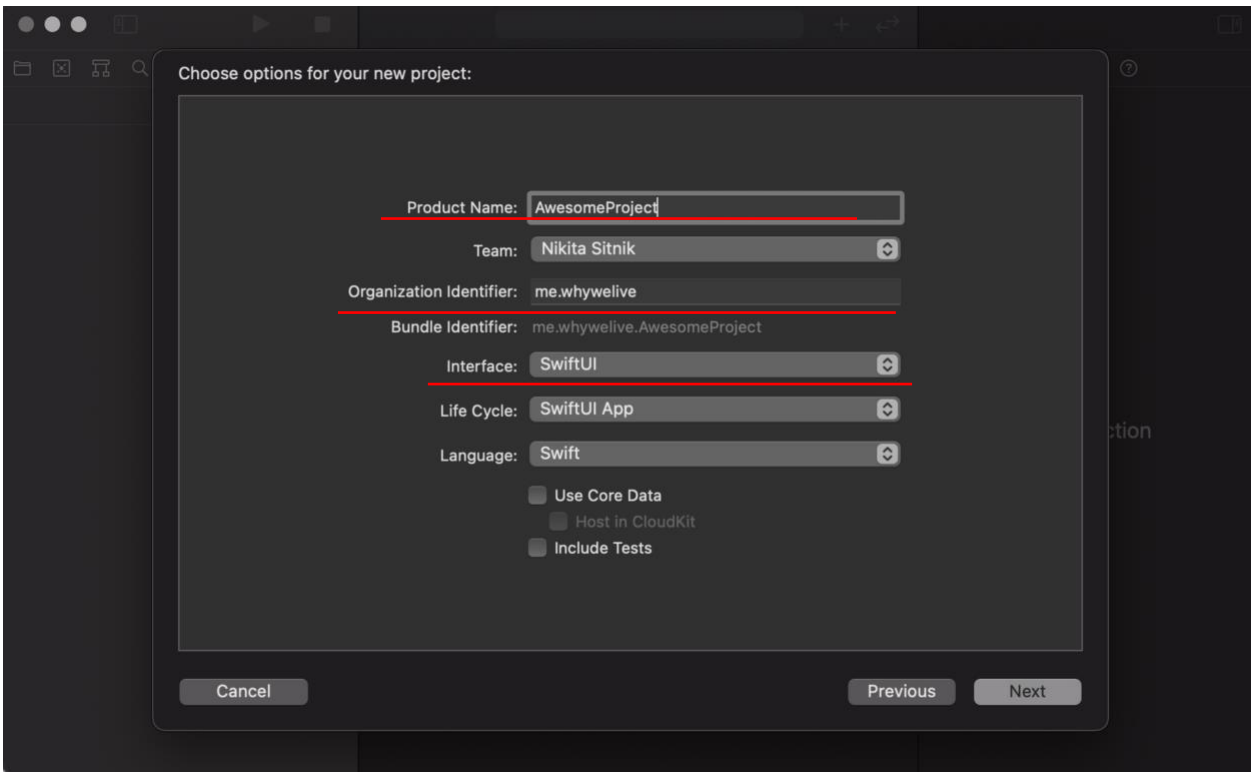
В главном меню **File - New - New Project** (с помощью горячих клавиш **⇧-⌘-N.**)

Теперь нам нужно произвести начальную настройку нашего нового проекта: указать название, используемый набор компонентов.

Во всплывшем окне выбираем вкладку **iOS** и шаблон **App.**



Next



В данном окне присутствует несколько параметров, на которые стоит обратить внимание.

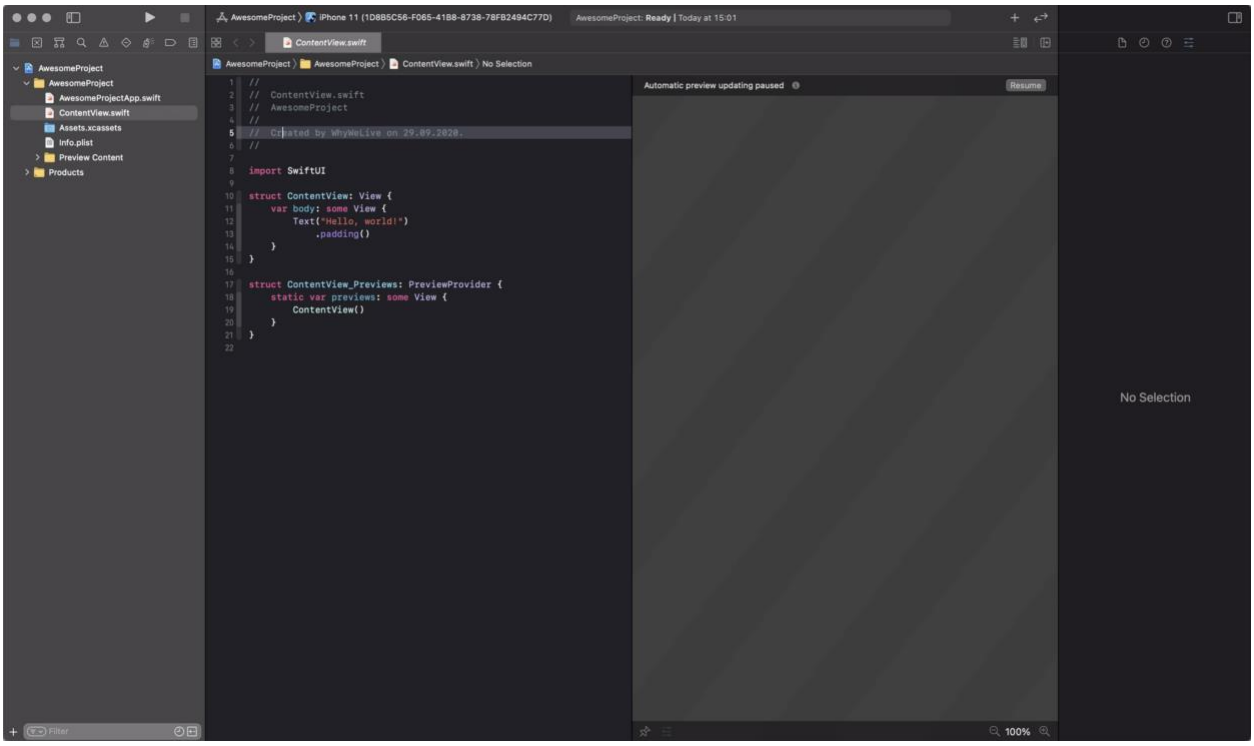
1. **Product Name** — это название вашего нового проекта; каждый раз стоит указывать уникальное название, чтобы в дальнейшем можно было легко различать их между собой;
2. **Organization Identifier** — уникальный идентификатор вашей организации, помните про то, что оно должно быть уникальным. Вам можно использовать любой другой, например, `ru.имяфамилия` — замените `имяфамилия` на ваше реальное имя и фамилию, написанные на английском языке;

3. **Interface / Life Cycle** — это набор компонентов для создания нашего интерфейса, в данном курсе мы будем использовать современный **SwiftUI**.

Next

Теперь выбираем место на нашем компьютере, чтобы туда поместить новый проект. Если вы решили разместить его на рабочем столе, то так делать не надо — лучше выберите, например, **Documents** и нажмите Create

Вы попали на экран, в котором мы будем разрабатывать приложение, но пока что давайте ознакомимся с его компонентами:



- Левая боковая панель используется для отображения файловой структуры проекта, а правая — для некоторых параметров, пока ее закроем, нажав на кнопку в правом верхнем углу окна;
- В центральной части окна находятся самые главные инструменты для разработки приложения: редактор кода и Canvas (Preview) — инструмент, который позволяет вам запустить ваше приложение прямо в **Xcode** (чтобы его активировать, нужно нажать на кнопку **Resume** в его верхней части);
- А верхняя панель в свою очередь позволяет вам, например, запустить уже готовое приложение на выбранном симуляторе или посмотреть на его статус.

Что такое View и модификаторы?

Это компонент, который представляет собой часть пользовательского интерфейса вашего приложения и предоставляет модификаторы, которые вы используете для настройки представлений. Вы можете создавать свои компоненты, комбинируя несколько готовых, — например, Text, используемый в данном примере; также любые другие пользовательские компоненты, которые вы создадите.

```
struct MyView: View {
    var body: some View {
        Text("Hello, World!")
    }
}
```

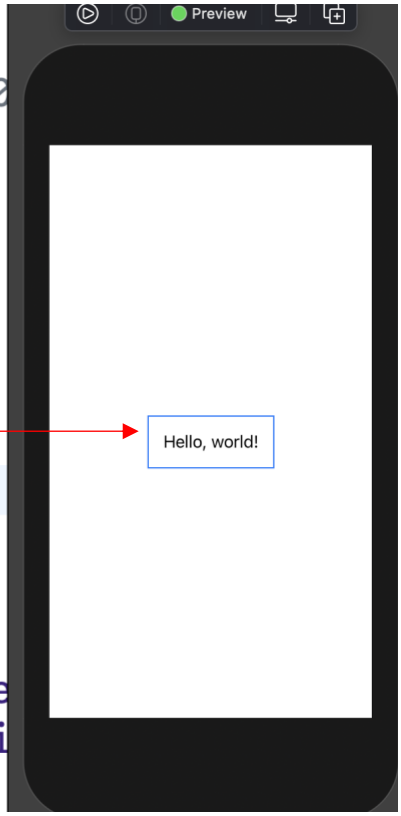
При создании нового проекта у вас заготовлена одна структура, в который мы и будем сначала работать.

```
1  //
2  //  ContentView.swift
3  //  Lesson1
4  //
5  //  Created by admin on 24.10.2020.
6  //
7
8  import SwiftUI
9
10 struct ContentView: View {
11     var body: some View {
12         Text("Hello, world!")
13             .padding()
14     }
15 }
16
17 struct ContentView_Previews: PreviewProvider {
18     static var previews: some View {
19         ContentView()
20     }
21 }
22
```

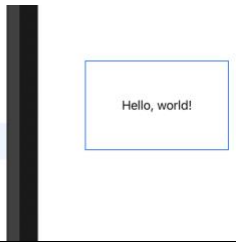
Компонент Text

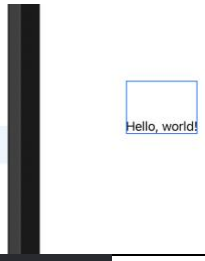
Компонент **Text** отрисовывает строку в интерфейсе приложения, используя при этом шрифт, который вы укажите, в противном случае стандартный шрифт данной платформы.

```
//
//  Created by admin on 24.10.20
//

import SwiftUI

struct ContentView: View {
    var body: some View {
        Text("Hello, world!")
            .padding()
    }
}

struct ContentView_Previews: Pre
    static var previews: some Vi
        ContentView()
```



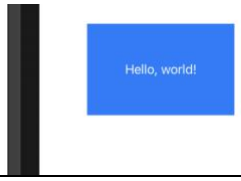
Модификаторы

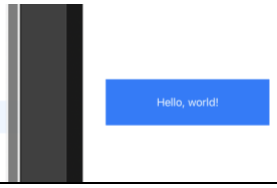
.padding(50)	Отступ по всем краям
	`struct ContentView: View {`
	`var body: some View {`
	`Text("Hello, world!")`
	`.padding(50)`
	`}`
	`}`
	 

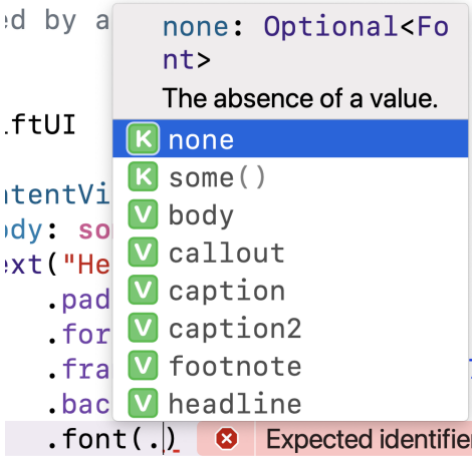
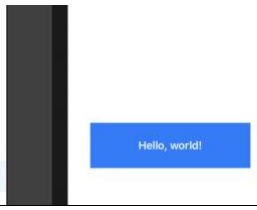
.padding(.top,50)	Отступ только по верхнему краю
	`struct ContentView: View {`
	`var body: some View {`
	`Text("Hello, world!")`
	`.padding(.top,50)`
	`}`
	`}`
	 

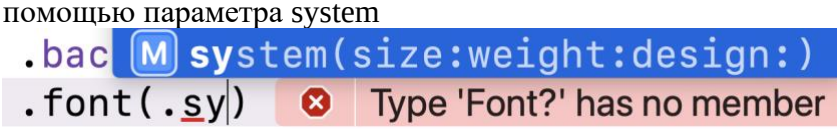
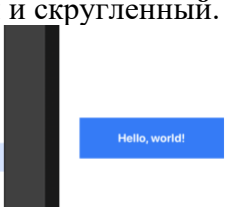
```
Text("Hello,world!")
    .padding()
    .padding(.top, 50) //сверху
    .padding(.bottom, 70) //снизу
    .padding(.leading, 60) //слева
    .padding(.trailing, 10)//справа
    .padding(.horizontal,10) //слева и справа
    .padding(.vertical, 20)//сверху и снизу
```

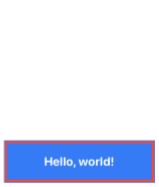
.foregroundColor(.red)	Цвет текста
	`Text("Hello, world!")`
	`.padding(.top,50)`
	`.foregroundColor(.red)`

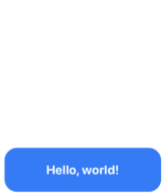
.background(Color.blue)	Цвет фона у компонента
	`Text("Hello, world!")`
	`.padding(50)`
	`.foregroundColor(.white)`
	`.background(Color.blue)`
	 

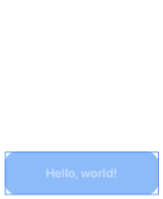
.frame(width:200, height:300)	Размеры компонента (рамка)
	`Text("Hello, world!")`
	`.padding(50)`
	`.foregroundColor(.white)`
	`.frame(width:250, height:70)`
	`.background(Color.blue)`
	 

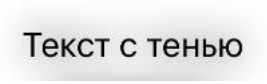
.font(.footnote)	Можно задать стиль шрифта. В модификаторе .font в параметрах после точки выбирается нужный стиль.
	 
	`Text("Hello, world!")`
	`.padding(50)`
	`.foregroundColor(.white)`
	`.frame(width:250, height:70)`
	`.background(Color.blue)`
	`.font(.headline)`
	 

.font(.system(
size:20,
weight:.light,
design:
rounded))
	Можно задать размер шрифта, стиль его начертания с помощью параметра system
	 
	Например размер шрифта 20, полужирный и скругленный.
	`Text("Hello, world!")`
	`.foregroundColor(.white)`
	`.frame(width:250, height:70)`
	`.background(Color.blue)`
	`.font(.system(size: 20, weight: .bold,`
	`design: .rounded))`
	 
	Здесь задаем тонкое начертание шрифта и шрифт семейства serif
	`Text("Hello, world!")`
	`.foregroundColor(.white)`
	`.frame(width:250, height:70)`
	`.background(Color.blue)`
	`.font(.system(size: 20, weight: .light,`
	`design: .serif))`

.border(Color.red, width: 5)	Цвет рамки и толщина рамки
	`Text("Hello, world!")`
	`.foregroundColor(.white)`
	`.frame(width:250, height:70)`
	`.background(Color.blue)`
	`.font(.system(size: 20, weight: .bol`
	`design: .rounded))`
	`.border(Color.red, width: 5)`
	 

.cornerRadius(8)	Радиус скругления
	`Text("Hello, world!")`
	`.foregroundColor(.white)`
	`.frame(width:250, height:70)`
	`.background(Color.blue)`
	`.font(.system(size: 20, weight: .bo`
	`design: .rounded))`
	`.cornerRadius(20)`
	 

.opacity(0.5)	прозрачность компонента
	`Text("Hello, world!")`
	`.foregroundColor(.white)`
	`.frame(width:250, height:70)`
	`.background(Color.blue)`
	`.font(.system(size: 20, weight: .bo`
	`design: .rounded))`
	`.cornerRadius(20)`
	`.opacity(0.5)`
	 

.shadow(color: Color.black, radius: 10, x: 0, y: 10)	Text("Текст с тенью") .shadow(color: Color.black, radius: 10, x: 0, y: 0) 

Стэки

Это набор контейнеров, который помогает нам выстраивать компоненты определенным образом.

```
VStack {
    Text("Hello, world!")
    Text("Hello, world!")
}
```

VStack — выстраивает все дочерние компоненты вертикально, по умолчанию выравнивает относительно центра.

```
HStack {
    Text("Hello, world!")
    Text("Hello, world!")
}
```

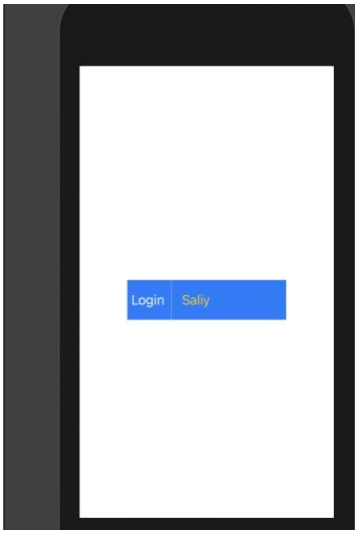
HStack — выстраивает все дочерние компоненты горизонтально, по умолчанию выравнивает относительно центра.

```
ZStack {
    Text("Hello, _____!")
    Text("_____, world!")
}
```

HStack — выстраивает все дочерние компоненты поверх друг друга, выравнивая их по обоим осям.

Пример: горизонтальный стек с разделителем

```
HStack{
    Text("Login").padding(.leading,5)
    Divider()
        .background(Color.white)
    Text("Saliy")
        .padding(.leading,5)
        .foregroundColor(.yellow)
}
.frame(width: 200,
       height: 50,
       alignment: .leading)
.background(Color.blue)
.foregroundColor(.white)
```



В примере показано как цвет фона и текста можно задать сразу у стека, и они будут действовать на все элементы внутри стека.

Помимо горизонтального стека здесь используется разделитель Divider(), у которого можно также изменить цвет.

Выравнивание в стеке

По умолчанию элементы внутри VStack центрированы. Выполним выравнивание их по левой стороне. Для этого мы можем добавить круглые скобки сзади VStack и использовать аргумент «alignment», чтобы изменить режим выравнивания элементов, заключенных в VStack. Для выравнивания видов с левой стороны мы используем опцию .leading. Мы также увеличиваем вертикальный интервал текстов, используя параметр «spacing».

```
VStack(alignment: .leading, spacing: 10) {
//…
}
.frame(width: 300, height: 150)
```

Spacer

Это компонент, который расширяет родительский компонент вдоль его основной оси, например стека, или по обеим осям, если компонент не содержится в стеке.

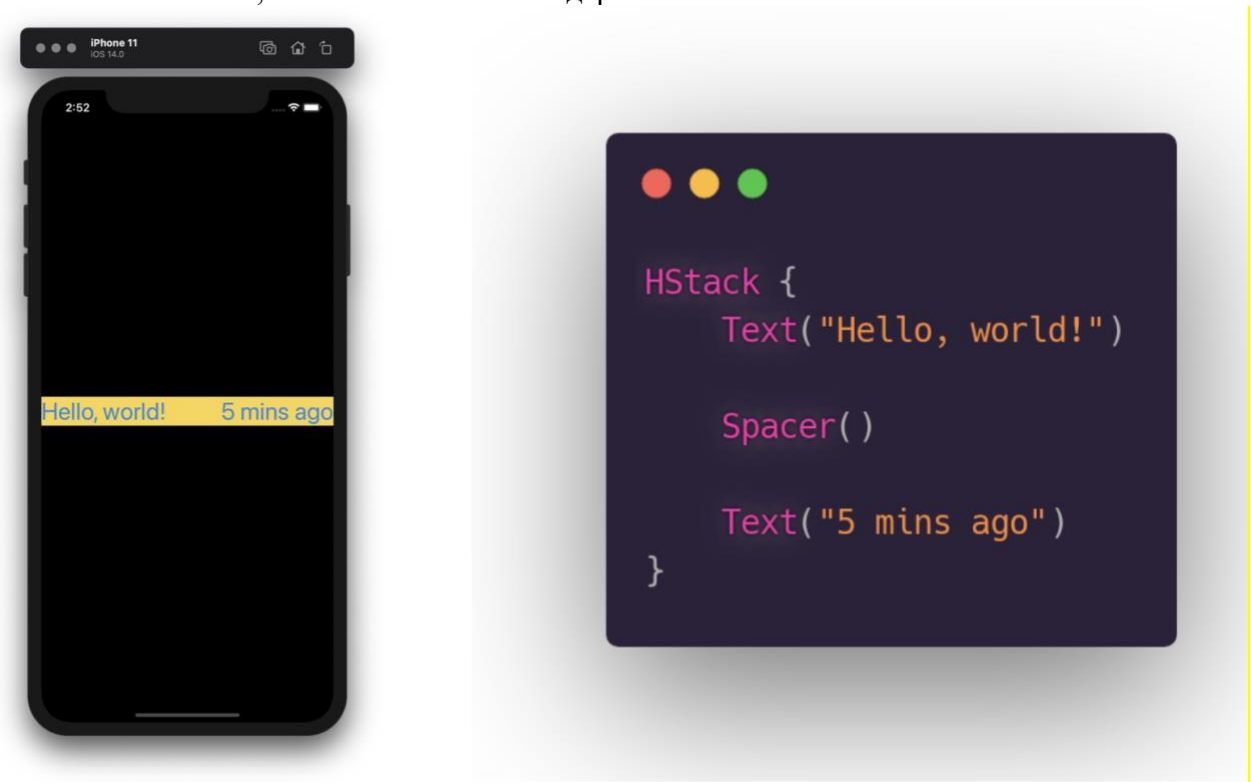
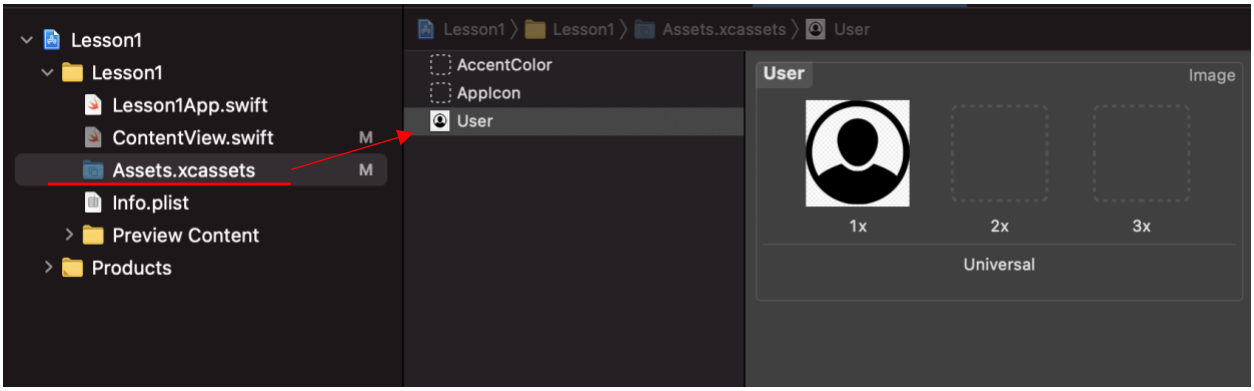


Image.

Картинка из ресурсов

Чтобы загрузить любую картинку в Image необходимо сначала загрузить ее в ресурсы проекта. Для этого в левой части в структуре проекта найдите папку Assets и перетяните из папки нужное вам изображение



Дальше мы можем использовать картинку из ресурсов по ее имени. Для задания размеров используется модификатор .resizable() и .frame

```
Image("User")
    .resizable()
    .frame(width: 30, height: 30)
    .padding(10)
```

В новых версиях XCode к имени картинки из русурсов можно обратится через точку:

```
Image(.user)
```

Для изменения размера изображения используется модификатор ***resizable***.
По умолчанию изображение изменяет размер с помощью метода ***skretch***(растягивания). Это означает, что исходное изображение будет масштабироваться для заполнения всего экрана (за исключением верхней и нижней областей)

Изображение на всю область экрана

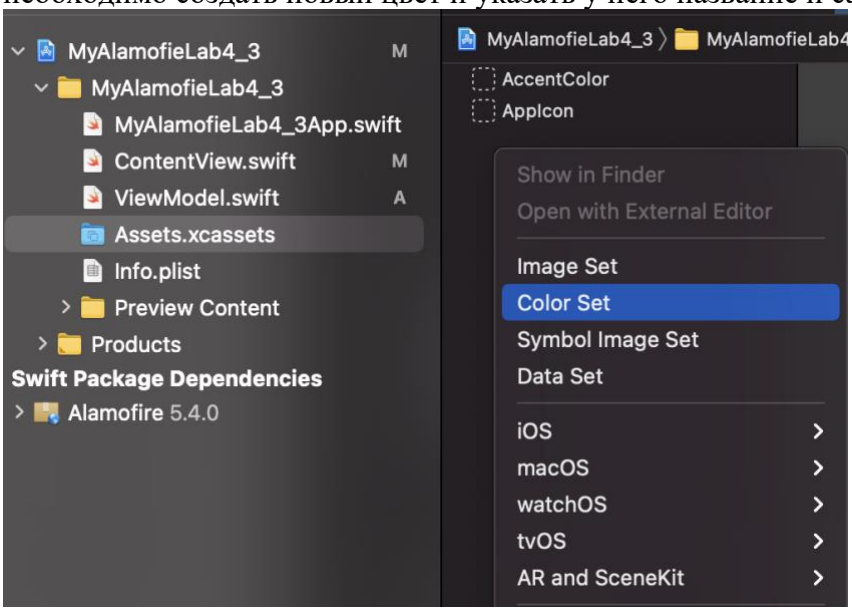
Технически говоря, изображение заполняет всю безопасную область в соответствии с определением iOS. Понятие *безопасная зона (safe area)* существует довольно долгое время, что определяет область просмотра, которая безопасна для размещения нашего ***UI*** компонента. Безопасная область — это область просмотра, которая исключает верхнюю панель (т.е. status bar) и нижнюю. С помощью области безопасности можно предотвратить случайное скрытие компонента *UI*, например, панели состояния, навигационной панели и панели табуляции.
При этом, если вы хотите отобразить изображение в полноэкранном режиме, вы можете игнорировать безопасную область, установив модификатор *edgesIgnoringSafeArea*.

```
struct ContentView: View {
    var body: some View {
      Image("Енот")
       .resizable()
       .edgesIgnoringSafeArea(.all)
    }
}
```

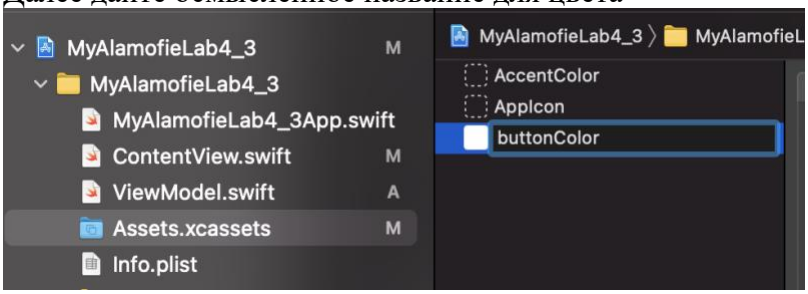
Вы можете игнорировать безопасную зону для определенного края. Например, чтобы игнорировать безопасную область для верхнего края, можно передать в качестве параметра *.top*. В примере *.all*, что означает игнорирование безопасной области для всех границ.

Нестандартный цвет для компонента

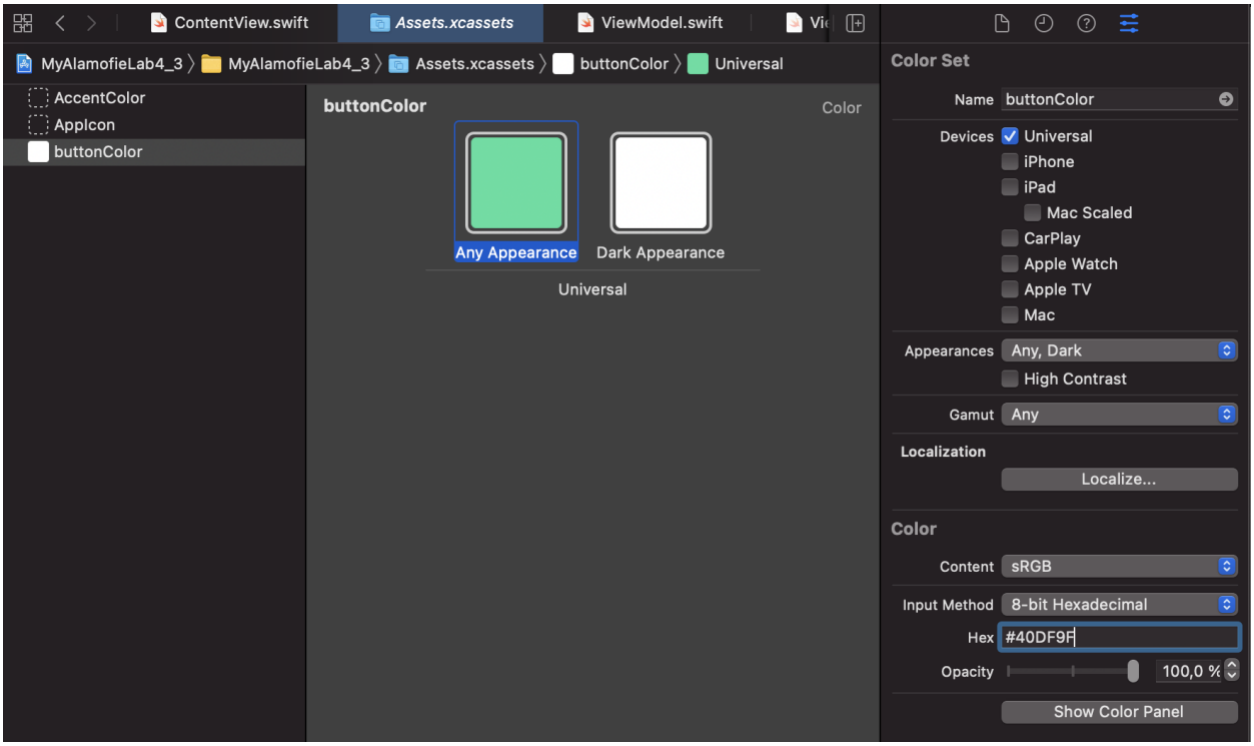
Если вам необходимо задать нестандартный цвет фона или текста изображения, в папке Assets необходимо создать новый цвет и указать у него название и сам цвет в шестнадцатеричном



Далее дайте осмысленное название для цвета



Далее необходимо задать сам цвет. Для этого щелкните AnyAppearence и правом окне параметров выберите **Input Method** = 8-bit Hexadecimal и введите код цвета в шестнадцатеричном формате в поле **Hex** (или Input Method = 8-bit (0-255) и введите код цвета в RGB формате).



В файле ContentView создадим расширение для цвета extension и добавим в него статичную константу buttonColor

```
extension Color{

    static let buttonColor = Color("buttonColor")

}
```

Вы создали свой цвет и добавили его в копилку к Color. Теперь надо его применить, например в модификаторе .background к любому компоненту указать название созданного цвета.

```
.background(Color.buttonColor)
```

В новых версиях XCode достаточно просто создать цветовую константу и обратится к ней также, как к обычному системному, через «.». Если в названии цвета есть слово Color в конце, то его нужно пропустить.

```
.background(.button)
```

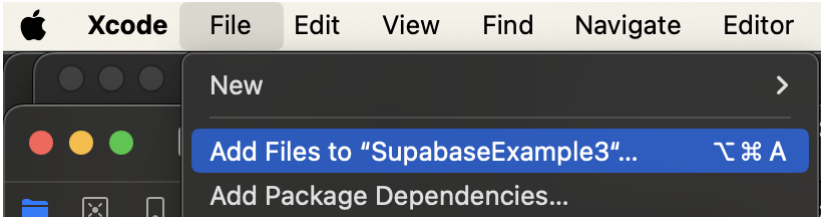
Заливка всего экрана нестандартным цветом .edgesIgnoringSafeArea()

Если нужен не стандартный цвет заливки, этот цвет нужно задать в Accets и в коде указать его так:

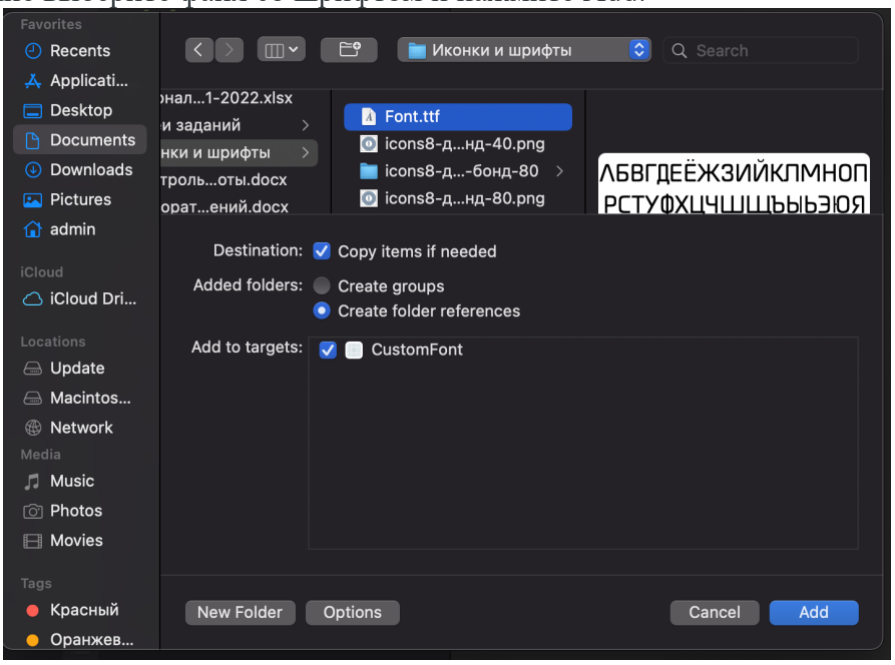
```
ZStack{

    Color.buttonColor.edgesIgnoringSafeArea(.all)

    // остальной код view


}
```

Кастомный шрифт

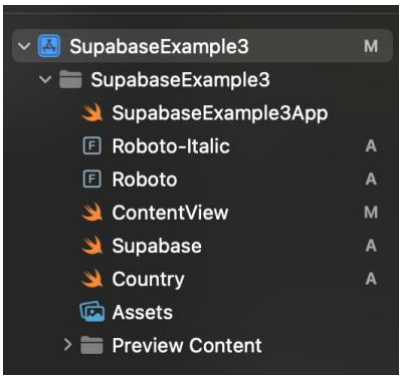
В главном меню XCode выберите пункт File – Add Files to “Project…”…



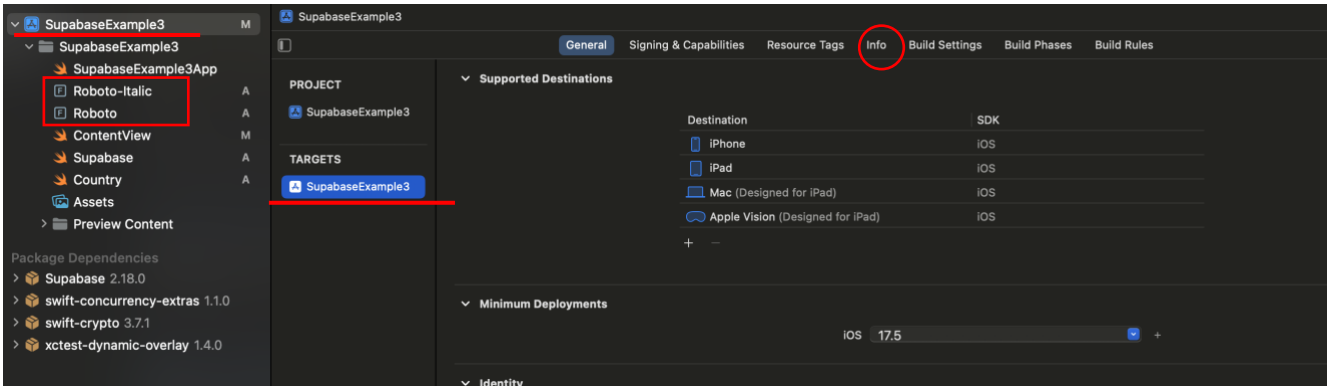
В открывшемся окне выберите файл со шрифтом и нажмите Add.



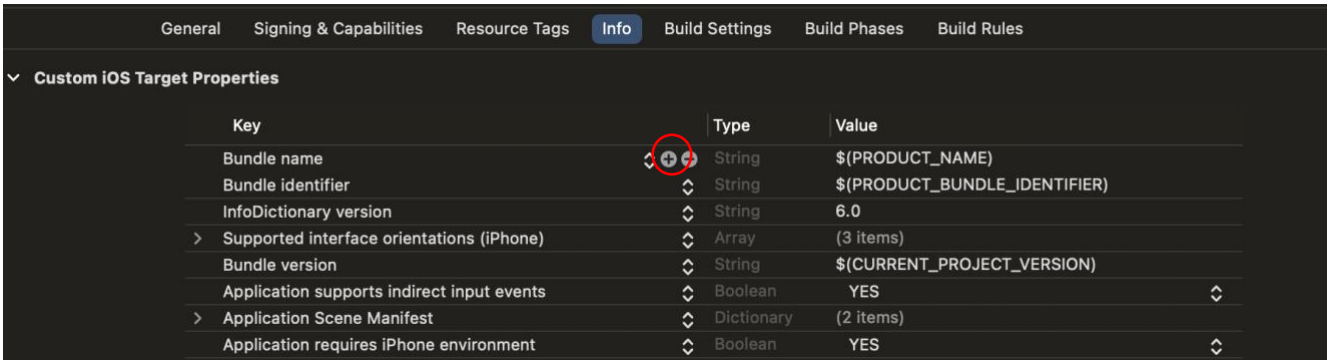
Шрифты добавятся в структуру проекта



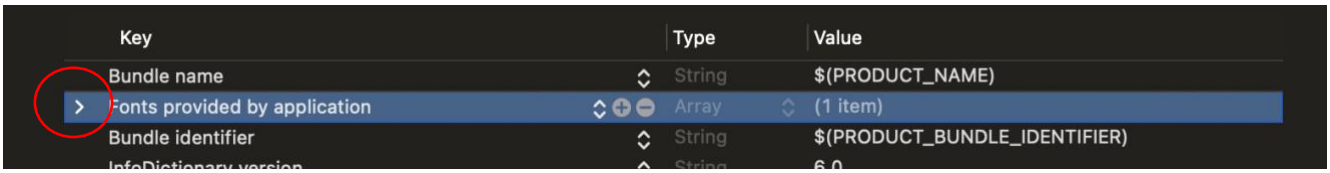
Чтобы использовать шрифт, надо внести его в info.plist проекта. Для этого щекните по названию проекта в структуре проекта. Откроется раздел с настройками. В нем откройте вкладку Info



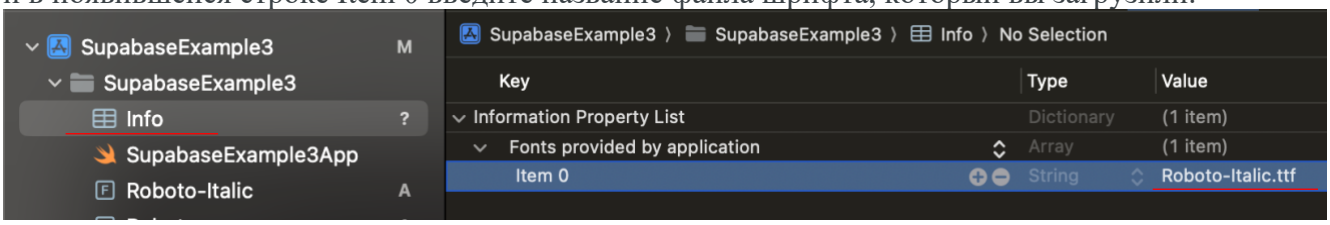
На вкладке Info щелкните по кнопке “+” , которая появляется при наведении на первую строку. Выберите пункт Fonts provider by application и нажмите Enter.



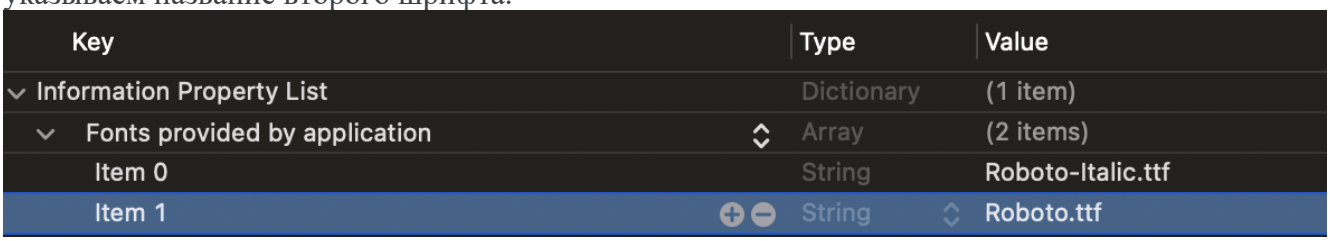
Далее в этом подпункте мы можем добавлять столько шрифтов, сколько надо. Для этого необходимо развернуть список Fonts provider by application (нажать на стрелочку слева)



и в появившейся строке Item 0 введите название файла шрифта, который вы загрузили.



Если надо добавить еще один шрифт, то в списке Fonts provider by application нажимаем “+” и указываем название второго шрифта.



Для использования шрифта в программе следует его указать явно:

```
Text("Привет, мир!")
  .font(.custom("Roboto", size: 33))
```

Элементы управления

Button

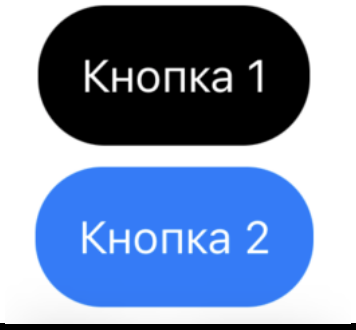
Это компонент, который выполняет определенное действие по нажатию.
Чтобы создать кнопку необходимо указать два параметра:

- **action** — это замыкание или отдельный метод, который будет выполнен, когда пользователь нажмет на эту кнопку. Код размещается внутри фигурных скобок.
- **label** — это View, которое будет отображено на нашем интерфейсе.

Два варианта оформления кнопки. Обратите внимание на поведение кнопки в процессе запуска программы.

```
Button("Кнопка 1")
{
  print("Действие кнопки 1")
```

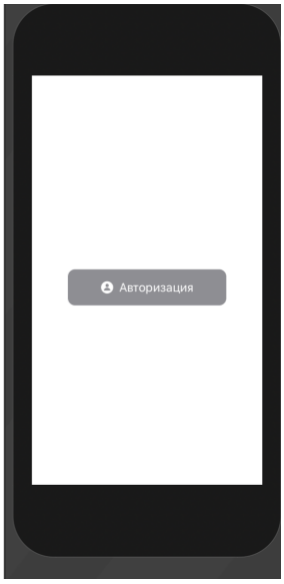
```
}
 .padding()
 .foregroundColor(.white)
 .background(RoundedRectangle(cornerRadius: 25.0))
```




```
Button(action:{print("Действие кнопки 2")})
{
  Text("Кнопка 2")
    .padding()
    .foregroundColor(.white)
    .background(RoundedRectangle(cornerRadius: 25.0))
}
```

Пример 1: кнопка со стандартной картинкой

```
struct ContentView: View {
    var body: some View {
        Button(action: {print("Hello")})
        {
            HStack{
                Image(systemName: "person.circle.fill")
                Text("Авторизация")
            }
            .frame(width: 200, height: 30)
            .foregroundColor(.white)
            .padding(10)
            .background(Color.gray)
            .cornerRadius(10)
        }
    }
}
```



@State – пишется перед объявлением переменной и позволяет менять значение переменной в рамках структуры. Структура отслеживает состояние переменной и перерисовывает экран при необходимости.

Переменные объявляются перед var body

TextField

- это простой элемент управления, отображающий редактируемый текстовый интерфейс.

Поскольку TextField позволяет пользователю вводить текст, ему также необходим способ сохранения введенного текста в переменной состояния, которую затем можно использовать для чтения ввода. Внешний вид TextField можно дополнительно настроить с помощью TextFieldStyle.

Чтобы инициализировать TextField, вам необходимо указать два параметра:

1. строку-заполнитель Placeholder, которая будет выведена в компонент до ввода текста.
2. привязка к переменной @State, которая будет хранить значение, введенное в TextField.

Пример 3:

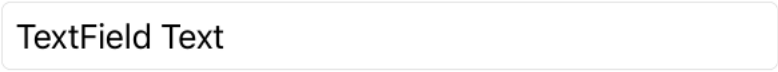
```
struct ContentView: View {
    @State var name: String = "Иван"
    var body: some View {
        VStack {
                Text("Привет, \(name)!")
            TextField("Введите ваше имя", text: $name)
        }
    }
}
```

Чтобы считать значение из поля TextField заводим переменную name. По умолчанию переменная имеет значение. Переменная должна быть обернута в оболочку свойств @State для того, чтобы она была изменяемой в рамках структуры.
В вертикальном стеке помещается 2 компонента. В текстовом поле будет выведено «Привет, Иван!»
При вводе текста в textField будет меняться значение переменной name и автоматически перерисовываться текст в компоненте выше.

Пример 4. Оформление с помощью .textFieldStyle

```
struct ContentView: View {
    @State var login: String = "TextField Text"
    var body: some View {
            TextField("Логин", text: $login)
            .textFieldStyle(RoundedBorderTextFieldStyle())
            .padding(.all, 20)
    }
}
```

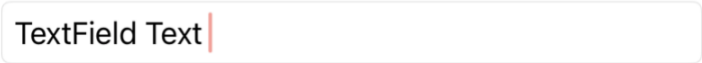
В результате textField будет обведен скругленной рамкой.



Пример 5. Выделение курсора цветом

```
struct ContentView: View {
   @State var login: String = "TextField Text"
   var body: some View {
         TextField("Логин", text: $login)
         .textFieldStyle(RoundedBorderTextFieldStyle())
         .padding(.all, 20)
                  .accentColor(.red)
   }
}
```

В результате при вводе текста курсор будет подсвечен красным цветом.



Пример 6. Фон со скруглением без .textFieldStyle

```
TextField("Логин", text: $login)
    .padding(20)
    .background(Color.red)
    .cornerRadius(10)
```

Пример 7. Выравнивание текста по центру

```
TextField("Placeholder Text", text: $text)
     .padding(.all, 20)
     .multilineTextAlignment(.center)
```

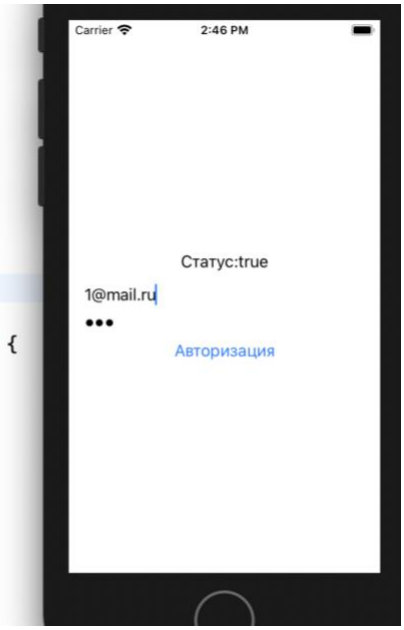
Пример 8. Нижний регистр в поле ввода

```
TextField("Placeholder Text", text: $text)
     .padding(.all, 20)
     .autocapitalization(.none)
```

SecureField.

Чтобы пароль не отображался при вводе, вместо TextField используется SecureField. Параметры идентичны.

```
struct ContentView: View {
    @State var login: String = ""
    @State var password: String=""
    @State var status: Bool = false

    var body: some View {
        VStack{
            Text("Статус:\(status.description)")
            TextField("Логин", text: $login)
            SecureField("Пароль", text: $password)
            Button(action: {
                if login=="1@mail.ru" && password=="111" {
                    status = true
                }

            }) {
                Text("Авторизация")
            }
        }.padding()
    }
}
```



Открытие гиперссылки

```
Link("google.com", destination: URL(string: "https://google.com")!)

Link(destination: URL(string: "https://ya.ru")!) {Text("Yandex")}
```

Кастомные компоненты:

Кастомный TextField – нестандартный цвет placeholder

```
struct CustomTextField: View{
    let placeholder: String
    @Binding var text: String
    var body: some View{
        ZStack{
            TextField(placeholder, text: $text)
                .padding()
                .background(Color.black)
                .cornerRadius(10)
                .padding(1)
                .background(Color.gray)
                .cornerRadius(10)
                .foregroundColor(.gray)

            if text == ""{
                HStack{
                    Text(placeholder)
                        .padding()
```

```
                .padding(.horizontal, 16)
                .foregroundColor(.gray)
            Spacer()
        }
    }
  }
}
```

Вызов кастомного компонента

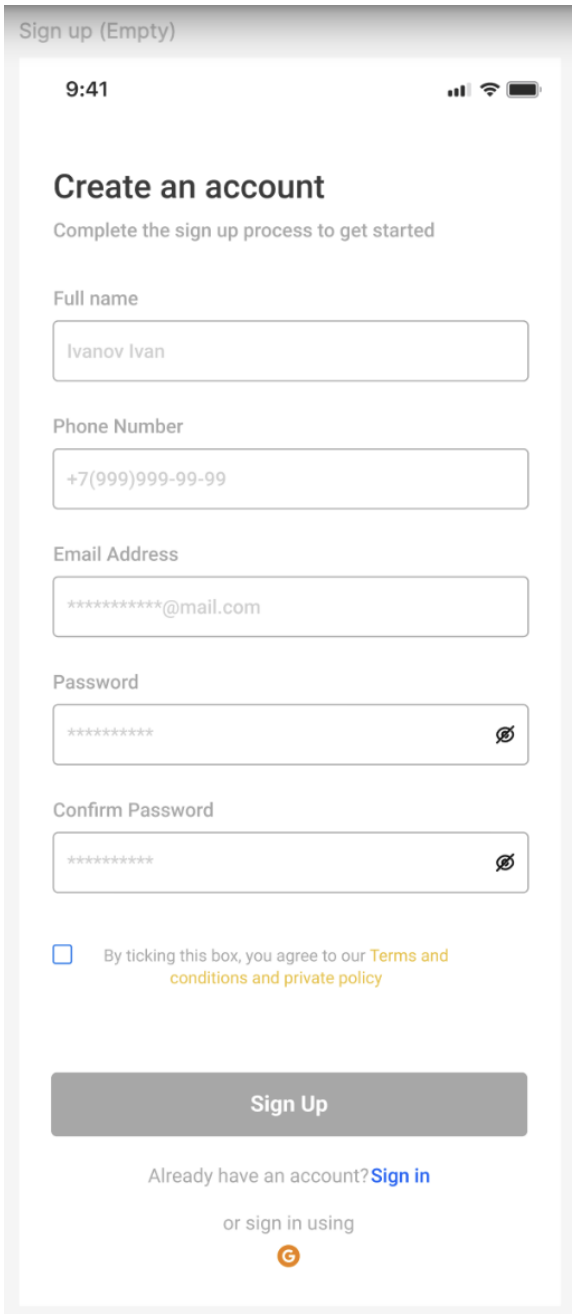
```
@State var name: String = “”
……
CustomTextField(placeholder: "Имя", text: $name)
```

C**heckbox** – разместить лучше в отдельном файле в папке Common

```
struct CheckBox: View {
  @Binding var value: Bool
  var body: some View {
    Group {
      if value {
        RoundedRectangle(cornerRadius: 2)
          .fill(Color.blue)
          .frame(width: 20, height: 20)
          .overlay {
            if value {
              Image(systemName: "checkmark")
                .font(.system(size: 10))
                .foregroundColor(.white)
            }
          }
      } else {
        RoundedRectangle(cornerRadius: 2)
          .stroke(Color.accentColor, lineWidth: 2)
          .frame(width: 14, height: 14)
      }
    }
```

```
        .onTapGesture {
            self.value.toggle()
        }
    }
}
```

Использование компонента в основном экране

```
…
@State private var checkBox: Bool = false
…
CheckBox(value: $checkBox)
```

Задание: попробуем реализовать один-два экран из задания чемпионата. Ссылка на макет https://www.figma.com/design/guyXW190UgIGSwE2CCyafu/OECH-APP-Final?node-id=1-2595&node-type=canvas