

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждения образования «БЕЛОРУССКИЙ
ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий
Кафедра Информационных систем и технологий
Специальность 1-40 05 01 Информационные системы и технологии
Специализация 1-40 05 01-03 Информационные системы и технологии (издательско-полиграфический комплекс)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к дипломной работе:

Управление мобильным приложением взглядом и мимикой лица

Дипломник _____ Юшкевич И. Н.

Руководитель проекта _____ Скребель А. С., ассистент

Заведующий кафедрой _____ Смелов В. В., к.т.н., доцент

Консультант _____ Семенова Л. С., преп.-стаж.

Нормоконтролер _____ Николайчук А. Н., преп.-стаж.

Дипломный проект защищен с оценкой _____

Председатель ГЭК _____ Дюбков В. К., к.т.н., доцент

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ
ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационных систем и технологий

Специальность 1-40 05 01 Информационные системы и технологии

Специализация 1-40 05 01-03 Информационные системы и технологии (издательско-полиграфический комплекс)

УТВЕРЖДАЮ

Заведующий кафедрой

_____ В.В. Смелов

« ____ » _____ 2023 г.

**ЗАДАНИЕ
на дипломную работу**

Юшкевич Илья Николаевич

(фамилия, имя, отчество)

1. Тема проекта: Управление мобильным приложением взглядом и мимикой лица

2. Тема утверждена приказом по университету от 17.03.2023 № 77-С

3. Срок сдачи студентом законченного проекта: 05.06.2023 г.

4. Исходные данные к проекту (требования к системе):

4.1 Назначение: библиотека, реализующая распознавание лица, взгляда и мимики с целью управления мобильным приложением. Библиотека предоставляет удобный интерфейс для внедрения распознавания, а также утилиты для отображения взгляда на экране и возможности управления приложением.

4.2 Основные функциональные возможности: распознавание лица и взгляда пользователя, проецирование взгляда на экран с помощью курсора, распознавание мимики лица для выполнения действий в приложении, адаптация жестов операционной системы для управления взглядом.

4.3 Пользовательские роли: Программист, реализующий библиотеку в своем приложении.

4.4 Целевая аудитория: люди с ограниченными возможностями.

4.5 Программная платформа: Swift 5, ARKit, CocoaPods, UIKit, XCFramework, Foundation, iOS 14+, iPhone X+, Alamofire, Charts, Firebase, SwiftGen, Swinject.

5. Содержание расчетно-пояснительной записки:

5.1 Реферат

5.2 Содержание

5.3 Введение

5.4 Раздел 1: постановка задачи и анализ аналогичных решений

5.5 Раздел 2: проектирование библиотеки

5.6 Раздел 3: реализация библиотеки

5.7 Раздел 4: тестирование библиотеки

5.8 Раздел 5: руководство программиста

5.9 Раздел 6: экономический раздел

5.10 Заключение

5.11 Список использованных источников

5.12 Приложения и графическая часть

6. Перечень графического материала (с точным указанием обязательных чертежей):

6.1 Диаграмма классов

6.2 API сессии дополненной реальности

6.3 API отслеживания мимики лица

6.4 API отслеживания взгляда

6.5 API распознавания речи

7. Консультанты по проекту с указанием относящихся к ним разделов проекта

Раздел	Консультант
Экономический раздел	Л. С. Семенова

8. Дата выдачи задания: _____

Руководитель _____ А.С. Скребель
(подпись)

Задание принял к исполнению _____ И.Н. Юшкевич
(подпись)

9. Календарный план

№ п/п	Наименование этапов дипломного проекта	Срок выполнения этапов проекта	Примечание
1	Постановка задачи и анализ аналогичных решений		
2	Проектирование библиотеки		
3	Реализация библиотеке		
4	Тестирование библиотеки		
5	Экономический раздел		
6	Рецензирование дипломной работы		

Дипломник _____ Руководитель проекта _____
(подпись) (подпись)

Реферат

Пояснительная записка содержит 76 страниц, 18 рисунков, 21 таблицу, 8 приложений.

БИБЛИОТЕКА ДЛЯ УПРАВЛЕНИЯ С ПОМОЩЬЮ ВЗГЛЯДА И МИМИКИ ЛИЦА, SWIFT, UIKIT, XCODE, COCOAPODS, AVFOUNDATION, FOUNDATION, ARKIT.

Целью данного дипломного проекта является разработка библиотеки для управления с помощью взгляда и мимики лица.

Пояснительная записка состоит из введения, семи разделов и заключения.

В первом разделе дипломного проекта приведен аналитический обзор предметной области и постановка задачи по теме дипломного проекта.

Во втором разделе описан выбор технологий для разработки приложения.

В третьем разделе описан процесс проектирования приложения.

В четвертом разделе описан процесс разработки и программной реализации библиотеки и тестового приложения.

В пятом разделе приведено руководство пользователя.

В шестом разделе описано тестирование библиотеки.

В седьмом разделе приводятся экономические обоснования разработанного программного модуля.

В заключении представлены итоги дипломного проекта и задачи, которые были решены в ходе разработки программного средства.

Графическая часть представлена в объеме 3 листов А3.

				БГТУ 00.00.ПЗ			
	ФИО	Подпись	Дата				
Разраб.	Юшкевич И. Н.			Реферат		Лит.	Лист
Пров.	Скребель А. С.						Листов
Н. контр.	Николайчук А.Н.						
УТВ.	Смелов В.В.						
				1111111, 2023			

Abstract

The explanatory note contains 76 pages, 18 figures, 21 tables, 8 appendices.

LIBRARY FOR CONTROL WITH THE HELP OF LOOK AND FACE MIMICS,
SWIFT, UIKIT, XCODE, COCOAPODS, AVFOUNDATION, FOUNDATION, ARKIT.

The purpose of this thesis project is to develop a library for control using gaze and facial expressions.

The explanatory note consists of an introduction, seven sections and a conclusion.

In the first section of the graduation project, an analytical review of the subject area and a problem statement on the topic of the graduation project are given.

The second section describes the choice of technologies for application development.

The third section describes the application design process.

The fourth section describes the development process and software implementation of the library and test application.

The fifth section contains the user manual.

The sixth section describes testing the library.

The seventh section provides economic justifications for the developed software module.

In conclusion, the results of the graduation project and the tasks that were solved during the development of the software are presented.

The graphic part is presented in the volume of 3 A3 sheets.

				БГТУ 00.00.ПЗ						
	ФИО	Подпись	Дата							
Разраб.	Юшкевич И.Н.			Abstract	Лит.		Лист	Листов		
Провер.	Скребель А.С..						1	1		
					11111111, 2023					
Н. контр.	Николайчук А.Н.									
Утв.	Смелов В.В.									

Содержание

Содержание.....	5
Введение.....	7
1 Постановка задачи и анализ аналогичных решений.....	8
1.1 Анализ и сравнительный обзор аналогов	8
1.1.1 Библиотека OpenCV	8
1.1.2 Библиотека Dlib.....	10
1.1.3 Face++ SDK.....	12
1.2 Постановка задачи.....	13
1.3 Выводы по разделу.....	13
2 Выбор технологий для разработки	14
2.1 Выбор языка программирования	14
2.2 Выбор средств разработки	14
2.3 Фреймворк ARKit.....	15
2.5 Выбор технологий и библиотек	16
2.5.1 Фреймворк UIKit	16
2.5.2 Формат XCFramework.....	17
2.5.3 Firebase Auth	18
2.5.4 Библиотека Charts.....	18
2.5.5 Менеджер пакетов CocoaPods.....	19
2.6 Выводы по разделу.....	19
3 Проектирование программного средства	20
3.1 Проектирование библиотеки.....	20
3.2 Проектирование приложения.....	27
3.3 Выводы по разделу.....	27
4 Реализация библиотеки.....	28
4.1 Задачи библиотеки	28
4.2 Разработка AR сессии	28
4.3 Разработка распознавания лица	31
4.4 Разработка распознавания взгляда	32
4.5 Распознавание речи.....	37
4.6 Разработка менеджера	39
4.7 Разработка тестового приложения	40
4.8 Выводы по разделу.....	42

				<i>БГТУ 00.00.ПЗ</i>		
	<i>ФИО</i>	<i>Подпись</i>	<i>Дата</i>	<i>Содержание</i>		
Разраб.	Юшкевич И.Н.					
Провер.	Скребель А.С.					
Н. контр.	Николайчук А.Н.					
УТВ.	Смелов В.В.			<i>11111111, 2023</i>		

5	Тестирование программного модуля	43
5.1	Ручное тестирование	43
5.2	Тестирование разрешений	44
5.3	Тестирование жестов системы	45
5.4	Тестирование входа по лицу	48
5.5	Выводы по разделу	49
6	Руководство пользователя	50
6.1	Общие сведения	50
6.2	Документация	50
6.3	Интеграция библиотеки	51
6.4	Отслеживание мимики	52
6.5	Отслеживание взгляда	53
6.6	Распознавание речи	54
6.7	Выводы по разделу	55
7	Технико-экономическое обоснование проекта	56
7.1	Общая характеристика разрабатываемого программного средства	56
7.2	Исходные данные для проведения расчетов и маркетинговый анализ	56
7.3	Обоснование цены программного средства	57
7.3.1	Расчёт затрат рабочего времени на разработку программного средства	58
7.3.2	Расчёт основной заработной платы	58
7.3.2	Расчёт дополнительной заработной платы	59
7.3.3	Отчисления в Фонд социальной защиты населения и Белгосстрах	59
7.3.4	Расчёт суммы прочих прямых затрат	59
7.3.4	Расчёт суммы накладных расходов	60
7.3.4	Сумма расходов на разработку программного средства	60
7.3.5	Расходы на реализацию	60
7.3.5	Расчет полной себестоимости	61
7.3.6	Определение цены, оценка эффективности	61
7.4	Вывод по разделу	63
	Заключение	64
	Список использованных источников	65
	ПРИЛОЖЕНИЕ А	67
	ПРИЛОЖЕНИЕ Б	69
	ПРИЛОЖЕНИЕ В	70
	ПРИЛОЖЕНИЕ Д	71
	ПРИЛОЖЕНИЕ Е	73
	ПРИЛОЖЕНИЕ Ж	74
	ПРИЛОЖЕНИЕ З	75

Введение

В современном мире компьютеры и мобильные устройства являются неотъемлемой частью повседневной жизни. Каждый день мы пользуемся различными приложениями для работы, общения и развлечения. Однако, не каждый человек может пользоваться мобильными устройствами с помощью рук из-за различных заболеваний.

В связи с этим существует необходимость в создании способа управления устройством без использования рук. Одним из таких способов является управление приложением с помощью взгляда и мимики лица. Эта технология основана на использовании камер и датчиков, которые позволяют определять положение глаз, узнавать выражение лица и определять направление взгляда. Благодаря этому пользователи могут управлять приложениями без непосредственного физического контакта с устройством.

Управление приложением с помощью взгляда и мимики лица уже нашло свое применение в некоторых областях, таких как автомобильная промышленность, медицина, развлечения и технологии виртуальной реальности. Однако данный подход еще не был реализован в самом популярном устройстве – смартфоне.

В данной дипломной работе будет реализована библиотека, встраивание которой в приложение позволит взаимодействовать с ним с помощью взгляда и мимики лица.

Для достижения поставленной цели необходимо решить следующие задачи:

- Выполнить анализ существующих аналогов;
- Произвести выбор инструментария и технологии проектирования;
- Выполнить проектирование структуры библиотеки;
- Выполнить программную реализацию;
- Провести тестирование библиотеки;
- Рассчитать экономические показатели.

Работа будет включать в себя анализ существующих решений и разработку новой библиотеки.

Результаты данной работы будут полезны в обществе, так как у людей с ограниченными возможностями появится возможность использовать мобильные устройства без использования рук.

				БГТУ 00.00.ПЗ							
	ФИО	Подпись	Дата								
Разраб.	Юшкевич И.Н.			Введение	Лит.			Лист		Листов	
Провер.	Скребель А.С.							1		1	
Н. контр.	Николайчук А.Н.				11111111, 2023						
Утв.	Смелов В.В.										

1 Постановка задачи и анализ аналогичных решений

1.1 Анализ и сравнительный обзор аналогов

Для реализации библиотеки для управления приложением с помощью взгляда и мимики лица, необходимо рассмотреть аналоги для обнаружения их достоинств и недостатков с целью улучшения качества продукта и его конкурентоспособности на рынке.

1.1.1 Библиотека OpenCV

OpenCV – это библиотека с открытым исходным кодом для обработки изображений и компьютерного зрения, которая предлагает множество функций для работы с изображениями и видео, а также функции для обработки и распознавания объектов. Она предоставляет простой интерфейс для использования и может работать на всех основных операционных системах.

Достоинства OpenCV:

- OpenCV содержит более 2500 алгоритмов компьютерного зрения и обработки изображений, что позволяет решать различные задачи.

- OpenCV имеет большое сообщество пользователей и разработчиков, что обеспечивает обширный набор документации и множество примеров использования.

- OpenCV имеет открытый исходный код и распространяется по лицензии BSD, что позволяет использовать библиотеку в коммерческих проектах и изменять ее код.

Недостатки OpenCV:

- OpenCV не оптимизирована для большинства языков программирования, она наиболее эффективна в использовании с языками C++ и Python.

- Для работы с OpenCV необходимо иметь базовые знания программирования и обработки изображений.

				БГТУ 00.00.ПЗ			
	ФИО	Подпись	Дата				
Разраб.	Юшкевич И.Н.			1 Постановка задачи и анализ аналогичных решений	Лит.	Лист	Листов
Провер.	Скребель А.С.					1	6
Н. контр.	Николайчук А.Н.						
Утв.	Смелов В.В.				11111111, 2023		

Примеры использования OpenCV:

- Создание систем распознавания лиц, автоматического фокусирования камеры и стабилизации изображения.
- Разработка системы машинного зрения для контроля качества на производстве.
- Распознавание дорожных знаков и лицензионных номеров на автомобилях.
- Разработка системы управления роботами на основе компьютерного зрения.
- Разработка игр с использованием распознавания жестов и лиц.

Для реализации управления с помощью взгляда и жестов лица с помощью OpenCV необходимо выполнить несколько пунктов.

Для начала, мы можем использовать OpenCV для распознавания лица пользователя в кадре с камеры. Это можно сделать при помощи алгоритмов Haar Cascade или Local Binary Patterns (LBP), которые содержатся в OpenCV.

На листинге 1.1 приведена функция распознавания лица с помощью алгоритма Haar Cascade на языке программирования Python.

```
def detect_face(gray):  
    face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')  
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)  
    if len(faces) == 0:  
        return None, None  
    (x, y, w, h) = faces[0]  
    return gray[y:y+w, x:x+h], faces[0]
```

Листинг 1.1 – Функция распознавания лица в OpenCV

Затем необходимо распознавать взгляд пользователя. Для этого можно использовать алгоритм Eye Gaze, который позволяет определить точку фокусировки глаз пользователя. Для этого мы можем использовать алгоритм Haar Cascade в функции распознавания взгляда, приведенной на листинге 1.2.

```
def detect_eyes(gray):  
    eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')  
    eyes = eye_cascade.detectMultiScale(gray, 1.3, 5)  
    if len(eyes) == 0:  
        return None, None  
    (x, y, w, h) = eyes[0]  
    return gray[y:y+w, x:x+h], eyes[0]
```

Листинг 1.2 – Функция распознавания взгляда в OpenCV

Для управления приложением на основе взгляда можно использовать библиотеку PyAutoGui для эмуляции кликов мыши в нужных областях экрана. PyAutoGui является модулем Python для автоматизации взаимодействия с графическим интерфейсом пользователя (GUI) на уровне системы. Он позволяет программно контролировать мышь и клавиатуру, а также предоставляет функции для работы с изображениями на экране. Например, если пользователь смотрит на кнопку, которая должна быть нажата, то мы можем эмулировать клик мышью на

этой кнопке. На листинге 1.3 приведен пример выполнения комбинации клавиш Command + Tab при улыбке пользователя.

```
smiles = cv2.CascadeClassifier('haarcascade_smile.xml')
smile = smiles.detectMultiScale(gray, scaleFactor=1.7, minNeighbors=20)
for (sx, sy, sw, sh) in smile:
    pyautogui.hotkey('command', 'tab')
```

Листинг 1.3 – Выполнение действий с помощью жестов лица в OpenCV

Полная реализация управления взглядом приведена в приложении А.

1.1.2 Библиотека Dlib

Dlib — одна из наиболее популярных библиотек с открытым исходным кодом для машинного обучения и компьютерного зрения, которая предназначена для разработки приложений в области распознавания объектов, нахождения лиц и оценки эмоционального состояния человека по изображению.

Достоинства Dlib:

- Dlib предоставляет множество реализаций алгоритмов машинного обучения для решения задач распознавания объектов и изображений, а также оценки эмоционального состояния.

- Библиотека содержит реализацию сверточных нейронных сетей (CNN), которые позволяют эффективно решать задачи распознавания объектов и часто используются в приложениях, использующих распознавание объектов.

- Dlib имеет простой и удобный интерфейс программирования для создания приложений для анализа изображений, которые могут быть использованы в различных сферах деятельности.

Недостатки Dlib:

- Dlib не так популярна, как OpenCV, и ее сообщество не такое обширное, что может затруднить доступность поддержки и обучения программистов.

- Большинство алгоритмов Dlib построены на машинном обучении, что означает необходимость иметь знания в этой области, чтобы использовать библиотеку.

Примеры использования Dlib:

- Распознавание объектов и лиц на изображениях.
- Оценка эмоционального состояния человека по изображению.
- Анализ видео для определения поведения объектов на экране.
- Распознавание глаз и определение точки взгляда.
- Определение возраста и пола по фотографии.

Один из примеров использования Dlib для управления приложением с помощью взгляда и жестов лица - это создание программы, которая управляет курсором мыши на основе положения головы пользователя.

Для начала мы можем использовать Dlib для распознавания лица пользователя на изображении. Для этого мы можем использовать предобученную модель детектора лица из библиотеки, как указано на листинге 1.4.

```
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
```

Листинг 1.4 – Инициализация модели распознавания лиц

Затем мы можем использовать модели из библиотеки Dlib для определения положения головы пользователя на основе обработки кадра видео. Например, модель 68 точек может использоваться для обнаружения ключевых точек на лице и определения положения глаз, носа и рта, что представлено на листинге 1.5.

```
# Detect faces in the grayscale frame
faces = detector(gray)
# If a face is detected
if len(faces) > 0:
    # Get the facial landmarks for the first face
    landmarks = predictor(gray, faces[0])
    # Get the position of the eyes
    left_eye = (landmarks.part(36).x, landmarks.part(36).y)
    right_eye = (landmarks.part(45).x, landmarks.part(45).y)
    # Calculate the center of the eyes
    eye_center = ((left_eye[0] + right_eye[0]) // 2, (left_eye[1] + right_eye[1]) //
2)
```

Листинг 1.5 – Распознавание лица и положения глаз

Далее мы можем использовать библиотеку PyAutoGui для управления курсором мыши в соответствии с положением головы пользователя. Например, если пользователь поворачивает голову вправо, мы можем двигать курсор вслед за его головой, а также при открытии рта мы можем выполнять действия. Пример такого использования библиотеки приведен на листинге 1.6.

```
# Get the position of the mouth
mouth = (landmarks.part(62).x, landmarks.part(62).y)
# If the mouth is open wide enough
if mouth[1] - eye_center[1] > 20:
    # Do something here, like simulate a mouse click
    # Increment the count
    count += 1
    # If the count reaches a certain value
    if count == 10:
        # Do something else here, like close the application
        # Reset the count
        count = 0
```

Листинг 1.6 – Выполнение клика при открытии рта

Пример реализации управления приложением с помощью Dlib приведен в приложении Б.

Таким образом, Dlib является отличной библиотекой для различных приложений в области компьютерного зрения и машинного обучения, особенно когда требуется решать задачи, связанные с анализом изображений, распознаванием объектов и оценкой эмоционального состояния.

1.1.3 Face++ SDK

Face++ SDK – это библиотека для распознавания и анализа лиц, разработанная компанией Megvii. Она используется разработчиками для создания приложений с функциями распознавания лиц, проверки подлинности.

Достоинства:

- Предоставляет широкий набор функций для анализа, включая распознавание лиц, анализ выражений лица, распознавание эмоций, определение возраста и пола, идентификацию и аутентификацию.
- Имеет хорошую точность распознавания.
- Хорошо документирован, с подробными инструкциями по использованию.
- Легко интегрируется с другими инструментами разработки благодаря различным API.

Недостатки:

- Платформа является коммерческой, поэтому для использования в продуктивной работе требуется оплата.
- Некоторые функции требуют обучения моделям, что может быть сложным и затратным процессом.

Примеры использования:

- Разработка системы наблюдения за посетителями, которая определяет их возраст, пол и настроение.
- Система полицейской безопасности, которая использует Face++ SDK для идентификации и аутентификации людей в общественных местах.

Пример использования Face++ SDK для управления приложением с помощью взгляда и жестов лица может выглядеть следующим образом.

Так как платформа является коммерческой, необходимо инициализировать библиотеку с помощью API ключа, как указано на листинге 1.7.

```
# Initialize Face++ API
api_key = '<your_api_key>'
api_secret = '<your_api_secret>'
face_api = facepp.API(api_key=api_key, api_secret=api_secret)
```

Листинг 1.7 – Инициализация библиотеки

Затем у программиста появляется возможность распознавать лицо, глаза а также нос пользователя, что отображено в листинге 1.8.

```
# Detect faces and eyes using Face++ API
result = face_api.detect(image_file=img, return_landmark=1)
# Get the position of the eyes and mouth
left_eye_pos = result['faces'][0]['landmark']['left_eye'][0]
right_eye_pos = result['faces'][0]['landmark']['right_eye'][0]
mouth_pos = result['faces'][0]['landmark']['mouth'][0]
# Calculate the center of the eyes and the position of the nose
eye_center = ((left_eye_pos['x'] + right_eye_pos['x']) // 2, (left_eye_pos['y'] +
right_eye_pos['y']) // 2)
nose_pos = result['faces'][0]['landmark']['nose'][0]
```

Листинг 1.8 – Распознавание лица, глаз и носа пользователя

В листинге 1.9 указано, как для управления приложением можно использовать положение рта и глаз, что позволяет выполнять действия на устройстве.

```
# Check if the mouth is open wide enough
if mouth_pos['y'] - eye_center[1] > 20:
    # Perform some action, like simulating a mouse click or closing the window
    pass
```

Листинг 1.9 – Распознавание положения рта и глаз для управления

Пример реализации управления приложением с помощью Face++ SDK приведен в приложении В.

1.2 Постановка задачи

В процессе поиска аналогов не было выявлено библиотек, которые бы позволили управлять мобильным приложением.

Целью данного дипломного проекта является создание нативной библиотеки для отслеживания взгляда для устройств под управлением ОС iOS.

При реализации дипломного проекта были поставлены следующие задачи:

- Реализация отслеживания взгляда пользователя;
- Реализация отслеживания мимики пользователя;
- Реализация проецирования взгляда пользователя на экран устройства;
- Регистрация мимики лица для отслеживания только необходимых жестов;
- Адаптация жестов операционной системы, например, прокрутка списков;

Библиотека создается с целью исследования возможностей датчиков и камеры TrueDepth, а также для помощи людям с ограниченными возможностями с использованием мобильных устройств под управлением iOS.

1.3 Выводы по разделу

В данном разделе были рассмотрены библиотеки, позволяющие реализовать управление приложением с помощью взгляда и мимики лица. Были выявлены их сильные и слабые стороны с целью улучшения качества и конкурентоспособности разрабатываемого продукта. Также в разделе был проведен анализ требований к разрабатываемому продукту. Исходя из результатов анализа, был сделан вывод о целесообразности разработки нативной библиотеки для устройств под управлением операционной системы iOS, а также были поставлены задачи, реализация которых является важной частью в написании дипломного проекта.

2 Выбор технологий для разработки

В данном подразделе будет обоснован выбор языка программирования, среды разработки и платформы, которые будут использованы в разработке приложения.

2.1 Выбор языка программирования

Для разработки библиотеки был выбран язык Swift, так как он является наиболее актуальным языком для разработки библиотек и приложений для iOS устройств. Его альтернативой может являться Objective-C, однако он не обладает необходимым AR инструментарием для реализации задач.

Swift – это язык программирования, разработанный Apple в 2014 году. Он был создан как замена Objective-C для разработки приложений для операционной системы iOS, macOS, watchOS и tvOS. Swift быстро получил популярность среди разработчиков благодаря своей безопасности, производительности, легкости использования и доступности.

Swift был спроектирован с учетом безопасности, что делает его защищенным от ошибок, связанных с утечкой памяти, за счет использования особого механизма работы с ней – Automatic Reference Counting.

Swift является более производительным языком, чем Objective-C. Производительность была достигнута путем использования экзистенциальных контейнеров и переноса основных типов данных на стек.

Swift можно легко интегрировать с существующим Objective-C кодом. Что достигается с помощью особых оберток над Swift классами. Это позволяет использовать производительный Swift код с такими преимуществами Objective-C, как, например, method swizzling.

2.2 Выбор средств разработки

Для реализации библиотеки для управления приложением с помощью взгляда и мимики лица была выбрана платформа iOS SDK, которая включает в себя все необходимые фреймворки для работы с компонентами пользовательского интерфейса, работы с микрофоном, а также немало важным является наличие возможностей для разработки приложений в AR.

				БГТУ 00.00.ПЗ							
	ФИО	Подпись	Дата								
Разраб.	Юшкевич И.Н.			2 Выбор технологий для разработки	Лит.			Лист		Листов	
Провер.	Скребель А.С.							1		6	
Н. контр.	Николайчук А.Н.				11111111, 2023						
Утв.	Смелов В.В.										

Одним из ключевых фреймворков, входящих в iOS SDK, является Foundation. Foundation – это фреймворк, предоставляющий основные инструменты для разработки приложений на платформах Apple. Он включает в себя API для работы с текстом, строками, датами и временем, коллекциями, файлами и сетью. Foundation также предоставляет набор базовых классов, таких как NSObject, NSData и NSNumber, которые являются основой для более сложных классов и API.

Некоторые ключевые компоненты фреймворка Foundation:

- NSString и NSMutableString: Эти классы предоставляют API для работы со строками и их манипуляции, включая поиск, замену, разделение строк и т.д.

- NSArray и NSMutableArray: Эти классы предоставляют API для работы с коллекциями объектов, включая сортировку, фильтрацию, поиск и манипуляции элементами.

- NSDate и NSCalendar: Эти классы предоставляют API для работы с датами и временем, включая форматирование, сравнение и арифметические операции.

- NSFileManager: Этот класс предоставляет API для работы с файловой системой, включая создание, удаление, перемещение и копирование файлов и папок.

- NSURLSession: Этот класс предоставляет API для работы с сетью, включая загрузку и отправку данных через сеть.

Foundation также включает в себя множество других классов и API, таких как NSNotificationCenter, UserDefaults и NSProcessInfo, которые предоставляют базовые инструменты для создания приложений на платформах Apple.

Использование фреймворка Foundation позволяет разработчикам быстро и эффективно создавать приложения на платформах Apple.

2.3 Фреймворк ARKit

ARKit - это фреймворк в iOS, который предназначен для создания приложений с дополненной реальностью. ARKit использует камеру и датчики устройства для отслеживания местоположения и ориентации в пространстве, а также для определения расположения объектов в реальном мире.

Он поддерживает множество функций и возможностей для создания интерактивных и визуально потрясающих приложений.

ARKit может определить и отслеживать поверхности в реальном мире, что позволяет размещать объекты на таких поверхностях и сохранять их местоположение и ориентацию в пространстве.

Фреймворк использует переднюю камеру устройства для определения лиц и их выражений, что может быть использовано для создания приложений различных категорий, таких как маскировочные приложения, приложения для селфи.

Данный фреймворк сохраняет данные об объектах в реальном мире, чтобы пользователям перезапуск приложения не нужно было как то повторять инициализацию расположения объектов и поверхностей.

ARKit может определить освещение в реальном мире, что позволяет создавать более реалистичный визуальный эффект объектов в AR.

Он также предоставляет инструменты для взаимодействия пользователя с дополненной реальностью, такие как жесты, зум, вращение.

ARKit является надежным и производительным фреймворком в iOS, который открывает широкие возможности для создания приложений различных типов с использованием дополненной реальности.

2.5 Выбор технологий и библиотек

2.5.1 Фреймворк UIKit

UIKit – это фреймворк в iOS, который предоставляет набор инструментов и компонентов интерфейса для создания пользовательских приложений. UIKit является одним из основных фреймворков в iOS и включает в себя множество возможностей для создания визуально привлекательного и функционального интерфейса приложения.

Фреймворк включает в себя множество компонентов и объектов управления, таких как кнопки, текстовые поля, таблицы, контейнеры, визуальные эффекты и многое другое. Он также поддерживает анимацию и графический дизайн, что позволяет создавать интерактивные и эффектные приложения.

UIKit предоставляет инструменты для отображения текстов, изображений, видео и другой информации на экране, включая анимацию и эффекты. Он также предоставляет контейнеры, которые позволяют управлять расположением и оптимизацией для различных устройств, а также автоматическое управление размерами элементов интерфейса.

Фреймворк из коробки использует MVC (Model-View-Controller) архитектуру, которая разделяет приложение на три компонента: модели данных, представление пользовательского интерфейса и контроллеры, которые связывают модели и представления.

Модель (Model) представляет сущность, которую мы хотим отобразить в программном приложении. Она может быть представлена в виде объекта, базы данных или другого источника данных. Модель содержит информацию о данных и правила, которые ограничивают доступ к этим данным.

Представление (View) отвечает за отображение информации на экране. Оно получает данные из модели и отображает их пользователю в удобном виде. Кроме того, представление может обеспечивать возможность изменения данных, например, через формы или диалоговые окна.

Контроллер (Controller) является посредником между моделью и представлением. Он обрабатывает запросы на изменение данных из представления и изменяет модель соответствующим образом. Контроллер также отвечает за обработку действий пользователя, например, ввода текста или нажатия кнопки.

UIKit позволяет создавать интерфейс различными способами. Storyboard – это графическое представление пользовательского интерфейса вашего приложения. Он позволяет визуально располагать сразу все компоненты интерфейса приложения и соединять их друг с другом. Это удобный способ создания приложений, позволяющий разработчику не писать много кода для создания интерфейса.

Создание интерфейса с помощью кода также является возможным. Разработчик может задать необходимый внешний вид и поведение компонентов интерфейса вызовами методов из UIKit. Этот способ подходит для более гибкого управления созданием пользовательского интерфейса, для решения конкретных задач, которых не предусмотрено в готовых компонентах.

UIKit – это мощный и гибкий фреймворк в iOS, который используется для создания различных типов приложений, от игр и приложений для работы с графикой, до криптовалютных кошельков. Он предоставляет множество возможностей, которые позволяют создавать красивый и функциональный пользовательский интерфейс, обеспечивающий комфортную работу пользователей.

2.5.2 Формат XCFramework

XCFramework — это новый формат фреймворков, представленный компанией Apple в iOS 12 и macOS 10.14 Mojave. XCFramework объединяет два типа библиотек: динамические библиотеки (dylibs) и статические библиотеки (static libraries).

Основное преимущество XCFramework заключается в возможности создания универсальных библиотек, которые могут быть использованы на разных архитектурах процессоров и платформах. Это сильно упрощает процесс разработки многоплатформенных приложений и существенно уменьшает размер приложения.

XCFramework состоит из нескольких слинкованных файлов разных типов, при этом каждый файл может быть скомпилирован под различные архитектуры и платформы. При создании XCFramework можно указать несколько архитектур (например, x86_64, arm64) и платформ (iOS, macOS, watchOS, tvOS), при этом каждая из архитектур и платформ может быть представлена как динамическая или статическая библиотека.

Кроме того, в XCFramework есть возможность интегрировать дополнительные заголовочные файлы и ресурсы, что делает его более удобным в использовании.

Основные преимущества использования XCFramework:

- Возможность создания универсальных библиотек, поддерживающих различные архитектуры и платформы.
- Сокращение размера приложения, что особенно важно для мобильных приложений.
- Упрощение работы с различными библиотеками и зависимостями в проектах.
- Возможность интеграции дополнительных ресурсов и заголовочных файлов.

Недостатки использования XCFramework:

- Некоторые сторонние библиотеки могут не поддерживать новый формат XCFramework, поэтому возможны трудности с интеграцией.
- В случае использования только одной платформы или архитектуры, XCFramework может быть несколько более сложным в использовании, чем простые динамические или статические библиотеки.

XCFramework является новым и универсальным инструментом для разработки кроссплатформенных приложений и библиотек на основе платформ

Apple. Он сильно упрощает процесс разработки и обеспечивает большую гибкость в использовании.

2.5.3 Firebase Auth

Firebase Auth - это компонент в наборе инструментов Firebase от компании Google, который предоставляет ряд функций аутентификации пользователя в приложениях для Android, iOS и веб-приложений. Firebase Auth облегчает процесс аутентификации пользователей в приложении, предоставляя множество функций, включая проверку подлинности электронной почты и пароля, вход через социальные сети, многофакторную аутентификацию и так далее.

Firebase Auth использует процедуры OAuth 2.0 и OpenID Connect для поддержки аутентификации сторонних приложений, а также встроенную аутентификацию Google-аккаунта.

В Firebase Auth можно использовать следующие методы аутентификации:

- Аутентификация по электронной почте и паролю. Пользователи могут регистрироваться с указанием своих электронных адресов и паролей.
- Вход через социальную сеть. Firebase Auth поддерживает вход через Facebook, Twitter, Google, GitHub, LinkedIn и Apple.
- Многофакторная аутентификация. Firebase Auth поддерживает использование кодов подтверждения и других устройств для улучшения безопасности приложения.
- Анонимная аутентификация. Пользователи могут войти в приложение без регистрации.

Firebase Auth также предоставляет API для управления пользователями, включая создание, обновление и удаление аккаунтов, а также управление учётными записями в приложении и доступ к API сторонних приложений.

2.5.4 Библиотека Charts

Charts iOS - это мощная и простая в использовании библиотека для создания диаграмм, графиков и других инфографических элементов в приложениях для iOS. Она предлагает большой выбор типов графиков и диаграмм, а также массу опций для настройки стиля, цветовой схемы и других параметров.

Библиотека Charts iOS написана на языке Swift и может быть интегрирована в любой проект для iOS различной сложности. С ее помощью можно создавать следующие типы графиков:

- Линейные графики
- Гистограммы
- Круговые диаграммы
- Столбчатые графики
- Свечные графики

Каждый из этих типов графиков имеет много настроек и настроек стиля, которые можно изменить, чтобы получить желаемый результат.

Некоторые функции библиотеки Charts iOS:

- Поддержка множества типов графиков, включая линейные графики, гистограммы, круговые диаграммы.
- Гибкие настройки стиля и цветовой схемы графиков.
- Можно создавать красивые инфографики, используя стандартные опции и пользовательские параметры.

2.5.5 Менеджер пакетов CocoaPods

CocoaPods – это система для управления библиотеками для Apple-платформ, включая iOS, macOS, watchOS и tvOS. С ее помощью разработчикам удобно устанавливать и использовать сторонние библиотеки в своих проектах.

Основная цель CocoaPods - упростить установку и обновление библиотек в проектах iOS и macOS. Разработчики могут использовать командную строку для установки библиотек, затем CocoaPods автоматически интегрирует их в Xcode-проекты.

Преимущества CocoaPods:

- Большое количество библиотек доступное для установки.
- Простая установка и обновление библиотек.
- Возможность использования различных версий библиотек для разных проектов.
- Возможность создавать и публиковать свои собственные библиотеки.

Некоторые возможные недостатки CocoaPods:

- Дополнительные зависимости в проектах.
- Сложность тестирования и отслеживания ошибок.
- Возможные проблемы с совместимостью библиотек.
- Необходимость постоянного обновления и управления зависимостями.

Тем не менее, CocoaPods продолжает оставаться одним из самых популярных инструментов управления библиотеками в экосистемах Apple. Он является удобным и гибким инструментом для использования сторонних библиотек в проектах, простой в настройке и использовании.

2.6 Выводы по разделу

В результате обзора технических средств и анализа поставленной задачи для реализации дипломного проекта была выбрана платформа *iOS SDK*, так как данная платформа предоставляет доступ к нативным средствам для управления системой.

В качестве средства для реализации отслеживания взгляда и мимики лица был выбран нативный фреймворк ARKit, так как он линкуется с приложением динамически и, следовательно, предустановлен в устройстве и не требует дополнительной памяти для приложения, а также дает возможность использовать все возможности TrueDepth камеры, которая включает в себя несколько датчиков.

Архитектурным паттерном для проекта был выбран MVVM, так как он позволяет разделить бизнес логику от логики представления, в отличие от паттерна MVC, реализованном в фреймворке UIKit.

3 Проектирование программного средства

Задачей дипломного проекта является разработка библиотеки, позволяющей внедрить управление с помощью взгляда и мимики лица в мобильное приложение.

Перед началом разработки библиотеки необходимо определить цели и задачи, а также продумать все варианты ее использования.

Библиотека предусматривает возможность отслеживания взгляда пользователя, проецирования его взгляда на экран, распознавание мимики лица для выполнения действий. Также библиотека предоставляет интерфейс для использования распознавания речи для ввода текста в текстовые поля приложения.

Программист с помощью данной библиотеки сможет внедрить в свое приложение распознавание взгляда, мимики лица и речи как отдельные независимые модули, так и как общую систему. Использование библиотеки предоставит ему как более удобное API для реализации данных функций, так и стандартные реализации.

Также в рамках дипломного проекта будет разработано приложение, которое будет демонстрировать функционал библиотеки, а также даст возможность ее тестирования. Так как для тестирования библиотеки необходима камера TrueDepth, ее автоматизированное тестирование невозможно. Ввиду этого, приложение также предоставит возможность ручного тестирования библиотеки.

3.1 Проектирование библиотеки

Библиотека представляет собой набор классов и протоколов, реализующих необходимый функционал и предоставляющих программисту доступ к событиям. Также в библиотеке присутствуют вспомогательные классы и расширения базовых классов для более удобного и изолированного взаимодействия с системой.

API библиотеки реализовано с помощью поведенческого шаблона «Делегирование». Это техника, в которой объект выражает определенное поведение снаружи, но в реальности делегирует ответственность за реализацию этого поведения связанному объекту.

Паттерн делегирования обеспечивает механизм отвлечения от реализации и контроля желаемого действия. Класс, вызываемый для выполнения действия, не выполняет его, а фактически делегирует вспомогательному классу. Потребитель не имеет или не требует знания фактического класса, выполняющего действие, только контейнера, к которому производится вызов.

Данный паттерн в языке Swift позволяет подписывать пользовательские классы под классы библиотеки, что в дает возможность библиотеке выполнять код и отправлять события пользователю, которые он сможет обрабатывать в своем коде.

				БГТУ 03.00.ПЗ					
	ФИО	Подпись	Дата						
Разраб.	Юшкевич И.Н.			3 Проектирование программного модуля			Лит.	Лист	Листов
Провер.	Скребель А.С.								
Н. контр.	Николайчук А.Н.						11111111, 2023		
Утв.	Смелов В.В.								

Библиотекой будут реализованы следующие функции:

- распознавание взгляда пользователя;
- распознавание лица и мимики пользователя;
- проецирование взгляда на экран устройства с помощью курсора;
- адаптация жестов ОС (например, прокручивание списка);
- распознавание речи.

Взаимодействие библиотеки с внешним кодом происходит за счет событий.

Работа библиотеки основана на трех основных функциях – распознавание взгляда, распознавание лица и речи. Эти функции разделены на классы, что позволяет придерживаться принципа единственной ответственности и сделать код библиотеки менее связанным.

Класс «*Session*», представленный в таблице 3.1, отвечает за создание и управление AR сессией, которая позволяет отслеживать все описанные функции.

Таблица 3.1 – Структура класса «*EyeTracker*»

Название	Тип	Описание
arSession	ARSession	Объект AR сессии, отслеживающий все функции
configuration	ARFaceTrackingConfiguration	Конфигурация AR сессии
delegates	[SessionDelegate]	Массив объектов-подписчиков, ожидающих события
isSessionInProgress	Bool	Свойство, отображающее статус сессии
start(ARFaceTrackingConfiguration, ARSession.RunOptions)	Функция	Функция для старта AR сессии с определенной конфигурацией и параметрами
end()	Функция	Функция для остановки AR сессии
session(ARSession, ARFrame)	Bool	Функция, оповещающая подписчиков об изменениях в сессии

Данный класс является оберткой над классом ARKit «*ARSession*», основные используемые свойства, функции и делегаты которого будут перечислены далее.

Таблица 3.2 – Структура класса «*ARSession*»

Название	Тип	Описание
delegate	ARDelegate	Объект-подписка на текущую AR сессию
configuration	ARFaceTrackingConfiguration	Конфигурация AR сессии
options	ARSession.RunOptions	Свойство, отвечающее за настройки запуска сессии
run(ARFaceTrackingConfiguration, ARSession.RunOptions)	Функция	Функция для запуска объекта ARSession
pause(ARSession, ARFrame)	Функция	Функция для остановки AR сессии

Также класс содержит массив делегатов. Данный тип содержит метод, который вызывается при каждом событии от Session.

Таблица 3.3 – Структура протокола «*SessionDelegate*»

Название	Тип	Описание
sessionDidUpdate(ARSession, ARFrame)	Функция	Функция оповещения о событии со стороны сессии

В результате объект класса Session является оберткой над нативным инструментом. Данная обертка позволила предоставить дополнительные возможности по управлению событиями в виде SessionDelegate. Далее был реализован класс для регистрации мимики лица «*FaceExpression*», представленный в таблице 3.4.

Таблица 3.4 – Структура класса «*FaceExpression*»

Название	Тип	Описание
blendShape	ARFaceAnchor.BlendShapeLocation	Тип мимики
minValue	Float	Минимальное значение мимики для срабатывания события
MaxValue	Float	Максимальное значение мимики для срабатывания события

Далее необходимо реализовать делегат, который бы уведомлял приложение о появлении одного из зарегистрированных типов мимики. Этим делегатом является протокол «*FaceTrackerDelegate*», представленный в таблице 3.5.

Таблица 3.5 – Структура протокола «*FaceTrackerDelegate*»

Название	Тип	Описание
faceTracker(FaceTracker, FaceExpression)	Функция	Функция оповещения о распознанной мимике

За отслеживание лица и его мимики отвечает класс «*FaceTracker*», свойства и методы которого описаны в таблице 3.6.

Таблица 3.6 – Структура класса «*FaceTracker*»

Название	Тип	Описание
session	Session	Объект AR сессии, отслеживающий все функции
expressions	[FaceExpression]	Массив зарегистрированных в приложении типов мимики

Окончание таблицы 3.6

Название	Тип	Описание
lastActionDate	Date	Время последнего действия. Необходимо для исключения случайных действий
delegates	[FaceTrackerDelegate]	Массив объектов-слушателей
instantiateFaceExpression(FaceExpression)	Функция	Функция регистрации типа мимики в приложении
removeFaceExpression(FaceExpression)	Функция	Функция удаления типа мимики из приложения
setDelegate(FaceTrackerDelegate)	Функция	Функция для добавления объектов-слушателей
removeDelegate(FaceTrackerDelegate)	Функция	Функция для удаления объектов-слушателей
checkExpression	Функция	Функция проверки соответствия типа мимики из массива типу мимики пользователя
sessionDidUpdate(ARSession, ARFrame)	Функция	Функция, оповещающая объекты-слушатели о событии

Отслеживание взгляда реализовано в классе «*EyeTracker*». Класс позволяет отслеживать взгляд и перемещать курсор на экране. Свойства и методы класса «*EyeTracker*» представлены в таблице 3.7

Таблица 3.7 – Структура класса «*EyeTracker*»

Название	Тип	Описание
session	Session	Объект AR сессии, отслеживающий все функции
pointer	Pointer?	Курсор для отслеживания взгляда на экране
delegates	[EyeTrackerDelegate]	Массив объектов-слушателей
positionLogs	[CGPoint]	Массив точек, где находился курсор. Необходимы для интерполяции точек с целью создания более плавного движения курсора
lastPositionIndex	Int	Индекс последней позиции в positionLogs
showPointer(UIWindow, PointerConfiguration)	Функция	Функция отображения курсора на экране

setDelegate(FaceTrackerDelegate)	Функция	Функция для добавления объектов-слушателей
----------------------------------	---------	--

Окончание таблицы 3.7

Название	Тип	Описание
removeDelegate(FaceTrackerDelegate)	Функция	Функция для удаления объектов-слушателей
getGazePosition(ARFrame, CGSize) -> CGPoint?	Функция	Функция поиска точки, в которой находится взгляд в данном кадре
sessionDidUpdate(ARSession, ARFrame)	Функция	Функция, оповещающая объекты-слушатели о событии

Для реализации курсора был создан класс «*Pointer*», описание свойств и методов которого представлено в таблице 3.8

Таблица 3.8 – Структура класса «*Pointer*»

Название	Тип	Описание
window	UIWindow?	Окно приложения, к которому крепится курсор
pointer	UIView	Графическая реализация курсора
show(PointerConfiguration)	Функция	Функция, отображающая курсор на выбранном окне с указанной конфигурацией
move(CGPoint)	Функция	Функция, перемещающая курсор в указанную точку

Указатель имеет свою конфигурацию, которая описана в таблице 3.9.

Таблица 3.9 – Структура класса «*PointerConfiguration*»

Название	Тип	Описание
color	UIColor?	Цвет курсора
size	CGSize	Размер курсора
cornerRadius	Double	Закругление углов курсора
cornerCurve	CALayerCornerCurve	Вид углов курсора
zPosition	CGFloat	Положение курсора на экране относительно других графических элементов

Дополнительно класс «*EyeTracker*» использует класс «*Plane*», описанный в таблице 3.10, для выполнения вычислений на плоскостях.

Таблица 3.10 – Структура класса «*Plane*»

Название	Тип	Описание
----------	-----	----------

normal	simd_float3?	Положение нормали к плоскости
--------	--------------	-------------------------------

Окончание таблицы 3.10

Название	Тип	Описание
dist	Float	Расстояние от плоскости до точки

Также в таблице 3.11 описан класс «Ray», определяющий луч.

Таблица 3.11 – Структура класса «Ray»

Название	Тип	Описание
origin	simd_float3?	Точка начала луча
direction	simd_float3?	Направление луча

Для оповещения объектов о событии создан протокол «EyeTrackerDelegate».

Таблица 3.12 – Структура протокола «EyeTrackerDelegate»

Название	Тип	Описание
eyeTracker(EyeTracker, TrackingState, FaceExpression)	Функция	Функция оповещения об изменении положения взгляда

Исходя из таблицы 3.12 видно, что используется параметр TrackingState. Данный параметр представляет собой два перечисления, которые указывают на состояние взгляда относительно экрана мобильного телефона. Перечисление описано в таблице 3.13.

Таблица 3.13 – Структура перечисления «TrackingState»

Название	Тип	Описание
screenIn(CGPoint)	TrackingState	Взгляд находится на экране
screenOut(Edge, CGPoint)	TrackingState	Взгляд находится вне экрана. Параметр Edge оповещает о том, с какой стороны экрана вышел взгляд

Для реализации последней функции – распознавание речи, был создан класс «SpeechRecognition». Его описание приведено в таблице 3.14.

Таблица 3.14 – Структура класса «SpeechRecognition»

Название	Тип	Описание
error	RecognizerError	Перечисление, указывающее на род ошибки распознавания
transcript	String	Содержит распознанную речь пользователя
audioEngine	AVAudioEngine?	Аудио-движок, отвечающий за захват аудио

		через микрофоны	встроенные
--	--	--------------------	------------

Окончание таблицы 3.14

Название	Тип	Описание
request	SFSpeechAudioBufferRecognitionRequest?	Запрос на обработку аудиоданных
task	SFSpeechRecognitionTask?	Задача по обработке аудиоданных. Необходима для асинхронного выполнения обработки на выделенном потоке
recognizer	SFSpeechRecognizer?	Объект, распознающий речь и создающий ее транскрипт
startTranscribing()	Функция	Функция начала распознавания речи
stopTranscribing()	Функция	Функция остановки распознавания
prepareEngine()	Функция	Функция инициализации аудио-движка
transcribe()	Функция	Функция транскрипции аудиопотока
reset()	Функция	Функция сброса аудио-движка
recognitionHandler()	Функция	Функция, обрабатывающая ошибки, которые могут возникнуть во время обработки аудиопотока
hasAuthorizationToRecognize()	Функция	Функция, проверяющая право на обработку аудиопотока для распознавания речи
hasPermissionToRecord()	Функция	Функция, проверяющая право на доступ к микрофону

Описанные выше таблицы в полной мере удовлетворяют всем функциональным требованиям разрабатываемой библиотеки, а также соответствуют принципам SOLID, что впоследствии позволит программистам

создавать более гибкое программное обеспечение на основе данной библиотеки. Диаграмма классов библиотеки представлена в приложении Г.

3.2 Проектирование приложения

Тестовое приложение будет реализовано на языке Swift с помощью фреймворка UIKit. Данное приложение будет содержать нативные средства для построения iOS приложения, что позволит протестировать библиотеку в условиях, приближенных к реальным.

Приложение будет реализовано на архитектуре MVVM, что даст возможность отделить бизнес логику от визуального отображения, а также позволит убедиться в корректности реализации библиотеки, так как в случае ошибок при проектировании, компоненты библиотеки будут проникать в слой бизнес-логики, то есть ViewModel.

Приложение будет состоять из следующих компонент:

- два вида навигационных меню (стек и коллекция);
- коллекция;
- текстовые поля;
- графики;
- переключатели;
- кнопки;

Данные компоненты являются наиболее распространенными в мобильных приложениях, что даст возможность обобщить наибольшее количество ручных тестов в одном мобильном приложении.

Приложение будет обладать следующим функционалом:

- просмотр курсов валют за определенные промежутки времени;
- конвертация валют;
- вход в аккаунт при помощи FaceID;
- добавление транзакций в аккаунт авторизованного пользователя.

3.3 Выводы по разделу

В данном разделе описан процесс разработки архитектуры библиотеки. Были обозначены все классы, необходимые для корректной реализации библиотеки. Также была определена структура и архитектура тестового приложения.

4 Реализация библиотеки

4.1 Задачи библиотеки

Задачей дипломного проекта является разработка библиотеки, позволяющей внедрить управление с помощью взгляда и мимики лица в мобильное приложение.

Программист с помощью данной библиотеки сможет внедрить в свое приложение распознавание взгляда, мимики лица и речи как отдельные независимые модули, так и как общую систему. Использование библиотеки предоставит ему как более удобное API для реализации данных функций, так и стандартные реализации.

Также в рамках дипломного проекта будет разработано приложение, которое будет демонстрировать функционал библиотеки, а также даст возможность ее тестирования. Так как для тестирования библиотеки необходима камера TrueDepth, ее автоматизированное тестирование невозможно. Ввиду этого, приложение также предоставит возможность ручного тестирования библиотеки.

Библиотекой будут реализованы следующие функции:

- распознавание взгляда пользователя;
- распознавание лица и мимики пользователя;
- проецирование взгляда на экран устройства с помощью курсора;
- адаптация жестов ОС (например, прокручивание списка);
- распознавание речи.

Приложение будет обладать следующим функционалом:

- просмотр курсов валют за определенные промежутки времени;
- конвертация валют;
- вход в аккаунт при помощи FaceID;
- добавление транзакций в аккаунт авторизованного пользователя.

Исходя из определенных функций приложения и библиотеки, можно приступить к их реализации.

4.2 Разработка AR сессии

Основой данной библиотеки является AR сессия. Она позволяет использовать камеру для распознавания различных объектов, их сканирования и проецирования. Однако стандартная реализация в фреймворке ARKit является довольно обобщенной и не предоставляет как необходимых ограничений, так и дополнительных функций, которые потребуются при разработке модулей распознавания лица и взгляда. Поэтому была реализована обертка над ARSession.

				БГТУ 04.00.ПЗ			
	ФИО	Подпись	Дата				
Разраб.	Юшкевич И.Н.			4 Реализация программного модуля	Лит.	Лист	Листов
Провер.	Скребель А.С.					1	15
Н. контр.	Николайчук А.Н.						
УТВ.	Смелов В.В.				11111111, 2023		

Основные задачи данной обертки – ограничить количество одновременно созданных AR сессий, предоставить стандартную конфигурацию сессии, а также реализовать отправку модифицированных событий.

Реализация сессии состоит из трех файлов, изображенных на рисунке 4.1.

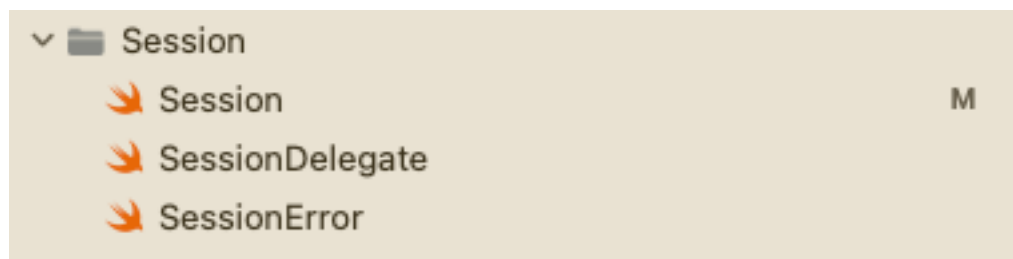


Рисунок 4.1 – Структура сессии

Файл SessionError содержит в себе перечисление SessionError, показанное в листинге 4.1, которое включает в себя все ошибки, связанные с работой сессии.

```
enum SessionError: Error {  
    case arNotSupported  
    case noSessionsInProgress  
}
```

Листинг 4.1 – Ошибки сессии

Данные ошибки возникают при вызове методов запуска и остановки сессии.

Файл SessionDelegate содержит в себе протокол-делегат SessionDelegate, который реализует отправку событий об обновлении сессии в классы, реализующие отслеживание взгляда и лица. Протокол представлен в листинге 4.2.

```
public protocol SessionDelegate {  
    func sessionDidUpdate(_ session: ARSession, frame: ARFrame)  
}
```

Листинг 4.2 – Делегат сессии

В файле Session реализован класс Session. Класс имеет методы start, который проверяет наличие активных сессий и, при отсутствии активных сессий, инициализирует новую с полученной конфигурацией и начинает отслеживание стандартной реализации ARSession, реализованной в ARKit. А в противном случае выбрасывает ошибку о наличии активной сессии. Реализация данного метода представлена в листинге 4.3.

```

public func start(with config: ARFaceTrackingConfiguration? = nil,
                  options: ARSession.RunOptions =
[.resetTracking, .removeExistingAnchors]) throws {
    guard ARFaceTrackingConfiguration.isSupported else { throw
SessionError.arNotSupported }

    if isSessionInProgress { return }

    if let config {
        faceTrackingConfiguration = config
    }

    arSession.delegate = self
    arSession.run(faceTrackingConfiguration, options: options)

    isSessionInProgress = true
}

```

Листинг 4.3 – Реализация метода запуска сессии

Вторым методом в классе `Session` является метод остановки сессии, представленный в листинге 4.4. Данный метод также проверяет наличие текущих сессий и, при наличии активной сессии, останавливает ее. В противном случае метод выбрасывает ошибку об отсутствии сессий.

```

public func end() throws {
    guard isSessionInProgress else { throw
SessionError.noSessionsInProgress }

    arSession.pause()
    isSessionInProgress = false
}

```

Листинг 4.4 – Реализация метода остановки сессии

При изменениях в `ARSession`, класс-обертка вызывает метод делегата, описанный выше. Пример вызова метода делегата приведен в листинге 4.5.

```

public func session(_ session: ARSession, didUpdate frame: ARFrame) {
    delegates.forEach { delegate in
        delegate.sessionDidUpdate(session, frame: frame)
    }
}

```

Листинг 4.5 – Вызов метода делегата

Таким образом была реализована обертка над классом `ARSession`. Она позволила наложить ограничения на количество сессий, а также добавить обработку пользовательских событий с помощью паттерна делегирования. API сессии представлено в приложении Д.

4.3 Разработка распознавания лица

Файловая структура распознавания лица представлена на рисунке 4.2.

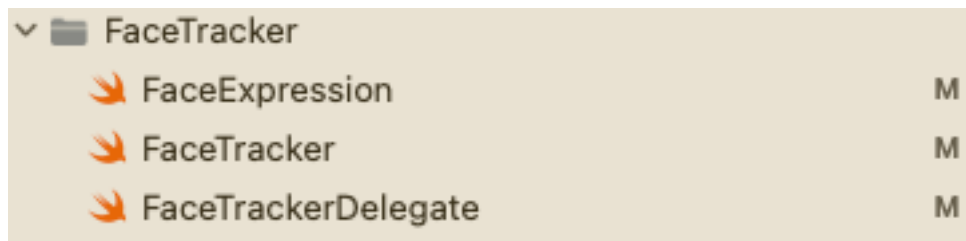


Рисунок 4.2 – Структура распознавания лиц

Класс `FaceExpression` представлен на листинге 4.6. Он отвечает за регистрацию новых мимик лица. Он определяет тип выражения, верхний и нижний порог его определения. Также была реализована функция сравнения выражений для возможности их определения и удаления.

```
public final class FaceExpression: Equatable {
    public let blendShape: ARFaceAnchor.BlendShapeLocation
    public let minValue: Float
    public let maxValue: Float

    public init(blendShape: ARFaceAnchor.BlendShapeLocation, minValue:
Float, maxValue: Float) {
        self.blendShape = blendShape
        self.minValue = minValue
        self.maxValue = maxValue
    }

    public static func == (lhs: FaceExpression, rhs: FaceExpression) ->
Bool {
        guard
            lhs.blendShape == rhs.blendShape,
            lhs.minValue == rhs.minValue,
            lhs.maxValue == rhs.maxValue
        else { return false }

        return true
    }
}
```

Листинг 4.6 – Класс определения выражения лица

Делегат `FaceTrackerDelegate` определяет событие, отправляемое объектам-подписчикам, которые отслеживают изменения выражения лица.

Класс `FaceTracker` отвечает за определение мимики лица. Основным методом данного класса является реализация `SessionDelegate`. Метод, представленный на листинге 4.7, выжидает интервал в 0.5 секунд для предотвращения ложных

срабатываний, проверяет событие от сессии и, при нахождении зарегистрированной мимики, отправляет событие всем объектам-подписчикам, которые зарегистрировали данное действие.

```
public func sessionDidUpdate(_ session: ARSession, frame: ARFrame) {
    guard
        let faceAnchor = frame.anchors.first as? ARFaceAnchor,
        Date().timeIntervalSinceNow -
lastActionDate.timeIntervalSinceNow >= 0.5
    else { return }
    lastActionDate = Date()
    expressions.forEach { expression in
        if checkExpression(expression, faceAnchor: faceAnchor) {
            delegates.forEach { delegate in
                delegate?.faceTracker(self, didUpdateExpression:
expression)
            }
        }
    }
}
```

Листинг 4.7 – Реализация отслеживания мимики лица

Таким образом было реализовано распознавание лица и его выражений. API мимики лица представлено в приложении Е.

4.4 Разработка распознавания взгляда

Файловая структура распознавания лица представлена на рисунке 4.3.

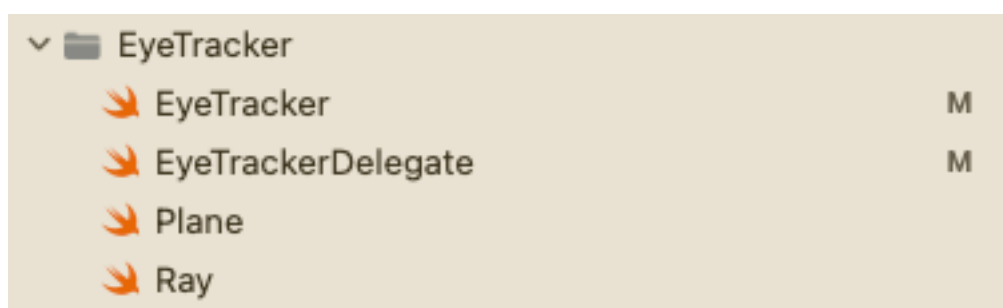


Рисунок 4.3 – Структура распознавания взгляда

Классы Plane и Ray, представленные на листингах 4.8 и 4.9 соответственно, отвечают за представление взгляда пользователя в виде проекции луча на поверхность. Данные классы имеют модификатор internal, так как не используются вне модуля. Их использование будет рассмотрено при рассмотрении класса EyeTracker, где они используются для вычисления точки на экране мобильного телефона, в которую смотрит пользователь.

```

struct Plane {
    var normal: simd_float3
    var dist: Float

    init(p1: simd_float3, p2: simd_float3, p3: simd_float3) {
        let p21 = p2 - p1
        let p32 = p3 - p2

        normal = cross(p21, p32)
        normal = normalize(normal)
        dist = dot(normal, p1)
    }
}

```

Листинг 4.8 – Класс описывающий поверхность

```

struct Plane {
    var normal: simd_float3
    var dist: Float

    init(p1: simd_float3, p2: simd_float3, p3: simd_float3) {
        let p21 = p2 - p1
        let p32 = p3 - p2

        normal = cross(p21, p32)
        normal = normalize(normal)
        dist = dot(normal, p1)
    }
}

```

Листинг 4.9 – Класс описывающий луч

Делегат `EyeTrackerDelegate` определяет событие, отправляемое объектам-подписчикам, которые отслеживают изменения положения взгляда.

Класс `EyeTracker` отвечает за определение направления взгляда пользователя. Основным методом данного класса является метод `getGazePosition`. Метод определяет положение взгляда пользователя исходя из кадра, полученного от камеры. На листинге 4.10 показано получение координат лица из фрейма AR сессии.

```

guard let faceAnchor = frame.anchors.first as? ARFaceAnchor else {
    return nil }

```

Листинг 4.10 – Получение координат лица

Далее для реализации отслеживания взгляда необходимо получить матрицы преобразования для левого и правого глаза пользователя. Это возможно сделать исходя из полученных координат лица. Объект `ARFaceAnchor` содержит

необходимые поля `rightEyeTransform` и `leftEyeTransform`, которые выражают положение зрачков глаз в собственной системе координат.

Однако для корректного проецирования взгляда на экран необходимо преобразовать положение зрачков из собственной системы координат в координаты экрана. Для достижения данного результата сначала нужно перевести взгляд из собственной системы координат в систему мировых координат. Достигается это за счет перемножения матриц преобразований, который имеют вид матрицы 4x4, представленный на рисунке 4.4. На листинге 4.11 приведено данное преобразование в методе `getGazePosition`.

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & a_3 & b_1 \\ a_4 & a_5 & a_6 & b_2 \\ a_7 & a_8 & a_9 & b_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Рисунок 4.4 – Матрица преобразований

```
let rightEyeSimdTransform = simd_mul(faceAnchor.transform,
faceAnchor.rightEyeTransform)
let leftEyeSimdTransform = simd_mul(faceAnchor.transform,
faceAnchor.leftEyeTransform)
```

Листинг 4.11 – Преобразование координат в мировые

Так как координатная система камеры не совпадает с мировой системой координат, необходимо выполнить ряд преобразований, а также вычислить плоскость в мировой системе координат, относительно которой в последствии будет вычисляться падение лучей. Данные преобразования описаны в листинге 4.12.

```
var cameraTransform = frame.camera.transform

var translation = matrix_identity_float4x4
let p1 = cameraTransform.position
translation.columns.3.x = 1.0
cameraTransform = simd_mul(cameraTransform, translation)
let p2 = cameraTransform.position
translation = matrix_identity_float4x4
translation.columns.3.y = 1.0
cameraTransform = simd_mul(cameraTransform, translation)
let p3 = cameraTransform.position

let plane = Plane(p1: p1, p2: p2, p3: p3)
```

Листинг 4.12 – Вычисление плоскости из координат камеры

Путем перемножения матрицы преобразования камеры на матрицы трансформации по координатам X и Y с помощью функции `simd_mul`, мы получаем три точки на плоскости, по которым можно построить данную плоскость с помощью класса `Plane`, описанного ранее.

Далее необходимо получить проекцию на экран координат левого и правого глаза с учетом расстояния до экрана. Для этого необходимо построить лучи от глаз к плоскости экрана, полученной ранее. Листинг 4.13 описывает получение лучей для каждого глаза и проецирование их на плоскость в мировой системе координат. Это даст нам две точки на плоскости, отображающие взгляд пользователя на плоскости.

```
let rightRay = Ray(origin: rightEyeSimdTransform.position,
                  direction: -rightEyeSimdTransform.frontVector)
let leftRay = Ray(origin: leftEyeSimdTransform.position,
                  direction: -leftEyeSimdTransform.frontVector)

translation = matrix_identity_float4x4
translation.columns.3.z = rightRay.dist(with: plane) -
screenDisplacement
let rightEyeEndSimdTransform = simd_mul(rightEyeSimdTransform,
translation)
translation.columns.3.z = leftRay.dist(with: plane) -
screenDisplacement
let leftEyeEndSimdTransform = simd_mul(leftEyeSimdTransform,
translation)
```

Листинг 4.13 – Вычисление точек падения лучей на плоскость

Получив две точки на плоскости, необходимо вычислить одну среднюю точку взгляда пользователя в мировой системе координат. Это реализовано с помощью деления суммы матриц на 2, что представлено на листинге 4.14.

```
let eyesMidPoint = (rightEyeEndSimdTransform.position +
leftEyeEndSimdTransform.position) / 2
```

Листинг 4.14 – Получение точки взгляда пользователя в мировой системе координат

Так как мировая система координат и экранная система координат отличаются, как представлено на рисунках 4.5 и 4.6, необходимо выполнить проецирование вычисленной точки в систему координат экрана. Для этого в фреймворке `ARKit` реализован метод `projectPoint`, который принимает точку в мировой системе координат, ориентацию устройства, а также размеры плоскости экрана. Реализация данного преобразования представлена на листинге 4.15.

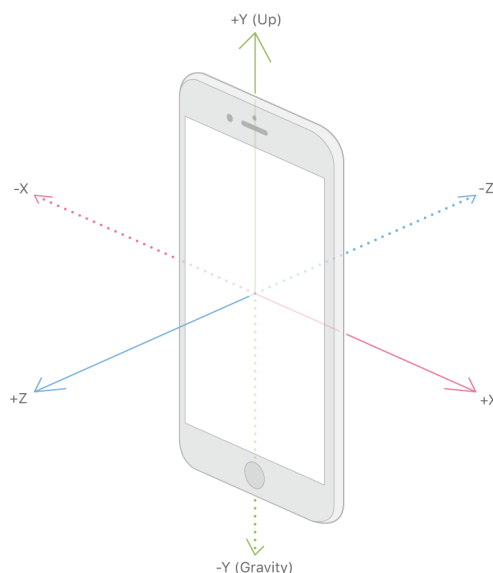


Рисунок 4.5 – Координатная система камеры



Рисунок 4.6 – Координатная система экрана

```
let screenPos = frame.camera.projectPoint(eyesMidPoint,
orientation: .portrait, viewportSize: viewport)
```

Листинг 4.15 – Преобразование точки из мировой в экранную систему координат

Таким образом было реализовано распознавание взгляда пользователя. API класса EyeTracker представлен в приложении Ж.

4.5 Распознавание речи

Файловая структура распознавания лица представлена на рисунке 4.7.

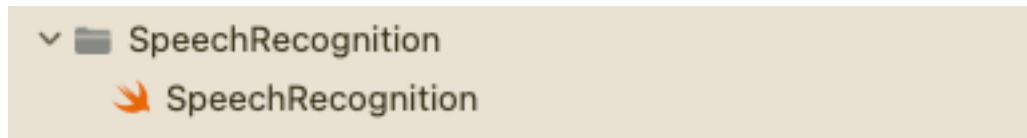


Рисунок 4.7 – Структура распознавания речи

Класс `SpeechRecognizer` отвечает за захват аудиопотока и создания транскрипта из его содержимого. Данный класс реализован с помощью двух фреймворков – `AVFoundation` и `Speech`, которые отвечают за захват аудиопотока и распознавание речи соответственно. Инициализатор класса `SpeechRecognizer`, представленный на листинге 4.16, инициализирует класс для распознавания речи, а также производит проверку на наличие разрешений для доступа к микрофону и отслеживанию речи.

```
public init() {
    recognizer = SFSpeechRecognizer()
    guard recognizer != nil else {
        transcribe(RecognizerError.nilRecognizer)
        return
    }

    Task {
        do {
            guard await
SFSpeechRecognizer.hasAuthorizationToRecognize() else {
                throw RecognizerError.notAuthorizedToRecognize
            }
            guard await
AVAudioSession.sharedInstance().hasPermissionToRecord() else {
                throw RecognizerError.notPermittedToRecord
            }
        } catch {
            transcribe(error)
        }
    }
}
```

Листинг 4.16 – Инициализатор класса `SpeechRecognizer`

Запросы на доступ к микрофону и распознаванию речи описаны в `plist` файле, содержание которого изображено на рисунке 4.8.

Key	Type	Value
Information Property List	Dictionary	(4 items)
Privacy - Microphone Usage Description	String	Allow microphone access to use speech recognition
Privacy - Speech Recognition Usage Description	String	Allow speech recognition access to transcribe your voice
Privacy - Camera Usage Description	String	Allow camera access to use eye tracking
Application Scene Manifest	Dictionary	(2 items)

Рисунок 4.8 – Разрешения на доступ к ресурсам системы

Основной функцией данного класса является функция `transcribe`, которая производит транскрипцию голосовой дорожки в текст. Ее реализация представлена в листинге 4.17. Данная функция инициализирует аудио-движок, который выполняет транскрипцию дорожки и возвращает результат в качестве замыкания.

```
private func transcribe() {
    guard let recognizer, recognizer.isAvailable else {
        self.transcribe(RecognizerError.recognizerIsUnavailable)
        return
    }

    do {
        let (audioEngine, request) = try Self.prepareEngine()
        self.audioEngine = audioEngine
        self.request = request
        self.task = recognizer.recognitionTask(with: request,
resultHandler: { [weak self] result, error in
            self?.recognitionHandler(audioEngine: audioEngine, result:
result, error: error)
        })
    } catch {
        self.reset()
        self.transcribe(error)
    }
}
```

Листинг 4.17 – Функция запуска транскрипции

Функция `prepareEngine`, описанная в листинге 4.18, конфигурирует движок, определяя шину, с которой будет считываться аудиодорожка, а также размер аудиобуфера.

```
let recordingFormat = inputNode.outputFormat(forBus: 0)
inputNode.installTap(onBus: 0, bufferSize: 1024, format:
recordingFormat) { (buffer: AVAudioPCMBuffer, when: AVAudioTime) in
    request.append(buffer)
}
```

Листинг 4.18 – Функция запуска транскрипции

Также при возникновении ошибок `SpeechRecognizer` возвращает в качестве результата транскрипции сообщение об ошибке. Данный функционал представлен в листинге 4.19.

```
nonisolated private func transcribe(_ error: Error) {
    var errorMessage = ""
    if let error = error as? RecognizerError {
        errorMessage += error.message
    } else {
        errorMessage += error.localizedDescription
    }
    Task { @MainActor [errorMessage] in
        transcript = "<< \(errorMessage) >>"
    }
}
```

Листинг 4.19 – Обработка ошибок транскрипции

Таким образом было реализовано распознавание речи пользователя. API класса `SpeechRecognizer` представлено в приложении 3.

4.6 Разработка менеджера

Для корректной работы библиотеки в рамках одной сессии, а также для реализации единой точки входа, было принято решение разработать класс `TrackingManager`, реализующий паттерн синглтон. Класс, представленный в листинге 4.20, позволяет запустить системы для отслеживания взгляда и мимики при помощи метода `start`, который принимает объект текущей сессии для ее последующей привязки к каждой системе.

```
public class TrackingManager {
    public static let shared = TrackingManager()

    public var eyeTracker = EyeTracker()
    public var faceTracker = FaceTracker()

    private init() { }

    public func start(with session: Session) {
        eyeTracker = EyeTracker(session: session)
        faceTracker = FaceTracker(session: session)
    }
}
```

Листинг 4.20 – Реализация класса `TrackingManager`

Данный класс позволяет использовать один объект системы для каждого экрана, в который впоследствии будут внедрены функции библиотеки.

4.7 Разработка тестового приложения

Для тестирования библиотеки было разработано приложение. Приложение реализовано на архитектуре MVVM, что дает возможность отделить бизнес логику от визуального отображения, а также позволит убедиться в корректности реализации библиотеки, так как в случае ошибок при проектировании, компоненты библиотеки будут проникать в слой бизнес-логики, то есть ViewModel.

Для встраивания библиотеки в приложение был использован менеджер зависимостей CocoaPods. Для этого в корне проекта вызывается команда `pod init`, которая создает рабочее пространство, включающее в себя проект и его зависимости.

Рассмотрим имплементацию библиотеки на примере класса `RateViewController`. Данный класс отвечает за отрисовку интерфейса и взаимодействие с ним в рамках фреймворка UIKit.

Так как UIKit является императивным фреймворком для написания пользовательского интерфейса, в нем нет разделения на файл разметки, инициализирующий объекты на экране, и файл для управления объектами на этой разметке. Инициализация всех объектов и управление ими производится в одном классе, как представлено в листинге 4.21.

```
private let manager = TrackingManager.shared

private var subscriptions = Set<AnyCancellable>()

private let tableView: UITableView = {
    let tableView = UITableView()

    tableView.translatesAutoresizingMaskIntoConstraints = false
    tableView.register(RateTableViewCell.self, forCellReuseIdentifier:
Constants.rateCellIdentifier)
    tableView.separatorStyle = .none

    return tableView
}()

private let refreshControl: UIRefreshControl = {
    let refreshControl = UIRefreshControl()

    return refreshControl
}()
```

Листинг 4.21 – Инициализация объектов пользовательского интерфейса

Для управления жизненным циклом экрана используются предопределенные в классе-родителе функции. В листинге 4.22 представлено добавление отслеживания взгляда и мимики лица при открытии экрана и его удаление при закрытии экрана.

```

override func viewDidAppear(_ animated: Bool) {
    super.viewDidAppear(animated)
    setupEyeTracking()
}

override func viewWillDisappear(_ animated: Bool) {
    super.viewWillDisappear(animated)
    removeEyeTracking()
}

```

Листинг 4.22 – Управление жизненным циклом экрана

Для интеграции библиотеки на экран достаточно подписать класс под протоколы, которые выполняют роль API библиотеки. Достигается это путем расширения класса и композиции интерфейсов, как показано в листинге 4.23.

```

extension RatesViewController: EyeTrackerDelegate, FaceTrackerDelegate {
}

```

Листинг 4.23 – Имплементация библиотеки

Таким образом приложение подписывается на события, испускаемые библиотекой, и может на них реагировать.

Рассмотрим реализацию отслеживания мимики лица для выполнения действий в приложении. Листинг 4.24 показывает, что приложение, получая зарегистрированную мимику лица от библиотеки, отправляет ее всем обработчикам распознавания взгляда, чтобы он обработал мимику с учетом взгляда.

```

public func faceTracker(_ faceTracker: FaceTracker, didUpdateExpression
expression: FaceExpression) {
    manager.eyeTracker.delegates.forEach { delegate in
        delegate?.eyeTracking(manager.eyeTracker, didUpdateState:
manager.eyeTracker.state, with: expression)
    }
}

```

Листинг 4.24 – Обработка событий об изменении мимики

Далее обработчик взгляда с учетом полученной мимики выполняет действия, определенные программистом в приложении. Например, в листинге 4.25 при открытии рта будет открыта ячейка таблицы, на которую смотрит пользователь, а при вынесении взгляда за пределы экрана, список начнет прокручиваться на 10 пунктов вверх или вниз, в зависимости от того, какое событие было получено от библиотеки. При отсутствии зарегистрированной мимики в моменте нахождения взгляда в рамках экрана, никаких действий выполняться не будет.

```

func eyeTracking(_ eyeTracker: EyeTracking.EyeTracker, didUpdateState
state: EyeTracking.EyeTracker.TrackingState, with expression:
EyeTracking.FaceExpression?) {
    switch state {
    case .screenIn(let point):
        guard let expression else { return }
        switch expression.blendShape {
        case .jawOpen:
            hitCell(at: point)
        default:
            return
        }
    case .screenOut(let edge, _):
        switch edge {
        case .left, .right:
            return
        case .top:
            scrollView(-6)
        case .bottom:
            scrollView(6)
        }
    }
}

```

Листинг 4.25 – Обработка событий о перемещении взгляда

Таким образом возможно реализовать любое необходимое поведение для каждого элемента пользовательского интерфейса. Однако стоит учитывать, что фреймворк UIKit не поддерживает симуляцию нажатия в конкретной точке, поэтому есть необходимость реализации нажатия на элементы пользовательского интерфейса различным образом для каждого элемента фреймворка.

4.8 Выводы по разделу

В ходе данного раздела была разработана библиотека для определения лиц, мимики, взгляда, а также распознавания речи. Был подробно расписан функционал библиотеки и методы его реализации. Также было разработано тестовое приложение, на примере которого далее будет протестирована работа библиотеки.

5 Тестирование программного модуля

Тестирование программного обеспечения (ПО) является неотъемлемой частью жизненного цикла разработки ПО. Любая проблема с функциональностью в программном обеспечении может привести к серьезным последствиям, таким как потеря времени, ресурсов, репутации, а также к значительным затратам на исправление некачественной разработки. Поэтому своевременная проверка того, что программный продукт выполняет заявленные функции и не содержит критических ошибок в основных сценариях использования, является очень важной задачей.

Тестирование ПО – это проверка соответствия между реальным поведением программы и ее ожидаемым поведением.

В широком смысле слова тестирование – это одна из техник контроля качества, включающая активности:

- планирование работ;
- разработка тестов;
- выполнение тестов;
- анализ полученных результатов.

Если говорить о целях тестирования, то это прежде всего повышение вероятности:

- что система, которую мы разработали, решает заявленные пользователем проблемы и не создает новых;
- что система будет работать правильно при известных предлагаемых обстоятельствах;
- что система будет соответствовать описанным требованиям.

Помимо этого, тестирование предоставляет нам актуальную информацию о состоянии системы на данный момент.

5.1 Ручное тестирование

Так как библиотека опирается на возможности камеры и микрофона, автоматизировать ее тестирование невозможно. В связи с этим необходимо разработать тест-кейсы для ручного тестирования разработанной библиотеки. Таблица 5.1 содержит весь перечень тест-кейсов, которые необходимо выполнить для покрытия всего функционала библиотеки. Это позволит поддерживать надежность кода при дальнейшей разработке функций, а также проверить корректность выполнения реализованных ранее функций при изменениях.

				БГТУ 05.00.ПЗ									
	ФИО	Подпись	Дата										
Разраб.	Юшкевич И.Н.			5 Тестирование программного модуля			Лит.		Лист		Листов		
Провер.	Скребель А.С.									1		7	
Н. контр.	Николайчук А.Н.						11111111, 2023						
УТВ.	Смелов В.В.												

Таблица 5.1 – Функциональные тест-кейсы библиотеки

Описание теста	Ожидаемый результат	Статус
Отклонение разрешения на доступ к камере	Отсутствие работы библиотеки. Приложение остается в рабочем состоянии и управляется касанием	Успешно
Отклонение разрешения на доступ к микрофону	Отсутствие распознавания речи. Распознавание взгляда и мимики работает	Успешно
Вывод взгляда за пределы экрана в списке (вверх или вниз)	Пролистывание списка в сторону, в которую был выведен взгляд	Успешно
Вывод курсора за пределы экрана (влево или вправо)	Переход на следующую или предыдущую страницу	Успешно
Открытие рта на ячейке таблицы	Переход к содержимому ячейки	Успешно
Улыбка на экране содержимого ячейки	Экран закрывается. Отображается таблица	Успешно
Голосовой ввод в текстовое поле	Текстовое поле заполняется введенными голосом данными	Успешно
Сканирование лица для входа в аккаунт	Пользователь аутентифицируется в системе	Успешно
Заккрытие левого или правого глаза на экране с переключателем	Переключатель меняет свое состояние в зависимости от того, какой глаз закрыт	Успешно

Данные тест-кейсы позволяют протестировать все возможности библиотеки в рамках разработанного тестового приложения.

5.2 Тестирование разрешений

Для многих приложений отказ в доступе к некоторым сенсорам или данным телефона может полностью поменять пользовательский опыт. Например, если функционирование приложения зависит от того, где находится пользователь, то отказ в получении локационных данных может сделать использование такого приложения просто бессмысленным. В случае разработанной библиотеки, при отсутствии какого-либо разрешения большая часть ее функционала будет недоступна пользователю. Поэтому важно предупредить пользователя о преимуществах, которые он получает при согласии на доступ к функциям.

Приложение предупреждает пользователя и объясняет необходимость использования камеры устройства с помощью всплывающего окна с описанием. При отсутствии доступа к датчику, приложение продолжает свою работу без использования данного датчика, как показано на рисунке 5.1.

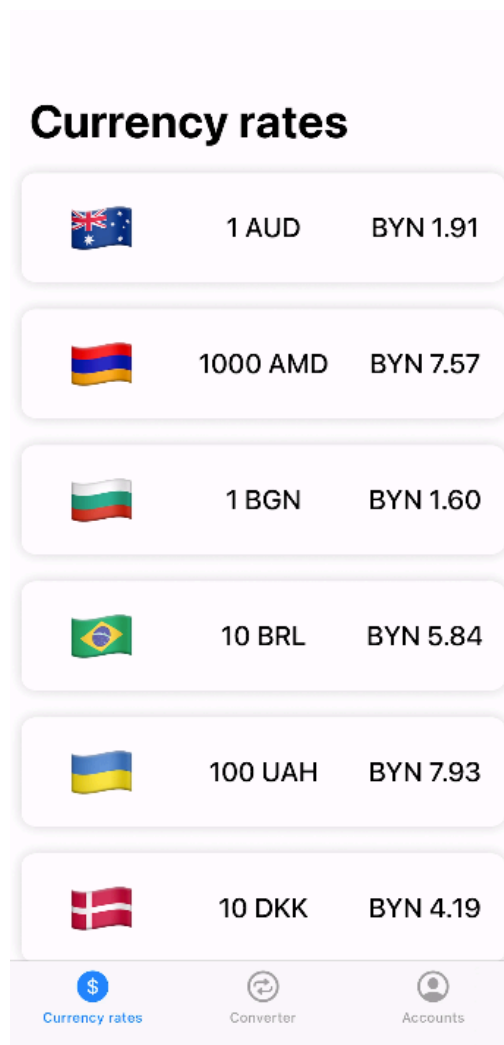


Рисунок 5.1 – Пример работы приложения без доступа к камере

Так же приложение ведет себя при отсутствии разрешений на доступ к микрофону и распознаванию речи.

5.3 Тестирование жестов системы

Все без исключения приложения используют жесты, предоставляемые операционной системой, например, пролистывание списков. Работа с приложением была бы невозможна, если бы не было возможности реализовать данное поведение с помощью разработанной библиотеки. Библиотека предоставляет все ресурсы для того, чтобы корректно реализовать пролистывание списков различными способами. На рисунке 5.2 изображено пролистывание списка вниз при перемещении курсора за нижнюю границу экрана.

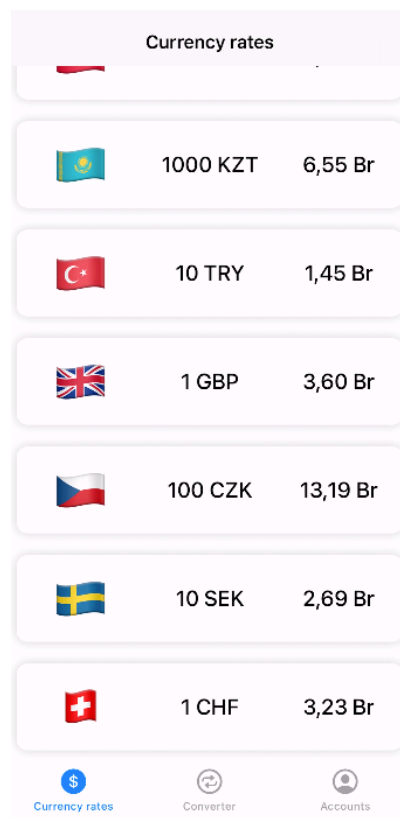


Рисунок 5.2 – Пример пролистывания списка выводом курсора

С помощью вывода курсора также реализовано перемещение по вкладкам приложения. При перемещении курсора влево или вправо, как показано на рисунке 5.3 приложение будет открывать пользователю новые вкладки.

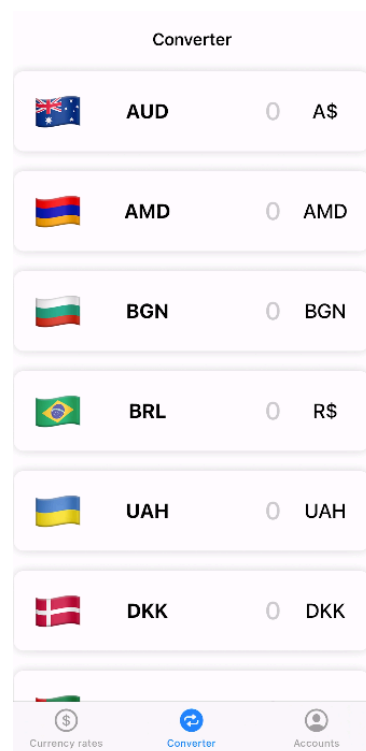


Рисунок 5.3 – Пролистывание вкладок с помощью взгляда

Также приложение обрабатывает мимику лица. В данном случае, отраженном на рисунке 5.4, оно открывает подробное описание валюты при наведении курсора на ячейку и открытии рта.

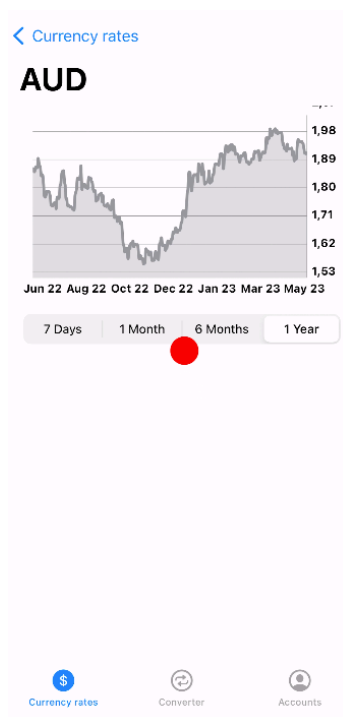


Рисунок 5.4 – Открытие подробностей с использованием мимики

Улыбка пользователя приводит к корректному закрытию окна подробностей и возврату пользователя к главному экрану приложения. Данное поведение изображено на рисунке 5.5.







Currency rates		
	1 AUD	1,91 Br
	1000 AMD	7,57 Br
	1 BGN	1,60 Br
	10 BRL	5,84 Br
	100 UAH	7,93 Br
	10 DKK	4,19 Br
Currency rates Converter Accounts		

Рисунок 5.5 – Закрытие подробностей с помощью улыбки

В окне подробностей библиотека корректно отслеживает закрытие глаз пользователя, что переключает диапазон графика, как показано на рисунке 5.6. Также библиотека контролирует случайные моргания и не производит действий при их распознавании.



Рисунок 5.6 – Смена диапазона графика при помощи моргания

5.4 Тестирование входа по лицу

Для полного управления приложением исключительно мимикой лица была предусмотрена возможность аутентификации с помощью лица. Face ID — это результат объединения аппаратных и программных компонентов Apple. Камера TrueDepth захватывает данные лица, проецируя на него и анализируя несколько тысяч невидимых точек. Таким образом устройство составляет подробную структурную карту лица, а также его изображение в инфракрасном спектре. Фрагмент системы Neural Engine, защищенный модулем Secure Enclave, преобразует структурную карту и инфракрасное изображение в математическое представление, которое сравнивается с зарегистрированными данными лица. Возможность авторизации с помощью данной технологии продемонстрирована на рисунке 5.7.

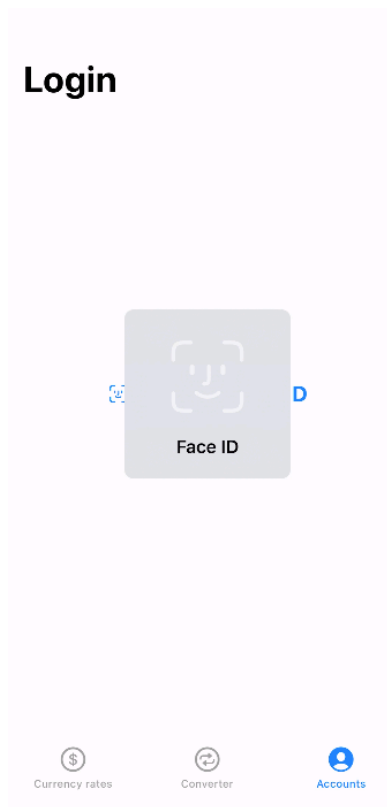


Рисунок 5.7 – Авторизация с помощью лица

Таким образом было проведено ручное тестирование всех компонент разработанной библиотеки. Все тесты прошли успешно.

5.5 Выводы по разделу

В данном разделе был описан процесс тестирования библиотеки для управления приложением с использованием взгляда и мимики лица. Тестирование проводилось по предварительно написанным тест-кейсам, а его результаты помещены в таблицу 5.1.

Полученные результаты при ручном и автоматическом функциональном тестировании показали, что разработанное программное средство работает правильно и корректно.

6 Руководство пользователя

В данном разделе дипломного проектирования, будут рассмотрены основные аспекты интеграции разработанной библиотеки в приложение.

6.1 Общие сведения

Дипломное проектирование заключается в создании библиотеки, встраиваемой в приложение с целью реализации в нем управления с помощью взгляда, мимики лица и голосового ввода.

6.2 Документация

При разработке библиотеки важным фактором является наличие документации для нее внутри среды разработки. Это позволяет программисту быстрее получать информацию о необходимых ему функциях библиотеки, а также дает возможность программистам, участвующим в разработке библиотеки, быстрее понять необходимость разрабатываемой ими функции.

В данной библиотеке документация написана с помощью аннотаций, представленных в листинге 6.1. Документация также генерируется в формате DocS, что позволяет просматривать ее из любой точки программы или библиотеки. Пример отображения документации представлен на рисунке 6.1.

```
/**
 * Method responsible for any frame updates from ARSession.
 * - Parameters:
 * - parameter session: Represents ``ARSession`` in progress. Its status
 * can be checked with ``isSessionInProgress`` property in ``Session`` class.
 * - parameter frame: Updated ``ARFrame``.
 */
```

Листинг 6.1 – Аннотация для создания документации

				БГТУ 06.00.ПЗ								
	ФИО	Подпись	Дата									
Разраб.	Юшкевич И.Н.			6 Руководство пользователя			Лит.		Лист		Листов	
Провер.	Скребель А.С.									1	6	
Н. контр.	Николайчук А.Н.						11111111, 2023					
Утв.	Смелов В.В.											

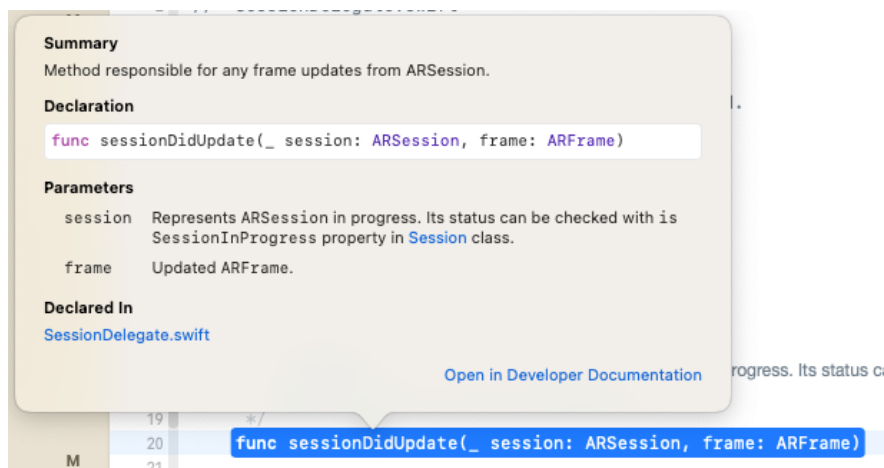


Рисунок 6.1 – Отображение документации в среде разработки

6.3 Интеграция библиотеки

Для интеграции библиотеки разработчик должен установить менеджер пакетов CocoaPods, а также интерпретатор языка Ruby.

Далее разработчику необходимо инициализировать менеджер пакетов в своем проекте с помощью команды `pod init`. После вызова данной команды в папке проекта появится рабочее пространство, содержащее проект и все его зависимости, а также файл Podfile, который содержит названия всех зависимостей проекта.

Для добавления библиотеки в качестве зависимости проекта необходимо открыть Podfile и указать его в качестве одной из зависимостей. Пример приведен на рисунке 6.2.

```
# Uncomment the next line to define a global platform for your project
platform :ios, '15.0'
workspace 'converterapp.xcworkspace'

use_frameworks!

def shared_pods
  pod 'Alamofire'
  pod 'FirebaseAnalytics'
  pod 'Firebase/Crashlytics'
  pod 'Firebase/RemoteConfig'
  pod 'FirebaseAuth'
  pod 'Swinject'
  pod 'EyeTracking', :path => '../EyeTracking'
end

target 'converterapp' do
  xcproj 'converterapp'
  pod 'SwiftGen', '~> 6.0'
  pod 'Charts', '~> 4.1.0'
  shared_pods
end

target 'converterappCore' do
  xcproj 'Modules/converterappCore/converterappCore.xcproj'
  shared_pods
end
```

Рисунок 6.2 – Добавление библиотеки в качестве зависимости

После добавления библиотеки в Podfile необходимо обновить зависимости в рабочем пространстве с помощью команды `pod install`.

Далее можно открыть рабочее пространство и проверить корректность импорта зависимости, написав `import EyeTracking` в любом файле, как показано на рисунке 6.3.

```
import EyeTracking
```

Рисунок 6.3 – Импорт библиотеки в существующий проект

Для корректной работы библиотеки необходимо инициализировать ее при старте программы. Осуществляется это с помощью инициализации сессии и запуска менеджера в классе AppDelegate, отвечающем за жизненный цикл приложения. Пример отображения документации представлен на рисунке 6.2.

```
func application(_ application: UIApplication,
didFinishLaunchingWithOptions launchOptions:
[UIApplication.LaunchOptionsKey: Any]?) -> Bool {
    keychainService.save("111111", service: "Firebase", account:
"default@mail.ru")
    UserDefaults.standard.set("default@mail.ru", forKey: "latestEmail")
    FirebaseApp.configure()

    FirebaseRemoteConfig.shared.fetchParameters()

    do {
        try session.start()
    } catch let error {
        print(error)
    }

    TrackingManager.shared.start(with: session)
    return true
}
```

Листинг 6.2 – Инициализация библиотеки в приложении

В результате программист получает инициализированную библиотеку с единой точкой входа в классе TrackingManager, которая готова к использованию в любой точке приложения.

6.4 Отслеживание мимики

Для реализации отслеживания мимики необходимо определить точку входа для библиотеки в классе, который рисует пользовательский интерфейс, как это реализовано в листинге 6.3 на примере класса RatesViewController.

```
private let manager = TrackingManager.shared
```

Листинг 6.3 – Инициализация точки входа для класса

Далее программисту необходимо подписать класс на события класса, отслеживающего мимику. Делается это путем добавления класса-слушателя в

качестве делегата класса, отслеживающего мимику, а также регистрации типов мимики, которые необходимо отслеживать, как показано на листинге 6.4.

```
private func setupEyeTracking() {
    manager.eyeTracker.setDelegate(self)
    manager.faceTracker.setDelegate(self)
    manager.faceTracker.initiateFaceExpression(FaceExpression(blendShape:
        .jawOpen, minValue: 0.3, maxValue: 1))
}
```

Листинг 6.4 – Регистрация класса для отслеживания мимики

Для реализации собственного поведения при получении события, необходимо реализовать метод интерфейса `FaceTrackerDelegate`. Пример реализации метода для класса `RatesViewController` приведен в листинге 6.5.

```
extension RatesViewController: EyeTrackerDelegate, FaceTrackerDelegate {
    public func faceTracker(_ faceTracker: FaceTracker, didUpdateExpression
        expression: FaceExpression) {
        manager.eyeTracker.delegates.forEach { delegate in
            delegate?.eyeTracking(manager.eyeTracker, didUpdateState:
                manager.eyeTracker.state, with: expression)
        }
    }
}
```

Листинг 6.5 – Обработчик события мимики

Таким образом реализуется отслеживание мимики лица на стороне приложения с использованием разработанной библиотеки.

6.5 Отслеживание взгляда

Для реализации отслеживания взгляда необходимо также определить точку входа и подписать класс на события класса, отслеживающего взгляд, как это было реализовано в предыдущем пункте. В то же время отслеживание взгляда не требует дополнительных регистраций, как например регистрация мимики, без которой невозможна работа класса, отслеживающего мимику. Для реализации обработчика событий для взгляда достаточно реализовать метод делегата `EyeTrackerDelegate`. Реализация данного делегата представлена в листинге 6.6.

```

extension RatesViewController: EyeTrackerDelegate, FaceTrackerDelegate {
    func eyeTracking(_ eyeTracker: EyeTracking.EyeTracker, didUpdateState
state: EyeTracking.EyeTracker.TrackingState, with expression:
EyeTracking.FaceExpression?) {
        switch state {
        case .screenIn(let point):
            guard let expression else { return }
            switch expression.blendShape {
            case .jawOpen:
                hitCell(at: point)
            default:
                return
            }
        case .screenOut(let edge, _):
            switch edge {
            case .left, .right:
                return
            case .top:
                scrollView(-6)
            case .bottom:
                scrollView(6)
            }
        }
    }
}

```

Листинг 6.6 – Обработчик события взгляда

В результате проведенных действий разработчик получает возможность управлять приложением в зависимости от направления взгляда пользователя.

6.6 Распознавание речи

Библиотека позволяет инициировать распознавание речи для любого элемента интерфейса, включая кнопки и переключатели. Однако наиболее распространенное применение распознавание речи имеет при заполнении текстовых полей. Рассмотрим реализацию текстового поля с распознаванием речи на примере класса-наследника UITextField.

Для начала необходимо инициализировать распознавание речи с помощью публичного конструктора, как показано в листинге 6.7.

```
private var speechRecognizer = SpeechRecognizer()
```

Листинг 6.7 – Инициализация распознавания текста

Затем при начале изменения текстового поля необходимо запустить распознавание речи, как указано в листинге 6.7.

```
@objc
private func editingDidBegin() {
    speechRecognizer.resetTranscript()
    speechRecognizer.startTranscribing()
}
```

Листинг 6.7 – Старт распознавания текста

При завершении изменения текстового поля необходимо остановить распознавание речи и предоставить результат пользователю внутри текстового поля, что реализовано в листинге 6.8.

```
@objc
private func editingDidEnd() {
    speechRecognizer.stopTranscribing()
    text = speechRecognizer.transcript
}
```

Листинг 6.8 – Остановка распознавания текста

В результате данных операций разработчик получает текстовое поле, управляемое с помощью голосового ввода.

6.7 Выводы по разделу

Руководство пользователя было составлено в соответствии с функционалом разработанной библиотеки. В разделе описаны основные аспекты работы с библиотекой, а также разъяснена работа с документацией данной библиотеки.

7 Технико-экономическое обоснование проекта

7.1 Общая характеристика разрабатываемого программного средства

При выполнении данного дипломного проект была разработана библиотека для управления мобильным приложением с помощью взгляда и мимики лица, а также тестовое приложение для подтверждения работоспособности библиотеки. Основной целью библиотеки является предоставление возможности управлять приложением без использования рук для, например, людей с ограниченными возможностями.

Данный дипломный проект был разработан при помощи iOS SDK и фреймворка ARKit на языке Swift версии 5. Также при разработке были задействованы фреймворк для реализации пользовательского интерфейса UIKit и библиотека для распознавания речи SpeechRecognizer.

Разработанное программное обеспечение является уникальным, так как найденные аналоги применимы только к персональным компьютерам и их имплементация в мобильных устройствах невозможна. Нативная реализация для платформы на языке Swift делает библиотеку более быстрой, оптимизированной и гибкой в рамках операционных систем Apple. Библиотеку можно достаточно просто портировать на такие операционные системы, как macOS, iPadOS и tvOS. Также библиотеку можно портировать на язык Objective-C с помощью классов-оберток, что позволит охватить большее количество программистов и программных решений.

По результатам анализа применяемых продуктами-аналогами стратегий монетизации следует выбрать стратегию монетизации: годовая подписка на использование продукта, где компания-разработчик внедряет в свое программное обеспечение разработанную библиотеку с целью охвата большего количества пользователей.

7.2 Исходные данные для проведения расчетов и маркетинговый анализ

Источниками исходных данных для данных расчетов выступают действующие законы и нормативно-правовые акты. Исходные данные для расчета приведены в таблице 7.1.

Таблица 7.1 – Исходные данные для расчета

Наименование показателя	Условные обозначения	Норматив
Численность разработчиков, чел	Ч _р	1,00

				БГТУ 07.00.ПЗ							
	ФИО	Подпись	Дата								
Разраб.	Юшкевич И.Н.			7 Технико-экономическое обоснование проекта				Лит.	Лист	Листов	
Провер.	Скребель А.С.									1	8
Консульт.	Семёнова Л.С							11111111, 2023			
Н. контр.	Николайчук А.Н.										
Утв.	Смелов В.В.										

Окончание таблицы 7.1

Наименование показателя	Условные обозначения	Норматив
Норматив дополнительной заработной платы, %	$H_{дз}$	15,00
Ставка отчислений в Фонд социальной защиты населения, %	$H_{фсзн}$	34,00
Ставка отчислений в БРУСП «Белгосстрах», %	$H_{бгс}$	0,40
Норматив прочих прямых затрат, %	$H_{пз}$	20
Норматив накладных расходов, %	$H_{обп,обх}$	10
Норматив расходов на реализацию, %	H_{pp}	7

7.3 Обоснование цены программного средства

В современных рыночных экономических условиях программное средство (ПС) выступает преимущественно в виде продукции организаций, представляющей собой функционально завершенные и имеющие товарный вид, реализуемые покупателям по рыночным отпускным ценам. Все завершенные разработки являются научно-технической продукцией.

Широкое применение вычислительных технологий требует постоянного обновления и совершенствования ПС. Выбор эффективных проектов ПС связан с их экономической оценкой и расчетом экономического эффекта, который может определяться как у разработчика, так и у пользователя.

У разработчика экономический эффект выступает в виде чистой прибыли от реализации ПС, остающейся в распоряжении организации, а у пользователя – в виде экономии трудовых, материальных и финансовых ресурсов, получаемой за счет:

- снижения трудоемкости расчетов и алгоритмизации программирования и отладки программ;
- сокращения расходов на оплату машинного времени и других ресурсов на отладку программных средств;
- снижения расходов на материалы;
- ускорение ввода в эксплуатацию новых систем;
- улучшения показателей основной деятельности в результате использования передовых программных средств.

Стоимостная оценка библиотеки у разработчиков предполагает определение затрат, что включает следующие статьи:

- заработная плата исполнителей – основная и дополнительная;
- отчисления в Фонд социальной защиты населения и обязательному страхованию от несчастных случаев на производстве и профессиональных заболеваний;
- расходы на материалы и комплектующие;
- расходы на спецоборудование;
- расходы на оплату машинного времени;
- общепроизводственные и общехозяйственные расходы;
- расходы, связанные с установкой и адаптацией (реализационные или коммерческие расходы).

На основании затрат рассчитывается себестоимость и отпускная цена разработанной библиотеки.

7.3.1 Расчёт затрат рабочего времени на разработку программного средства

Ниже приведена таблица с информацией о затратах рабочего времени на разработку программного средства.

Таблица 6.2 – Затраты рабочего времени на разработку ПС

Содержание работ	Затраты рабочего времени, часов
Проектирование библиотеки	8
Настройка окружения для разработки	24
Написание логики отслеживания взгляда и его проецирования на экран	56
Реализация управления взглядом	32
Реализация отслеживания мимики лица	40
Реализация управления мимикой лица	24
Реализация распознавания речи	40
Реализация тестового приложения	60
Тестирование библиотеки	48
Всего	332

Результаты таблицы будут использованы далее для расчета заработной платы.

7.3.2 Расчёт основной заработной платы

Для определения величины основной заработной платы, был проведен анализ заработных плат для iOS специалистов. В итоге было установлено, что средняя месячная заработная плата на позиции junior составляет 2 100 рублей, часовая ставка составляет 9,375 руб./час.

Согласно таблице 6.2, проект разрабатывался одним специалистом на протяжении 196 часов. Таким образом, основная заработная плата будет рассчитываться по формуле 7.1.

$$C_{\text{оз}} = T_{\text{раз}} \cdot C_{\text{зп}} \cdot K_{\text{раз}}, \quad (7.2)$$

где $C_{\text{оз}}$ – основная заработная плата, руб.;

$T_{\text{раз}}$ – время разработки (часов);

$C_{\text{зп}}$ – средняя часовая ставка руб./час;

$K_{\text{раз}}$ – количество разработчиков, человек.

$$C_{\text{оз}} = 332 \cdot 9,375 = 1837,50 \text{ руб}$$

В дальнейшем для других расчётов используется основная заработная плата, рассчитанная по указанной выше методике.

7.3.2 Расчёт дополнительной заработной платы

Дополнительная заработная плата на конкретное программное средство включает выплаты, предусмотренные законодательством о труде, и определяется по нормативу в процентах к основной заработной плате по формуле 7.2

$$C_{\text{дз}} = \frac{C_{\text{оз}} \cdot H_{\text{дз}}}{100}, \quad (7.2)$$

где $C_{\text{оз}}$ – основная заработная плата, руб.;

$H_{\text{дз}}$ – норматив дополнительной заработной платы, %.

$$C_{\text{дз}} = 1837,50 \cdot 15 / 100 = 275,63 \text{ руб.}$$

7.3.3 Отчисления в Фонд социальной защиты населения и Белгосстрах

Отчисления в Фонд социальной защиты населения и Белгосстрах (ФСЗН) определяются в соответствии с действующими законодательными актами по нормативу в процентном отношении к фонду основной и дополнительной зарплаты исполнителей и вычисляются по формуле 7.3

$$C_{\text{фсзн}} = \frac{(C_{\text{оз}} + C_{\text{дз}}) \cdot H_{\text{фсзн}}}{100}, \quad (7.3)$$

где $C_{\text{оз}}$ – основная заработная плата, руб.;

$C_{\text{дз}}$ – дополнительная заработная плата, руб.;

$H_{\text{фсзн}}$ – ставка отчислений в Фонд социальной защиты населения, %.

Отчисления в БРУСП «Белгосстрах» вычисляются по формуле 7.4

$$C_{\text{бгс}} = \frac{(C_{\text{оз}} + C_{\text{дз}}) \cdot H_{\text{бгс}}}{100} \quad (7.4)$$

$$C_{\text{фсзн}} = (1837,50 + 275,63) \cdot 34 / 100 = 718,46 \text{ руб.}$$

$$C_{\text{бгс}} = (1837,50 + 275,63) \cdot 0,4 / 100 = 8,45 \text{ руб.}$$

Таким образом, общие отчисления в БРУСП «Белгосстрах» составили 8,45 руб., а в фонд социальной защиты населения – 718,46 руб.

7.3.4 Расчёт суммы прочих прямых затрат

Сумма прочих затрат $C_{\text{пз}}$ определяется как произведение основной заработной платы исполнителей на конкретное программное средство $C_{\text{оз}}$ на норматив прочих затрат в целом по организации $H_{\text{пз}}$ согласно формуле 7.5

$$C_{пз} = \frac{C_{оз} \cdot H_{пз}}{100} \quad (7.5)$$

$$C_{пз} = 1837,50 \cdot 20 / 100 = 367,50 \text{ руб.}$$

7.3.4 Расчёт суммы накладных расходов

Сумма накладных расходов $C_{обп, обх}$ – произведение основной заработной платы исполнителей на конкретное программное средство $C_{оз}$ на норматив накладных расходов в целом по организации $H_{обп, обх}$ согласно формуле 7.6

$$C_{обп, обх} = \frac{C_{оз} \cdot H_{обп, обх}}{100} \quad (7.6)$$

Все данные необходимые для вычисления есть, поэтому можно определить сумму накладных расходов:

$$C_{обп, обх} = 1837,50 \cdot 10 / 100 = 183,75 \text{ руб.}$$

7.3.4 Сумма расходов на разработку программного средства

Сумма расходов на разработку веб-приложения C_p определяется как сумма основной и дополнительной заработных плат исполнителей на конкретное программное средство, отчислений на социальные нужды, расходов на материалы, расходов на оплату машинного времени, суммы прочих затрат и суммы накладных расходов согласно формуле 7.7.

$$C_p = C_{оз} + C_{дз} + C_{фсзн} + C_{бгс} + C_{пз} + C_{обп, обх} \quad (7.7)$$

$$C_p = 1837,50 + 275,63 + 718,46 + 8,45 + 367,50 + 183,75 = 3391,29 \text{ руб.}$$

Сумма расходов на разработку веб-приложения была вычислена на основе данных, рассчитанных ранее в данном разделе.

Таким образом, сумма расходов на разработку библиотеки составила 3391,29 рублей.

7.3.5 Расходы на реализацию

Сумма расходов на реализацию программного средства C_{pp} определяется как произведение суммы расходов на разработку на норматив расходов на реализацию H_{pp} , и находится по формуле 7.8.

$$C_{pca} = \frac{C_p \cdot H_{pp}}{100}, \quad (7.8)$$

$$C_{pp} = 3391,29 \cdot 7 / 100 = 237,39 \text{ руб.}$$

Все проведенные выше расчеты необходимы для вычисления полной себестоимости проекта.

7.3.5 Расчет полной себестоимости

Полная себестоимость C_{π} определяется как сумма двух элементов: суммы расходов на разработку C_p и суммы расходов на сопровождение и адаптацию веб-приложения C_{pca} согласно формуле 7.9

$$C_{\pi} = C_p + C_{pca}, \quad (7.9)$$

где C_{π} – полная себестоимость веб-приложения, руб.;

C_p – сумма расходов на разработку веб-приложения, руб.;

C_{pca} – сумма расходов на сопровождение и адаптацию веб-приложения, руб.

$$C_{\pi} = 3391,29 + 237,39 = 3628,68 \text{ руб.}$$

Полная себестоимость составила 3628,68 рублей.

7.3.6 Определение цены, оценка эффективности

Так как монетизация продукта осуществляется путем предоставления библиотеки для управления приложением. Необходимо определить сумму денежных поступлений и окупаемость затрат на разработку библиотеки, т.е. целесообразность и эффективность.

Для этого необходимо приблизительно рассчитать количество подписок/установок/продаж на основании следующих данных о продуктах-аналогах:

1. *OpenCV* – библиотека с открытым исходным кодом для обработки изображений и компьютерного зрения, которая предлагает множество функций для работы с изображениями и видео, а также функции для обработки и распознавания объектов. Она предоставляет простой интерфейс для использования и может работать на всех основных операционных системах.

2. *Dlib* – одна из наиболее популярных библиотек с открытым исходным кодом для машинного обучения и компьютерного зрения, которая предназначена для разработки приложений в области распознавания объектов, нахождения лиц и оценки эмоционального состояния человека по изображению.

3. *Face++ SDK* – библиотека для распознавания и анализа лиц, разработанная компанией Megvii. Она используется разработчиками для создания приложений с функциями распознавания лиц, проверки подлинности.

По результатам анализа применяемых продуктами-аналогами стратегий монетизации следует выбрать стратегию монетизации: предоставление услуги по интеграции библиотеки в продукт. Были выбраны следующие характеристики для показателей качества рассматриваемого программного продукта и программного продукта конкурента:

- 1) Юзабилити – насколько удобно использовать API библиотеки
- 2) Функциональность – количество инноваций, внедряемых в приложение
- 3) Отсутствие багов – отсутствие несущественных ошибок в работе.

Расчет показателей качества базового и нового продуктов согласно балловому методу приводится в таблице 7.3.

Таблица 7.3 – Оценка качества программного средства

Показатель качества	Весовой коэффициент	Разрабатываемый продукт	Продукт-аналог 1	Продукт-аналог 2	Продукт-аналог 3
Юзабилити	0,4	7	9	6	5
Функциональность	0,5	9	7	7	7
Отсутствие багов	0,1	7	8	7	6
Всего	1	7,7	8	6,7	6

Исходя из таблицы 7.3 можно сделать вывод что наш продукт выгодно отличается от продуктов наших конкурентов и имеет выгодный баланс возможностей, дизайна и качества кода.

Расчёт прогнозного количества заказов программного средства рассчитывается исходя из среднего значения заказов по каждому аналогу. Количество заказов K аналогов при монетизации методом покупок услуги, скорректированная на оценку показателей качества, рассчитывается по формуле 7.10:

$$K_1 = (K_0 / T_0 \cdot \text{ИР}) / \text{ИК} \quad (7.10)$$

где K_0 – количество заказов ПС конкурента;

T_0 – количество лет существования приложения;

ИР – показатель рассматриваемого программного продукта;

ИК – показатель программного продукта конкурента.

$$\begin{aligned} K_1 &= (5\,000 / 23 \cdot 7,7) / 8 = 244,57 \text{ (заказов в год)}, \\ K_2 &= (5\,000 / 21 \cdot 7,7) / 6,7 = 316,16 \text{ (заказов в год)}, \\ K_3 &= (5\,000 / 6 \cdot 7,7) / 6 = 1038,46 \text{ (заказов в год)}, \\ K &= (244,57 + 316,16 + 1038,46) / 3 = 533,06 \approx 533 \text{ (заказов в год)}. \end{aligned}$$

Цена подписки Π нового продукта рассчитывается исходя из среднего значения цены по каждому аналогу. Цена Π установки аналога, скорректированная на оценку показателей качества, рассчитывается по формуле:

$$\Pi_1 = (\Pi_0 \cdot \text{ИР}) / \text{ИК}, \quad (7.11)$$

где Π_0 – цена программного продукта конкурента,

ИР – показатель рассматриваемого программного продукта,

ИК – показатель программного продукта конкурента.

$$\begin{aligned} \Pi_1 &= (0 \cdot 7,7) / 8 = 0 \text{ рубля} \\ \Pi_2 &= (0 \cdot 7,7) / 6,7 = 0 \text{ рубля} \\ \Pi_3 &= (3480 \cdot 7,7) / 6 = 4336,62 \text{ рубля} \\ \Pi &= (0 + 0 + 4336,62) / 3 = 1445,54 \text{ рубля} \end{aligned}$$

По данным расчета рыночной цены рассмотренных конкурентов такого решения, если среднее количество заказов 533 за год, то денежные поступления от продажи подписки год составят 770571,63 рублей.

Количество покупателей продукта необходимых для окупаемости Π_n вычисляется по формуле 7.12:

$$P_{\Pi} = C_{\Pi} / \Pi, \quad (7.12)$$

$$P_{\Pi} = 3628,68 / 1445,53 = 3 \text{ покупателей.}$$

7.4 Вывод по разделу

Разработка программного средства, осуществляемая одним программистом в течении 332 часов, при заданных условиях обойдется в 3628,68 руб. Реализации данного программного средства будет приносить годовые денежные поступления от продажи подписки в размере 770571,63 рублей и окупится при продаже 3 услуг в год. Необходимо написать сколько в среднем установок в год, ориентировочно количество установок 5000, отсюда делаем предположение что проект окупится.

Необходимость разработки библиотеки, обусловлена увеличением внимания к проблемам людей с ограниченными возможностями. Библиотека может занять нишу с малым количеством конкурентов, а, учитывая превосходство в функционале над конкурентами, она может стать востребованным продуктом на рынке.

Заключение

В рамках работы над проектом был проведен обзор аналогичных решений, выбраны язык и платформа для разработки библиотеки, а также тестового приложения к ней.

Были написаны тест-кейсы, реализация которых позволила повысить качество кода в библиотеке.

В пояснительной записке приведено руководство пользователя по интеграции библиотеки в проект и использованию отдельных ее частей в нем.

В экономической части пояснительной записки были проведены расчеты, которые показали целесообразность реализации данной библиотеки и ее монетизации, а также ее конкурентоспособность на рынке.

Конечный продукт предоставляет возможность управления приложением с помощью взгляда и мимики лица, что позволяет конечным пользователям использовать мобильное приложение без использования рук.

Программное средство соответствует целям дипломного проекта, реализует все поставленные перед ним задачи.

				БГТУ 00.00.ПЗ		
	ФИО	Подпись	Дата	Заключение		
Разраб.	Юшкевич И.Н.					
Провер.	Скребель А.С.					
Н. контр.	Николайчук А.Н.					
УТВ.	Смелов В.В.			11111111, 2023		
				Лит.	Лист	Листов
					1	1

Список использованных источников

- 1 Hoster.by [Электронный ресурс] // hoster.by. – Режим доступа: <https://hoster.by/>. – Дата доступа: 11.04.2023.
- 2 HostFly [Электронный ресурс] // hostfly.by. – Режим доступа: <https://www.hostfly.by/>. – Дата доступа: 11.04.2023.
- 3 ActiveCloud [Электронный ресурс] // activecloud.by. – Режим доступа: <https://www.activecloud.by/>. – Дата доступа: 12.04.2023.
- 4 DomainBy [Электронный ресурс] // domain.by. – Режим доступа: <https://domain.by/>. – Дата доступа: 12.04.2023.
- 5 Либерти Д. Язык программирования C# // Программирование на C#. – Санкт-Петербург. – 2003: Символ-Плюс. – С. 26. – 688 с. – ISBN 5-93286-038-3.
- 6 Введение в ASP.NET Core MVC – Краткое описание [Электронный ресурс] / metanit.com. – Режим доступа: <https://metanit.com/sharp/aspnet5/3.1.php/>. – Дата доступа: 08.04.2023.
- 7 Обзор .NET Core [Электронный ресурс] // learn.microsoft.com – Режим доступа: <https://learn.microsoft.com/ru-ru/dotnet/core/introduction>. – Дата доступа: 15.04.2023.
- 8 Декостер К. Pro NuGet (Expert's Voice in Microsoft) – Нью-Йорк. – 2012: Apress – С. 40. – 256 с. – ISBN 1-43024-191-8.
- 9 Microsoft Visual Studio [Электронный ресурс] // microsoft.com. – Режим доступа: <https://visualstudio.microsoft.com/ru/>. – Дата доступа 15.04.2023.
- 10 Краузе Дж. Mastering Windows Server 2019 – Бирмингем, Великобритания. – 2019: Packt Publishing С. 16. – 406 с. – ISBN 978-1-78980-453-9.
- 11 RFC 2616 Hypertext Transfer Protocol – HTTP 1/1 [Электронный ресурс] // tools.ietf.org. – Режим доступа: <https://tools.ietf.org/html/rfc2616/>. – Дата доступа: 16.04.2023.
- 12 SQL Server technical documentation [Электронный ресурс] // learn.microsoft.com. – Режим доступа: <https://learn.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver16>. – Дата доступа: 16.04.2023.
- 13 Введение в Entity Framework Core [Электронный ресурс] // metanit.com. – Режим доступа: <https://metanit.com/sharp/entityframeworkcore/1.1.php/>. – Дата доступа: 17.04.2023.
- 14 LINQ to SQL [Электронный ресурс] // metanit.com. – Режим доступа: <https://metanit.com/sharp/adonet/4.1.php> – Дата доступа: 17.04.2023.
- 15 FluentValidation Documentation [Электронный ресурс] // docs.fluentvalidation.net – Режим доступа: <https://docs.fluentvalidation.net/en/latest/> – Дата доступа: 17.04.2023.

				БГТУ 00.00.ПЗ			
	ФИО	Подпись	Дата	Список использованных источников			
Разраб.	Юшкевич И.Н.						
Провер.	Скребель А.С.						
Н. контр.	Николайчук А.Н.						
УТВ.	Смелов В.В.			11111111, 2023			

16 Учебник. Начало работы с Razor Pages в Asp.Net Core [Электронный ресурс] // learn.microsoft.com – Режим доступа: <https://learn.microsoft.com/ru-ru/aspnet/core/tutorials/razor-pages/razor-pages-start?view=aspnetcore-7.0&tabs=visual-studio>. – Дата доступа 17.04.2023.

17 Bootstrap (front-end framework) [Электронный ресурс] / getbootstrap.com – Режим доступа: <https://getbootstrap.com/docs/5.0/layout/grid/>. – Дата доступа: 09.04.2023.

18 Гранд К. CSS in Depth – Нью-Йорк. – 2018: Manning – С. 37. – 364 с. – ISBN 9-78161729-345-0.

19 Openprovider documentation [Электронный ресурс] // docs.openprovider.com – Режим доступа: <https://docs.openprovider.com/doc/all> – Дата доступа: 17.04.2023.

20 Академик [Электронный ресурс] / dic.academic.ru. – Режим доступа: <https://dic.academic.ru/dic.nsf/ruwiki/146913/>. – Дата доступа: 11.04.2023.

21 Тестирование. Фундаментальная теория [Электронный ресурс] // habr.com. – Режим доступа: <https://habr.com/ru/post/279535/>. – Дата доступа 01.05.2023.

22 About xUnit.net [Электронный ресурс] // xunit.net. – Режим доступа: <https://xunit.net/>. – Дата доступа: 02.05.2023.

ПРИЛОЖЕНИЕ А

Реализация отслеживания лица в OpenCV

```
import cv2
import numpy as np
import pyautogui
# Функция для детектирования лица с использованием алгоритма Haar Cascade
def detect_face(gray):
    face_cascade =
cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
    faces = face_cascade.detectMultiScale(gray, 1.3, 5)
    if len(faces) == 0:
        return None, None
    (x, y, w, h) = faces[0]
    return gray[y:y+w, x:x+h], faces[0]
# Функция для детектирования глаз с использованием алгоритма Haar Cascade
def detect_eyes(gray):
    eye_cascade = cv2.CascadeClassifier('haarcascade_eye.xml')
    eyes = eye_cascade.detectMultiScale(gray, 1.3, 5)
    if len(eyes) == 0:
        return None, None
    (x, y, w, h) = eyes[0]
    return gray[y:y+w, x:x+h], eyes[0]
# Считываем поток видео с камеры
cap = cv2.VideoCapture(0)
# Цикл для обработки каждого кадра видео
while True:
    ret, frame = cap.read()
    if not ret:
        break
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # Детектирование лица
    face, rect = detect_face(gray)
    if rect is not None:
        (x, y, w, h) = rect
        # Детектирование глаз
        eye_gray, eyes_rect = detect_eyes(gray[y:y+h, x:x+w])
        # Определение координат глаз
        if eyes_rect is not None:
            (ex, ey, ew, eh) = eyes_rect
            gaze_x = ex + ew//2 + x
            gaze_y = ey + eh//2 + y
            # Управление приложением
            if gaze_x >= 100 and gaze_x <= 200 and gaze_y >= 100 and gaze_y
<= 200:
                pyautogui.click()
            # Детектирование жестов лица
            smiles = cv2.CascadeClassifier('haarcascade_smile.xml')
            smile = smiles.detectMultiScale(gray, scaleFactor=1.7,
minNeighbors=20)
            for (sx, sy, sw, sh) in smile:
```

```
        pyautogui.hotkey('command', 'tab')
# Отображение кадра с маркерами
cv2.imshow('Face Detection', frame)
# Закрытие окна по нажатию на клавишу 'q'
if cv2.waitKey(1) & 0xFF == ord('q'):
    break
# Освобождение ресурсов и закрытие окна
cap.release()
cv2.destroyAllWindows()
```

ПРИЛОЖЕНИЕ Б

Реализация отслеживания лица в Dlib

```
import dlib
import cv2
import numpy as np
# Set up the detector and the predictor
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
# Set up the video capture
cap = cv2.VideoCapture(0)
count = 0
while True:
    # Read the frame from the camera
    ret, frame = cap.read()
    # Convert the frame to grayscale
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    # Detect faces in the grayscale frame
    faces = detector(gray)
    # If a face is detected
    if len(faces) > 0:
        # Get the facial landmarks for the first face
        landmarks = predictor(gray, faces[0])
        # Get the position of the eyes
        left_eye = (landmarks.part(36).x, landmarks.part(36).y)
        right_eye = (landmarks.part(45).x, landmarks.part(45).y)
        # Calculate the center of the eyes
        eye_center = ((left_eye[0] + right_eye[0]) // 2, (left_eye[1] +
right_eye[1]) // 2)
        # Draw a circle at the center of the eyes
        cv2.circle(frame, eye_center, 5, (0, 255, 0), -1)
        # Get the position of the mouth
        mouth = (landmarks.part(62).x, landmarks.part(62).y)
        # If the mouth is open wide enough
        if mouth[1] - eye_center[1] > 20:
            # Do something here, like simulate a mouse click
            # Increment the count
            count += 1
            # If the count reaches a certain value
            if count == 10:
                # Do something else here, like close the application
                # Reset the count
                count = 0
    # Display the frame
    cv2.imshow("frame", frame)
    # If the user presses "q"
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
# Release the capture and destroy the windows
cap.release()
cv2.destroyAllWindows()
```

ПРИЛОЖЕНИЕ В

Реализация отслеживания лица в Face++ SDK

```
import cv2
import facepp
# Initialize Face++ API
api_key = '<your_api_key>'
api_secret = '<your_api_secret>'
face_api = facepp.API(api_key=api_key, api_secret=api_secret)
# Open the video capture device
cap = cv2.VideoCapture(0)
# Main loop
while True:
    # Read an image from the video capture device
    ret, img = cap.read()
    # Detect faces and eyes using Face++ API
    result = face_api.detect(image_file=img, return_landmark=1)
    # Get the position of the eyes and mouth
    left_eye_pos = result['faces'][0]['landmark']['left_eye'][0]
    right_eye_pos = result['faces'][0]['landmark']['right_eye'][0]
    mouth_pos = result['faces'][0]['landmark']['mouth'][0]
    # Calculate the center of the eyes and the position of the nose
    eye_center = ((left_eye_pos['x'] + right_eye_pos['x']) // 2,
(left_eye_pos['y'] + right_eye_pos['y']) // 2)
    nose_pos = result['faces'][0]['landmark']['nose'][0]
    # Check if the mouth is open wide enough
    if mouth_pos['y'] - eye_center[1] > 20:
        # Perform some action, like simulating a mouse click or closing the
window
        pass
    # Display the image with annotations
    cv2.imshow('face tracking', img)
    # If the 'q' key is pressed, quit the program
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
# Clean up
cap.release()
cv2.destroyAllWindows()
```

ПРИЛОЖЕНИЕ Д

API AR сессии

```
public func start(with config: ARFaceTrackingConfiguration? = nil,
                  options: ARSession.RunOptions =
[.resetTracking, .removeExistingAnchors]) throws {
    guard ARFaceTrackingConfiguration.isSupported else { throw
SessionError.arNotSupported }

    if isSessionInProgress { return }

    if let config {
        faceTrackingConfiguration = config
    }

    arSession.delegate = self
    arSession.run(faceTrackingConfiguration, options: options)

    isSessionInProgress = true
}

public func end() throws {
    guard isSessionInProgress else { throw
SessionError.noSessionsInProgress }

    arSession.pause()
    endTime = Date().timeIntervalSince1970
    isSessionInProgress = false
}

public func sessionDidUpdate(_ session: ARSession, frame: ARFrame) {
    let viewport = UIScreen.main.bounds.size

    guard let smoothPos = getGazePosition(frame: frame, viewport:
viewport) else { return }

    if UIScreen.main.bounds.contains(smoothPos) {
        state = .screenIn(point: smoothPos)
    } else {
        switch (smoothPos.x, smoothPos.y) {
        case (_, ...0):
            state = .screenOut(.top, point: smoothPos)
        case (_, viewport.height...):
            state = .screenOut(.bottom, point: smoothPos)
        case (...0, _):
            state = .screenOut(.left, point: smoothPos)
        case (viewport.width..., _):
            state = .screenOut(.right, point: smoothPos)
        default:
            fatalError("Invalid state")
        }
    }
}
```



```
delegates.forEach { delegate in
    delegate?.eyeTracking(self, didUpdateState: state, with: nil)
}

pointer?.move(to: smoothPos)
}
```

ПРИЛОЖЕНИЕ Е

API мимики лица

```
public init(blendShape: ARFaceAnchor.BlendShapeLocation, minValue: Float,
maxValue: Float) {
    self.blendShape = blendShape
    self.minValue = minValue
    self.maxValue = maxValue
}
public func initiateFaceExpression(_ expression: FaceExpression) {
    if expressions.first(where: { $0.blendShape == expression.blendShape
}) != nil { return }
    expressions.append(expression)
}

public func removeFaceExpression(_ expression: FaceExpression) {
    expressions.removeAll { $0 == expression }
}

public func setDelegate(_ delegate: EyeTrackerDelegate?) {
    delegates.append(delegate)
}

public func removeDelegate(_ delegate: EyeTrackerDelegate?) {
    delegates = delegates.filter { $0 != delegate }
}

func faceTracker(_ faceTracker: FaceTracker, didUpdateExpression
expression: FaceExpression)
```

ПРИЛОЖЕНИЕ Ж

API отслеживания взгляда

```
public func showPointer(window: UIWindow, with config:
PointerConfiguration) {
    pointer = Pointer(window: window)
    pointer?.show(with: config)
}

public func setDelegate(_ delegate: EyeTrackerDelegate?) {
    delegates.append(delegate)
}

public func removeDelegate(_ delegate: EyeTrackerDelegate?) {
    delegates = delegates.filter { $0 !== delegate }
}

func eyeTracking(_ eyeTracker: EyeTracker, didUpdateState state:
EyeTracker.TrackingState, with expression: FaceExpression?)
```

ПРИЛОЖЕНИЕ 3

API распознавания речи

```
public init() {
    recognizer = SFSpeechRecognizer()
    guard recognizer != nil else {
        transcribe(RecognizerError.nilRecognizer)
        return
    }

    Task {
        do {
            guard await SFSpeechRecognizer.hasAuthorizationToRecognize()
else {
            throw RecognizerError.notAuthorizedToRecognize
        }
        guard await
AVAudioSession.sharedInstance().hasPermissionToRecord() else {
            throw RecognizerError.notPermittedToRecord
        }
    } catch {
        transcribe(error)
    }
}

@MainActor public func startTranscribing() {
    Task {
        await transcribe()
    }
}

@MainActor public func stopTranscribing() {
    Task {
        await reset()
    }
}

@MainActor public var transcript: String = ""
```