DOI 10.37882/2223-2966.2025.04.23

ДИЗАЙН-СИСТЕМА ДЛЯ ПРИЛОЖЕНИЙ ПОД ПЛАТФОРМУ IOS

DESIGN SYSTEM FOR APPLICATIONS ON THE IOS PLATFORM

M. Konovalov
V. Amosov
A. Petrov

Summary. The article describes a new design system for applications on the iOS platform. It presents a study of existing solutions and identifies their shortcomings. A new development process is proposed, aimed at increasing design efficiency using ready-made components and templates. This reduces the labor costs associated with creating new elements and screen forms and allows any changes made to the design system to be automatically propagated throughout the entire application. This approach simplifies the process of updating and maintaining relevance, as well as enhances consistency in appearance and interaction across the entire application. Users navigate the interface more quickly when it adheres to a unified visual and functional style.

Keywords: design system, Swift, iOS, Human Interface Guidelines, Accessibility, technology stack, UI components, framework, CI/CD processes, MVC architecture, pattern, token, theming, backward compatibility, versioning, testing.

Коновалов Михаил Андреевич
Санкт-Петербургский политехнический университет Петра Великого
konovalov2.ma@edu.spbstu.ru
Амосов Владимир Владимирович
Кандидат технических наук, доцент, Санкт-Петербургский политехнический университет Петра Великого
amosov_vv@spbstu.ru
Петров Александр Владимирович
Старший преподаватель, Санкт-Петербургский политехнический университет Петра Великого
petrov_av@spbstu.ru

Аннотация. Статья описывает новую дизайн-систему для приложений под платформу iOS. Приводится исследование существующих решений и выделяются их недостатки. Предлагается новый процесс разработки, направленный на повышение эффективности проектирования за счет использования готовых компонентов и шаблонов. Это сокращает трудозатраты на создание новых элементов, экранных форм и позволяет автоматически распространять любые изменения, внесенные в дизайн систему на все приложение, что упрощает процесс обновления и поддержания актуальности, а также повышает единообразие внешнего вида и взаимодействия в рамках всего приложения. Пользователи быстрее ориентируются в интерфейсе, если он подчиняется единому визуальному и функциональному стилю.

Ключевые слова: дизайн-система, Swift, iOS, Human Interface Guidelines, Accessibility, технологический стек, UI-компоненты, фреймворк, CI/CD процессы, архитектура MVC, паттерн, токен, темизация, обратная совместимость, версионирование, тестирование.

Введение

Современная разработка мобильных приложений требует создания эффективных решений для проектирования интерфейсов, которые отвечают требованиям удобства, гибкости и доступности. Разработка дизайн-системы актуальна в условиях увеличивающейся сложности мобильных приложений и растущей потребности в единообразии, эффективности и масштабируемости.

Однако текущие подходы имеют ряд недостатков, связанных с ограниченной адаптивностью, сложностью внедрения и поддержания консистентности.

Цель данной работы — предложить и опробовать новый процесс создания дизайн-системы, которая учитывает специфику iOS-приложений и повышает их пользовательскую ценность. А также подобрать необходимый стек и архитектуру для разработки.

Основные проблемы при создании дизайн-систем iOS

При создании дизайн-систем для iOS возникают несколько ключевых проблем. Несмотря на унифицированность платформы, необходимо учитывать разнообразие устройств, таких как iPhone и iPad, с различными размерами и типами экранов. Поддержание консистентности представляет сложность, поскольку требуется соблюдение единого стиля при обновлениях и добавлении нового функционала. Важно также отслеживать изменения внутри компонентов, чтобы сохранять однородный вид приложений. Интеграция дизайн-системы в существующие проекты часто вызывает трудности у команд разработчиков. Кроме того, необходимо учитывать требования доступности, обеспечивая соответствие стандартам для пользователей с ограниченными возможностями, такими как увеличение шрифта и поддержка озвучивания Voice Over. Поддержка темной темы требует дополнительного внимания к дизайну и технической реализации при переключении между светлой и темной

вариантами интерфейса. Все эти проблемы требуют решений, обеспечивающих адаптивность, удобство использования и бесшовную интеграцию дизайн-системы. Обзор существующих решений.

Существует несколько подходов к созданию дизайн-систем, которые применяются разработчиками iOS-приложений:

Human Interface Guidelines [1] (HIG) — руководство Apple, описывающее принципы создания интерфейсов. HIG устанавливает базовые правила, но не предоставляет готовых решений для разработки дизайн-системы.

Open source дизайн-системы [2]: Например, Material Design от Google или IBM Carbon. Эти системы рассматриваются, так как предоставляют структурированный подход к созданию интерфейсов и включают в себя инструменты, которые могут быть адаптированы для работы с iOS, несмотря на их изначальную ориентацию на другие платформы. Эти решения часто слишком обобщены и не учитывают особенностей экосистемы iOS.

Коммерческие дизайн-системы [3]: Многие компании разрабатывают собственные системы (Uber, Atlassian Design Syste), что приводит к значительным затратам на проектирование и поддержку. Так же эти дизайн системы зачастую содержат не достаточное количество компонентов. Кроме того, достаточно часто коммерческие дизайн системы невозможно пере использовать даже на другие продукты того же бренда.

Основной недостаток существующих решений — отсутствие гибкости и интеграции с инструментами разработки iOS. Они не учитывают все особенности платформы iOS.

Способы решения проблем

Для устранения выявленных недостатков и проблем в разработке дизайн-системы предлагается комплексный подход, начинающийся с исследования требований. В этом этапе проводится тщательный анализ продукта, определяется ключевые сценарии использования и создается дизайн основных базовых компонентов, которые будут применяться в большинстве сценариев. Далее осуществляется разработка собственных стилей, включающая создание палитры цветов, адаптированной под различные темы, а также разработку собственной системы шрифтов с поддержкой динамических шрифтов.

Следующим шагом является создание библиотеки компонентов, где разрабатываются универсальные UI-компоненты, соответствующие рекомендациям HIG. Эти компоненты создаются по принципу модульности, что позволяет разбивать их на более мелкие элементы, ускоряя процесс разработки новых компонентов и упрощая поддержку существующих. При этом обеспечивается наличие настраиваемых параметров для адаптации под различные размеры экранов и продукты, а также поддержка смены тем и функций доступности на уровне компонентов. Баланс достигается за счет строгого следования рекомендациям HIG, где стандартное поведение компонентов закладывается как основа, а кастомизация ограничивается рамками, не нарушающими согласованность дизайна. Это обеспечивается использованием предустановленных значений, ограничений параметров и строгим тестированием на соответствие базовым требованиям.

Интеграция дизайн-системы с инструментами разработки выполняется посредством использования autolayout или SwiftUI для создания адаптивных интерфейсов, снепшот тестирования для стабилизации внешнего вида компонентов и подключения к процессам CI/CD для автоматического тестирования компонентов. Для обеспечения поддержки и масштабируемости внедряется система версионирования для каждого отдельного компонента, разрабатывается документация, доступная как для дизайнеров, так и для разработчиков, а также создаются основные сценарии использования, такие как статусные экраны и экраны ошибок, для их унифицированного применения в рамках всего проекта.

Дополнительно создается приложение, в котором представлены все компоненты в различных конфигурациях, включая разные состояния, размеры, темы и настройки доступности. Это упрощает процесс тестирования компонентов и способствует лучшему пониманию их возможностей и способов использования, что значительно облегчает интеграцию дизайн-системы в существующие проекты.

Реализация дизайн системы

Дизайн система была реализована на языке программирования Swift [5] — родном языке платформы iOS. Она представляет из себя фреймворк распространяемый с помощью менеджера пакетов Cocoapods [6] и SwiftPackage [7]. Это позволяет распространить дизайн систему сразу на несколько проектов. Кроме того, имеется демо-приложение, в котором представлены все компоненты и возможности дизайн системы.

Для создания палитры была использована система токенов. Токены разбиваются на отдельные списки перечисления, которые относятся к определенной категории принадлежности. Например, список токенов для задания цвета текста, фона, иконок. Токен представляет из себя название цвета из палитры например primary или secondary и является абстракцией над настоящим значением цвета. Это нужно чтобы поддерживать различные

значения цветов в разных темах, кроме того, использование абстракции значительно упрощает рефакторинг.

Верстка компонентов реализована на основе системных компонентов нативного фреймворка UIKit. Кроме того, для поддержки нового фреймворка для верстки SwiftUI, компоненты дизайн системы были переведены на него с помощью подхода UIRepresentableView [8]. Это позволило не переписывать все компоненты на новый фреймворк.

Новые компоненты верстаются с SwiftUI. Для поддержки обратной совместимости с UIKit используется UIHostingControlle [9]. Это так же позволяет убрать поддержку двух реализаций компонента. При этом остается возможность полноценно использовать компонент в экранах реализованных с помощью разных фреймворков.

Архитектура реализации компонентов

Архитектура реализации компонентов представлена на Рисунке 1. Она заключается в декомпозиции компонентов на базовые элементы [10]. Данная архитектура имеет следующие преимущества:

- Компоненты разделены на несколько уровней (модулей), это позволяет пере использовать их и подменять реализацию при необходимости
- При изменении реализации на определенном уровне, изменения сразу попадают во все места использования компонента.

При реализации компонентов использовалась архитектура MVC [11].

С iOS 13 Apple добавила встроенную возможность поддержки темной темы. При создании объекта цвета можно было сразу указать значения для светлой и темной и при изменении система сама их применит. Но такой подход имеет один большой недостаток, невозможно добавить поддержку еще одной темы. Поэтому возникла необходимость реализации своего механизма.

Реализация темизации была основана на паттерне *Notifications* (рассылки) [12]. Схема работы представлена на Рисунке 2. Каждый компонент подписывается на события и при обновлении на основе значения токена из объекта палитры (или из токена шрифта) берется необходимое значение.

Аналогичный подход используется для реализации работы динамических шрифтов.

Стоит отметить, что на стороне дизайн системы происходит работа с загрузкой картинок внутри компонентов. Для этого создан объект AsyncImage, в который достаточно передать ссылку на изображение, и он самостоятельно загрузит его и отобразит. Для работы с изображениями был реализован отдельный сервис, который грузит изображение из сети, далее сохраняет его в кэш. Для оптимизации отрисовки изображений они дополнительно обрабатываются с использованием механизма downsampling [13]. Так как обработка изображений тяжелая операция, она вынесена в отдельную очередь с помощью фреймворка GCD [14].

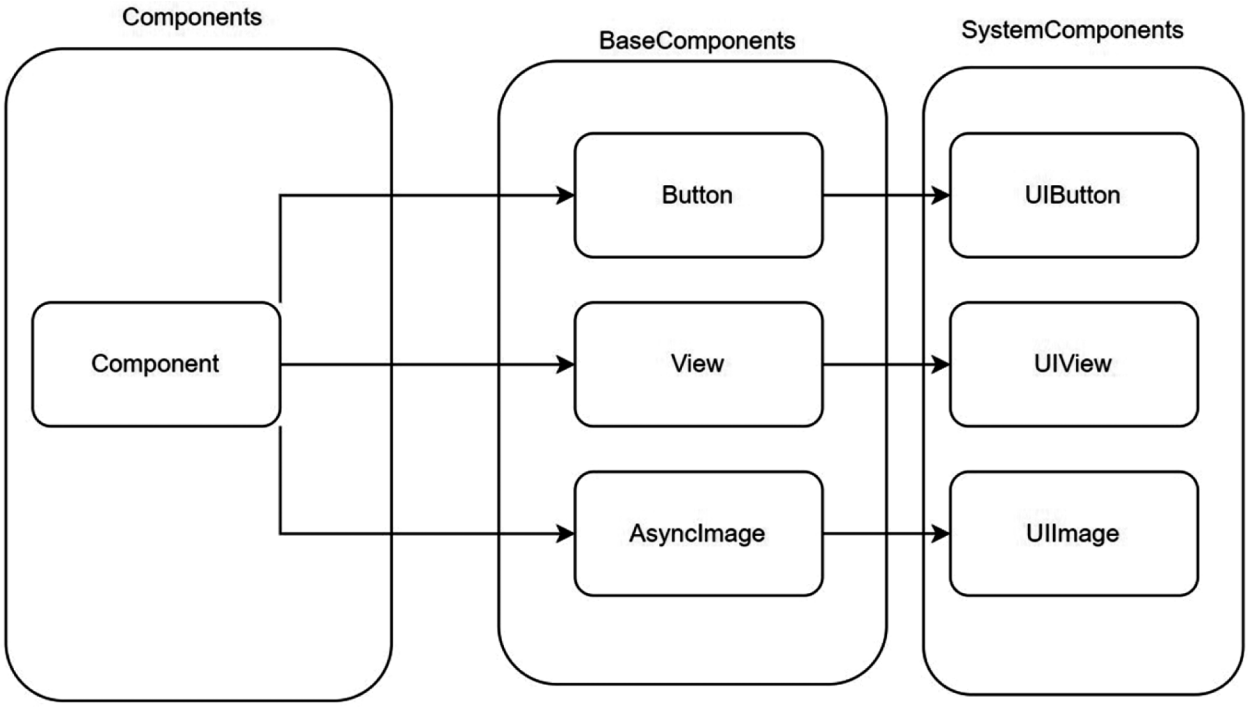

Рис. 1. Архитектура создания компонента

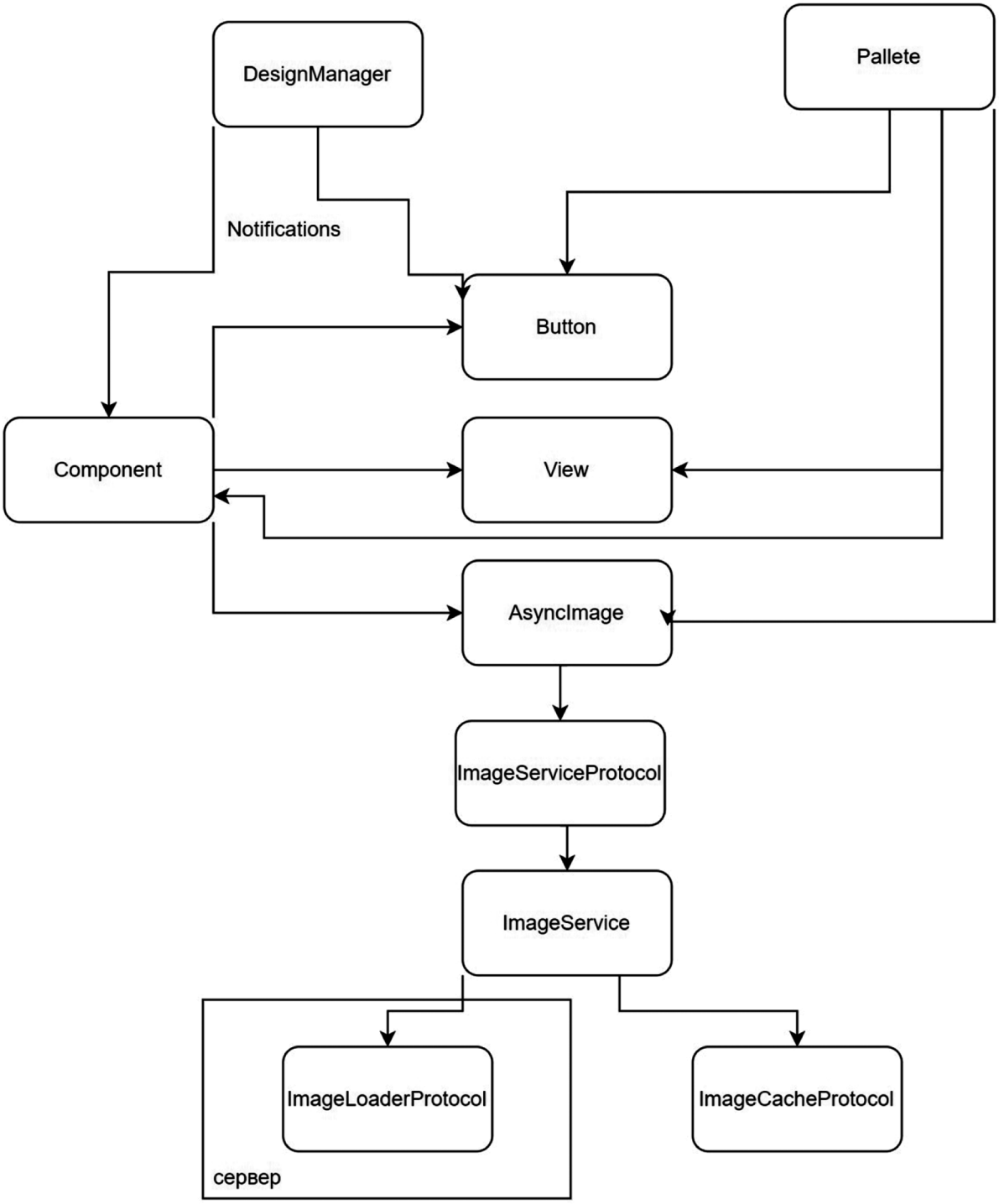

Рис. 2. Архитектура реализации темизации

Реализация обратной совместимости фреймворка дизайн-системы iOS

Одной из ключевых задач при разработке и поддержке дизайн-систем является обеспечение обратной совместимости. Это особенно важно в условиях, когда множество проектов и команд зависят от стабильности и предсказуемости фреймворка.

Для обеспечения обратной совместимости необходимо учитывать:

1. **Семантическое версионирование.** Использование семантического версионирования позволяет четко обозначать изменения в API. Формат версии MAJOR.MINOR.PATCH указывает на уровень изменений.
2. **Тестирование на совместимость.** Регулярное тестирование на совместимость с предыдущими

версиями позволяет выявлять потенциальные проблемы на ранних этапах. Для этого можно использовать инструменты, такие как:

XCTest[15] для модульного и интеграционного тестирования;

Snapshot Testing[16] (например, с использованием библиотеки Snapshot Testing) для визуального регрессионного тестирования.

3. **Ведение документации.** При внесении изменений, нарушающих обратную совместимость, важно предоставлять подробную документацию, которые помогут разработчикам адаптировать свой код к новой версии. Для этого в swift есть аннотация deprecated, которая подсвечивает устаревшую функциональность и подсказывает что использовать взамен.

Всё вышеперечисленное автоматизируется добавлением в проект CI/CD [17]. Можно настроить автоматическое поднятие версии библиотеки на основе изменений, добавить этап тестирования кода при внесении изменений.

Кроме того, можно настроить автоматический выпуск новой версии тестового приложения для фреймворка при наличии изменений.

Итоги

В статье рассмотрены существующие подходы к созданию дизайн-систем для iOS-приложений и предложен и опробован новый процесс разработки дизайн-системы, учитывающий специфику платформы. Были рассмотрены существующие дизайн системы. Предложенный новый процесс учитывает и решает проблемы присущие существующим решениям. Новый процесс разработки дизайн-системы реализован в виде фреймворка на языке программирования Swift, распространяемого с помощью менеджера пакетов Cocoapods и SwiftPackage. В результате анализа были выявлены основные требования, которым должна соответствовать программная реализация дизайн системы на платформе iOS. Предложена архитектура и технологический стек для реализации дизайн системы. Новый процесс разработки дизайн-системы направлен на повышение эффективности её проектирования за счёт использования готовых компонентов и шаблонов, это сокращает трудозатраты на создание новых элементов и позволяет любые изменения, внесенные в дизайн-систему, автоматически распространять на всё приложение. Это упрощает процесс обновления и поддержания актуальности, а также повышает единообразие или согласованность внешнего вида и взаимодействия в рамках всего приложения. Пользователи быстрее ориентируются в интерфейсе, если он подчиняется единому визуальному и функциональному стилю.

ЛИТЕРАТУРА

1. Apple. Human Interface Guidelines [Электронный ресурс]. — Режим доступа: https://developer.apple.com/design/human-interface-guidelines/. (дата обращения: 10.02.2025).
2. Material Design Guidelines [Электронный ресурс]. — Режим доступа: https://material.io/design. (дата обращения: 10.02.2025).
3. UXPin. «13 Best Design System Examples in 2024». — Режим доступа: https://www.uxpin.com/studio/blog/best-design-system-examples/.
4. SwiftUI Documentation [Электронный ресурс]. — Режим доступа: https://developer.apple.com/documentation/swiftui. (дата обращения: 10.02.2025).
5. The Swift Programming Language Book [Электронный ресурс]. — Режим доступа: https://www.swift.org/documentation/. (дата обращения: 10.02.2025).
6. Cocoapods documentation [Электронный ресурс]. — Режим доступа: https://cocoapods.org/. (дата обращения: 10.02.2025).
7. SPM Documentation [Электронный ресурс]. — Режим доступа: https://www.swift.org/documentation/package-manager/. (дата обращения: 10.02.2025).
8. UIViewRepresentable explained to host UIView instances in SwiftUI [Электронный ресурс]. — Режим доступа: https://www.avanderlee.com/swiftui/integrating-swiftui-with-uikit/. (дата обращения: 10.02.2025).
9. SwiftUI in UIKit View Controllers with UIHostingController [Электронный ресурс]. — Режим доступа: https://medium.com/@max.codes/use-swiftui-in-uikit-view-controllers-with-uihostingcontroller-8fe68dfc523b. (дата обращения: 10.02.2025).
10. Сидоров Д.А Comparative Analysis of Component-Based Architectures in Web Design for Scalable Applications. — Холодная наука, 2024. — 24 с.
11. Булыга А.И. Анализ и сравнение архитектурных шаблонов проектирования, используемых при разработке мобильных приложений для платформы iOS. — Научный журнал, № 1(63), 2022.
12. Nyisztor K., Nyisztor M. Design Patterns in Swift 5 — Razeware LLC, 2019. — 277 с.
13. iOS: Downsampling for Improved Performance [Электронный ресурс]. — Режим доступа: https://medium.com/codex/ios-downsampling-for-improved-performance-ed03b5e7627e. (дата обращения: 10.02.2025).
14. Todorov M. Modern Concurrency in Swift — Razeware LLC, 2021. — 273 с.
15. Apple Inc. XCTest Documentation. — Официальная документация Apple [Электронный ресурс]. — Режим доступа: https://developer.apple.com/documentation/xctest. (дата обращения: 10.02.2025).
16. GitHub. SnapshotTesting Library [Электронный ресурс]. — Режим доступа: https://github.com/pointfreeco/swift-snapshot-testing. (дата обращения: 10.02.2025).
17. Potter B. Continuous Integration and Delivery for Mobile Apps — O'Reilly Media, 2018. — 240 с.

18. Norman D.A. The Design of Everyday Things: Revised and Expanded Edition. — Basic Books, 2013.
19. Brown A. Designing for Accessibility. — Smashing Magazine. — Режим доступа: https://www.smashingmagazine.com/designing-for-accessibility/. (дата обращения: 10.02.2025).
20. Google. Material Design for iOS [Электронный ресурс]. — Режим доступа: https://material.io/design/platform-guidance/ios. (дата обращения: 10.02.2025).
21. Atlassian. Design System Documentation [Электронный ресурс]. — Режим доступа: https://atlassian.design. (дата обращения: 10.02.2025).
22. IBM. Carbon Design System [Электронный ресурс]. — Режим доступа: https://carbondesignsystem.com. (дата обращения: 10.02.2025).
23. Uber. Base Design System [Электронный ресурс]. — Режим доступа: https://baseweb.design. (дата обращения: 10.02.2025).
24. Lightning Design System [Электронный ресурс]. — Режим доступа: https://www.lightningdesignsystem.com/. (дата обращения: 10.02.2025).