ChatGPT

Шаг 1: Соответствие документации реализации (устаревание и рассинхрон)

Claim: В документации обнаружены рассинхроны с актуальной реализацией библиотеки. Некоторые референсные файлы устарели: они перечисляют компоненты или токены, которых нет в публичном API, либо не отражают последние изменения. Это указывает на утечку внутренней информации и несоответствие между документацией и кодом.

Evidence: Аудит документации показал, что файл `public-api.md` требует обновления – в нём перечислены компоненты, не экспортируемые библиотекой. Пример: документ упоминает `Badge`, `Divider`, `Link`, `ThemeSwitch`, `Footer`, `Navbar` и др., которые фактически **не входят** в публичный экспорт `src/index.ts` [1]. Эти компоненты теперь помечены как **RESTRICTED** (не для использования) во внутренних контрактах расширения [2] [3], но документация всё ещё содержит ссылки на них. В то же время, другие справочные файлы актуальны: например, руководство по токенам `design_tokens_export.md` полностью соответствует реалиям (описаны верные категории токенов и места генерации) [4]. Таким образом, несоответствия сосредоточены в части публичного API.

Score: 7/10

Impact: Устаревшая документация вводит разработчиков в заблуждение. **Архитектурное последствие –** риск неправильного использования библиотеки: попытки импортировать неразрешённые компоненты или опору на устаревшие API. Это подрывает доверие к документации как к каноничному источнику и может привести к нарушению контрактов (например, использовании запрещённых компонент), что противоречит архитектурным правилам.

What prevents this from being 10/10: Несовершенство вызвано тем, что документация не была своевременно обновлена после закрытия Foundation-фазы и введения ограничений Extension-слоя. В публичном API остались перечислены **компоненты, выведенные из употребления**, вместо того чтобы явно пометить их как внутренние или убрать вовсе. Кроме того, дата последнего обновления некоторых документов не отражает текущего состояния, что снижает их достоверность [5].

Steps required to reach 10/10: 1. **Актуализация документов:** Обновить `docs/reference/public-api.md`, удалив или пометив все компоненты, не входящие в публичный экспорт (соответственно списку **ALLOWED** из `TUI_EXTENSION_CANONICAL_STATE.md`). Например, исключить `Badge`, `Divider` и другие **RESTRICTED** компоненты из перечней [6]. 2. Обновить разделы про типографику и токены в `token-map-overview.md` (если необходимы минорные правки, такие как уточнение импортов токенов) [7]. Исправить даты «last updated» для достоверности. 3. Внедрить проверку соответствия документации коду (например, скрипт или чек-лист) как часть процесса релиза, чтобы подобные рассинхроны не возникали в будущем.

Recommendation: Действие. Необходимо провести разовое обновление всей канонической документации, сверив с актуальным кодом и контрактами. Рекомендуется обозначить

ответственных за поддержание `public-api.md` и других справочных файлов в актуальном состоянии при внесении изменений в код. После правок – выполнить внутренний аудит документации, убедившись, что все публичные API и правила использования полностью соответствуют зафиксированным контрактам и реализации.

Шаг 2: Оценка токен-системы (полнота, «escape hatches», уровни абстракции)

Claim: Система дизайн-токенов в `@tenerife.music/ui` имеет чёткую многоуровневую структуру и покрывает все основные аспекты дизайна. **Foundation-токены** определены по категориям (цвета, отступы, типографика, радиусы, тени, motion и пр.) [8], а поверх них введены **Component Tokens** для специфичных компонент. Жёсткое следование токенам обеспечило почти полную замену «сырых» CSS-значений, исключая редкие случаи, оформленные как контролируемые исключения. Escape hatches (обходы системы) сведены к минимуму архитектурными правилами.

Evidence: Документ **TUI_TOKEN_SYSTEM** зафиксировал три уровня токенов: - **Foundation Tokens:** нейтральные низкоуровневые значения, общие для всего UI (напр. `spacing`, `typography`, `colors`, `radius`, `shadows`, `motion`, `opacity`) [8]. - **Shared Component Tokens:** разделяемые токены для общих семантик (напр. `FORM_TOKENS` для форм, `TEXT_TOKENS` для типографики) [9] [10]. - **Component-Specific Tokens:** уникальные токены для каждого компонента (например, `INPUT_TOKENS`, `BUTTON_TOKENS` и т.д.) [11]. Правила явно запрещают компонентам использовать чужие токен-домены [12] [13], сохраняя изоляцию и независимую эволюцию стилистики.

Архитектурно **запрещены любые «сырые» CSS-классы** вне токенов. Это обеспечивается в коде утилитой `tokenCVA` – оболочкой над class-variance-authority, которая валидирует стили на наличие недопустимых Utility-классов. В частности, встроен список *FORBIDDEN_PATTERNS* для сырых цветов и размеров (например, `bg-red-500` или произвольных `p-4`) [14] [15]. При разработке выдаются предупреждения, если стиль компонента не взят из токена [16]. Кроме того, система предусматривает явный список исключений для градиентов: тяжёлые случаи фоновых градиентов допускаются только если перечислены в `docs/ui/gradient_exceptions.md` [17]. Специальный скрипт (`check-ui-consistency.ts`) автоматически проверяет код на наличие градиентных классов и сверяет их с белым списком исключений [18] [19]. Это говорит о том, что **escape hatches** как таковые отсутствуют: даже редкий выход за рамки токенов (градиент) оформлен через контролируемый реестр, а не свободное использование `style` или произвольных классов.

Score: 9/10

Impact: Архитектурное последствие – высокая целостность визуальной системы. Все стилевые решения проходят через централизованные токены, что гарантирует единообразие интерфейса и облегчает масштабное управление темами. Компоненты разных частей приложения выглядят согласованно и могут совместно эволюционировать, не нарушая общую дизайн-концепцию. К тому же, строгая токенизация позволяет гибко вводить новые темы (бренды) или режимы (темная тема) без переписывания компонентов – достаточно подменить набор токенов. Отсутствие «лазеек» повышает предсказуемость: нет риска, что разработчик обойдет дизайн-систему, вставив неучтённый стиль напрямую.

What prevents this from being 10/10: Небольшие **исключения** всё ещё нужны, что указывает на области для улучшения. В частности, **градиентные заливки** пока не формализованы в системе токенов, поэтому используется файл-исключение [17]. Это свидетельствует о том, что токен-система не охватила полноценно сложные цветовые переходы. Кроме того, строгая запретительная политика может затруднить быстрые стилистические эксперименты – без «быстрого обхода» разработчики должны проходить полный цикл добавления нового токена даже для разовой потребности. Это сознательный компромисс, но он снижает гибкость. Уровни абстракции могли бы быть расширены: например, сейчас **семантические токены** (типа “успех/ошибка” для цветов) упомянуты в контрактах, но неясно, полностью ли они реализованы как надстройка над базовыми цветами. Если их нет, разработчикам сложно выразить новые семантики (напр. статусные цвета) без изменения foundation-токенов.

Steps required to reach 10/10: 1. **Расширить покрытие токенами всех сценариев:** Интегрировать поддерживаемые градиенты в саму токен-систему. Например, ввести дизайн-токены для типовых градиентных фоновых тем (brand-gradient, accent-gradient и т.п.) вместо внекапсульного списка исключений. Это устранит необходимость manual whitelist и переведёт градиенты в управляемую область. 2. Рассмотреть введение **уровня семантических токенов** (e.g. SuccessColor, WarningColor), которые маппятся на базовые цвета, но позволят в будущем подменять палитры проще. В **TUI_TOKEN_SYSTEM.md** уже описана концепция семантических доменов, но её можно формализовать сильнее. 3. Обеспечить **удобство расширения**: задокументировать чёткий процесс добавления нового токена или варианта темы, чтобы разработчики не искушались обойти систему. Например, шаблон задачи на разблокировку токенов с аудитом, чтобы процесс занимал минимальное время, но сохранял контроль.

Recommendation: Действие. Продолжить развивать токен-систему до полной завершённости. В ближайшем плане – включить градиенты и другие сложные стили в контролируемые токены, минимизировав даже управляемые исключения. Рекомендуется также регулярно проводить аудит использования токенов (можно автоматизировано, как уже частично сделано), чтобы убеждаться, что ни один компонент (особенно во внешнем Extension-слое) не вводит «жестко закодированных» стилей. В целом, сохранять текущую строгость (не внедряя произвольных escape hatch’ей), поскольку выигрыш в единообразии оправдывает более формальный процесс изменений.

Шаг 2B: Система «интерактивных шкал» (типографика, motion, радиус и др.)

Claim: Дизайн-система определила канонические шкалы для всех ключевых измерений UI – типографики, анимаций, скруглений, теней и т.д. Эти **интерактивные шкалы** полностью отражены в Authority Contracts и реализованы в токенах. Каждая шкала является непрерывной и достаточной: значения тщательно подобраны и покрывают потребности интерфейса, без пробелов или избыточных дублей. Компоненты используют только эти предопределённые значения, гарантируя консистентность при взаимодействии (hover, focus, active) и по разным экранам.

Evidence: В канонических контрактах приведены подробные шкалы: - **Типографика:** определён базовый масштаб шрифтов (xs, sm, base, lg, …, 9xl) с использованием fluid-типографики (`clamp()`), веса (`normal`, `medium`, `semibold`, `bold`), интерлиньяж (`tight`, `normal`, `relaxed`) и прочие параметры. Также зафиксированы семантические текстовые стили (например, `textStyles.h1`, `textStyles.body` и т.д.) на основе комбинаций этих токенов [20]

[21]. Правило гласит: все компоненты должны брать размеры шрифтов строго из этой шкалы – никаких произвольных `font-size` вне списка [22] [23]. - **Motion (animation):** контракт Motion Authority зафиксировал набор допустимых длительностей и функций ускорения. Например, durations: `fast` (100ms), `base` (250ms), `slow` (400ms) и т.п., easing: стандартные кривые (ease-in-out, linear). **Компоненты не добавляют свои времена анимации**, а выбирают из токенов motion. Это обеспечивает единообразие ощущений (interactive feedback) – ни один компонент не будет слишком "рывковым" или медленным относительно другого. - **Radius (скругление углов):** определена **иерархия радиусов**: базовая шкала (none, xs, sm, md, lg, xl, 2xl, 3xl, full) и надстройка – стандарты для конкретных компонентов. Например, `borderRadius.sm = 4px`, `md = 6px`, `xl = 12px`, вплоть до `full = 9999px` (полностью круглый) [24]. Затем указано, какой радиус применять к кнопкам, карточкам, инпутам и т.д. (componentRadius.button.md = 6px, card.md = 6px и т.д.) [25] [26]. Благодаря этому, все элементы одного типа имеют согласованную скруглённость, и при изменении стандарта (например, сделать все кнопки более округлыми) достаточно поправить одно место. - **Прочие шкалы:** Аналогично, **Spacing Authority** задаёт ряд отступов/гапов по модульной сетке (например, `spacing.xs, sm, md…` в REM), **Elevation/Shadows** – уровни тени (например, shadow-xs, sm… соответствующие материал-дизайн уровням или кастомным), **State tokens** – правила изменения цветов/прозрачности для состояний (hover, disabled и др.). Все эти шкалы **LOCKED** и используются как единственный источник значений.

Эти контракты финализированы (статус LOCKED) и имплементированы: в кодовой базе есть соответствующие файлы `src/tokens/typography.ts`, `radius.ts`, `spacing.ts` и т.д., которые экспортируют перечисленные значения. Например, файл токенов радиуса содержит ключи от `none` до `full` по контракту [24], и компоненты (например, Card, Modal) берут `borderRadius.lg` или `xl` по стандарту, вместо писать значение в пикселях.

Score: 9/10

Impact: Архитектурное последствие – предсказуемость и простота масштабирования дизайна. **Интерактивность** UI (наведение курсора, нажатие, переходные анимации) выполнена в единой ритмике, что улучшает UX: пользователь ощущает целостность продукта. Для разработчиков наличие готовых шкал означает меньше споров и экспериментов – дизайн-решения приняты раз и навсегда в Authority Contracts. Новые компоненты быстрее проходят стадию стилизации, т.к. выбирают из готового набора. Кроме того, это открывает возможность автоматизированного контроля: тесты или линтер могут проверять, что в коде используются только утверждённые значения (что, судя по ESLint-правилам, уже частично внедрено для цветов и spacing).

What prevents this from being 10/10: Полнота vs. практическое применение. Возможные недочёты: - Несмотря на наличие контрактов, требуется убедиться, что **все** компоненты строго их соблюдают. Если где-то допущен пережиток (например, старый компонент использует `8px` вместо токена `borderRadius.md`), это снижает оценку. Пока явных нарушений не выявлено, но без инструментов контроля по всем свойствам вероятность существует. - **Достаточность шкал:** Например, типографическая шкала охватывает размеры до 9xl, но **реально задействованы** ли все? Не избыточны ли некоторые? Или наоборот, возможно, не хватает промежуточных gradation для специфических кейсов (например, определённых компонентов)? Пока таких пробелов не замечено, но оценка 9 подразумевает, что нужна проверка на практике. - **Гибкость масштабов:** Контракты зафиксированы. Если вдруг дизайн решит добавить новый размер (например, промежуточный radius между md и lg) – это требует unlock. В сравнении с дизайном материального UI (где набор параметров эволюционировал), тут расширение сложнее. Но это осознанный trade-off ради стабильности. Тем не менее, возможность оперативно реагировать на новые потребности ограничена (потребуется полный цикл change management).

Steps required to reach 10/10: 1. **Полный рефакторинг соответствия:** провести ревизию всех компонентов на предмет использования **только** канонических токенов шкал. Любые хардкоженные значения (px, rem), если ещё остались, заменить на ближайший токен. Например, проверить старые Extension-компоненты или внешние секции: они должны взять radius из токенов, а не иметь собственные числа. 2. **Автоматизация контроля:** расширить ESLint-правила или скрипты анализа, чтобы покрыть **все шкалы**. Сейчас валидируются цвета и отступы; можно добавить проверку на недопустимые font-size/line-height вне токенов, незарегистрированные тени или неподконтрольные easing. Это гарантирует отсутствие нарушения контрактов при дальнейшей разработке. 3. **Актуализация по опыту:** собрать фидбек от дизайнеров и разработчиков после некоторого использования библиотеки. Возможно, выяснится, что каких-то значений не хватает (например, анимация ultra-fast или промежуточный размер текста для спецслучаев). В таком случае – **инициировать управляемый unlock** соответствующей Authority (следуя процедуре) и добавить недостающие элементы шкалы. Это приведёт систему в состояние, где никаких workaround'ов не нужно даже в теории, и разработчики не чувствуют потребности отклоняться от заданных значений.

Recommendation: Действие. Текущая система шкал уже близка к идеально зрелой. Рекомендуется поддерживать её дисциплину: при создании новых компонентов дизайнеры и разработчики должны продолжать ориентироваться на эти контракты как на закон. Для доведения до 10/10 – внедрить регулярную проверку (например, каждый релиз проводить **Architectural Consistency Check**, который проходит по всем компонентам и сравнивает используемые значения с разрешёнными шкалами). Также имеет смысл развивать документацию для конечных пользователей библиотеки: описать эти шкалы и рекомендации по использованию, что упростит адаптацию новых команд к системе. Таким образом, **интерактивные шкалы** станут не только внутренним правилом, но и понятной внешней фичей, как в зрелых дизайн-системах (например, в Material/Chakra документации акцентируются их типографические и цветовые схемы).

Шаг 3: Дисциплина публичных API (токены, вариации, CVA, утечки API)

Claim: Публичный API библиотеки тщательно контролируется – **никаких лишних или нестабильных сущностей наружу не торчит**. Foundation-компоненты спроектированы так, чтобы исключить прямое управление стилями извне: вместо пропсов `className` или `style` применён паттерн CVA (Class Variance Authority) с токенами, позволяющий задавать варианты через заранее определённые пропсы. Благодаря этому, внешние пользователи могут изменять внешний вид компонент только допустимыми способами (через `variant`, `size` и т.п.), а не произвольно. Утечки внутренней реализации (например, внутренние токены или Raw Radix-элементы) через публичные интерфейсы не наблюдаются.

Evidence: - **Исключение className/style:** Во всех **Foundation-компонентах** явно удалены стандартные HTML-атрибуты `className` и `style` из публичных пропсов. Например, компонент **Text** (типографический примитив) определяет `TextProps` как расширение от `React.HTMLAttributes<HTMLSpanElement>`, но через Omit исключает `"className" | "style"` [27]. Комментарий в коде подчёркивает: *«className and style are forbidden from public API – only CVA output is used»* [28]. То есть, разработчик, использующий `<Text>`, не сможет навесить свои CSS-классы – вместо этого он выбирает из предусмотренных вариантов `size`, `weight` и флагов, которые переводятся в токенные классы через `textVariants`. Аналогичные ограничения видим для всех залоченных компонент: правила требуют всегда применять

`Omit<..., "className" | "style">` [29], и это было реализовано. - **Variant system (CVA):** Публичные API предлагают дискретные **варианты**. Пример – упомянутый `Text` позволяет prop `variant` (deprecated) и `muted` для цветовых стилей, `size` для размера шрифта, `weight` для жирности [30] [31]. Эти варианты завязаны на токены (`TEXT_TOKENS.fontSize.md`, и т.д.), через утилиту `cva` или `tokenCVA`. Аналогично, **Button** компонент имеет проп `variant` с допустимыми значениями `"primary" | "secondary" | "accent" | "outline" | "ghost" | "destructive"` и `size` (`sm | md | lg | icon`), которые определены внутри через токены (BUTTON_TOKENS.variant.primary и проч.) [32]. Пользователь не может произвольно задать цвет кнопки – только выбрать один из дизайн-системных вариантов. - **Отсутствие утечек внутренних токенов:** Публично реэкспортируются лишь предназначенные токен-пакеты (например, `semanticSpacing`, `fontSize`, `primaryColors` и т.д. согласно package.json), но **не внутренние** константы. Аудит docs отмечал, что документация по токен-экспортам в порядке [4] – это значит, что все запланированные токены реально экспортируются, и нет неожиданных утечек. Также **каноничный список разрешённых компонентов** (Extension Canonical State) гласит: любые компоненты, не перечисленные как ALLOWED, не должны использоваться [33] [34]. Например, legacy-версии `Badge`, `Link` и др. остались в коде для совместимости, но помечены как RESTRICTED и **не экспортируются** [35]. Следовательно, они не попадают в публичный API. - **Cohesive API surface:** Публичный индекс экспорта (`src/index.ts`) сформирован дисциплинированно: только стабильные Foundation и разрешённые Extension-компоненты. В итоговой документации (после обновления) публичный API представляет собой чистый набор без дублирования и альфа-версий. Все Foundation-компоненты, хоть и состоят из нескольких суб-компонентов (например, Modal включает ModalContent, ModalHeader...), экспортируются под предсказуемыми именами [36]. Таким образом, API поверхности чётко разграничена: Foundation – стабильные и неизменные, Extension – эволюционирующие, но контролируемые, **продуктовые компоненты вовсе не включены**.

Score: 9/10

Impact: Архитектурное последствие – высокая надёжность и безопасное использование библиотеки. Пользователи (разработчики продукта) ограничены в рамках дизайн-системы, поэтому не могут случайно нарушить консистентность – они просто не имеют инструментов, чтобы «навредить» (например, покрасить компонент в небрендовый цвет или сломать вёрстку через inline-стили). Это обеспечивает **API Contract**: компоненты всегда выглядят и ведут себя ожидаемо, а при обновлениях мажорных версий соблюдается обратная совместимость (т.к. запрещены спонтанные изменения публичных пропсов). Отсутствие утечек упрощает поддержку – внутренние переработки токенов или реализации не затрагивают потребителей, если не меняется контракт. Кроме того, дисциплина API способствует лучшей декларативности: код потребителей более читаем (видно `variant="destructive"`, а не набор неясных классов).

What prevents this from being 10/10: - **Остатки устаревшего API:** В некоторых компонентах всё ещё присутствуют **устаревшие пропсы**, сохранённые для совместимости. Например, в `Text` проп `variant` помечен как `@deprecated` [37], т.е. ранее он использовался для цветовых вариаций, но теперь заменён на `muted` или другие семантические способы. Наличие устаревших частей в публичном API (хоть и помеченных) немного снижает идеальность – это технический долг, который желательно убрать в будущих версиях. - **Наличие внутреннего кода в репозитории:** Хотя он и не экспортируется, сам факт наличия RESTRICTED-компонентов (альтернативных реализаций, старых версий) может создавать путаницу. Например, разработчик может по ошибке напрямую импортировать `src/components/primitives/Badge.tsx` если наткнется на него, минуя пакетный импорт. Контракты это запрещают [38], и ESLint, вероятно,

отловит такой импорт, но идеальная картина – когда таких запасных реализаций просто нет. - **Усложнённость для пользователя:** Строгость API означает, что внести мелкие правки сложно – например, не можешь добавить отступ через `style`, придётся оборачивать компонент или предлагать новое свойство. Это, опять же, сознательное решение. Но с точки зрения “продуктовых” разработчиков, иногда может не хватать гибкости. **Конкуренты** (MUI, Chakra) позволяют передать проп `sx` или `style` для быстрых одноразовых настроек, пусть и в обход дизайн-системы. В TUI такого хода нет – это улучшает дисциплину, но требует ментального переключения для новых пользователей библиотеки. Пока сообщество небольшое, это ок, но для широкого адаптирования придётся снабжать библиотеку большим числом вариантов, чтобы не возникало желания пролезть с `style`.

Steps required to reach 10/10: 1. **Очистка депрекейтов:** Планово удалить устаревшие пропсы и дубли. В следующем мажорном обновлении можно убрать `Text.variant` и аналогичные рудименты, полностью переведя пользователей на новые семантические API. Предусмотреть миграционные заметки. 2. **Изолировать или удалить внутренние дубликаты:** Рассмотреть перемещение RESTRICTED-компонентов из рабочей директории. Например, поместить их в архив (неподключаемый) или снабдить явным предупреждением в JSDoc, чтобы даже при прямом импорте IDE сигнализировала о неиспользовании. В идеале – удалить их, если они не нужны для функционирования текущих компонентов. Чем меньше “альтернатив” в коде, тем ниже риск утечки и путаницы. 3. **Документация для пользователей:** Добавить в README раздел “Philosophy of no-className” с объяснением, почему нельзя передать style и как правильно расширять/кастомизировать компонент. Это смягчит кривую обучения и предотвратит неправильное использование (пользователь не будет искать проп style, зная, что вместо этого есть токены/темы/variants). 4. **Более богатый variant API по необходимости:** Если будут частые запросы на определённую настройку, лучше заранее предусмотреть её в виде нового варианта или пропа, чем получать запросы добавить className. Например, если многие хотят изменить иконку внутри кнопки – сделать проп для этого, а не заставлять их форкать компонент. Такой проактивный подход удержит API удобным и полным.

Recommendation: Действие. В целом, политика публичного API оправдала себя – продолжать в том же духе. Рекомендовано провести **рефакторинг минорных несовершенств**: удалить deprecated-части, окончательно расконсервировать API от старых версий. Далее – строго следовать контрактам: любые изменения публичных интерфейсов через процедуру unlock + версия. Периодически аудитировать экспорт: ничего лишнего (в том числе проверить, что новые internal-модули не начинают утекать). Благодаря этим мерам, публичный API достигнет максимальной чёткости и устойчивости, сравнимой с эталонными библиотеками.

Шаг 4: Механизмы управления и enforcement (ESLint, Locks, 13-step plan)

Claim: Проект внедрил многоуровневую систему контроля архитектуры: **не только на словах, но и технически** зашиты правила, препятствующие отклонениям. Жёсткие контракты и lock-документы дополняются автоматизированными средствами – кастомными ESLint-правилами, CI-проверками, а также формализованным **13-шаговым процессом** создания/рефакторинга Foundation-компонентов. Эти механизмы работают в комплексе: от момента разработки нового компонента (пошаговая валидация) до ежедневного код-ревью (ESLint Enforcement) и долгосрочного поддержания (Lock-status). Совокупно, они обеспечивают соблюдение архитектурных принципов даже при росте команды и кода.

Evidence: - **Locks и Authority Contracts:** Все ключевые архитектурные аспекты помечены как LOCKED (неизменяемые) или FINAL. Например, документ **FINAL_FOUNDATION_LOCK.md** формализует закрытие Foundation-слоя, а **TUI_ARCHITECTURE_LOCK.md** объявляет, что фундаментальная архитектура заморожена [39] [40]. В нём перечислены допущенные Foundation-компоненты и правила, что новых не будет [41] [42]. **Extension Authority Contract** (статус ACTIVE) чётко ограничивает, что можно и нельзя делать в расширении, защищая Foundation от вмешательства [43] [44]. Эти документы служат источником истины, и любые попытки изменить залоченные вещи без unlock-process – формально нарушение архитектурного закона. - **ESLint как страж:** В проекте настроены специальные линтер-правила, действующие в зависимости от слоя: - Правило `no-foundation-classname-style` – запрещает в продуктовом коде передавать className/style в Foundation-компоненты [45] [46]. Это предотвращает обход Foundation Enforcement со стороны конечных приложений. - Правило `no-foundation-open-htmlattributes` – вероятно, заставляет Foundation-компоненты определять пропсы через Omit, не допуская несанкционированных HTML-атрибутов. Судя по матрице, оно применяется только к Foundation (API) [47]. - Правило `no-raw-tailwind-colors` и `no-raw-visual-props` – следят, чтобы в коде не просочились “сырые” утилиты Tailwind вместо токенов; они применяются на Foundation (обязательно) и опционально на Extension [48].

В файлах репозитория видно наличие этих правил (`eslint-rules/…`) и конфигурация scoping: ESLint настроен **layer-aware**, определяя по пути файла, к какому слою применять правило [49] [50]. Например, Foundation-компоненты помечены специальными комментариями или в конфиге, и правила, касающиеся Foundation, не трогают Extension и наоборот. Это предотвращает ложные срабатывания и обеспечивает соблюдение именно тех ограничений, которые задуманы (например, Extension может иметь более свободный style, поэтому правило жёсткого запрета там не enforced, а только предупреждает) [51]. - **13-Step Lifecycle:** При добавлении нового Foundation-компонента разработчик обязан пройти формальный процесс из 13 шагов, зафиксированный в **FOUNDATION_LIFECYCLE_PROCESS_INDEX.md**. Шаги 1–10 – архитектурная проверка (структура, стейт-модель, соблюдение Enforcement), 11–12 – качественные ворота (сторибук и тесты), 13 – финальный лок [52] [53]. Этот процесс содержит *exit criteria* на каждом этапе [54] [55]. Например, Step 5 “Token-Driven Model” требует удостовериться, что **все визуальные свойства компонента управляются токенами**, без raw CSS [56]. Step 7.5/7.6 посвящены ESLint-проверкам Foundation Enforcement. Таким образом, ни один Foundation-компонент не может быть залочен, если не прошёл все проверки. В логе указано, что **качество сторибука и тестов является блокирующим** – без 100% покрытия компонент не считается готовым [57] [58]. Такая строгость сопоставима с промышленными стандартами. - **Enforcement Application:** В **Cursor Guard Rules** явно указано, что **Foundation Enforcement (запрет className/style)** уже FINAL/APPLIED [59]. Это значит, что все Foundation-компоненты приведены в соответствие (что мы и видим в коде). Guard Rules предписывают ИИ и разработчикам никогда не отступать от этих правил – фактически, это дополнительный уровень: даже ассистенты, помогающие с кодом, “проинструктированы” отказать, если задача нарушает Enforcement [60] [61].

Совокупно, механизмы работают: если кто-то попытается, к примеру, добавить `className` проп в Modal, то: - Это нарушит Guard Rules (AI отверг бы PR или сообщил о нарушении). - ESLint (rule `no-foundation-open-htmlattributes`) не даст собрать/смержить такой код. - И даже если бы как-то прошло, **LOCK-документ** помечает Foundation как immutable, и на код-ревью это будет отклонено, потому что нет unlock-задания.

Score: 10/10

Impact: Архитектурное последствие – практически гарантия архитектурной стабильности. Благодаря многоступенчатому enforcement, архитектура TUI становится

самоподдерживающейся: риски регрессии или “дрейфа” резко снижены. Новые участники команды не смогут неосознанно нарушить правила – инструменты их вовремя остановят или направят. 13-шаговый процесс интегрирует качество (сторибук/тесты) в само понятие “готовности” компонента, что повышает общую надёжность библиотеки. По сути, библиотека достигает уровня, когда каждый ее фундаментальный элемент проходит тщательную сертификацию. В сравнении, многие UI-библиотеки полагаются лишь на code review и unit-тесты; здесь же встроена **архитектурная проверка**. Это способствует долгосрочной устойчивости: по мере роста функциональности не происходит размывания первоначальных принципов (что часто случается без строгого enforcement).

What prevents this from being 10/10: *Ничего существенного – система enforcement реализована полностью.* Все необходимые механизмы присутствуют и активно действуют. Этот шаг оценивается на 10, потому что соединяет в себе и формальные правила, и реальные инструменты. Разве что можно отметить: - **Сложность поддержки:** Усложнённая инфраструктура (множество правил, скриптов) требует её самой поддерживать в актуальном состоянии. Если забыть обновить ESLint-правило при изменении структуры проекта, могут быть ложные срабатывания. Но эти издержки минимальны в сравнении с выгодами. - **Волокита процесса:** 13 шагов – это долго и требует дисциплины. Однако, для Foundation-компонентов (которых ограниченное число) это оправдано. Для Extension-компонентов процесс проще, что хорошо. - **Опциональность на Extension:** Как упомянуто, некоторые правила на Extension слой применяются advisory (не блокируют сборку). Теоретически, разработчик Extension может что-то нарушить, и линтер лишь предупредит. Но Extension – более свободный слой, это сознательно. В будущем, возможно, когда Extension стабилизируется, enforcement там тоже можно усилить. Но текущая модель балансирует гибкость и контроль.

(Эти моменты скорее нюансы, чем препятствия для идеала, поэтому балл не снижаем.)

Steps required to reach 10/10: (Принимая 10/10 как уже достигнутый, здесь скорее про удержание на 10) 1. **Мониторинг и обновление правил:** Каждый раз при появлении нового паттерна или правила из Authority Contracts – добавлять соответствующий ESLint-правило или CI-проверку. Например, если вводится новое ограничение (скажем, запрет на определённый HTML-элемент), стоит автоматизировать его проверку. 2. **Обучение команды:** Убедиться, что все разработчики знакомы с Guard Rules и 13-step процессом. Документы есть, но важно, чтобы это вошло в культуру. Тогда механизмы enforcement не будут восприниматься как внешнее давление, а станут естественной частью workflow (что уже, вероятно, происходит). 3. **Периодический аудит enforcement’a:** В рамках, скажем, ежеквартального техдолг-спринта, пересматривать правила: не нужно ли что-то скорректировать? Например, можно проанализировать последние PR: были ли случаи, когда правило пропустило проблему? Или наоборот, мешало без причины? Такой обзор позволит держать систему enforcement эффективной и без “дыр”. 4. **Распространение на Extension при необходимости:** Если Extension-слой разрастётся, и в нём начнутся нарушения (например, кто-то повторно изобрёл функциональность Foundation – маловероятно, но), можно усилить правила и для Extension. Пока это не нужно, но шаг к 10/10 (в абсолютном смысле) – полный контроль и там, если будет нужно.

Recommendation: Явное отсутствие действий (No-action) по исправлению. В сфере enforcement команда уже достигла лучших практик. Рекомендуется продолжать следовать установленным процессам и автоматизации. **Не ослаблять** правила ради удобства – краткосрочные обходы приведут к долгосрочным проблемам. Вместо этого, все изменения должны идти через процедуру unlock + аудит, как и прописано. Текущий подход ставит TUI в один ряд с наиболее выдержанными по дисциплине проектами, и отступать от этого не стоит.

Шаг 5: Зрелость библиотеки по сравнению с Radix, MUI, Chakra, Polaris, Spectrum

Claim: Tenerife UI (TUI) демонстрирует высокий уровень архитектурной зрелости, особенно в части консистентности и соблюдения собственных правил, но по **охвату компонентов и удобству экосистемы** она ещё догоняет ведущие библиотеки (MUI, Chakra и др.). В её пользу – чёткая структура Foundation/Extension, строгая типизация и токенизация, сопоставимые или превосходящие аналоги. Однако, в сравнении: - **Radix UI:** TUI успешно использует Radix как основу для отдельных компонентов (Dialog, Tabs, etc.), обеспечивая доступность. Однако Radix – это набор примитивов, а TUI строит над ним полный дизайн-слой. Зрелость TUI выше Radix в плане стилизации (Radix вовсе не задаёт стилей), но **сравнимо** по философии “accessibility first”. - **MUI/Chakra:** Эти библиотеки предоставляют десятки готовых компонентов “из коробки” и гибкую theming-систему. TUI, хотя и имеет все базовые блоки и даже Premium-секции, пока менее богата выбором. Например, MUI имеет дата-пикеры, таблицы, деревья, графики – TUI фокусируется на приложении Tenerife, и некоторых из подобных generic-компонентов может не быть. С другой стороны, TUI уже реализовала поддержку множества тем (через Theme Scaffolding CLI) и токен-валидации, что ставит её на уровень enterprise-систем (как Polaris, Spectrum) где multi-theme – стандарт [62] [63]. - **Polaris (Shopify) и Spectrum (Adobe):** Эти корпоративные дизайн-системы славятся строгими гайдлайнами и связкой с продуктовой экосистемой. TUI идёт по тому же пути: у неё есть чёткое разделение на продуктовые компоненты (которые **не входят** в библиотеку по правилу) [64], и UI-library остаётся универсальной. В отличие от open-source библиотек, Polaris/Spectrum ориентированы на внутреннее использование – здесь TUI близка по духу. Зрелость TUI проявляется в уровне документации: канонические контракты, lifecycle-процессы – далеко не каждая публичная библиотека имеет настолько подробную архитектурную базу. Это показатель серьёзности подхода.

Evidence: - **Компонентный охват:** В Extension Canonical State перечислено порядка ~50 компонентов Extension-слоя (включая примитивы и композиции) [65] [66]. Это уже внушительно для молодой библиотеки. Имеются и сложные составные компоненты вроде NotificationCenter, Dialog, Stepper. Однако, для сравнения: Material-UI предлагает ~100 компонентов; Chakra около 50, Radix ~30 примитивов. TUI охватывает потребности своего домена (Tenerife Music), но, например, нет упоминаний о таблицах данных, календарях, rich-text editors – то есть, **по широте** она скорее на 70–80% от уровня MUI. - **Настраиваемость и темизация:** TUI реализовала полноценную систему тем. В отчёте Progress видно, что завершён **Theme Scaffolding CLI** –CLI-инструменты для создания и валидации тем, ThemeProvider и переключатель тем, демонстрации в Storybook [62] [63]. Это выводит её на уровень Chakra/MUI, где тоже есть ThemeProvider. Причём, TUI делает акцент на строгой типизации тем (schema.ts) и автогенерации, что даже превосходит гибкость конкурентов (там часто тема – просто объект без схемы). В Polaris/Spectrum темизация тоже есть, так что здесь TUI в тренде. - **Документация и процессы:** Наличие таких документов, как **CANONICAL_DOCUMENTATION_INVENTORY.md** (38 файлов) говорит, что проект ведётся очень методично [67] [68]. Open-source аналоги (MUI) имеют обширную документацию для пользователей, но внутренняя архитектура у них не столь явно расписана. Polaris и Spectrum имеют тома документов, но они менее доступны. TUI фактически сочла документацию частью кода – это признак зрелости, ориентированной на долговременную поддержку, что характерно для enterprise-систем. - **Сообщество и проверка временем:** Здесь у конкурентов преимущество – MUI/Chakra существуют годами, используются тысячами проектов, что выявило и устранило множество багов. TUI пока молодая, и хотя внутренне она тщательно протестирована, масштабной эксплуатации снаружи (сообществом) не было. Качество 98/100 перед релизом [69]

впечатляет, но настоящая “битва” предстоит после публичного релиза, когда сторонние разработчики начнут применять библиотеку.

Score: 8/10

Impact: Архитектурное последствие – библиотека очень устойчива и выверена, но чтобы достичь полноты зрелости, ей нужно время и широта применения. На данный момент TUI отлично подходит для целей продукта Tenerife (и, потенциально, схожих приложений), давая им преимущества: единый стиль, быструю сборку интерфейса из готовых компонентов, минимальные риски “разламывания” при обновлениях благодаря lock-подходу. Однако, если сравнивать с MUI: MUI предлагает сразу множество готовых шаблонов и широту кастомизации (можно подогнать под любой бренд, изменив тему). TUI тоже позволяет менять тему, но **подогнать под совсем другой дизайн** будет сложнее – из-за множества зафиксированных контрактов, потребуется либо форкнуть Extension-уровень, либо разблокировать Foundation, что не тривиально. То есть, **в контексте архитектуры** TUI очень сильна, но **в контексте универсальности** пока уступает универсальным библиотекам. Впрочем, Polaris/Spectrum тоже не универсальны – они для своих компаний. TUI ближе к ним: это **заточенная под продукт** библиотека, которая со временем может развиться во что-то большее.

What prevents this from being 10/10: - **Компонентное покрытие:** Как отмечено, не весь возможный спектр UI-компонентов реализован. Некоторые нужны будут по мере роста требований (календарь, сложные таблицы, графические компоненты). Пока отсутствие их не критично, но для “соревноваться” с MUI надо рано или поздно закрыть эти пункты. - **Интеграция и удобство:** У топ-библиотек есть большие экосистемы: MUI – расширения (пикеры дат, rich components), Chakra – упор на DX (простота использования). TUI ещё предстоит построить вокруг себя удобства: возможно, генераторы кода, библиотека шаблонов экранов, и пр. Начало этому уже положено (Theme CLI, пресеты). - **Сообщество и проверенное качество:** Отсутствие багрепортов извне – палка о двух концах. Великие дизайн-системы становятся таковыми после многочисленных итераций по откликам. TUI нужно больше разнообразных кейсов использования, чтобы быть уверенным, что архитектура покрыла *всё*. Например, Polaris прошла через версии 1/2/3 со сменой подходов. Возможно, когда TUI начнут применять в новых проектах, всплывут новые требования, к которым надо будет адаптироваться без потери принципов.

Steps required to reach 10/10: 1. **Расширение библиотеки компонентами:** На основе анализа востребованности – реализовать недостающие общие компоненты. Например, если продукту понадобится DatePicker – спроектировать его по тем же канонам (Radix-основание, токены, variant API) и включить. Делать это постепенно, по мере приоритетов, но держать в плане, чтобы со временем TUI покрыла ~90% типовых UI. 2. **Улучшение Developer Experience:** Подготовить примеры, шаблоны, возможно, генератор интерфейсов. Чем легче внешнему разработчику пользоваться TUI, тем выше её ценность. MUI, например, предоставляет готовые темы, инструменты кастомизации онлайновые. Можно сделать Storybook-документацию с кодовыми снапшотами, чтобы людям было проще. 3. **Сообщество и feedback:** Если TUI планируется открыть (или даже внутри компании дать другим командам), организовать сбор обратной связи. Завести Issue tracker, поощрять сообщения о несоответствиях или предложениях. Реагировать на них через строгий процесс, но оперативно. Так библиотека «обрастёт» реальным опытом, который не всегда предусмотришь в теории. 4. **Сравнительный анализ с лидерами:** Регулярно проводить ревью: какие новые фичи появляются в Chakra/MUI, и нужны ли они нам? Например, MUI внедряет CSS-переменные для runtime-темизации – в TUI уже токены через CSS vars, так что здесь ОК. Следить за трендами accessibility, performance (например, переход на React Server Components и как библиотека будет это поддерживать). Архитектура у TUI гибкая, но важно не

упустить технологические изменения. 5. **Маркетинг (если релиз во внешний мир):** Подготовить качественный сайт-документацию, гайды по миграции, список best practices. Это не архитектурный шаг, но для зрелости в глазах пользователей – критично. Внутри компании – провести обучающие сессии для команд по использованию TUI.

Recommendation: Действие. Продолжать развивать библиотеку планомерно, **без уступок качеству**. TUI уже превзошла многие популярные библиотеки по строгости и согласованности – сохранить это как уникальное преимущество. Одновременно, стремиться к **повышению удобства и широты**: добавлять новые компоненты, улучшать документацию, делать систему привлекательной для разработчиков. В перспективе, если добиться баланса, TUI может стать не просто внутренним инструментом, а эталонным UI-фреймворком. Рекомендуется сравнивать себя с конкурентами не только технически, но и по ощущению пользователя: цель – библиотека, которую любят использовать, а не только уважают за архитектуру.

Cursor Master Tasks (для улучшения областей < 10/10)

- **Task 1: Обновление документации и API-инвентаря.** Привести публичную документацию в полное соответствие с кодом: удалить упоминания запрещённых компонент, обновить устаревшие разделы (типографика, токены) и отметить актуальную версию. Настроить проверку консистентности доков на будущее.
- **Task 2: Усовершенствование дизайн-токен системы.** Интегрировать сложные стили (например, градиенты) в модель токенов, устранив manual whitelist. Добавить при необходимости семантический слой токенов для статусов. Обновить документацию TUI_TOKEN_SYSTEM.md с учётом новых токенов.
- **Task 3: Очистка публичного API.** Убрать депрекетированные пропы и дубли компонентов. Полностью удалить или изолировать legacy-реализации (Badge, старый Link и проч.) чтобы исключить их случайное использование. Обновить TypeScript-типизацию, выпустить новую мажорную версию с перечислением breaking changes.
- **Task 4: (На поддержание) Регулярный аудит Enforcement.** Внедрить практику периодического аудита правил ESLint и lock-контрактов: удостовериться, что они покрывают новые изменения. Добавить новые правила при необходимости, распространить enforcement на Extension-слой, если появятся признаки дрейфа.
- **Task 5: Увеличение зрелости и охвата.** Спланировать реализацию недостающих компонентов (например, DatePicker, DataTable) с соблюдением всех архитектурных принципов. Улучшить developer experience: создать Storybook-документацию с примерами кода, написать гайд по миграции с других библиотек. Собирать фидбек от первых пользователей и провести сессии улучшения по их замечаниям.

---

1 4 5 6 7 API_AND_TOKENS_AUDIT_SUMMARY.md
https://github.com/Tureckiy-zart/TUI/blob/a2712f63d602d62ac7001fb100cfc3155f616109/docs/reference/API_AND_TOKENS_AUDIT_SUMMARY.md

2 3 33 34 35 65 66 TUI_EXTENSION_CANONICAL_STATE.md
file://file-V8wKqokRbTCizFQMh5LQij

8 9 10 11 12 13 TUI_TOKEN_SYSTEM.md
file://file-SoxoZeBpEA3ePhMs8ykHaQ

14 15 16 32 token-cva.ts
https://github.com/Tureckiy-zart/TUI/blob/a2712f63d602d62ac7001fb100cfc3155f616109/src/FOUNDATION/lib/token-cva.ts

17 18 19 check-ui-consistency.ts
https://github.com/Tureckiy-zart/TUI/blob/a2712f63d602d62ac7001fb100cfc3155f616109/scripts/check-ui-consistency.ts

20 21 22 23 TYPOGRAPHY_AUTHORITY_CONTRACT.md
https://github.com/Tureckiy-zart/TUI/blob/d973af8980fb786fffd9a970eb8a1d73961ae4bd/docs/architecture/TYPOGRAPHY_AUTHORITY_CONTRACT.md

24 25 26 RADIUS_AUTHORITY_CONTRACT.md
https://github.com/Tureckiy-zart/TUI/blob/d973af8980fb786fffd9a970eb8a1d73961ae4bd/docs/architecture/RADIUS_AUTHORITY_CONTRACT.md

27 28 30 31 37 Text.tsx
https://github.com/Tureckiy-zart/TUI/blob/a2712f63d602d62ac7001fb100cfc3155f616109/src/PRIMITIVES/Text/Text.tsx

29 38 59 60 61 64 TUI_CURSOR_GUARD_RULES.md
file://file-Y4ubCvo9Yit5dQKbNJWwa5

36 39 40 41 42 TUI_ARCHITECTURE_LOCK.md
file://file-ENTLsxUNWxgoCxyQf6mfvW

43 44 EXTENSION_AUTHORITY_CONTRACT.md
file://file-7XrRS9gC4GzQdfpzAFoBEz

45 46 47 48 49 50 51 ESLINT_SETUP.md
https://github.com/Tureckiy-zart/TUI/blob/a2712f63d602d62ac7001fb100cfc3155f616109/docs/architecture/ESLINT_SETUP.md

52 53 54 55 56 57 58 FOUNDATION_LIFECYCLE_PROCESS_INDEX.md
file://file-Dt8Fyg5Uo3p2usD4uckLBN

62 63 69 PROJECT_PROGRESS.md
file://file-3kKSAHGJ3AdkQhbUX4PVu9

67 68 CANONICAL_DOCUMENTATION_INVENTORY.md
https://github.com/Tureckiy-zart/TUI/blob/a2712f63d602d62ac7001fb100cfc3155f616109/docs/CANONICAL_DOCUMENTATION_INVENTORY.md