FRONTEND РАЗРАБОТЧИК

# Чеклист Frontend разработчика

Практическая база для разработки интерфейсов: UX, компоненты, состояние, интеграции, доступность, производительность, тесты и релизы.

---

## UX и требования

Frontend отвечает не только за пиксели, но и за понятный путь пользователя: состояния, ошибки, загрузку, пустые экраны и реальные данные.

### Разобрать пользовательский сценарий

Понять путь пользователя, цель экрана и ключевые решения до написания компонента.

**Что это дает:** Разбор сценария помогает не построить красивый интерфейс, который не решает задачу пользователя.

**Как выполнить:**

1. Опишите вход в сценарий и ожидаемый результат.
2. Проверьте happy path, ошибки и пустые состояния.
3. Согласуйте спорные места с продуктом и дизайном.

**Критерии приемки:**

- Сценарий понятен.
- Состояния перечислены.
- Неясные требования вынесены в вопросы.

### Проверить интерфейс на реальном контенте

Убедиться, что текст, числа, длинные имена, пустые значения и локализация не ломают UI.

**Что это дает:** Реальный контент быстро показывает слабые места верстки и логики отображения.

**Как выполнить:**

1. Подставьте длинные имена, большие числа и пустые значения.
2. Проверьте переносы, обрезку и адаптив.
3. Согласуйте правила отображения unknown/null.

**Критерии приемки:**

- UI не ломается на длинном контенте.
- Пустые состояния оформлены.
- Текст не перекрывает элементы.

### Описать состояния интерфейса

Для каждого экрана предусмотреть loading, empty, error, success, disabled и permission states.

**Что это дает:** Полные состояния делают интерфейс предсказуемым и снижают количество аварийных решений после интеграции API.

**Как выполнить:**

1. Составьте список состояний для каждого блока.
2. Согласуйте тексты ошибок и пустых экранов.
3. Проверьте transitions между состояниями.

**Критерии приемки:**

- Состояния реализованы.
- Пользователь понимает, что происходит.
- Нет бесконечных loaders без действия.

---

## Компоненты

Компоненты должны быть предсказуемыми, изолированными и удобными для повторного использования без копирования логики и стилей.

## Собрать переиспользуемый компонент

Выделить UI-элемент с понятными props, states и ответственностью.

**Что это дает:** Переиспользуемый компонент ускоряет разработку и уменьшает расхождение интерфейса в разных частях продукта.

**Как выполнить:**

1. Определите назначение компонента.
2. Задайте минимальный набор props.
3. Покройте варианты: size, state, disabled, loading.

**Критерии приемки:**

- Компонент не содержит лишней бизнес-логики.
- Props документированы.
- Состояния можно проверить отдельно.

## Использовать design tokens

Применять согласованные цвета, отступы, типографику и радиусы вместо произвольных значений.

**Что это дает:** Design tokens сохраняют единый визуальный язык и упрощают изменение темы или дизайн-системы.

**Как выполнить:**

1. Найдите существующие CSS variables или theme tokens.
2. Заменяйте hardcoded значения токенами.
3. Проверяйте контраст и состояния hover/focus.

**Критерии приемки:**

- Компонент использует токены.
- Нет случайных цветов и отступов.
- Состояния соответствуют дизайн-системе.

### Документировать контракт компонента

Описать props, события, ограничения и примеры использования компонента.

**Что это дает:** Документация снижает неправильное использование компонента и ускоряет onboarding.

**Как выполнить:**

1. Опишите обязательные и опциональные props.
2. Добавьте примеры типовых сценариев.
3. Зафиксируйте ограничения и accessibility требования.

**Критерии приемки:**

- Есть примеры использования.
- Props понятны без чтения внутреннего кода.
- Ограничения явно указаны.

---

## Данные и состояние

Интерфейс должен корректно работать с loading, error, empty, optimistic updates и конфликтами данных.

### Интегрировать API с обработкой состояний

Подключить данные так, чтобы UI корректно показывал загрузку, ошибку, пустой результат и успех.

**Что это дает:** Качественная интеграция API делает интерфейс устойчивым к задержкам, ошибкам и неполным данным.

**Как выполнить:**

1. Используйте typed contract или схему ответа.
2. Обработайте loading, error, empty и success.
3. Проверьте retry и отмену запроса при необходимости.

**Критерии приемки:**

- Все состояния отображаются.
- Ошибки понятны пользователю.
- Нет гонок при быстром переключении.

### Настроить валидацию формы

Проверить пользовательский ввод на клиенте и корректно показать ошибки сервера.

**Что это дает:** Хорошая форма снижает количество ошибок, повторных отправок и обращений в поддержку.

**Как выполнить:**

1. Разделите клиентскую и серверную валидацию.
2. Показывайте ошибку рядом с полем.
3. Сохраняйте введенные данные после ошибки.

**Критерии приемки:**

- Ошибки связаны с полями.
- Submit защищен от дублей.
- Серверные ошибки отображаются понятно.

### Продумать кеш и обновление данных

Определить, когда данные кешируются, обновляются, инвалидируются или показываются оптимистично.

**Что это дает:** Правильный cache strategy делает интерфейс быстрым и не показывает пользователю устаревшее состояние после действий.

**Как выполнить:**

1. Определите stale time для данных.
2. Инвалидируйте кеш после мутаций.
3. Осторожно используйте optimistic update.

**Критерии приемки:**

- После действия UI показывает актуальные данные.
- Нет лишних запросов.
- Ошибки optimistic update откатываются.

---

## Качество интерфейса

Качество frontend видно пользователю сразу: скорость, стабильность, доступность, отзывчивость и отсутствие визуальных поломок.

## Проверить доступность интерфейса

Убедиться, что интерфейс доступен с клавиатуры, screen reader и корректной семантикой.

**Что это дает:** Accessibility улучшает продукт для большего числа пользователей и часто повышает общее качество HTML.

**Как выполнить:**

1. Проверьте keyboard navigation и focus order.
2. Используйте semantic HTML и labels.
3. Проверьте контраст и aria только там, где нужно.

**Критерии приемки:**

- Основные сценарии доступны с клавиатуры.
- Формы имеют labels.
- Контраст соответствует требованиям.

## Оптимизировать Core Web Vitals

Проверить LCP, INP и CLS на ключевых страницах.

**Что это дает:** Web Vitals влияют на пользовательское ощущение скорости и стабильности интерфейса.

**Как выполнить:**

1. Определите LCP element.
2. Сократите тяжелый JavaScript и долгие задачи.
3. Задайте размеры медиа и резервируйте место под динамические блоки.

**Критерии приемки:**

- LCP, INP и CLS измерены.
- Критические проблемы исправлены.
- Оптимизации проверены на реальных устройствах.

### Добавить тесты интерфейса

Покрыть критическую логику компонентными и e2e тестами.

**Что это дает:** Тесты помогают не ломать важные сценарии при рефакторинге и изменении API.

**Как выполнить:**

1. Выберите критические пользовательские сценарии.
2. Пишите тесты на поведение, а не на внутреннюю реализацию.
3. Добавьте e2e для главного business flow.

**Критерии приемки:**

- Критические сценарии покрыты.
- Тесты стабильны в CI.
- Падение теста показывает понятную причину.

---

## Сборка и релиз

Frontend-релиз требует контроля сборки, переменных окружения, совместимости браузеров, мониторинга ошибок и отката.

### Проверить production build

Убедиться, что сборка проходит и использует корректные переменные окружения.

**Что это дает:** Многие frontend-проблемы появляются не в dev-режиме, а в production build: env, minification, routes, assets.

**Как выполнить:**

1. Запустите production build локально или в CI.
2. Проверьте env variables и base path.
3. Откройте собранную версию и пройдите ключевой сценарий.

**Критерии приемки:**

- Build проходит.
- Нет ошибок assets и routes.
- Ключевые сценарии работают на production build.

## Настроить мониторинг frontend-ошибок

Собирать JS errors, failed requests и важный пользовательский контекст после релиза.

**Что это дает:** Мониторинг показывает реальные проблемы пользователей, которые не всегда воспроизводятся в тестовой среде.

**Как выполнить:**

1. Подключите error tracking.
2. Добавьте release version и sourcemaps.
3. Настройте алерты на рост ошибок.

**Критерии приемки:**

- Ошибки видны с версией релиза.
- Sourcemaps работают.
- Критические ошибки создают уведомление.

## Использовать feature flags для рискованных изменений

Выкатывать функциональность постепенно и иметь возможность быстро выключить ее без нового деплоя.

**Что это дает:** Feature flags снижают риск релиза и помогают тестировать изменения на ограниченной аудитории.

**Как выполнить:**

1. Оборачивайте рискованные функции флагом.
2. Опишите владельца и срок жизни флага.
3. Удаляйте устаревшие flags после стабилизации.

**Критерии приемки:**

- Флаг управляется без деплоя.
- Есть план удаления.
- Fallback состояние проверено.