Андреева Арина Александровна – магистрант, курс 2, кафедра «Цифровые технологии обработки данных», РТУ МИРЭА, г. Москва

Курочкин Вадим Сергеевич – магистрант, курс 2, кафедра «Цифровые технологии обработки данных», РТУ МИРЭА, г. Москва

Гетман Владислав Антонович – магистрант, курс 2, кафедра «Цифровые технологии обработки данных», РТУ МИРЭА, г. Москва

ПАТТЕРНЫ ОТКАЗОУСТОЙЧИВОСТИ МИКРОФРОНТЕНД-АРХИТЕКТУР В УСЛОВИЯХ ВЫСОКОЙ НАГРУЗКИ

Архитектура микрофронтендов рассматривается как логичное продолжение микросервисного подхода на уровень пользовательского интерфейса. Она позволяет разделить крупный веб-продукт на набор относительно независимых фронтенд-модулей, что упрощает масштабирование разработки и эволюцию технологического стека. Однако вопросы отказоустойчивости, особенно в условиях высокой нагрузки и частичных отказов, исследованы заметно слабее, чем организационные и модульные преимущества. В работе предлагается компактная модель отказов на уровне пользовательского интерфейса, вводится набор метрик (доступность интерфейса, доля частичной деградации, доля полностью сломанного интерфейса, коэффициент изоляции отказов и время восстановления интерфейса), а также рассматриваются паттерны повышения отказоустойчивости микрофронтенд-архитектур.

КЛЮЧЕВЫЕ СЛОВА Микрофронтенды, Отказоустойчивость, высоконагруженные системы, graceful degradation, fault injection.

Microfrontend architecture is considered a logical extension of the microservices approach to the user interface level. It enables the decomposition of a large web product into a set of relatively independent frontend modules, which simplifies development scaling and the evolution of the technology stack. However, issues of fault tolerance, especially under high load and partial failures, have received significantly less attention than the organizational and modular benefits. This paper proposes a compact failure model at the user interface level, introduces a set of metrics (interface availability, partial degradation rate, completely broken interface rate, failure isolation factor, and interface recovery time), and examines patterns for improving the fault tolerance of microfrontend architectures.

KEYWORDS Microfrontends, Fault Tolerance, Highly Loaded Systems, Graceful Degradation, Fault Injection.

ВВЕДЕНИЕ

Развитие веб-технологий приводит к постоянному росту требований к пользовательским интерфейсам крупных систем. Современные веб-приложения должны одновременно выдерживать высокую нагрузку, быстро обновляться в ответ на требования рынка и оставаться устойчивыми к отказам инфраструктуры и отдельных сервисов. На этом фоне всё более распространённым подходом к организации фронтенд части становится архитектура микрофронтендов, рассматриваемая как логичное продолжение микросервисной парадигмы на уровне пользовательского интерфейса.

Суть микрофронтенд подхода заключается в разделении монолитного фронтенд приложения на набор относительно независимых модулей (микрофронтендов), каждый из которых отвечает за свой бизнес-домен и может разрабатываться, тестироваться и развёртываться отдельной командой. Это позволяет уменьшить взаимозависимости между командами, ускорить выпуск изменений и упростить постепенную миграцию на новые технологии. Однако в условиях высокой нагрузки и сложной распределённой инфраструктуры подобное дробление интерфейса порождает новые вопросы, связанные с отказоустойчивостью и деградацией пользовательского опыта.

На практике крупные высоконагруженные системы чаще всего сталкиваются не с полными отказами, а с частичными инцидентами: деградацией одного из серверных сервисов, локальными сетевыми проблемами, ошибками конфигурации для отдельного домена или конфликтами версий общих библиотек. В такой ситуации от фронтенда ожидается не только сохранение работоспособности, но и способность к управляемой деградации (graceful degradation): пользовательский интерфейс должен корректно обрабатывать частичные сбои, ограничивать их влияние (blast radius) и, по возможности, автоматически восстанавливаться после кратковременных нарушений.

1. ЦЕЛЬ ИССЛЕДОВАНИЯ

Цель работы: проанализировать и продемонстрировать архитектурные решения, повышающие отказоустойчивость пользовательских интерфейсов в условиях высокой нагрузки.

Объект исследования – высоконагруженные веб-системы с распределённой архитектурой, ориентированные на массовое обслуживание пользователей через браузерный интерфейс.

Для достижения цели в работе решаются следующие задачи:

1. Проанализировать существующие подходы построения микрофронтенд архитектур и практик обеспечения отказоустойчивости веб-интерфейсов.
2. Сформировать модель отказов на уровне пользовательского интерфейса и ввести метрики для оценки отказоустойчивости.
3. Обозначить набор метрик, описывающих отказоустойчивость на уровне пользовательского интерфейса.
4. Выделить и описать паттерны, направленные на повышение устойчивости микрофронтендов.
5. Спроектировать экспериментальный стенд на основе микрофронтенд архитектуры и продемонстрировать влияние различных конфигураций архитектуры на предложенные метрики.

2. Микрофронтенды в контексте высоконагруженных веб-систем

2.1 Предметная область и особенности интерфейса

В рамках настоящего исследования рассматривается предметная область высоконагруженных веб-систем, ориентированных на массовое взаимодействие с пользователями через браузерный интерфейс. Под высоконагруженными системами понимаются программно-аппаратные комплексы, в которых обрабатывается значительный объём пользовательских запросов (десятки и сотни

тысяч запросов в минуту и более), присутствуют выраженные пиковые нагрузки, связанные с внешними факторами (маркетинговые кампании, распродажи, новости и события), имеются строгие требования к доступности (часто на уровне 99,9 % и выше) и времени отклика интерфейса. К такой категории относятся:

- крупные платформы электронной коммерции (интернет-магазины, маркетплейсы, агрегаторы);
- сервисы бронирования (авиабилеты, отели, мероприятия);
- финансовые и финтех-сервисы, предоставляющие пользователям доступ к операциям через веб-интерфейс;
- SaaS (Software as a Service)-продукты, основным каналом использования которых является веб-клиент.

Общая особенность таких систем – интерфейс не является «тонким клиентом». Это полноценное одностраничное приложение (SPA) или набор тесно связанных SPA, тесно взаимодействующих с множеством сервисов. Нагрузка на UI выражается не только в количестве запросов, но и в сложности состояния: большое число компонентов, завязанных на разные домены и внешние зависимости.

Традиционный фронтенд-монолит в этом контексте сталкивается с рядом ограничений: рост времени сборки и релизов, усиление рисков регрессий, сложность миграций на новые технологии. Возникает потребность в архитектуре, позволяющей разделить интерфейс по доменам и командам, сохранив при этом целостность пользовательского опыта.

2.2 Архитектура современных веб-систем

На пользовательском уровне традиционно использовались крупные SPA монолиты (рисунок 1), разрабатываемые в рамках одного репозитория и зачастую одной команды. По мере роста продукта такой подход начинает создавать проблемы.

Во-первых, замедляется доставка изменений, любая модификация требует пересборки и повторного тестирования всего приложения. Во-вторых, усложняется технологическая эволюция, переход на новый фреймворк или библиотеку требует

масштабных миграций. Также большое количество разработчиков и команд в одной кодовой базе приводят к обострению организационных проблем.

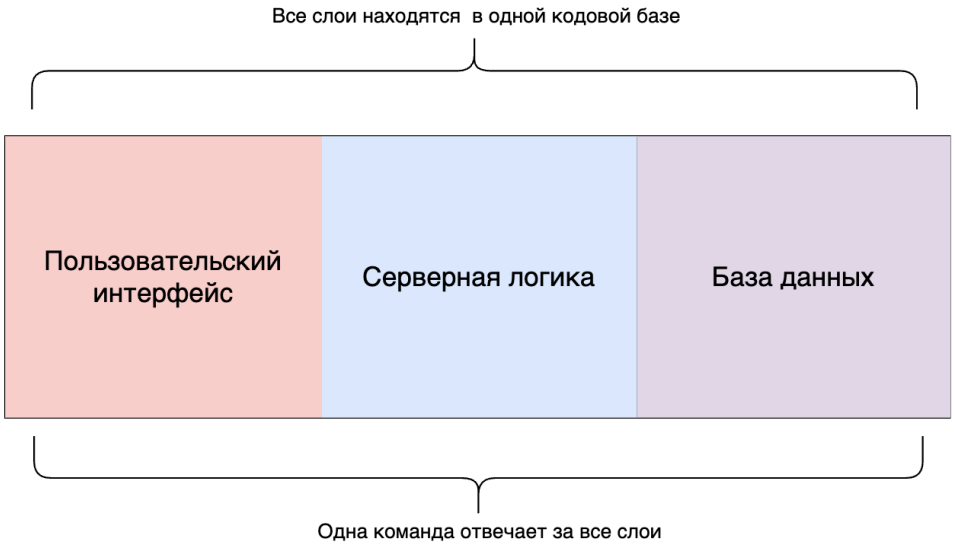

Рисунок 1 – Монолитная архитектура веб-приложения.

Эти ограничения стали одним из ключевых стимулов к применению архитектуры микрофронтендов, которая переносит принципы микросервисного проектирования на слой пользовательского интерфейса, что позволяет нескольким командам независимо друг от друга работать над частями большого продукта. В рамках микрофронтенд подхода пользовательский интерфейс разбивается на набор отдельных модулей, каждый из которых:

- соответствует определённому бизнес-домену (например, каталог товаров, корзина, оформление заказа, личный кабинет, административная панель);
- разрабатывается и сопровождается отдельной командой;
- может использовать собственный стек технологий (в разумных пределах, согласованных на уровне архитектуры и платформы);
- поставляется и развёртывается независимо от других модулей.

На рисунке 2 представлен пример микросервисной архитектуры.

Рисунок 2 – Микросервисная архитектура веб-приложения.

2.3 Требования к отказоустойчивости в высоконагруженных системах

Высоконагруженные веб-системы предъявляют повышенные требования к отказоустойчивости как серверной, так и части пользовательского интерфейса. Для предметной области, связанной с электронными продажами и финансовыми операциями, особенно критичны:

- доступность интерфейса. Невозможность оформить заказ или операцию даже в течение нескольких минут может приводить к существенным финансовым потерям;
- предсказуемость поведения. При частичных отказах или деградации пользователь должен получать понятную обратную связь (сообщения об ошибках, подсказки, возможности повторить попытку);
- ограничение области отказа. Сбой одной функциональной части (например, рекомендаций или истории заказов) не должен приводить к отказу критичных сценариев (оформление заказа, платежи).

Важно отметить, что в реальных условиях отказы редко носят характер «полной остановки» всей системы. Гораздо чаще происходят:

- деградация производительности отдельных сервисов;

- временная недоступность сторонних интеграций;
- ошибки конфигурации или обновления конкретного домена.

Поэтому особый интерес представляет именно поведение микрофронтенд интерфейса в условиях частичных отказов и его способность к управляемой деградации.

3 Модель отказоустойчивости и метрики интерфейса

3.1 Обеспечение отказоустойчивости: общие практики

Если рассматривать отказоустойчивость с точки зрения фронтенд разработки в целом, то можно выделить ряд устоявшихся практик:

– обработка ошибок на уровне запросов и компонентов с отображением понятных пользователю сообщений;

– использование тайм-аутов для сетевых запросов, чтобы предотвратить бесконечное ожидание ответа;

– fallback-интерфейсы (заглушки, упрощённые версии блоков, «скелетоны» вместо контента);

– кеширование данных для ускорения повторных запросов и поддержки сценариев при нестабильном соединении;

– разделение интерфейса на функциональные области, позволяющее в случае проблем с одной частью оставить остальные работоспособными.

Эти подходы, как правило, рассматриваются на уровне отдельных страниц или компонент, без явной привязки к архитектуре микрофронтендов как к целостному решению.

Анализируя публикации и практические кейсы, можно заметить несколько неявных предположений. Модульность автоматически = отказоустойчивость. Часто предполагается, что раз интерфейс разделён на микрофронтенды, то отказ одного из них не может повлиять на остальные. На практике это не всегда так:

shared-зависимости, глобальное состояние, общие обработчики ошибок и событийная шина могут создать скрытые связи.

Отказоустойчивость – зона ответственности» серверной инфраструктуры. Фронтенд рассматривается как тонкий слой, который лишь отображает результат, а вся логика по ретраям, деградации и резервированию перекладывается на серверную часть. В реальных высоконагруженных системах такая модель оказывается недостаточной: поведение интерфейса при частичных отказах критично для UX.

Отсутствие формализованных метрик для UI. Большинство метрик доступности и отказоустойчивости формулируются на уровне сервисов (доля успешных ответов, время отклика API), а не на уровне пользовательского интерфейса. Из-за этого сложно обосновывать архитектурные решения на стороне клиента с опорой на количественные данные.

Учитывая описанный контекст, в данной работе предлагается рассматривать отказоустойчивость не только как свойство серверного слоя, но и как качественный атрибут фронтенд архитектуры, ввести набор метрик, описывающих поведение микрофронтенд-интерфейса при частичных отказах, проанализировать и систематизировать паттерны, применимые именно на уровне микрофронтендов, а также продемонстрировать на экспериментальном стенде, как различные архитектурные конфигурации влияют на эти метрики.

3.2 Типы отказов и уровни последствий

Для корректного анализа отказоустойчивости пользовательского интерфейса необходимо выделить типовые классы отказов, с которыми сталкивается система в реальной эксплуатации. Выделим четыре основных типа отказов:

Отказ отдельного микрофронтенда – ошибка сборки (некорректная конфигурация сборщика, несовместимые версии библиотек), критическая runtime-ошибка, проблема с загрузкой артефактов или конфликт версий общих библиотек. В результате пользователь либо не видит соответствующую часть интерфейса,

либо сталкивается с белым блоком, бесконечной индикацией загрузки или полной недоступностью страницы.

Деградация или отказ конкретного сервиса – рост латентности (например, из-за блокировок в базе данных или перегрузки ресурса), увеличение доли ответов 5xx, временная недоступность эндпоинтов. Для фронтенда это выражается в медленной загрузке данных, ошибках при выполнении действий либо необходимости вывода сообщений о временной недоступности функций.

Сетевые проблемы – высокий RTT (время приема-передачи), потери пакетов, частичная недоступность CDN или API для части пользователей. Это может приводить к тому, что одни микрофронтенды загружаются быстрее, другие – существенно медленнее или не загружаются вовсе.

Ошибки интеграции – нарушения контрактов между фронтендом и сервисами, ошибки маршрутизации, ошибки в глобальном состоянии либо событийной шине. Подобные ошибки особенно характерны для систем с большим количеством команд и быстро эволюционирующей архитектурой.

Отметим, что описанные классы отказов могут проявляться как по отдельности, так и в комбинации, усиливая негативный эффект друг друга. Модель отказоустойчивости должна учитывать все перечисленные сценарии.

С практической точки зрения важно не только классифицировать причины отказов, но и оценивать их последствия для конечного пользователя. В рамках работы предлагается выделять 4 уровня последствий:

- отсутствие заметного влияния (пользователь не видит проблем);
- локальная частичная деградация (недоступен некритичный блок, основной сценарий выполняется);
- критичная частичная деградация (основной сценарий выполняется с трудностями, требуются дополнительные действия);

- полный отказ интерфейса (невозможно оформить заказ, выполнить оплату и т.д.).

Именно переходы между этими уровнями являются ключевыми: задача архитектора – спроектировать систему так, чтобы максимальное число инцидентов оказывалось на уровнях 1–2, а частота сценариев уровня 4 стремилась к минимуму.

3.3 Метрики отказоустойчивости интерфейса

Для количественной оценки поведения интерфейса в работе используются следующие метрики:

UI availability (UA) – доля пользовательских сессий, в которых все критичные компоненты для выбранного сценария были успешно отображены и перешли в работоспособное состояние в пределах заданного порога времени (SLA по времени загрузки). Высокое значение UA свидетельствует о том, что большинство пользователей получают полностью работоспособный интерфейс в разумные сроки.

Partial degradation rate (PDR) – доля сессий, в которых хотя бы один некритичный компонент работал в режиме деградации (заглушка, устаревшие данные), но основной сценарий был выполнен успешно. Положительным эффектом внедрения паттернов отказоустойчивости считается рост PDR при снижении BUR (см. далее), то есть перенос части сессий из состояния полного отказа в состояние частичной деградации.

Broken UI rate (BUR) – доля сессий, в которых интерфейс оказался непригоден для выполнения ключевого сценария: не отрисовались критичные компоненты, произошли непреодолимые ошибки, либо пользователь не смог завершить целевое действие. Основной целью применения паттернов отказоустойчивости является уменьшение BUR при сохранении или повышении UA.

Isolated failure ratio (IFR) – доля инцидентов, последствия которых были локализованы в пределах одного микрофронтенда (или заранее определённой

группы). Высокое значение IFR свидетельствует о том, что архитектура обеспечивает хорошую изоляцию отказов: проблемы чаще остаются локальными, не распространяясь на другие части интерфейса.

UI recovery time (URT) – время между переходом интерфейса в ошибочное или деградированное состояние и восстановлением нормальной работы (автоматически или после действия пользователя).

Эти показатели позволяют сравнивать различные конфигурации микрофронтенд-архитектуры с точки зрения устойчивости интерфейса к частичным отказам.

4. ПАТТЕРНЫ ОТКАЗОУСТОЙЧИВОСТИ В МИКРОФРОНТЕНД-АРХИТЕКТУРЕ

4.1 Паттерн client-side timeouts и повторные попытки

Первый базовый паттерн – явное ограничение времени ожидания ответов от сервисов на уровне пользовательского интерфейса. Для каждого важного запроса задаётся верхний предел времени ожидания, по истечении которого запрос принудительно прекращается и переводит компонент в состояние ошибки.

В сочетании с ограниченным количеством повторных попыток (retries) это позволяет:

- избегать бесконечной индикации загрузки;
- удерживать время реакции интерфейса в разумных пределах;
- явным образом фиксировать факт ошибки в логах фронтенда.

Важно избегать чрезмерно агрессивных повторных попыток, чтобы не усиливать нагрузку на уже деградировавший сервис.

4.2 Паттерн fallback-UI и единая модель состояний

Fallback-UI расширяет предыдущий паттерн, определяя, как именно интерфейс должен выглядеть в случае ошибки. Вместо разрозненных сообщений и «ломаной»

верстки вводится единая модель состояний интерфейса (loading, success, error, degraded) и соответствующий fallback-UI:

- унифицированные компоненты для состояния error (короткое понятное сообщение, кнопка «повторить»);
- режимы degraded, при которых временно скрываются или упрощаются некритичные блоки (например, рекомендации, промо-блоки), но сохраняется возможность оформить заказ или выполнить основную операцию;
- использование локального кеша (последнего успешного состояния) для отображения данных в degraded mode.

Таким образом, часть сессий, которые иначе приводили бы к полному отказу, переводится в управляемую частичную деградацию.

4.3 Паттерн degraded mode (режимы деградации)

Degraded mode предполагает, что компонент или целый микрофронтенд может функционировать в упрощённом режиме, если часть зависимостей недоступна.

Например, при недоступности сервиса рекомендаций на карточке товара отображается только основная информация и кнопка «добавить в корзину», а блок рекомендаций скрывается или при проблемах с получением истории заказов личный кабинет показывает только базовую информацию о пользователе и кнопки выхода/изменения пароля;

Применение режима деградации позволяет уменьшать долю полностью сломанных сценариев, переводя их в категорию частичной, но управляемой деградации.

4.4 Паттерн локального кеширования

Локальное кеширование на пользовательском интерфейсе (localStorage, IndexedDB, Cache API) является важным компонентом стратегии отказоустойчивости. При первых успешных запросах данные сохраняются локально, далее при временной недоступности сервиса интерфейс может использовать закешированные данные.

Таким образом, пользователь получает возможность продолжить работу, даже если информация потенциально устаревшая.

При этом необходимо явно задавать правила устаревания, информировать пользователя о возможной неактуальности данных, и при необходимости определять перечень сущностей, для которых такой подход допустим.

Кеширование особенно эффективно в сочетании с degraded mode и повторными попытками: интерфейс может сначала показать кеш, затем, по мере восстановления сервисов, обновить данные.

4.5 Паттерн архитектурной изоляции микрофронтендов

Для снижения риска каскадных отказов важную роль играет архитектурная изоляция микрофронтендов:

- минимизация shared-слоя (общие остаются только базовые инфраструктурные компоненты – дизайн-система, логирование, аналитика);
- отказ от общего глобального состояния, в которое могут записывать несколько микрофронтендов;
- явное управление совместно используемыми зависимостями при динамической загрузке модулей;
- использование Web Components или других механизмов, ограничивающих влияние одного микрофронтенда на другой.

Такая изоляция повышает коэффициент IFR и уменьшает вероятность того, что сбой в одном домене нарушит работу остальных.

5. ПРОЕКТ ЭКСПЕРИМЕНТАЛЬНОГО СТЕНДА И МЕТОДИКИ ИССЛЕДОВАНИЯ

5.1 Цели и гипотезы эксперимента

Для получения экспериментальной оценки влияния различных архитектурных решений и паттернов отказоустойчивости на поведение пользовательского

интерфейса в условиях частичных отказов и высокой нагрузки эксперимента могут быть сформулированы следующие гипотезы:

Гипотеза 1. Введение на фронтенде явных client-side timeouts и унифицированного fallback-UI приводит к снижению доли полностью сломанных сессий (Broken UI rate, BUR) по сравнению с базовой конфигурацией без указанных паттернов.

Гипотеза 2. Добавление degraded-режимов и локального кеширования критически важных данных (поверх client-side timeouts и fallback-UI) дополнительно снижает BUR и уменьшает время восстановления интерфейса (UI recovery time, URT), при этом увеличивая долю частичной деградации (Partial degradation rate, PDR).

Гипотеза 3. Усиление архитектурной изоляции микрофронтендов (сокращение shared-слоя, минимизация глобального состояния, явный контроль зависимостей) повышает коэффициент изоляции отказов (Isolated failure ratio, IFR), то есть увеличивает долю инцидентов, затрагивающих только один микрофронтенд.

5.2 Архитектура экспериментального стенда

Для демонстрации предложенного подхода был реализован упрощённый экспериментальный стенд интернет-магазина (Рисунок 3). На уровне пользовательского интерфейса одно shell-приложение динамически подгружало четыре микрофронтенда: Catalog MF (каталог товаров), Product MF (карточка товара), Cart MF (корзина и оформление заказа) и Profile MF (личный кабинет).

Микрофронтенды взаимодействуют с соответствующими сервисами (Catalog Service, Cart Service, Profile Service, Product Service) через REST-API. Между фронтендом и сервисами реализован слой имитации отказов (fault injection), позволяющий управляемо замедлять ответы и возвращать ошибки для выбранных эндпоинтов.

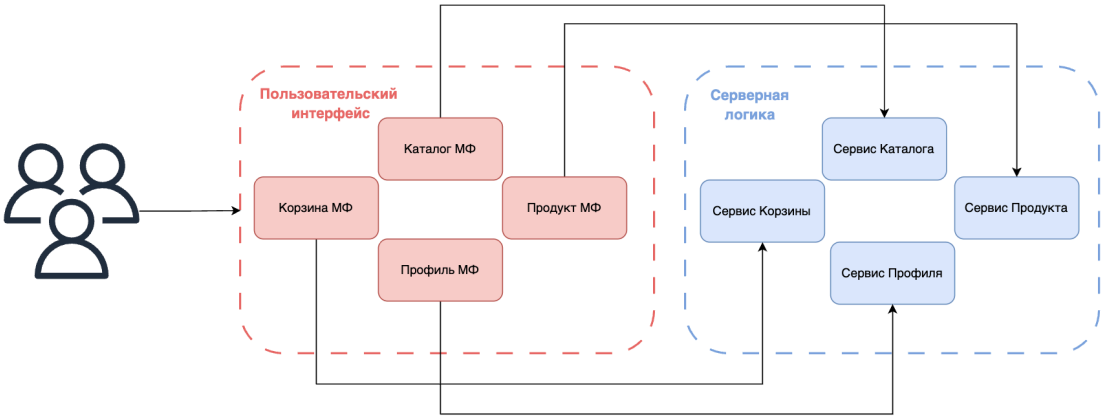

Рисунок 3 – Схема экспериментального стенда интернет-магазина.

5.2 Конфигурации архитектуры и сценарий отказа

В работе для удобства сравнительной оценки вводятся четыре конфигурации архитектуры, которые обозначаются как Baseline, Resilience-L1, Resilience-L2 и High-Isolation. Эти названия не являются стандартными терминами, а используются как условные обозначения уровней применения различных паттернов отказоустойчивости:

- Baseline – отсутствие явных client-side timeouts, минимальная обработка ошибок, отсутствие degraded-режимов, расширенный shared-слой.
- Resilience-L1 – добавлены client-side timeouts для запросов, реализована унифицированная обработка ошибок и fallback-UI.
- Resilience-L2 – поверх L1 реализованы degraded-режимы (отключение некритичных блоков при ошибках) и локальное кеширование ключевых данных.
- High-Isolation – дополнительно к паттернам L1/L2 минимизирован shared-слой и исключено общее глобальное состояние, что усиливает архитектурную изоляцию микрофронтендов.

В качестве типового сценария отказа рассматривалась частичная недоступность сервиса корзины (Partial Outage): для части запросов к Cart Service с вероятностью порядка 30 % возвращались ответы с кодом 5xx при сохранении нормальной

работы остальных сервисов. Для каждой конфигурации выполнялись имитационные прогоны пользовательских сценариев (просмотр каталога, переход к карточке товара, добавление в корзину, попытка оформить заказ), на основе которых оценивались значения метрик.

Для иллюстрации была реализована имитационная модель стенда (рисунок 4), позволяющая генерировать события успешных и неуспешных сценариев оформления заказа при различных конфигурациях архитектуры (рисунок 5)

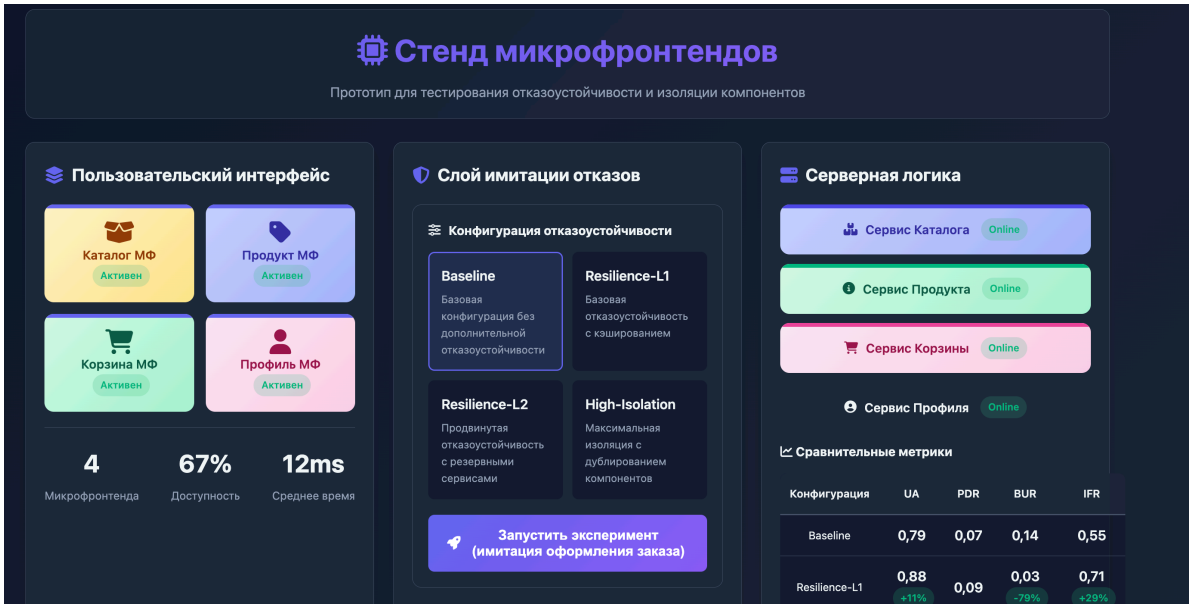

Рисунок 4 – Имитационная модель стенда.

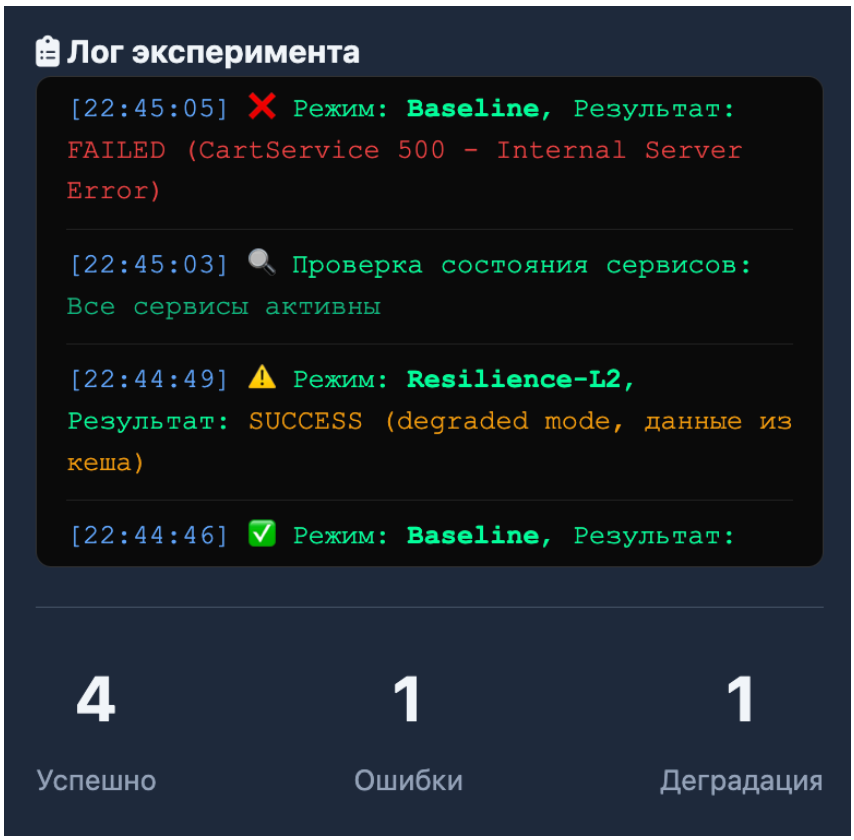

Рисунок 5 – Логи успешных и неуспешных сценариев.

5.3 Примерные значения метрик

Таблица 1 содержит примерные усреднённые значения метрик отказоустойчивости интерфейса для описанного сценария Partial Outage.

Таблица 1 – Примерные значения метрик отказоустойчивости интерфейса для различных конфигураций архитектуры

Конфигурация	**UI availability (UA)**	**Partial degradation rate (PDR)**	**Broken UI rate (BUR)**	**Isolated failure ratio (IFR)**	**UI recovery time (URT), c**
Baseline	0,79	0,07	0,14	0,55	6,1
Resilience-L1	0,88	0,09	0,03	0,71	4,0
Resilience-L2	0,93	0,11	0,02	0,79	2,8
High-Isolation	0,91	0,10	0,03	0,86	3,1

Из таблицы видно, что переход от базовой конфигурации к Resilience-L1 (timeouts и fallback-UI) приводит к заметному росту доступности интерфейса (UA) и более чем четырёхкратному снижению доли полностью сломанных сессий (BUR). Среднее время восстановления интерфейса сокращается примерно с 6,1 до 4,0 секунд за счёт отказа от «бесконционных» ожиданий и явной обработки ошибок.

Конфигурация Resilience-L2 демонстрирует наилучшие значения по метрикам UA и BUR: большинство сессий завершаются успешно, а часть потенциальных отказов переводится в режим контролируемой частичной деградации (рост PDR). Время восстановления интерфейса сокращается до 2,8 секунд благодаря использованию degraded-режимов и локального кеша.

Конфигурация High-Isolation по доступности близка к Resilience-L2, но обеспечивает максимальный коэффициент изоляции отказов (IFR = 0,86): большинство инцидентов ограничиваются одним микрофронтендом и не затрагивают остальные области интерфейса. Это подтверждает эффективность архитектурной изоляции как механизма ограничения области воздействия отказов.

5.4 Влияние паттернов на доступность интерфейса

Сравнение конфигураций показывает, что:

– в базовой конфигурации Baseline интерфейс ведёт себя наименее предсказуемо: при деградации или частичных отказах сервиса корзины пользователи часто сталкиваются с длительной загрузкой, отсутствием реакции на действия и невозможностью завершить оформление заказа;

– в конфигурации Resilience-L1 введение client-side timeouts и fallback-UI приводит к сокращению числа сессий, в которых пользователь наблюдает «зависший» интерфейс. Ошибки становятся явными, появляются сообщения и кнопки для повторных попыток. Это повышает субъективное восприятие стабильности системы, даже если часть операций временно недоступна;

– в конфигурации Resilience-L2 за счёт degraded-режимов и кеширования достигается дальнейший рост доступности: во многих случаях, когда в базовой

конфигурации пользователь оказывался бы в ситуации полного отказа, он теперь может продолжить работу в ограниченном режиме (например, оформить заказ без рекомендаций).

Таким образом, гипотеза о положительном влиянии паттернов L1/L2 на доступность интерфейса подтверждается: система реже выглядит для пользователя полностью недоступной.

6. ЗАКЛЮЧЕНИЕ

В настоящей работе рассмотрен вопрос обеспечения отказоустойчивости микрофронтенд-архитектур в условиях высокой нагрузки и частичных отказов. На основе анализа предметной области и существующих подходов сформулирована модель отказов на уровне пользовательского интерфейса и введён набор метрик, ориентированных на оценку поведения UI: UI availability, Partial degradation rate, Broken UI rate, Isolated failure ratio и UI recovery time.

Систематизированы паттерны отказоустойчивости, применимые к микрофронтенд-архитектурам: client-side timeouts, fallback-UI, degraded mode, локальное кеширование, архитектурная изоляция и явная коммуникация между микрофронтендами. На примере упрощённого стенда интернет-магазина показано, что применение этих паттернов позволяет уменьшить долю полностью сломанных сессий, улучшить показатели доступности интерфейса и сократить время восстановления, а также повысить изоляцию отказов между микрофронтендами.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Клеппман М. Высоконагруженные приложения. Программирование, масштабирование, поддержка. — СПб.: Питер, 2022. — 640 с.
2. Ричардсон К. Микросервисы. Паттерны разработки и рефакторинга. — СПб.: Питер, 2025. — 544 с.

3. Нейгард М. Release it! Проектирование и дизайн ПО для тех, кому не всё равно. — СПб.: Питер, 2016. — 320 с.

4. Таненбаум Э., ван Стеен М. Распределенные системы. Принципы и парадигмы / пер. с англ. — СПб.: Питер, 2003. — 876 с.

5. Peltonen S., Mezzalira L., Taibi D. Motivations, benefits, and issues for adopting Micro-Frontends: A Multivocal Literature Review // Information and Software Technology. — 2021. — Vol. 136. — Art. 106571. — DOI: 10.1016/j.infsof.2021.106571.

6. Jackson C. Micro Frontends [Электронный ресурс]. — MartinFowler.com, 2019. (дата обращения: 17.12.2025).

7. Webpack. ModuleFederationPlugin [Электронный ресурс]. — Официальная документация webpack. (дата обращения: 17.12.2025).

8. Webpack. Module Federation (concepts) [Электронный ресурс]. — Официальная документация webpack. (дата обращения: 17.12.2025).

9. Chaos Engineering Upgraded [Электронный ресурс]. — Netflix TechBlog, 25.09.2015. (дата обращения: 17.12.2025).

10. Web Vitals [Электронный ресурс]. — web.dev (Google), 2020–2024. (дата обращения: 17.12.2025).

REFERENCES

1. Kleppmann, M. (2022). Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems. St. Petersburg, Russia: Piter. (Original work published in English).

2. Richardson, C. (2025). Microservices Patterns: With examples in Java. St. Petersburg, Russia: Piter. (Original work published in English).

3. Nygard, M. T. (2016). Release It!: Design and Deploy Production-Ready Software (2nd ed.). St. Petersburg, Russia: Piter. (Original work published in English).

4. Tanenbaum, A. S., & van Steen, M. (2003). Distributed Systems: Principles and Paradigms. (Translated from English). St. Petersburg, Russia: Piter.

5. Peltonen, S., Mezzalira, L., & Taibi, D. (2021). Motivations, benefits, and issues for adopting Micro-Frontends: A Multivocal Literature Review. Information and Software Technology, 136, 106571.

6. Jackson, C. (2019). Micro Frontends [Online]. MartinFowler.com. (Accessed: Dec. 17, 2025)

7. Webpack. (n.d.). ModuleFederationPlugin [Online]. Webpack Official Documentation. (Accessed: Dec. 17, 2025).

8. Webpack. (n.d.). Module Federation (concepts) [Online]. Webpack Official Documentation. (Accessed: Dec. 17, 2025).

9. Netflix Technology Blog. (2015, September 25). Chaos Engineering Upgraded [Online]. (Accessed: Dec. 17, 2025).

10. Google Developers. (2020-2024). Web Vitals [Online]. web.dev. (Accessed: Dec. 17, 2025).