МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

УТВЕРЖДАЮ
И.о. директора института
перспективной инженерии,
канд. техн. наук, доцент
А.А. Порохня

ФОНД ОЦЕНОЧНЫХ СРЕДСТВ

для проведения текущего контроля успеваемости и промежуточной аттестации по дисциплине

Высоконагруженные и распределенные системы

Направление подготовки	09.03.04 Программная инженерия
Направленность (профиль)	Разработка и сопровождение программного обеспечения
Форма обучения	очная
Год начала обучения	2026
Реализуется в	7 семестре

Введение

1. Назначение: Оценочные материалы (оценочные средства) предназначены для проведения текущего контроля успеваемости и промежуточной аттестации обучающихся, и представляют собой совокупность контрольно-измерительных материалов и методов их использования для измерения уровня достижения обучающимся установленных результатов обучения.

2. ФОС является приложением к программе дисциплины «Высоконагруженные и распределенные системы»

3. Разработчик Потапов И.Р., ассистент межинститутской базовой кафедры департамента цифровых, робототехнических систем и электроники

4. Проведена экспертиза ФОС.

Члены экспертной группы:

Председатель: Новикова Елена Николаевна, зав. межинститутской базовой кафедрой

(ФИО, должность)

Члены комиссии:

Николаев Евгений Иванович, доцент департаменты цифровых и робототехнических систем и электроники.

(ФИО, должность)

Винокурский Д. Л., доцент департамента цифровых, робототехнических систем и электроники

(ФИО, должность)

Представитель организации-работодателя: А. В. Новиков, начальник отдела АСУТП ООО «ЕНДС-Ставрополь»

(ФИО, должность)

Экспертное заключение: Оценочные материалы (оценочные средства) по дисциплине «Паттерны проектирования» составлены в соответствии с требованиями самостоятельно устанавливаемого Федерального государственного образовательного стандарта высшего образования по направлению 09.03.04 «Программная инженерия» и позволяют оценить уровень сформированности компетенции ПК-1, ПК-2, ПК-4.

5. Срок действия ФОС определяется сроком реализации образовательной программы.

1. Описание показателей и критериев оценивания на различных этапах их формирования, описание шкал оценивания

Уровни сформированности компетенци(ий), индикатора (ов)	Дескрипторы			
	Минимальный уровень не достигнут (Неудовлетворительно) 2 балла	Минимальный уровень (удовлетворительно) 3 балла	Средний уровень (хорошо) 4 балла	Высокий уровень (отлично) 5 баллов
Компетенция: **ПК-1** – Способен осуществлять управление программными проектами: планировать, контролировать сроки и ресурсы, применять системы контроля версий и инструменты управления задачами				
Результаты обучения по дисциплине (модулю): *Индикатор:* ПК-1$_{ИД-1}$ – Применяет методы декомпозиции работ (WBS), оценки трудоёмкости (PERT, Planning Poker) и календарного планирования (диаграмма Ганта, критический путь) для формирования плана проекта	Не владеет методами декомпозиции WBS. Не может назвать методы оценки трудоёмкости (PERT, Planning Poker). Не знает, что такое диаграмма Ганта и критический путь. Не умеет формулировать SMART-цели.	С помощью преподавателя/шаблонов строит простейшую WBS (до 10 задач). Использует только один метод оценки (например, экспертную оценку). Может построить диаграмму Ганта для небольшого проекта, но не выделяет критический путь. Формулирует цели, но не все критерии SMART соблюдены.	Самостоятельно выполняет декомпозицию проекта среднего размера (20–30 задач) с иерархией. Применяет PERT для оценки неопределённости и Planning Poker в команде. Строит диаграмму Ганта с учётом зависимостей и находит критический путь. Анализирует интересы ключевых стейкхолдеров. Формулирует измеримые критерии успеха.	Глубоко владеет всеми методами: WBS до 3–4 уровней, комбинирует PERT и COCOMO II для оценки трудоёмкости. Проводит фасилитацию Planning Poker с распределённой командой. Оптимизирует календарный план за счёт сжатия критического пути (crashing/fast-tracking). Выявляет скрытых стейкхолдеров и управляет их ожиданиями. Цели и KPI проекта полностью SMART, с учётом рисков.
Результаты обучения по дисциплине (модулю): *Индикатор* ПК-1$_{ИД-2}$ – Использует системы контроля версий (Git, Git Flow) и платформы совместной	Не умеет работать с Git (только clone/add/commit). Не знает Git Flow. Не имеет опыта с Pull Requests и CI/CD.	Выполняет базовые операции: создание feature-ветки, слияние в develop, разрешение простых конфликтов (с помощью IDE). Может создать Pull Request в GitHub/GitLa	Организует командную работу по Git Flow: управляет ветками release, hotfix, защищает основную ветку правилами. Настраивает простой CI/CD пайплайн (сборка, юнит-тесты) через GitLab CI или	Внедряет Git Flow в команде с автоматической проверкой согласованности (commitlint, семантические версии). Настраивает многостадийные CI/CD пайплайны (сборка → тесты → линтеры → статический анализ → деплой на стенд). Организует процесс code review

разработки (GitLab, GitHub) для организации командной работы		b, но не настраивает CI/CD. Участвует в code review как комментатор.	GitHub Actions. Проводит code review, даёт содержательные замечания. Разрешает сложные конфликты с перебазированием (rebase).	с обязательными проверками покрытия кода и прохождения всех чеков. Управляет конфликтами при параллельной разработке нескольких релизных веток.
Результаты обучения по дисциплине (модулю): *Индикатор:* ПК-1$_{ИД-3}$ – Применяет инструменты управления задачами (Jira, Trello, YouTrack) для отслеживания прогресса (burndown chart), ведения бэклога и контроля исполнения	Не работал с Jira/Trello. Не знает понятий «бэклог», «спринт», «story points». Не может прочитать burndown chart.	Создаёт задачи в Trello с базовыми статусами. Использует готовые шаблоны спринтов в Jira, но не оценивает задачи в story points. Понимает, как выглядит burndown chart, но не интерпретирует отклонения. Ведёт бэклог как простой список.	Самостоятельно ведёт бэклог: приоритизирует задачи по MoSCoW, оценивает их в story points с командой. Планирует спринты, отслеживает burndown chart, выявляет отклонения (например, сгорание медленнее идеального). Корректирует план спринта за счёт перемещения задач. Идентифицирует простые риски (зависимости от смежников).	Управляет проектом в Jira с расширенными настройками: кастомизированные доски, автоматизация переходов, создание отчётов по скорости команды (velocity). Анализирует burndown и burnup диаграммы для прогнозирования завершения релиза. Проводит ретроспективы, корректирует процесс оценки. Управляет рисками с использованием матрицы вероятности/воздействия, формирует план смягчения.
Компетенция: **ПК-2** – Способен выполнять проектирование и разработку архитектуры программных систем среднего и крупного масштаба с применением современных паттернов и методов проектирования				
Результаты обучения по дисциплине (модулю): *Индикатор:* ПК-2$_{ИД-1}$ – Выделяет и анализирует функциональные и нефункциональные требования (FURPS+), выбирает	Не знает FURPS+. Не может перечислить нефункциональные требования. Не различает архитектурные стили.	По списку перечисляет некоторые требования (производительность, безопасность). Может назвать монолит и микросервисы, но не обосновывает выбор. При оценке RPS и latency	Применяет FURPS+ для классификации: функциональность, удобство, надёжность, производительность, поддерживаемость. Собирает количественные требования: ожидаемый RPS (например, 1000), p99 latency (<100 мс), доступность 99.9%. Выбирает	Проводит полноценный сбор NFR с участием стейкхолдеров, документирует сценарии качества (по методологии ATAM). Использует метрики: RPS на каждый эндпоинт, SLA/SLO, бюджет отказов. Выбирает гибридные стили: модульный монолит с выделением serverless-функций для пиковых нагрузок, EDA с Kafka для

архитектурный стиль (монолит, микросервисы, EDA, Clean Architecture) на основе сценариев качества		ориентируется интуитивно.	архитектурный стиль: монолит для низкой нагрузки, микросервисы для масштабируемости, EDA для асинхронных сценариев. Обосновывает выбор сравнением.	событий. Аргументирует компромиссы между стоимостью, сложностью и производительностью.
Результаты обучения по дисциплине (модулю): *Индикатор:* ПК-2$_{ИД-2}$ – Документирует архитектуру с использованием нотаций C4 и UML, фиксирует архитектурные решения в формате ADR (Architecture Decision Record)	Не знаком с C4 и UML. Не знает, что такое ADR. Не создавал архитектурные диаграммы.	Может нарисовать диаграмму контекста (C4 level 1) с одним внешним актёром и системой. Использует отдельные элементы UML (класс, стрелка). Знает структуру ADR (заголовки: контекст, решение, последствия), но не заполняет содержательно.	Создаёт диаграммы C4 levels 1–3 (контекст, контейнеры, компоненты). Использует UML-диаграммы последовательности для ключевых сценариев (например, оформление заказа). Пишет ADR с фиксацией выбора (например, «использовать PostgreSQL вместо MongoDB»), перечисляет плюсы/минусы.	Полное документирование: C4 level 1–4 (включая код – ключевые классы и интерфейсы) с использованием Structurizr или PlantUML. В ADR фиксирует не только решение, но и отвергнутые альтернативы, компромиссы, дату, статус, ссылки на связанные решения. Визуализирует развёртывание в Kubernetes или облаке с помощью UML-диаграмм развёртывания.
Результаты обучения по дисциплине (модулю): *Индикатор:* ПК-2$_{ИД-3}$ – Применяет порождающие, структурные и поведенческие паттерны GoF, а также архитектурные паттерны (CQRS, Event Sourcing,	Не знает паттерны CQRS, Event Sourcing, Saga, Circuit Breaker. Не может объяснить компенсирующие транзакции.	Теоретически знаком с определениями, но не реализовывал. Может написать простой Circuit Breaker с ручным переключением состояний. Понимает, что Saga управляет распределёнными транзакциями, но путает	Реализует CQRS в рамках монолита: разделяет Command (изменение) и Query (чтение) с разными моделями. Использует Event Sourcing с сохранением событий в событийной базе (например, EventStoreDB) и восстановлением состояния через replay. Применяет Saga оркестрацию с	Проектирует CQRS на уровне распределённой системы с раздельными хранилищами (command DB и read DB), синхронизацией через событийный брокер. Внедряет Event Sourcing с оптимизациями (снапшоты, индексация событий, параллельный replay). Реализует оба типа Saga (хореография через события, оркестрация с восстановлением после сбоев).

Saga, Circuit Breaker)		хореографию и оркестрацию.	явным координатором и компенсирующими шагами. Настраивает Circuit Breaker (библиотеки Resilience4j, Hystrix) с корректной логикой переходов.	Применяет Circuit Breaker с мониторингом, fallback-стратегиями и интеграцией с сервис-мешом (например, Istio). Тестирует поведение при эмуляции сбоев.
Результаты обучения по дисциплине (модулю): *Индикатор:* ПК-2$_{ИД-6}$ – Проектирует распределённые системы с учётом CAP-теоремы и PACELC, выбирая компромиссы между согласованностью, доступностью и устойчивостью к разделению	Не знает CAP и PACELC. Не может объяснить разницу между согласованностью и доступностью. Не работает с распределённым кэшем.	Формулирует CAP: из трёх свойств можно выбрать только два. Знает примеры CP (ZooKeeper) и AP (Cassandra). Понимает, что синхронная репликация даёт согласованность, но увеличивает задержку. Может настроить простой Redis как одноузловой кэш.	Анализирует систему по CAP и PACELC: например, при отсутствии разрыва выбирает между задержкой и консистентностью. Выбирает стратегию репликации: синхронная для критичных данных (банки), асинхронная для лайков. Применяет consistent hashing для шардирования (Redis Cluster). Проектирует распределённое кэширование с инвалидацией по паттерну «write-through/cache-aside».	Глубоко обосновывает выбор компромиссов для конкретной предметной области: использует настраиваемые уровни консистенции (Cassandra: ONE/QUORUM/ALL), сочетает синхронную и асинхронную репликацию в разных зонах. Реализует шардирование с виртуальными узлами и динамическим масштабированием (consistent hashing). Проектирует многоуровневое кэширование (L1 – локальный Caffeine, L2 – Redis Cluster) с опорой на Pub/Sub для инвалидации. Оценивает влияние PACELC на дизайн API (например, eventual consistency для чтения, strong для записи).
Компетенция: **ПК-4** – Способен выполнять проектирование и разработку архитектуры программных систем среднего и крупного масштаба с применением современных паттернов и методов проектирования				
Результаты обучения по дисциплине (модулю): *Индикатор:*	Не знает методов тест-дизайна. Не писал unit-тесты. Не работал с TestContainers.	Может написать несколько unit-тестов для простой функции	Системно применяет классы эквивалентности и граничные значения для строк, коллекций.	Владеет всеми методами тест-дизайна, включая таблицы решений, попарное тестирование, причинно-следственные

ПК-4$_{ИД-1}$ – Применяет методы тест-дизайна (классы эквивалентности, граничные значения, pairwise) и пишет unit-, интеграционные и E2E-тесты с использованием JUnit, pytest, Jest, Cypress, Selenium		(например, сложение). Применяет классы эквивалентности и граничные значения только на простых числовых диапазонах. Знает о pairwise, но не использует. Интеграционные тесты выполняет вручную или через заглушки.	Использует pairwise для сокращения числа тестов (например, 5 параметров → 6 комбинаций). Пишет unit-тесты с TDD (красный-зелёный-рефакторинг) на JUnit/pytest. Реализует интеграционные тесты с TestContainers для PostgreSQL/Kafka, проверяет реальные запросы.	диаграммы. Применяет BDD с Cucumber/JBehave. Пишет сложные интеграционные тесты с TestContainers, включая инициализацию схемы, миграции, тестирование компенсирующих транзакций. Организует E2E-тесты с Cypress/Selenium в CI/CD, работает с фикстурами и мокингом внешних API (WireMock). Достигает покрытия кода >80% с критическими сценариями.
Результаты обучения по дисциплине (модулю): *Индикатор:* ПК-4$_{ИД-2}$ – Использует инструменты отладки (GDB, IDE debugger), профилирования (perf, JProfiler) и нагрузочного тестирования (JMeter, k6) для выявления узких мест	Не использовал профайлеры и нагрузочные инструменты. Не знает метрик p95/p99.	Может запустить готовый сценарий JMeter с фиксированным числом потоков. Читает в отчёте среднее время отклика, но не различает процентили. Использует базовый дебаггер (точки остановки).	Создаёт сценарий нагрузочного тестирования в k6: задаёт RPS, длительность, проверки (checks). Анализирует результаты: p95, p99, throughput, проценты ошибок. Применяет профилировщик (JProfiler) для поиска медленных методов и блокировок. Проводит SAST с помощью SonarQube или Checkmarx (базовый уровень).	Пишет сложные нагрузочные сценарии: постоянный RPS, ступенчатый рост, тестирование стабильности (soak test). Интерпретирует взаимосвязь p99 latency и GC, утечек памяти. Использует async profiler для анализа сэмплов в продакшене (без остановки). Комбинирует SAST и DAST (OWASP ZAP) в CI, обрабатывает ложные срабатывания. Находит узкие места на уровне сети/базы данных через трассировку (Jaeger).
Результаты обучения по дисциплине (модулю): *Индикатор:* ПК-4$_{ИД-3}$ – Проводит нагрузочное, стресс-	Не знает chaos-инжиниринг, FMEA. Не рассчитывал MTBF/MTTR.	Понимает цель chaos-тестирования, но не проводил. Может назвать MTBF как	Проводит FMEA для основных компонентов системы (сервис, БД, кэш) с оценкой критичности. Запускает простые chaos-	Внедряет chaos-инжиниринг как практику: эксперименты в production с взрыв-радиусом (blast radius) и автоматической остановкой. Использует Chaos

тестирование и оценку надёжности (MTBF, MTTR) с применением chaos-инжиниринга, статического (SAST) и динамического (DAST) анализа безопасности		среднее время между отказами. FMEA применяет только на уровне «что будет, если упадет база данных».	эксперименты в среде staging (например, kill pod в Kubernetes, сетевые задержки 100 мс). Проверяет, что система восстанавливается (auto-healing). Рассчитывает MTBF по логам отказов и MTTR по времени восстановления.	Mesh/Gremlin для сложных сценариев: перегрузка CPU, потеря пакетов 30%, истечение DNS, сбой часов. Оценивает устойчивость через метрики: время обнаружения отказа, время частичного/полного восстановления. Комбинирует FMEA с анализом видов последствий отказов и приоритизирует эксперименты. Рассчитывает качество ПО по совокупности: покрытие кода (>85%), плотность дефектов (<0.5 на KLOC), MTBF (например, >100 часов), MTTR (<10 мин). Демонстрирует улучшение этих метрик после внедрения тестирования и chaos-экспериментов.

ОЦЕНОЧНЫЕ СРЕДСТВА ДЛЯ ПРОВЕРКИ УРОВНЯ СФОРМИРОВАННОСТИ КОМПЕТЕНЦИЙ

Номер задания	Правильный ответ	Содержание вопроса	Компетенция
1.	2	Какое из утверждений о блокирующем (thread-per-request) и неблокирующем (event loop) серверах верно? 1. При увеличении числа соединений неблокирующий сервер всегда потребляет больше памяти, чем блокирующий. 2. Неблокирующий сервер более эффективен для I/O-связанных задач при большом количестве соединений. 3. Блокирующий сервер не может использовать несколько ядер процессора. 4. Неблокирующий сервер всегда показывает более высокий RPS, даже при одном соединении.	ПК-2
2.	3	Для чего в алгоритме consistent hashing используются виртуальные узлы? 1. Для увеличения числа ключей, хранимых в кэше. 2. Для обеспечения синхронной репликации данных. 3. Для более равномерного распределения ключей между физическими узлами. 4. Для уменьшения времени поиска узла по ключу.	ПК-2
3.	1	Какая основная проблема возникает при использовании асинхронной репликации master-slave? 1. Возможность потери ещё не среплицированных данных при падении мастера. 2. Невозможность масштабирования чтения. 3. Снижение пропускной способности записи. 4. Отсутствие поддержки репликации на уровне БД.	ПК-2
4.	4	Какое утверждение о hash-шардировании и range-шардировании верно? 1. Hash-шардирование лучше подходит для диапазонных запросов. 2. Range-шардирование всегда даёт равномерное распределение ключей. 3. При добавлении нового шарда в hash-шардировании перемещается минимальное количество ключей. 4. Hash-шардирование обеспечивает равномерное распределение, но неэффективно для диапазонных запросов.	ПК-2

5.	2	В паттерне Saga (хореография) компенсирующая транзакция – это: 1. Операция, которая повторяет основную транзакцию с другими параметрами. 2. Операция, логически отменяющая эффект предыдущей локальной транзакции. 3. Механизм автоматического перезапуска всех сервисов. 4. Синхронный вызов для отката БД.	ПК-2
6.	1	Что из перечисленного НЕ является состоянием узла в алгоритме Raft? 1. Coordinator. 2. Follower. 3. Candidate. 4. Leader.	ПК-2
7.	3	Как в алгоритме Token Bucket, реализованном через Lua в Redis, атомарно проверяется и уменьшается количество токенов? 1. Через последовательность команд GET и SET с оптимистичной блокировкой. 2. Через команду INCR с проверкой в клиенте. 3. Весь процесс вычислений и обновления выполняется в одном Lua-скрипте. 4. Через транзакцию MULTI/EXEC с условными командами.	ПК-2
8.	3	Какое состояние Circuit Breaker позволяет отправить ограниченное количество проверочных запросов для восстановления сервиса? 1. Closed. 2. Open. 3. Half-Open. 4. Broken.	ПК-2
9.	2	Какая основная цель distributed tracing (например, OpenTelemetry + Jaeger)? 1. Сбор метрик использования процессора. 2. Выявление узких мест и задержек в распределённых вызовах. 3. Шифрование трафика между сервисами. 4. Автоматическое масштабирование контейнеров.	ПК-4
10.	4	Что такое backpressure в реактивных потоках (Reactive Streams)? 1. Механизм ускорения потребителя за счёт буферизации. 2. Способ блокировки производителя при переполнении очереди. 3. Автоматическое увеличение числа потребителей. 4. Механизм, позволяющий потребителю сигнализировать производителю о своей готовности к приёму данных.	ПК-2

11.	2	Для обеспечения семантики at-least-once в Kafka consumer необходимо: 1. Установить enable.auto.commit=true и максимальный интервал коммита. 2. Отключить авто-коммит и фиксировать offset только после успешной обработки сообщения. 3. Использовать транзакционный продюсер. 4. Установить replication.factor=1.	ПК-2
12.	1	Как работает алгоритм Smooth Weighted Round Robin (SWRR) в балансировщике? 1. Выбирается узел с максимальным current_weight, который затем уменьшается на сумму весов всех узлов. 2. Запросы распределяются строго по кругу в порядке возрастания весов. 3. Каждый узел обслуживает фиксированное количество запросов подряд, равное его весу. 4. Запросы направляются только на узлы с наибольшим весом.	ПК-2
13.	3	Зачем при установке TTL в кэше добавляется случайный разброс (jitter)? 1. Для увеличения срока жизни ключей. 2. Для уменьшения нагрузки на Redis. 3. Для предотвращения одновременного истечения большого количества ключей и лавинной нагрузки на БД. 4. Для гарантии синхронного обновления кэша.	ПК-2
14.	2	Что такое «stale read» при использовании read replicas? 1. Чтение данных, которые ещё не были записаны в master. 2. Чтение устаревших данных из-за задержки асинхронной репликации. 3. Чтение данных из реплики с последующей записью в master. 4. Ошибка чтения при отказе реплики.	ПК-2
15.	4	В архитектуре CQRS с асинхронными проекциями обновление read model происходит: 1. Синхронно в той же транзакции, что и команда. 2. Немедленно через прямой вызов из command side. 3. Только при ручном запуске администратором. 4. Асинхронно, через обработку событий из очереди, что приводит к eventual consistency.	ПК-2
16.	2	Какая основная цель chaos engineering?	ПК-4

		1. Увеличение нагрузки на систему для проверки максимальной производительности. 2. Преднамеренное внесение отказов для проверки устойчивости и выявления слабых мест. 3. Автоматическое восстановление после сбоев с помощью ИИ. 4. Создание полной изоляции между сервисами.	
17.	1	Что из перечисленного НЕ является обязательным компонентом типичного highload-стека (Nginx + app + Redis + PostgreSQL + мониторинг) при горизонтальном масштабировании? 1. FTP-сервер для загрузки файлов. 2. Балансировщик нагрузки. 3. Система сбора метрик (Prometheus). 4. Кэш (Redis).	ПК-1 / ПК-2
18.	3	Согласно CAP-теореме, распределённая система может одновременно обеспечивать только два из трёх свойств. Какие? 1. Consistency, Availability, Partition tolerance. 2. Consistency, Atomicity, Durability. 3. Consistency, Availability, Partition tolerance. 4. Latency, Throughput, Consistency.	ПК-2
19.	1	Модель PACELC расширяет CAP, добавляя компромисс в отсутствие сетевого разделения (Else): 1. Latency vs Consistency. 2. Availability vs Partition tolerance. 3. Throughput vs Latency. 4. Security vs Availability.	ПК-2
20.	4	Какое из утверждений о CI/CD и контейнеризации для высоконагруженных систем верно? 1. Контейнеризация увеличивает время запуска приложений, поэтому не используется. 2. CI/CD автоматизирует только сборку кода, но не развёртывание. 3. Docker Compose не подходит для локальной разработки многокомпонентных систем.	ПК-1

		4. Контейнеризация упрощает воспроизводимость окружения и масштабирование через оркестрацию.	
21.	2	Какая стратегия повторных попыток (retry) наиболее эффективна при временных сетевых сбоях и не перегружает систему? 1. Фиксированное количество мгновенных повторов (3 раза подряд). 2. Экспоненциальная задержка (exponential backoff) с ограничением числа попыток. 3. Бесконечные повторения с интервалом 1 мс. 4. Однократный повтор через случайное время.	ПК-2
22.	3	Какой инструмент чаще всего используется в связке с Prometheus для визуализации метрик и построения дашбордов? 1. Jaeger. 2. Elasticsearch. 3. Grafana. 4. Kibana.	ПК-4
23.	4	В Raft для того чтобы кандидат стал лидером, ему необходимо: 1. Получить одобрение всех узлов кластера. 2. Иметь самый длинный лог среди всех узлов. 3. Ожидать истечения таймаута у других кандидатов. 4. Получить голоса от большинства узлов (quorum).	ПК-2
24.	1	Что такое RPO (Recovery Point Objective) в контексте отказоустойчивости? 1. Максимально допустимый объём потерянных данных при сбое. 2. Время восстановления системы после отказа. 3. Вероятность успешного переключения на резервный узел. 4. Среднее время между отказами.	ПК-4
25.	2	При проектировании высоконагруженной системы выбор между синхронной и асинхронной репликацией данных – это компромисс между: 1. Производительностью чтения и производительностью записи. 2. Согласованностью (консистентностью) и задержкой (латентностью) / доступностью. 3. Сложностью и стоимостью. 4. Безопасностью и скоростью.	ПК-2

26.	3	В событийно-ориентированной архитектуре (event-driven) при использовании Kafka для доставки сообщений между микросервисами, какое свойство обеспечивается партиционированием топика? 1. Гарантированная доставка exactly-once. 2. Синхронное выполнение всех подписчиков. 3. Возможность масштабировать потребление параллельно на несколько экземпляров. 4. Автоматическое шифрование сообщений.	ПК-2
27.	1	Что из перечисленного является недостатком сквозной записи (write-through) в кэше? 1. Увеличение задержки записи из-за ожидания записи в кэш и основное хранилище (или очередь). 2. Невозможность кэшировать данные, которые часто обновляются. 3. Обязательная потеря данных при перезапуске кэша. 4. Отсутствие поддержки TTL.	ПК-2
28.	4	Какой паттерн используется для предотвращения каскадных отказов при вызове внешнего нестабильного сервиса? 1. Bulkhead. 2. Retry. 3. Saga. 4. Circuit Breaker.	ПК-2
29.	2	Какая стратегия кэширования записей лучше всего подходит для систем, где важна максимальная производительность записи и допустима редкая потеря данных (например, логгирование)? 1. Write-through. 2. Write-behind (write-back). 3. Write-around. 4. Read-through.	ПК-2
30.	3	В контексте CAP-теоремы, если в распределённой системе происходит сетевое разделение (partition), то необходимо выбирать между: 1. Производительностью и согласованностью. 2. Доступностью и производительностью.	ПК-2

		3. Доступностью и согласованностью. 4. Безопасностью и доступностью.	
31.	1	Какая метрика является наиболее информативной для оценки эффективности кэширования? 1. Hit ratio (отношение количества обращений к кэшу, завершившихся попаданием, к общему числу обращений). 2. Абсолютное количество ключей в кэше. 3. Время жизни TTL в секундах. 4. Частота сброса кэша на диск.	ПК-4
32.	4	Что из перечисленного НЕ относится к методам обеспечения отказоустойчивости в высоконагруженных системах? 1. Репликация данных. 2. Автоматический перезапуск упавших процессов. 3. Health checks и таймауты. 4. Увеличение размера оперативной памяти на каждом узле.	ПК-2
33.	2	В системах контроля версий (Git) для организации командной работы над высоконагруженным проектом наиболее предпочтительной моделью ветвления является: 1. Одна долгоживущая ветка main, в которую все коммитят напрямую. 2. Git Flow или GitHub Flow с короткоживущими фича-ветками, код-ревью и CI-проверками. 3. Хранение всех изменений только в локальных репозиториях. 4. Создание отдельной ветки для каждого разработчика без слияний.	ПК-1
34.	3	Какое утверждение об инструменте автоматизации CI/CD (например, GitLab CI, Jenkins, GitHub Actions) верно? 1. CI/CD может только запускать юнит-тесты, но не выполнять развёртывание. 2. CI/CD не требует наличия Docker-контейнеров. 3. CI/CD позволяет автоматически собирать, тестировать и развёртывать приложение при каждом изменении кода. 4. CI/CD используется только для мониторинга производительности.	ПК-1
35.	1	Что такое «replication lag» в контексте master-slave репликации? 1. Задержка между моментом применения записи на мастере и её появлением на реплике.	ПК-2

		2. Время ответа slave на запрос чтения. 3. Количество потерянных транзакций при падении master. 4. Максимальный размер очереди репликации.	
36.	4	Какая из перечисленных схем распределённой трассировки (distributed tracing) поддерживается OpenTelemetry и позволяет передавать контекст между сервисами? 1. W3C TraceContext. 2. B3 (Zipkin). 3. Jaeger Propagator. 4. Все перечисленные.	ПК-4
37.	2	Для чего в системах реального времени (например, чатах) используются очереди сообщений (Kafka, RabbitMQ)? 1. Для хранения двоичных файлов. 2. Для асинхронной доставки сообщений между компонентами, повышения отказоустойчивости и развязывания сервисов. 3. Для балансировки нагрузки на уровне TCP. 4. Для шифрования трафика.	ПК-2
38.	3	При нагрузочном тестировании с помощью wrk параметр -c (connections) определяет: 1. Количество потоков нагрузки. 2. Длительность теста. 3. Количество одновременных соединений. 4. Таймаут запроса.	ПК-4
39.	2	Какую основную проблему решает паттерн Backpressure в реактивных системах? 1. Потерю данных при сетевых обрывах. 2. Защиту потребителя от переполнения буфера, когда производитель генерирует данные быстрее, чем потребитель успевает обрабатывать. 3. Увеличение пропускной способности за счёт сжатия данных. 4. Синхронизацию часов между узлами.	ПК-2
40.	1	В архитектуре на основе микросервисов для обеспечения согласованности данных в распределённой транзакции (например, создание заказа и резервирование товара) чаще всего используется паттерн:	ПК-2

		1. Saga. 2. Two-Phase Commit (2PC). 3. Consensus (Raft/Paxos). 4. Eventual consistency без транзакций.	
41.	4	Какое из утверждений о мониторинге highload-систем верно? 1. Достаточно собирать только средние значения метрик (средняя задержка, средний RPS). 2. Метрики не нужно хранить, достаточно смотреть в реальном времени. 3. Логи и трейсы не коррелируют с метриками. 4. Необходимо собирать метрики по перцентилям (p50, p99), ошибкам и отслеживать тренды, а также использовать трейсинг для распределённых запросов.	ПК-4
42.	2	Что такое «graceful degradation» в контексте отказоустойчивости? 1. Немедленное завершение всех запросов при сбое. 2. Способность системы продолжать работу в урезанном режиме, отключая второстепенные функции, при отказе части компонентов. 3. Автоматическое масштабирование числа экземпляров. 4. Перезагрузка всех сервисов по расписанию.	ПК-2
43.	3	Какой подход позволяет снизить количество конфликтов при записи в распределённой системе с шардированием? 1. Использование синхронной репликации всех шардов. 2. Применение optimistic locking без шардирования. 3. Выбор ключа шардирования, обеспечивающего равномерное распределение нагрузки, и использование идемпотентных операций. 4. Увеличение числа копий данных в каждой партиции.	ПК-2
44.	1	Для чего в конвейере CI/CD используется стадия статического анализа кода (линтинг, анализ типов)? 1. Для выявления потенциальных ошибок и нарушения стандартов кодирования до выполнения тестов. 2. Для замены юнит-тестов. 3. Для автоматического исправления всех ошибок. 4. Для измерения производительности приложения.	ПК-1 / ПК-4
45.	2	Какая команда Docker используется для масштабирования сервиса в режиме Docker Swarm или Compose?	ПК-1

		1. docker run --scale 2. docker compose up --scale <service>=<count> (или docker service scale в Swarm) 3. docker container scale 4. docker deploy --replicas	
46.	4	Какая из перечисленных баз данных наиболее подходит для хранения временных рядов метрик (например, для Prometheus) в высоконагруженной системе? 1. PostgreSQL. 2. MongoDB. 3. Redis. 4. VictoriaMetrics / Thanos (совместимые с Prometheus).	ПК-2
47.	2	Что такое «split-brain» в распределённых системах и как его предотвращают? 1. Разделение базы данных на шарды – предотвращается выбором хорошего ключа шардирования. 2. Ситуация, когда два узла одновременно считают себя лидерами – предотвращается с помощью алгоритмов консенсуса (Raft, Paxos) и эпох (term). 3. Перегрузка сети между дата-центрами – предотвращается кэшированием. 4. Потеря кворума – предотвращается увеличением числа реплик.	ПК-2
48.	4	Как правильно интерпретировать p99 latency, равный 250 мс? 1. 99% запросов выполняются медленнее 250 мс. 2. 1% запросов выполняются быстрее 250 мс. 3. Средняя задержка составляет 250 мс. 4. 99% запросов выполняются быстрее 250 мс, а 1% – медленнее.	ПК-4
49.	3	Какой из паттернов проектирования распределённых систем описывает разделение ответственности между стороной команды (изменение данных) и стороной запроса (чтение)? 1. Saga. 2. Event Sourcing. 3. CQRS (Command Query Responsibility Segregation). 4. Circuit Breaker.	ПК-2
50.	1	При развёртывании highload-приложения в production для управления контейнерами на нескольких серверах наиболее часто используется: 1. Kubernetes. 2. Docker Compose (в однопользовательском режиме).	ПК-1

		3. Docker Machine. 4. Podman в режиме rootless.	
51.	2	Какое утверждение о плановой (blue-green) стратегии развёртывания верно? 1. Приложение обновляется на работающем экземпляре без остановки. 2. Существуют два окружения (blue и green), трафик переключается на новую версию после её проверки. 3. Обновление выполняется последовательно на каждом узле кластера. 4. Новая версия развёртывается поверх старой с сохранением сетевых правил.	ПК-1
52.	3	Что такое «semaphore» в контексте ограничения параллелизма (concurrency limiting) в высоконагруженных системах? 1. Инструмент для измерения задержек. 2. Тип очереди сообщений. 3. Синхронизирующая конструкция, ограничивающая количество одновременных доступов к ресурсу. 4. Алгоритм балансировки нагрузки.	ПК-2
53.	1	Какая характеристика отличает распределённый rate limiter от локального? 1. Состояние счётчика или токенов хранится и синхронизируется между несколькими узлами (например, через Redis). 2. Локальный limiter не может ограничивать скорость. 3. Распределённый limiter всегда точнее локального. 4. Распределённый limiter не требует сети.	ПК-2
54.	4	Для чего используется инструмент wrk или bombardier? 1. Для развёртывания контейнеров. 2. Для статического анализа кода. 3. Для сбора метрик из production. 4. Для нагрузочного тестирования HTTP-серверов.	ПК-4
55.	2	Каким образом в системе с eventual consistency (например, на основе read replicas) можно гарантировать, что клиент прочитает свои собственные записи (read your writes) сразу после их выполнения? 1. Увеличить число реплик. 2. Направлять чтения клиента на master в течение некоторого времени после его записи. 3. Использовать только синхронную репликацию.	ПК-2

		4. Отключить кэширование на клиенте.	

2. Описание шкалы оценивания

Оценка успеваемости студентов по дисциплине осуществляется в ходе текущего контроля и промежуточной аттестации. Для проверки уровня сформированности компетенций используется совокупность оценочных средств, позволяющих оценить знания, умения и навыки, приобретённые студентами в процессе изучения дисциплины.

Основными формами текущего контроля являются:

– выполнение и защита лабораторных работ;

– устные опросы (собеседования) по темам занятий;

– проверка индивидуальных заданий;

– тестирование по отдельным разделам дисциплины.

Промежуточная аттестация проводится в форме зачета с оценкой и включает выполнение практического задания и/или ответы на теоретические вопросы.

3. Критерии оценивания компетенций заданий открытого типа

Оценка «отлично» выставляется студенту, если в ходе выполнения лабораторных работ и итогового проекта продемонстрировано умение самостоятельно планировать этапы разработки распределённой системы, декомпозировать задачи на подзадачи (например, реализация шардирования, настройка репликации, интеграция мониторинга) и оценивать трудозатраты. Студент использовал систему контроля версий (Git) с осмысленной историей коммитов, ветвлением (feature branches, pull requests) и код-ревью. Применял инструменты управления задачами (Jira, Trello или GitHub Projects) для отслеживания прогресса, назначения приоритетов и фиксации технического долга. В отчёте представлены временные диаграммы (Gantt) или планы спринтов, обоснование распределения ресурсов (память, CPU, сеть) между компонентами системы. Компетенция ПК-1 освоена на высоком уровне.

Оценка «хорошо» выставляется студенту, если студент демонстрирует умение планировать основные этапы работы, использовать систему контроля версий (регулярные коммиты, понятные сообщения), но история коммитов может быть не полностью структурированной (например, отсутствуют отдельные ветки под задачи). Инструменты управления задачами применялись, но не в полной мере (например, только список задач без приоритетов и сроков). В отчёте присутствует общий план работ, однако детальная декомпозиция и оценка ресурсов выполнены не для всех этапов. Компетенция ПК-1 освоена на среднем уровне.

Оценка «удовлетворительно» выставляется студенту, если студент демонстрирует базовое умение использовать систему контроля версий (коммиты есть, но сообщения неинформативны, ветвление не применялось), планирование работ ограничивается перечнем лабораторных заданий без разбивки на задачи. Инструменты управления задачами не использовались либо использовались формально. В отчёте отсутствуют планы сроков и ресурсов, не показано распределение нагрузки между участниками (при групповой работе). Допущены ошибки в оценке сложности задач (например, завышение или занижение времени на реализацию паттернов). Компетенция ПК-1 освоена на минимальном уровне.

Оценка «неудовлетворительно» выставляется студенту, если студент не использовал систему контроля версий либо использовал её нерегулярно (один коммит на всю работу), не умеет планировать задачи, не проводил декомпозицию и не оценивал ресурсы. Инструменты управления задачами не применялись. В отчёте отсутствует какая-либо информация об управлении проектом, сроках или распределении работ. Компетенция ПК-1 не сформирована.

Оценка «отлично» выставляется студенту, если имеются результаты полного демонстрирования умений применять архитектурные паттерны высоконагруженных распределённых систем (Saga, CQRS, Circuit Breaker, Backpressure, Consistent Hashing), методы согласованности (CAP/PACELC, Raft), стратегии масштабирования (шардирование, репликация, read replicas), механизмы отказоустойчивости (retry, chaos engineering) и наблюдаемости (tracing, метрики). Студент обосновывает выбор конкретных паттернов и технологий

(очереди сообщений, кэши, балансировщики) с детальным анализом компромиссов (trade-offs) и оценкой технического долга, а также проводит рефакторинг кода с учётом требований к производительности и масштабируемости. Представленный отчёт не содержит ошибок, все задания лабораторных работ выполнены верно, код соответствует принципам построения распределённых систем (чистая архитектура, изоляция компонентов, идемпотентность, обработка ошибок), покрыт тестами (модульными, интеграционными, нагрузочными). Компетенция ПК-2 освоена на высоком уровне.

Оценка «хорошо» выставляется студенту, если имеются результаты демонстрирования умений применять основные архитектурные паттерны highload, проводить рефакторинг с учётом масштабируемости и обосновывать выбор паттернов и технологий (например, почему выбрана асинхронная репликация, а не синхронная; почему Redis, а не Memcached). Представленный отчёт содержит незначительные ошибки (неточности в описании протоколов, отсутствие некоторых несущественных метрик), но это не влияет на общее понимание решения. Возможны небольшие недочёты в реализации (неполное покрытие тестами, не оптимальная конфигурация пулов соединений), однако ключевые требования к работоспособности и отказоустойчивости выполнены. Компетенция ПК-2 освоена на среднем уровне.

Оценка «удовлетворительно» выставляется студенту, если имеются результаты демонстрирования базовых умений применять паттерны распределённых систем, но допущены ошибки в выборе или реализации (например, выбор неподходящей стратегии шардирования, отсутствие механизма backpressure, неправильная настройка health checks). Анализ компромиссов (CAP/PACELC) представлен поверхностно, недостаточно проработаны вопросы согласованности и отказоустойчивости. Представленный отчёт содержит ошибки, не позволяющие в полной мере оценить качество разработки; ключевые определения (eventual consistency, replication lag, split brain) и выводы по нагрузочному тестированию представлены не в полном объёме. Компетенция ПК-2 освоена на минимальном уровне.

Оценка «неудовлетворительно» выставляется студенту, если отсутствуют результаты демонстрирования умений применять архитектурные паттерны высоконагруженных систем, рефакторинг с учётом производительности не проведён, выбор паттернов, стратегий шардирования, репликации и кэширования не обоснован. Представленный отчёт содержит много грубых ошибок (неверное понимание CAP, отсутствие схемы взаимодействия сервисов), отсутствуют ключевые определения, результаты нагрузочного тестирования, а также работающий код или он не соответствует заданию. Компетенция ПК-2 не сформирована.

Оценка «отлично» выставляется студенту, если для разработанных высоконагруженных компонентов выполнено полное покрытие тестами: юнит-тесты (изоляция модулей), интеграционные тесты (взаимодействие сервисов, очередей, БД) и нагрузочные тесты (wrk, bombardier, k6) с анализом полученных метрик (RPS, p99 latency, error rate, hit ratio). Реализованы автоматизированные сценарии chaos-тестирования (отказ узлов, задержки сети) с фиксацией поведения системы. Проведена отладка производительности с использованием профилировщиков (cProfile, py-spy) и tracing (Jaeger). Оценка качества включает измерение утечек памяти, проверку идемпотентности операций и безопасности (например, rate limiting для предотвращения атак). Представленный отчёт содержит подробные результаты тестирования, графики и обоснование выбора тестовых сценариев. Компетенция ПК-4 освоена на высоком уровне.

Оценка «хорошо» выставляется студенту, если выполнены базовые тесты (юнит-тесты на ключевые функции и интеграционные тесты для основных сценариев), проведено нагрузочное тестирование, но не для всех эндпоинтов или при неполном диапазоне нагрузок. Использованы инструменты автоматизации тестирования (pytest, JUnit, k6), но отсутствуют тесты на отказоустойчивость (chaos) или профилирование производительности. Оценка качества проведена, но без детального анализа выбросов латентности или утечек

памяти. В отчёте присутствуют результаты тестов, но возможны незначительные погрешности в интерпретации метрик. Компетенция ПК-4 освоена на среднем уровне.

Оценка «удовлетворительно» выставляется студенту, если тестирование ограничивается ручными проверками или минимальным набором юнит-тестов, не покрывающим все сценарии (например, отсутствуют тесты на конкурентность или на обработку ошибок). Нагрузочное тестирование проведено формально (один тип нагрузки без изменения параметров). Инструменты автоматизации использованы частично, отладка производительности не проводилась. Оценка качества ПО выполнена поверхностно, безопасность не проверялась. Отчёт содержит ошибки в тестовых данных или в выводах о производительности. Компетенция ПК-4 освоена на минимальном уровне.

Оценка «неудовлетворительно» выставляется студенту, если тестирование отсутствует или выполнено несистемно, не использованы инструменты автоматизации, нагрузочное тестирование не проводилось. Студент не умеет оценивать качество и надёжность распределённой системы, не может интерпретировать метрики производительности. В отчёте отсутствуют результаты тестирования, либо они содержат грубые ошибки. Компетенция ПК-4 не сформирована.

4. Критерии оценивания компетенций заданий закрытого типа

Оценка «отлично» выставляется студенту, если он правильно ответил на 88-100% от общего количества заданий. Компетенция ПК-1 освоена на высоком уровне.

Оценка «хорошо» выставляется студенту, если он правильно ответил на 72-87% от общего количества заданий. Компетенция ПК-1 освоена на среднем уровне.

Оценка «удовлетворительно» выставляется студенту, если он правильно ответил на 53-71% от общего количества заданий. Компетенция ПК-1 освоена на минимальном уровне.

Оценка «неудовлетворительно» выставляется студенту, выставляется студенту, если он правильно ответил на менее 53% от общего количества заданий. Компетенция ПК-1 не сформирована.

Оценка «отлично» выставляется студенту, если он правильно ответил на 88-100% от общего количества заданий. Компетенция ПК-2 освоена на высоком уровне.

Оценка «хорошо» выставляется студенту, если он правильно ответил на 72-87% от общего количества заданий. Компетенция ПК-2 освоена на среднем уровне.

Оценка «удовлетворительно» выставляется студенту, если он правильно ответил на 53-71% от общего количества заданий. Компетенция ПК-2 освоена на минимальном уровне.

Оценка «неудовлетворительно» выставляется студенту, выставляется студенту, если он правильно ответил на менее 53% от общего количества заданий. Компетенция ПК-2 не сформирована.

Оценка «отлично» выставляется студенту, если он правильно ответил на 88-100% от общего количества заданий. Компетенция ПК-4 освоена на высоком уровне.

Оценка «хорошо» выставляется студенту, если он правильно ответил на 72-87% от общего количества заданий. Компетенция ПК-4 освоена на среднем уровне.

Оценка «удовлетворительно» выставляется студенту, если он правильно ответил на 53-71% от общего количества заданий. Компетенция ПК-4 освоена на минимальном уровне.

Оценка «неудовлетворительно» выставляется студенту, выставляется студенту, если он правильно ответил на менее 53% от общего количества заданий. Компетенция ПК-4 не сформирована.