Edu Platform

Документ оценки архитектуры

Авторы документа:

Киселев Иван
Федоров Артём

Проверяющее лицо:

Пётр Андреевич Белкин

Содержание

1. Описание проекта в целом

1.1. Название проекта

Образовательная платформа.

1.2. Контекст задачи и среда функционирования системы

Проект "Образовательная платформа" представляет собой онлайн-платформу, предназначенную для изучения пользователями различных курсов. Основная задача проекта заключается в предоставлении пользователям доступа к курсам и учебным материалам, а учителям возможности выложить курсы, управлять финансовыми транзакциями и уведомлениями и следить за обратной связью.

Система функционирует в онлайн-среде, обеспечивая доступ пользователей через веб-браузеры. Пользователи могут регистрироваться, просматривать курсы, покупать курсы, оставлять отзывы и получать уведомления о новых материалах или изменениях в системе.

1.3. Рамки и цели проекта

Целью проекта является создание полнофункциональной образовательной платформы (автоматизированной системы, согласно требованиям и ТЗ) , обеспечивающей пользователям удобное изучение различных курсов.

Система должна предоставить клиентам возможность просматривать курс, приобретать доступ к курсам и их частям по отдельности, изучать купленные курсы, прослеживать свой прогресс, получать рассылки информации о курсе и оставлять комментарии.

Пользователь-учитель, сможет создавать и редактировать свои шаблоны курсов, публиковать их и получать часть прибыли с их продажи. Учитель также может получать информацию о продажах его курсов и делать рассылки участникам курсов.

Пользователи с ролями модератора смогут просматривать курсы (и их содержимое) и обрабатывать заявки на публикацию курсов. Модераторы при желании также смогут посылать рассылки на почты учителей и участникам курсов.

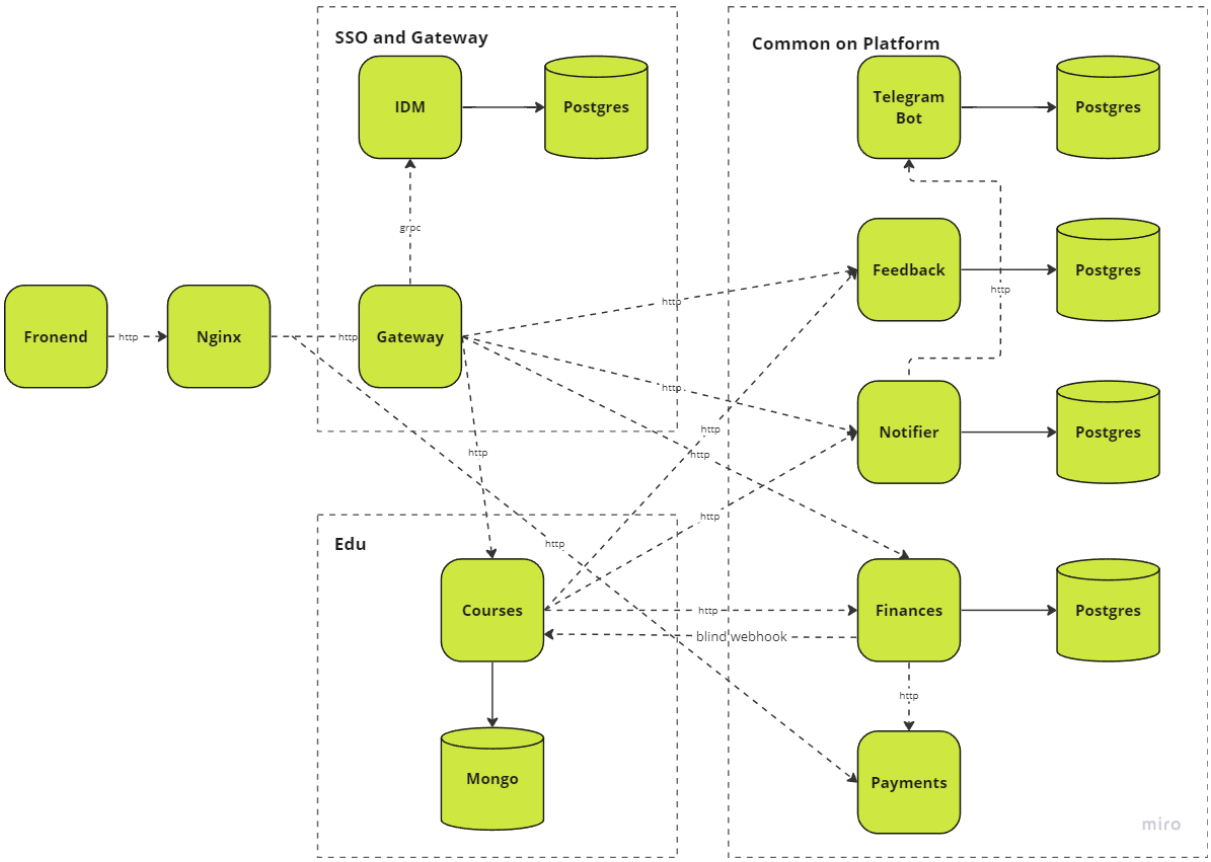
Пользователям являющимися админами будет предоставлена возможность управления аккаунтами и ролями пользователей.

Целью текущей разработки является создание системы, основные функции включают в себя регистрацию пользователей, управление и просмотр курсов, обработку платежей, отправку SMTP уведомлений. Также подключить для нее мониторинг и логирование. Система должна быть платформой, которую можно

расширишь в виде интеграции нового функционала и предметной области, а также корпоративных приложений.

В текущую разработку не входит подробный анализ и проектирование безопасности и работы с высокой нагрузкой (однако, конечно, проектирование будет учитывать и эти факторы, хоть и не слишком подробно со стороны требований и интересов), а также проектирование внедрения систем мониторинга, так как их реализация будет уникальной и уже сделана в шаблоне микросервиса на Go.

2. Описание выбранной архитектуры



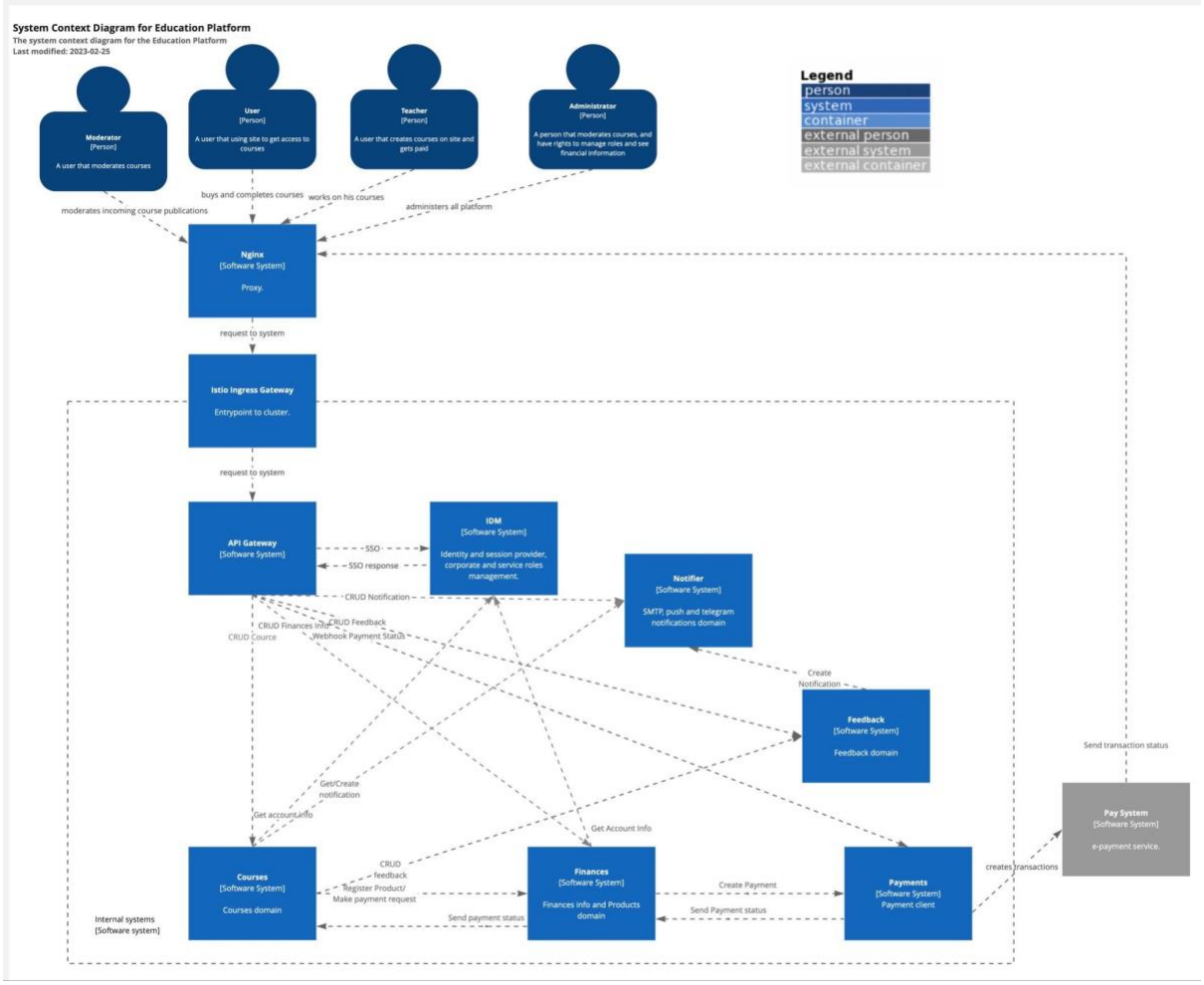
В ходе автоматизации бизнес процессов создания продажи доступов к учебным материалам создается данная информационная система состоящая из клиентской и серверной части, а также инфраструктуры мониторинга и внешних сервисов (платежный сервис Юмани telegram).

Информационная система разбита на следующие следующие системы:

1. Frontend - обеспечивает интерфейс взаимодействия с пользователями, разрабатывается на React.js + TypeScript.
2. Gateway - является промежуточным слоем между фронтендом и бэкендом, обеспечивает маршрутизацию запросов, разрабатывается на Go.
3. IDM (Identity Manager): Отвечает за управление и аутентификацию пользователей, разрабатывается на Java.
4. Courses: Бэкенд для управления курсами, разрабатывается на Go.
5. Finances: Управление финансовыми транзакциями и платежами, разрабатывается на Go.
6. Payments: Отвечает за взаимодействие с платежными системами, разрабатывается на Python.
7. Notifier: Обеспечивает отправку уведомлений пользователям, разрабатывается на Java.
8. Feedback: Модуль для сбора обратной связи от пользователей, разрабатывается на Java.
9. Telegram Bot (часть системы Feedback): Позволяет взаимодействовать с платформой через Telegram, разрабатывается на Python.
10. Системы для реализации мониторинга, трассировок (Prometheus, Grafana, Open Telemetry + Jaeger)
11. Базы данных (mongoDB, postgresql)

Взаимодействие между подсистемами будет вестись асинхронно по протоколу HTTP. Все запросы от конечного пользователя будут поступать от клиентской части Frontend через Gateway (API Gateway) к остальной серверной части. Системы серверной части организованы согласно принципам микросервисной архитектуры и разбиты по доменам (DDD) и независимым общим функциям, например, продажа продукта, отправления уведомлений авторизация (такие функции могут использоваться и новыми бизнес доменами при расширении функционалами автомаизированной системы).

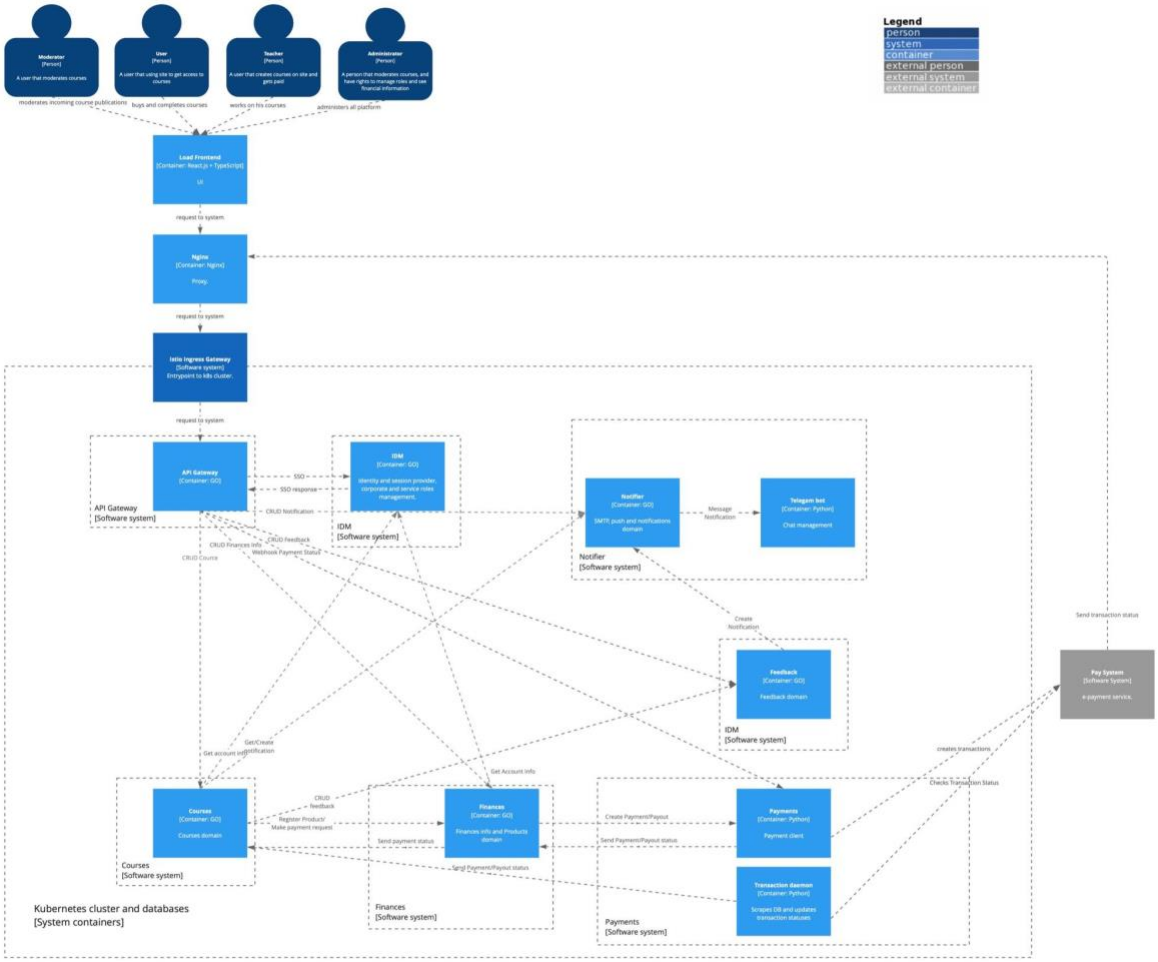
C4 model Context level:



Данная модель представляет разбиение системы на компоненты-подсистемы и показывает связь между ними

Представление развертывания можно отлично видеть на моделе Container уровня C4 model

System Container Diagram for Education Platform
The system container diagram for the Education Platform
Last modified: 2023-02-25



3. Описание бизнес-целей разработки системы

Система должна предоставить клиентам возможность просматривать курс, приобретать до- ступ к курсам и их частям по отдельности, изучать купленные курсы, прослеживать свой про- гресс, получать рассылки информации о курсе и оставлять комментарии.

Клиент, получив доступ, сможет размещать курсы и получит роль "учитель". Данный поль- зователь, учитель, сможет создавать и редактировать свои шаблоны курсов, публиковать их и получать часть прибыли с их продажи. Учитель также может получать информацию о продажах его курсов и делать рассылки участникам курсов.

Пользователи с ролями модератора смогут просматривать курсы (и их содержимое) и обрабатывать заявки на публикацию курсов. Модераторы при желании также смогут посылать рассылки на почты учителей и участникам курсов.

Пользователям являющимися админами будет предоставлена возможность управления ак- каунтами и ролями пользователей.

4. Приоритизированные требования по атрибутам качества в виде сценариев

Атрибуты качества и выражающие их сценарии (отображение 1 к n)

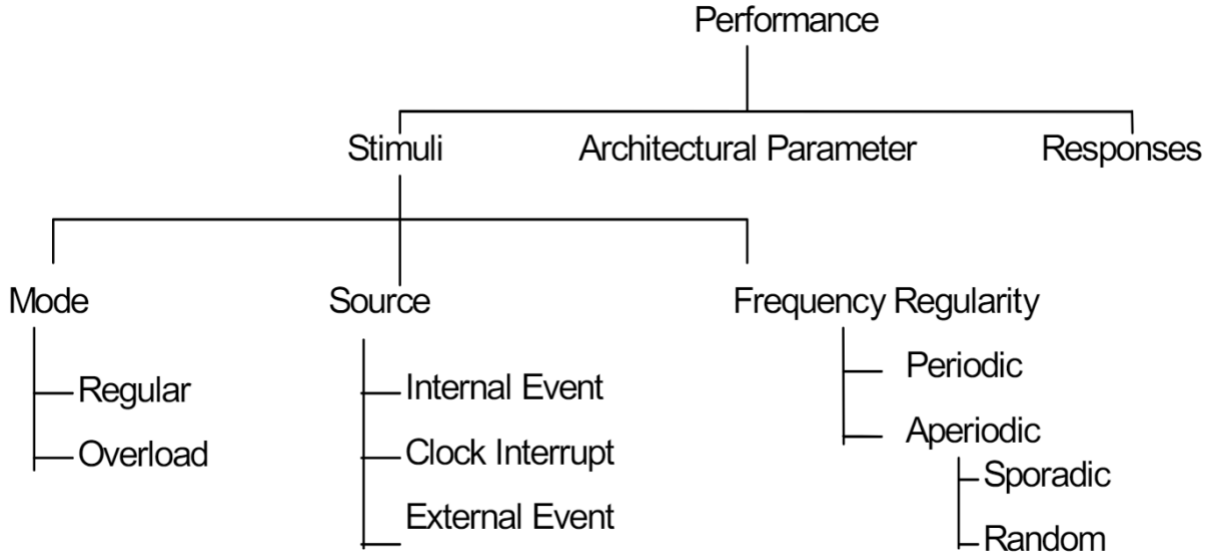
1) Функциональные требования к составу выполняемых функций.
    a) Система должна полностью реализовать функции в ее бэклоге (бэклоге продукта). См. User Story Mapping. TODO
    b) Каждая подсистема должна реализовать функции ее бэклога. См. User Story Mapping. TODO
2) Требования к безопасности, система не должна быть уязвима к
    a) XSS атакам
    b) CSRF атакам
    c) легкому подбору паролей
    d) RCE
    e) атакам на сети (DDOS)
    f) защита секретов
        i) Сценарий: злоумышленный разработчик, которому нельзя напрямую работать с БД, пытается достать пароль от нее из конфига.
3) Требования к удобству использования
4) Требования к надежности
    a) Отказоустойчивость обработки покупки продукта пользователем
        i) Сценарий: пользователь переводит деньги и в нашей системе моргает сеть
        ii) Сценарий: пользователь переводит деньги и в нашей системе падает питание
5) Требования к производительности
    a) время отклика
        i) Сценарий-требование: RPS 1к, SLO времени ответа — 60 мс, SLO успешности ответа — 98%
        ii) Сценарий-требование: RPS 100, SLO времени ответа — 25 мс, SLO успешности ответа — 99%
    b) использование ресурсов
        i) Сценарий: использование CPU в простое
        ii) Сценарий: использование CPU в при RPS 100
    c) эффективность
    d) масштабируемость
        i) Сценарий: увеличение более 1к PRS, что нужно выделить?
6) Требования к поддержке
    a) сопровождение
    b) модульность
    c) расширяемость

Приоритизированные атрибуты качества и выражающие их сценарии (отображение 1 к n)

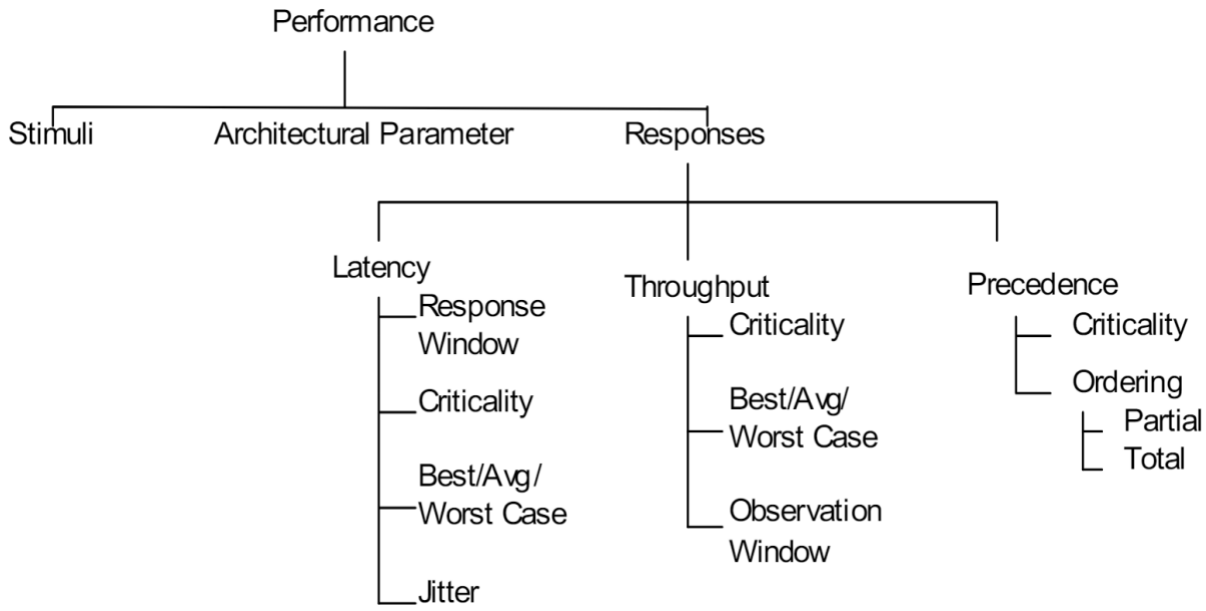
1) Функциональные требования к составу выполняемых функций.
    a) Система должна полностью реализовать функции в ее бэклоге (бэклоге продукта). См. User Story Mapping. TODO
    b) Каждая подсистема должна реализовать функции ее бэклога. См. User Story Mapping. TODO
2) Требования к безопасности, система не должна быть уязвима к
    a) XSS атакам
    b) CSRF атакам
    c) легкому подбору паролей
    d) RCE
    e) атакам на сети (DDOS)
    f) защита секретов
        i) Сценарий: злоумышленный разработчик, которому нельзя напрямую работать с БД, пытается достать пароль от нее из конфига.
3) Требования к удобству использования
4) Требования к надежности
    a) Отказоустойчивость обработки покупки продукта пользователем
        i) Сценарий: пользователь переводит деньги и в нашей системе моргает сеть
        ii) Сценарий: пользователь переводит деньги и в нашей системе падает питание
5) Требования к производительности
    a) время отклика
        i) Сценарий-требование: RPS 1к, SLO времени ответа — 60 мс, SLO успешности ответа — 98%
        ii) Сценарий-требование: RPS 100, SLO времени ответа — 25 мс, SLO успешности ответа — 99%
    b) использование ресурсов
        i) Сценарий: использование CPU в простое
        ii) Сценарий: использование CPU в при RPS 100
    c) эффективность
    d) масштабируемость
        i) Сценарий: увеличение более 1к PRS, что нужно выделить?
6) Требования к поддержке
    a) сопровождение
    b) модульность
    c) расширяемость
        i) Сценарий: добавление функционала требующего транзакции между микросервисами

Также допускается дальше уточнять сценарии и добавлять в соответствии с этими деревьями:

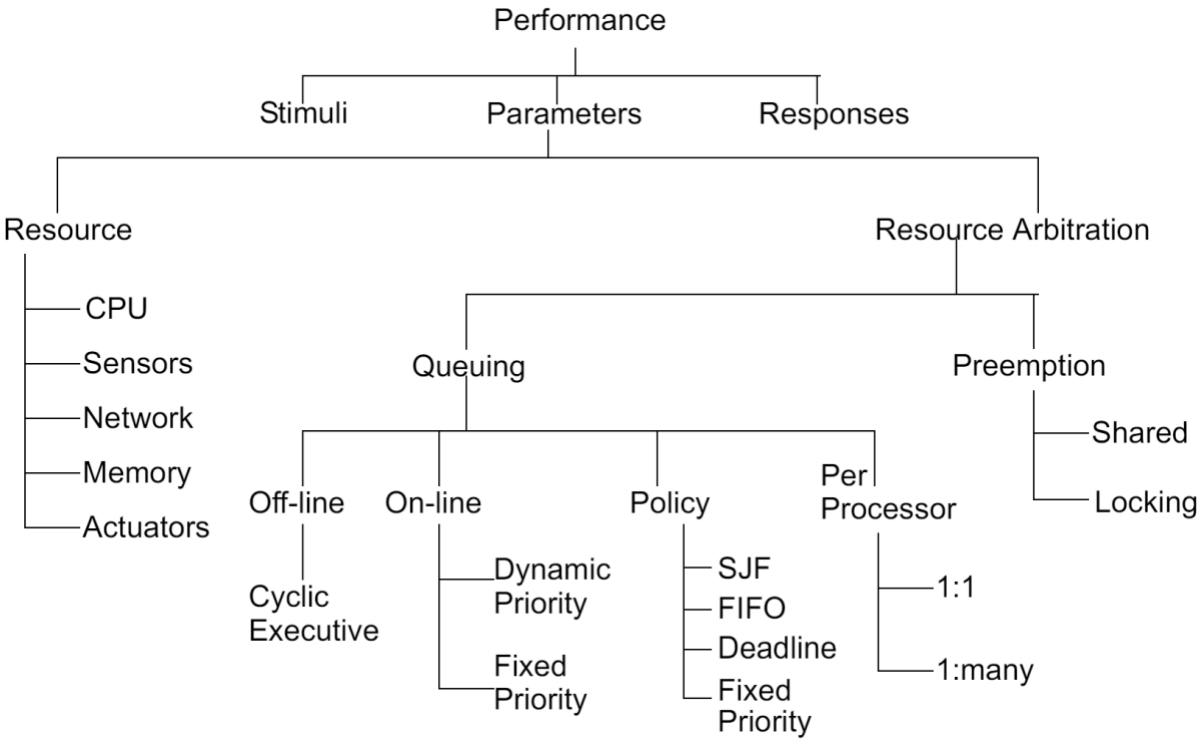
Уточняющие деревья характеристики (сценариев) производительности — Стимулы (Stimuli):



Дерево характеристики сценариев производительности — Ответы (Responses):



Дерево характеристики сценариев производительности — Архитектурные решения (Architectural Decisions/Parameters):


Performance
Stimuli
Parameters
Responses
Resource
CPU
Sensors
Network
Memory
Actuators
Resource Arbitration
Queuing
Preemption
Shared
Locking
Off-line
On-line
Policy
Per Processor
Cyclic Executive
Dynamic Priority
Fixed Priority
SJF
FIFO
Deadline
Fixed Priority
1:1
1:many

5. Набор рисков и не-рисков

* Риск — архитектурное решение, которое может привести к нежелательным последствиям в свете имеющихся требования по атрибутам качества.

Основные риски системы:

1) Разбиение системы Courses на отдельные микросервисы с Feedback:

   Описание: Разделение функционала системы Courses на отдельные микросервисы может привести к сложностям в управлении и согласовании изменений между ними. Возможны проблемы с целостностью данных и согласованностью операций.

   Риск: Увеличение сложности разработки, ухудшение производительности из-за частого обращения к разным микросервисам, возможные проблемы с консистентностью данных.

   Компромисс: Легкость масштабирования отдельных компонентов системы, возможность быстрой разработки и внедрения новых функций благодаря модульной архитектуре.

2) Использование только HTTP в качестве стандарта транспортной модели для общения между фронтенд-составляющей и серверной частью:

   Описание: Ограничение только использования HTTP может быть недостаточным для обеспечения оптимальной производительности и расширяемости системы, особенно при увеличении нагрузки или необходимости взаимодействия с другими протоколами.

   Риск: Ограничение функциональности, недостаточная производительность, сложности при взаимодействии с другими сервисами или системами.

   Компромисс: Простота разработки и обслуживания, универсальность и распространенность протокола HTTP.

3) Использование собственного Gateway вместо NGINX:

   Описание: Разработка и поддержка собственного API Gateway может потребовать значительных усилий и ресурсов, особенно в сравнении с использованием уже существующего и проверенного решения, такого как NGINX.

   Риск: Увеличение времени разработки и поддержки, риск возникновения ошибок в собственном решении, возможные проблемы с масштабируемостью и производительностью.

   Компромисс: Полный контроль над функциональностью и конфигурацией, возможность адаптации под конкретные потребности системы.

4) Использование Docker в качестве внутренней сети вместо Kubernetes:

Описание: Использование Docker в качестве внутренней сети может быть менее эффективным по сравнению с оркестраторами контейнеров, такими как Kubernetes, особенно при необходимости управления множеством контейнеров и автоматизации процессов.

Риск: Усложнение управления контейнерами, возможные проблемы с масштабируемостью и надежностью, увеличение сложности конфигурации и обслуживания.

Компромисс: Простота развертывания и управления контейнерами, низкий порог входа для новых разработчиков, гибкость конфигурации среды разработки и тестирования.

5) Выбор микросервисной архитектуры:

Описание: Решение о выборе.

Чувствительность: решение сильно влияет на стоимость разработки размер и сложность большей части кодовой базы

Риск: Появление необходимых транзакций между микросервисами приведет к необходимости написания саг (а также написании хореографии или орекстрации), что потребует больших ресурсов.

Компромисс: Монолит, однако он не подойдет для написания на разных языках.

6) Использование микросервисной архитектуры:

Описание: Внедрение микросервисной архитектуры может привести к сложностям в управлении и координации между множеством независимых сервисов.

Возможные последствия:

- Усложнение конфигурации и развертывания, возможные проблемы с целостностью данных, увеличение нагрузки на сеть и снижение производительности из-за частых запросов между сервисами.
- Неудобно тестировать: сложность организации end-to-end тестирования из-за множества взаимодействующих сервисов.
- Неудобно использовать транзакции Saga: сложность обеспечения атомарности и согласованности изменений в нескольких сервисах.
- Повышенная сложность мониторинга и отладки: необходимость отслеживания работы множества независимых компонентов, что может усложнить обнаружение и исправление проблем.
- Дополнительные накладные расходы на сетевое взаимодействие: из-за большого количества запросов между сервисами возможно увеличение нагрузки на сеть и задержек в ответах.

Хорошие стороны выбора:

- Удобно разрабатывать по DDD.
- Расширение системы: возможность добавления новых функций и сервисов без изменения всей системы.
- Балансировка нагрузки: возможность равномерного распределения нагрузки между сервисами.
- Отказоустойчивость: снижение влияния отказа одного сервиса на работу всей системы.

7) Использование языка Go в микросервисах:

Описание: Использование Go может обеспечить высокую производительность и эффективное использование ресурсов, но может также внести определенные сложности из-за особенностей языка.

Возможные последствия:

Плюсы:

- Высокая производительность: Go известен своей скоростью выполнения и низким потреблением ресурсов, что особенно важно для микросервисов.
- Простота конкурентного программирования: встроенная поддержка goroutines и channels упрощает реализацию асинхронных операций и параллельных вычислений.

Минусы:

- Относительная новизна: недостаток опыта и ресурсов по сравнению с более устоявшимися языками программирования может вызвать проблемы с обучением и поддержкой разработчиков.
- Ограниченные возможности стандартной библиотеки: Go имеет компактную стандартную библиотеку, что может потребовать использования сторонних библиотек для некоторых задач.

8) Использование языка Java в микросервисах:

Описание: Java широко используется в микросервисных архитектурах благодаря своей надежности, масштабируемости и богатой экосистеме инструментов.

Возможные последствия:

Плюсы:

- Обширная экосистема: Java имеет огромное количество библиотек и инструментов, что облегчает разработку и поддержку микросервисов.
- Широкая поддержка: Java является одним из самых популярных языков программирования, что обеспечивает доступность разработчиков и обширное сообщество для поддержки.

Минусы:

- Значительные накладные расходы: из-за управления виртуальной машиной Java (JVM) и большого объема памяти Java-программы могут потреблять больше ресурсов, чем программы, написанные на других языках.
- Длинные времена развертывания: запуск и инициализация JVM могут занять значительное время, что может повлиять на время отклика микросервиса.

9) Использование языка Python в микросервисах:

Описание: Python известен своей простотой, гибкостью и широким применением в различных областях, но может также вносить определенные сложности из-за динамической типизации и управления памятью.

Возможные последствия:

Плюсы:

- Простота и выразительность: Python обладает простым и интуитивно понятным синтаксисом, что упрощает разработку и поддержку кода.
- Большое сообщество: Python имеет активное сообщество разработчиков и обширную коллекцию библиотек и фреймворков, что облегчает разработку микросервисов.

Минусы:

- Низкая производительность: Python может быть медленнее других языков из-за интерпретации кода и управления памятью.
- Ограниченная поддержка для многопоточности: из-за GIL (Global Interpreter Lock) многопоточность в Python может быть менее эффективной по сравнению с другими языками.

10) Использование чистой архитектуры в микросервисах:

Описание: Чистая архитектура обеспечивает разделение системы на слои с четко определенными зависимостями, что облегчает тестирование, поддержку и масштабирование.

Возможные последствия:

Плюсы:

- Улучшенная структурированность: чистая архитектура способствует выделению основных бизнес-логики и уменьшает зависимость от конкретных технологий.
- Улучшенная тестируемость: разделение системы на слои позволяет проводить модульное и интеграционное тестирование отдельных компонентов.

Минусы:

- Дополнительные накладные расходы на разработку: внедрение чистой архитектуры может потребовать дополнительных усилий и ресурсов на проектирование и реализацию.
- Усложнение начального этапа разработки: необходимость строго следовать принципам чистой архитектуры может вызвать задержки на начальном этапе разработки.

Не-риски:

1) Использование микросервисной архитектуры и DDD:

Описание: Использование микросервисной архитектуры и принципов DDD может способствовать улучшению масштабируемости, гибкости и модульности системы.

Не-риск: Стандартное решение по повышению гибкости разработки, улучшение расширяемости и поддерживаемости системы.

Хорошие стороны: Легкость масштабирования отдельных компонентов системы, возможность быстрой разработки и внедрения новых функций благодаря модульной архитектуре.

2) Использование Prometheus, Grafana, Open Telemetry + Jaeger для мониторинга и трассировок:

Описание: Использование набора инструментов для мониторинга и трассировки позволит обеспечить высокую надежность и производительность системы, а также облегчит выявление и устранение проблем.

Не-риск: Улучшение отслеживаемости и аналитики, повышение надежности и производительности системы.

Хорошие стороны: Централизованное управление мониторингом и трассировкой, быстрая диагностика проблем и улучшение качества обслуживания пользователей.

3) Использование различных баз данных (MongoDB, PostgreSQL):

Описание: Использование различных типов баз данных позволит выбирать наиболее подходящее хранилище данных для конкретных задач, обеспечивая гибкость и оптимальную производительность.

Не-риск: Обеспечение гибкости и масштабируемости хранения данных, оптимизация производительности и надежности.

Хорошие стороны: Поддержка разнообразных типов данных и запросов, возможность выбора наиболее эффективного хранилища для каждого конкретного случая.

6. Набор рискованных тем проекта

> Прослеживается одна основная рискованная тема — проблема наличия кодовой базы на разных языках и усложнение системы произошедшее вследствие использования микросервисной архитектуры, в особенности для транзакций, наблюдаемости и тестирования.

Также Bus Factor и проблемы неимения k8s и проработанной другой инфраструктуры.

7. Отображение архитектурных решений на требования по качеству

Атрибуты качества:

- Производительность
- Расширяемость
- Сопровождение

Решение:

- Микросервисы

Атрибуты качества:

- Производительность
  - Эффективность
- Расширяемость
- Сопровождение
- Bus Factor

Решение:

- Выбор языка Java

Атрибуты качества:

- Производительность
- Расширяемость
- Сопровождение
- Bus Factor

Решение:

- Выбор языка GO

Атрибуты качества:

- Поддержка
  - Расширяемость

Решение:

- Выбор Чистой архитектуры

Другие отображения по критическим сценариям в следующем пункте

8. Множество чувствительных и компромиссных точек в проекте

1) Сценарий: RPS 100, SLO времени ответа — 25 мс, SLO успешности ответа — 99%
   Атрибуты качества:
   Производительность
   Решения:
   Микросервисы | S1, C1, R1
   HTTP | S1, R1

S1 решение очень чувствительно, так как напрямую коррелирует с стоимостью разработки и многими атрибутами производительности.
R1 риск, что если увеличится цепочка запросов для ответа на эндпоинт сервиса курсов, то время ответа превысит 25 мс, так как в среднем одно обращение по сети прибавляет 5 мс обработки.
C1 выражает выбор микросервисов для возможности написать систему на разных языках в текущей команде

S2 решение менее чувствительно, так как разница между принятыми протоколами не столь большая для среднего по характеристикам запроса.
R2 Риск, что при появлении больших в размере сущностей , например урока с большим текстом, мы не сможем быстро гнать такие пакеты по HTTP.

2) Сценарий: использование CPU в простое

Атрибуты качества:
Эффективность
Решения:
Выбор языка Java | S1, C1, R1

S1 высокая чувствительность, это сильно влияет на бизнес драйвер стоимости железа
R1 риск, для небольших сервисов Java много требует CPU при небольшом количестве запросов относительно стоимости запроса (JMV не легковесна)
C1 разворачиваем на одном сервере в одном JVM, если возможно, но теряем плюсы расположения на разных серверах и невозможность развертки кубера для такого компромисса.

3) Сценарий: увеличение более 1к PRS, сколько нужно выделить мощностей

Атрибуты качества:
Производительность — масштабируемость
Решения:
Выбор docker вместо k8s | S1, R1

S1, на данном этапе это не очень чувствительно, однако важно учесть в соответствии с планируемым ростом
R1 риск, при резком увеличении требований к производительности по количеству запросов для сервисом на Python и в меньшей степени для сервисов на Java придется писать механизм репликации или управления несколькими инстансами сервисов (один сервис на Python выдержит до 3к подключений за раз без FastAPI)

4) Сценарий:

   Пользователь переводит деньги и в нашей системе моргает сеть.

   Пользователь переводит деньги и в нашей системе падает питание.

Атрибуты качества:

Отказоустойчивость

Решения:

Паттерн transactional outbox | S1, NR1

S1, очень чувствительно так как у заказчика нету ресурсов на карды пользовательской поддержки
NR1 не риск, выбрано отличное решение которое хорошо масштабируется, выводит отчетность происходящего и легко мониториться.

5) Сценарий: добавление функционала требующего транзакции между микросервисами

Атрибуты качества:

Расширяемость

Решения:

Микросервисная архитектура S1, R1, 1.C1

S1 высокая чувствительность, это влияет на стоимость и скорость будущей разработки (TTP time to production новой фичи)
R1 риск, для такой функции в системе не продуманы системы управления транзакциями. Придется писать оркестрацию или хореографию впервые и это будет долго и дорого.

10. Итог

Система имеет много чувствительных точек и рисков из-за компромисса на возможность включения команд на разных языках в проект.
Микросервисы в текущем окружении также несут риск, поэтому стоит учесть доработки для этих проблем в документе доработки.